User Manual

for MPC574XG FLS Driver

Document Number: UM35FLSASR4.2 Rev0002R1.0.0
Rev. 1.0.0

Contents

Section number	Title	Page

Chapter 1 Revision History

Table 1-1. Revision History

Revision	Date	Author	Description
1.0.0	17-Feb-2017	Tien Huu Nguyen	Initial Version for Calypso RTM 1.0.0 ASR 4.2 Release.

Chapter 2 Introduction

This User Manual describes NXP Semiconductor AUTOSAR Flash (FLS) driver for MPC574XG.

AUTOSAR FLS driver configuration parameters and deviations from the specification are described in FLS Driver chapter of this document. AUTOSAR FLS driver requirements and APIs are described in the AUTOSAR FLS driver software specification document.

2.1 Supported Derivatives

The software described in this document is intented to be used with the following microcontroller devices of NXP Semiconductor .

Table 2-1. MPC574XG Derivatives

NXP Semiconductor	MPC5748G_LQFP176, MPC5748G_MAPBGA256, MPC5748G_MAPBGA324, MPC5747G_LQFP176, MPC5747G_MAPBGA256, MPC5747G_MAPBGA324, MPC5746G_LQFP176, MPC5746G_MAPBGA256, MPC5746G_MAPBGA324, MPC5748C_LQFP176, MPC5748C_MAPBGA256, MPC5748C_MAPBGA324, MPC5747C_LQFP176, MPC5747C_MAPBGA256, MPC5747C_MAPBGA324, MPC5746C_LQFP176, MPC5746C_MAPBGA256, MPC5746C_MAPBGA324, MPC5746C_MAPBGA100, MPC5745C_LQFP176, MPC5745C_MAPBGA256, MPC5745C_MAPBGA100, MPC5744C_LQFP176, MPC5744C_MAPBGA256,

Table 2-1. MPC574XG Derivatives

	MPC5744C_MAPBGA100, MPC5746B_LQFP176, MPC5746B_MAPBGA256, MPC5746B_MAPBGA100, MPC5744B_LQFP176, MPC5744B_MAPBGA256, MPC5744B_MAPBGA100, MPC5745B_LQFP176, MPC5745B_MAPBGA256, MPC5745B_MAPBGA100

All of the above microcontroller devices are collectively named as MPC574XG .

2.2 Overview

AUTOSAR (AUTomotive Open System ARchitecture) is an industry partnership working to establish standards for software interfaces and software modules for automobile electronic control systems.

AUTOSAR

- paves the way for innovative electronic systems that further improve performance, safety and environmental friendliness.
- is a strong global partnership that creates one common standard: "Cooperate on standards, compete on implementation".
- is a key enabling technology to manage the growing electrics/electronics complexity. It aims to be prepared for the upcoming technologies and to improve cost-efficiency without making any compromise with respect to quality.
- facilitates the exchange and update of software and hardware over the service life of the vehicle.

2.3 About this Manual

This Technical Reference employs the following typographical conventions:

Boldface type: Bold is used for important terms, notes and warnings.

Italic font: Italic typeface is used for code snippets in the text. Note that C language modifiers such "const" or "volatile" are sometimes omitted to improve readability of the presented code.

Notes and warnings are shown as below:

Note

This is a note.

2.4 Acronyms and Definitions

Table 2-2. Acronyms and Definitions

Term	Definition
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
DEM	Diagnostic Event Manager
DET	Default Error Tracer
ECC	Error Correcting Code
VLE	Variable Length Encoding
N/A	Not Applicable
MCU	Micro Controller Unit
ECU	Electronic Control Unit
EEPROM	Electrically Erasable Programmable Read-Only Memory
FEE	Flash EEPROM Emulation
FLS	Flash
XML	Extensible Markup Language

2.5 Reference List

Table 2-3. Reference List

#	Title	Version
1	AUTOSAR 4.2 Rev0002FLS Driver Software Specification Document.	V4.2.2
2	MPC5748G Reference Manual	Rev. 5, 12/2016
3	MPC5746C Reference Manual	Rev. 4, 12/2016
4	MPC5748G_1N81M_Rev.2 (official document) (1N81M)	Jun-16
5	MPC5748G_1N81M_0N78S_Comparison_Summary_v2_0 (internal document) (1N81M, 0N78S)	31.10.2016
6	MPC5746C_1N06M_Rev.4 (official document) (1N06M)	Jul-16
7	MPC5746C_cut1.1_cut2.0_cut2.1_comparison_v0 (internal document) (1N06M, 0N84S, 1N84S)	14-Sep-16
8	C3M_cut2.1_new_errata_20170113 (internal document) (1N84S)	13-Jan-17

Chapter 3
Driver

3.1 Requirements

Requirements for this driver are detailed in the AUTOSAR 4.2 Rev0002FLS Driver Software Specification document (See Table Reference List).

3.2 Driver Design Summary

- Linear Address.

The FLS driver provides services for reading, writing and erasing flash memory and it combines configured flash memory sectors into one linear address space.

The FLS module shall combine all available flash memory areas into one linear address space, it will always start at address 0 and continues without any gap.

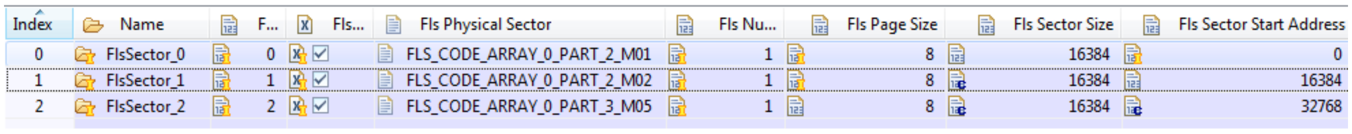
Example:

Suppose user want to configure following sectors:

Table 3-1. Sectors details Example

FlsPhysicalSector	Fls Physical Start Address	Fls Sector Size
FLS_CODE_ARRAY_0_PART_1_L02	0xFD0000	0x8000
FLS_CODE_ARRAY_0_PART_0_L04	0xFE0000	0x10000

The FlsSector List should be configured in the following way:



Index	Name	F...	Fls...	Fls Physical Sector	Fls Nu...	Fls Page Size	Fls Sector Size	Fls Sector Start Address
0	FlsSector_0	0	✓	FLS_CODE_ARRAY_0_PART_2_M01	1	8	16384	0
1	FlsSector_1	1	✓	FLS_CODE_ARRAY_0_PART_2_M02	1	8	16384	16384
2	FlsSector_2	2	✓	FLS_CODE_ARRAY_0_PART_3_M05	1	8	16384	32768

Figure 3-1. Fls Sector List

As you can see "Fls Sector Start Address" for FlsSector_0 will be 0 and "Fls Sector Start Address" for FlsSector_1 will be 0x4000 (16384)

If user want to write FLS_CODE_ARRAY_0_PART_2_M02, user need to write to the logical address 0x4000 - 0x7FFF (32767).

If user want to erase it, user need to erase sector from address 0x4000 with size 0x4000.

Note: The user do not need to calculate the "Fls Sector Start Address" and "Fls Sector Size" they can be automatically computed.

- Page Programming Size.

The FLS driver for the platform MPC574XG supports three different write.

- Double Word Write (8 bytes) = FLS_WRITE_DOUBLE_WORD

- Page Write (32 bytes) = FLS_WRITE_PAGE

- Quad Page Write (128 bytes) = FLS_WRITE_QUAD_PAGE

User has to select any one option from the above as desired for user's application. The Default value of this is a Double Word Write (8 bytes).

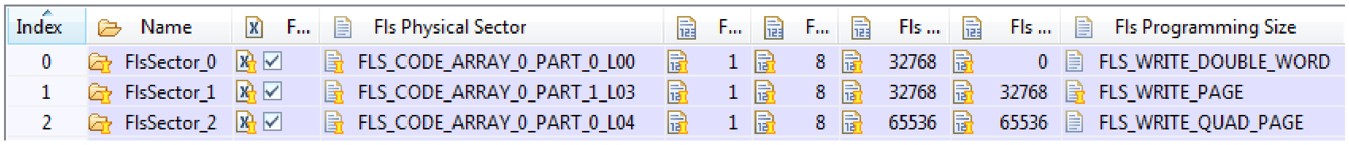
Example:

Suppose user want to configure following sectors:

Table 3-2. Page Programming Size Example

FlsPhysicalSector	Fls Page programming Size
FLS_CODE_ARRAY_0_PART_0_L00	FLS_WRITE_DOUBLE_WORD
FLS_CODE_ARRAY_0_PART_1_L03	FLS_WRITE_PAGE
FLS_CODE_ARRAY_0_PART_0_L04	FLS_WRITE_QUAD_PAGE

The FlsSector List should be configured in the following way:



Index	Name	F...	Fls Physical Sector	F...	F...	Fls ...	Fls ...	Fls Programming Size
0	FlsSector_0	✓	FLS_CODE_ARRAY_0_PART_0_L00	1	8	32768	0	FLS_WRITE_DOUBLE_WORD
1	FlsSector_1	✓	FLS_CODE_ARRAY_0_PART_1_L03	1	8	32768	32768	FLS_WRITE_PAGE
2	FlsSector_2	✓	FLS_CODE_ARRAY_0_PART_0_L04	1	8	65536	65536	FLS_WRITE_QUAD_PAGE

Figure 3-2. Fls Sector List

- Unlocking FLS Sectors.

The Flash memory physical sectors that are going to be modified by Fls driver (i.e. erase and write operations) have to be unlocked for a successful operation.

If it is not handled by FLS driver must be setup on an application level. Unlock only those physical sectors that will be modified by Fls driver operation.

It is recommended to configure only those FlsPhysicalSector(s) that are required by upper layer module FEE. As FLS driver can access only those configured FlsPhysicalSector(s) and can not modify rest of Flash address space.

Note: If the microcontroller is in user mode, be sure that the Flash memory controller registers are accessible.

For more information please refer to the 'Memory Protection Unit' and 'Register Protection' chapters in the device reference manual.

- Application's tasks.

It is responsibility of integrator/application to ensure that MCU-wide parameters like voltage supply etc. are according to and in limits specified in MCU documentation. Integrator/application is responsible to implement additional functionality that cancel any on-going erase/write Fls jobs if MCU conditions are not in such limits.

3.3 Hardware Resources

Available hardware resources.

3.3.1 MPC574XG Flash Banks/Arrays, Sectors details

MPC574XG has up to 6 MB of flash memory consisting of the main space and the independent 16KB block of one-time-programmable (OTP) flash memory included to support systems that require non-volatile memory for security features or system initialization information. The independent 16KB block is referred to as "UTest space".

Table 3-3. MPC574XG sectors details

Type	Sector name	Sector Size (KB)
FLS_CODE_ARRAY_0_PART_0_L00	L00	32

Table continues on the next page...

Table 3-3. MPC574XG sectors details (continued)

FLS_CODE_ARRAY_0_PART_0_L01	L01	32
FLS_CODE_ARRAY_0_PART_1_L02	L02	32
FLS_CODE_ARRAY_0_PART_1_L03	L03	32
FLS_CODE_ARRAY_0_PART_0_L04	L04	64
FLS_CODE_ARRAY_0_PART_1_L05	L05	64
FLS_CODE_ARRAY_0_PART_2_M00	M00	16
FLS_CODE_ARRAY_0_PART_2_M01	M01	16
FLS_CODE_ARRAY_0_PART_2_M02	M02	16
FLS_CODE_ARRAY_0_PART_2_M03	M03	16
FLS_CODE_ARRAY_0_PART_3_M04	M04	16
FLS_CODE_ARRAY_0_PART_3_M05	M05	16
FLS_CODE_ARRAY_0_PART_3_M06	M06	16
FLS_CODE_ARRAY_0_PART_3_M07	M07	16
FLS_CODE_ARRAY_0_PART_2_M08	M08	32
FLS_CODE_ARRAY_0_PART_3_M09	M09	32
FLS_CODE_ARRAY_0_PART_6_LG00	LG00	256
FLS_CODE_ARRAY_0_PART_6_LG01	LG01	256
FLS_CODE_ARRAY_0_PART_6_LG02	LG02	256
FLS_CODE_ARRAY_0_PART_6_LG03	LG03	256
FLS_CODE_ARRAY_0_PART_6_LG04	LG04	256
FLS_CODE_ARRAY_0_PART_6_LG05	LG05	256
FLS_CODE_ARRAY_0_PART_6_LG06	LG06	256
FLS_CODE_ARRAY_0_PART_6_LG07	LG07	256
FLS_CODE_ARRAY_0_PART_7_LG08	LG08	256
FLS_CODE_ARRAY_0_PART_7_LG09	LG09	256
FLS_CODE_ARRAY_0_PART_7_LG10	LG10	256
FLS_CODE_ARRAY_0_PART_7_LG11	LG11	256
FLS_CODE_ARRAY_0_PART_7_LG12	LG12	256
FLS_CODE_ARRAY_0_PART_7_LG13	LG13	256
FLS_CODE_ARRAY_0_PART_7_LG14	LG14	256
FLS_CODE_ARRAY_0_PART_7_LG15	LG15	256
FLS_CODE_ARRAY_0_PART_8_LG16	LG16	256
FLS_CODE_ARRAY_0_PART_8_LG17	LG17	256
FLS_CODE_ARRAY_0_PART_8_LG18	LG18	256
FLS_CODE_ARRAY_0_PART_9_LG19	LG19	256
FLS_CODE_ARRAY_0_PART_9_LG20	LG19	256
FLS_CODE_ARRAY_0_PART_9_LG21	LG21	256
FLS_UTEST_ARRAY_0_PART_0_L00	L00	16
FLS_CODE_ARRAY_0_PART_0_SHSM00	SHSM00	16
FLS_CODE_ARRAY_0_PART_0_HSM02	HSM02	64

Table continues on the next page...

Table 3-3. MPC574XG sectors details (continued)

FLS_CODE_ARRAY_0_PART_1_HSM03	HSM03	64
FLS_DATA_ARRAY_0_PART_4_HSM00	HSM00	16
FLS_DATA_ARRAY_0_PART_5_HSM01	HSM01	16

3.3.2 Flash memory physical sectors unlock example

Below is the code example that unlocks all Flash memory physical sectors.

```
void Fls_SectorsUnlock(void)
{
    /* FLASHMEM0_CF0_A unprotect/unlock */
    *((volatile uint32*)(0xFFFE0010)) = 0xC000FFFC; /* FLASHMEM_LOCK0 */
    *((volatile uint32*)(0xFFFE0014)) = 0xFFFFFFF0; /* FLASHMEM_LOCK1 */
    *((volatile uint32*)(0xFFFE0018)) = 0xFFFFC000; /* FLASHMEM_LOCK2 */

}
```

3.4 Deviation from Requirements

The driver deviates from the AUTOSAR FLS Driver software specification in some places.

There are also some additional requirements (on top of requirements detailed in AUTOSAR FLS Driver software specification) which need to be satisfied for correct operation.

Table 3-4. Deviations Status Column Description

Term	Definition
N/A	Not Available
N/T	Not Testable
N/S	Out of Scope
N/I	Not Implemented
N/F	Not Fully Implemented
I/D	Implemented with Deviation

Below table identifies the AUTOSAR requirements that are not fully implemented, implemented differently, or out of scope for the driver.

Table 3-5. Driver Deviations Table

SW Requirement ID	Status	Description	Notes
SWS_Fls_00107	N/F	The Fls module shall comply with the following file structure: Figure1 File include structure (see Figure image2.emf)	Not fully compliant
SWS_Fls_00004	N/F	(see Table Table_d2e28879.html)	FLS_E_TIMEOUT not implemented; FLS_E_UNEXPECTED_FLASH_ID is applicable only for external Fls driver
SWS_Fls_00015	I/D	If development error detection for the module Fls is enabled: the function Fls_Init shall check the (hardware specific) contents of the given configuration set <continue>	The check of the CRC computed over selected parameters of the configuration set is done independently of the development error detection setting. The setting itself just controls the DET reporting (FLS_E_PARAM_CONFIG).
SWS_Fls_00319	N/A	The production error code FLS_E_UNEXPECTED_FLASH_ID shall be reported when the expected flash ID is not matched (see FLS144).	Applicable only for external Fls driver
SWS_Fls_00144	N/A	During the initialization of the external flash driver, the FLS module shall check the hardware ID of the external flash device against the corresponding published parameter. <continue>	Applicable only for external Fls driver
SWS_Fls_00272	I/D	If development error detection for the module Fls is enabled: the function Fls_MainFunction shall provide a timeout monitoring for the currently running job, that is it shall supervise the deadline of the read / compare / erase or write job.	The timeout monitoring is provided independently of development error detection setting. Instead its own pre-compile switch is provided (see Form FlsTimeouts). Addtionally, when properly enabled, the DEM event respective to the failed operation (FLS_E_ERASE_FAILED or FLS_E_WRITE_FAILED) is reported.
SWS_Fls_00359	I/D	If development error detection for the module Fls is enabled: the function Fls_MainFunction shall check, whether the configured maximum erase time <continue>	The same as for SWS_Fls_00272.
SWS_Fls_00360	I/D	If development error detection for the module Fls is enabled: the function Fls_MainFunction shall check, whether the expected maximum write time <continue>	The same as for SWS_Fls_00272.
SWS_Fls_00361	I/D	The development error code FLS_E_TIMEOUT shall be reported when the timeout supervision of a read, write, erase or compare job failed.	The same as for SWS_Fls_00272. See also SWS_Fls_00362's Note.
SWS_Fls_00362	I/D	If development error detection for the module Fls is enabled: the function Fls_MainFunction shall check, whether the expected maximum read / compare <continue>	Timeout check was implemented only for the operations whose termination is HW-dependent (erase/write).
SWS_Fls_00215	N/F	The FLS module's flash access routines shall only disable interrupts and wait for the completion of the erase / write command if necessary (that is if it has to be ensured that no other code is executed in the meantime).	Only RTE plug-in has the ability to enable/disable interrupts. Additionaly there is possibility to alter default behaviour and have Erase/Write jobs asynchronous, i.e. Fls_MainFunction function doesn't wait

Table continues on the next page...

Table 3-5. Driver Deviations Table (continued)

SW Requirement ID	Status	Description	Notes
			(block) for completion of the erase sector/ page write operation(s).
SWS_Fls_00217	N/A	The FLS module shall add a device specific base address to the address type Fls_AddressType if necessary.	Unclear concept: device specific base address Not used
SWS_Fls_00208	N/F	The FLS module shall combine all available flash memory areas into one linear address space (denoted by the parameters FlsBaseAddress and FlsTotalSize).	Unclear Pourpose. FlsBaseAddress and FlsTotalSize not used. Impacted requitements are: SWS_Fls_00221, SWS_Fls_00020, SWS_Fls_00226, SWS_Fls_00026, SWS_Fls_00239, SWS_Fls_00097, SWS_Fls_00244, SWS_Fls_00150,
ECUC_Fls_00169	N/I	FlsBaseAddress	FlsBaseAddress not used. Unclear purpose
ECUC_Fls_00170	N/I	FlsTotalSize	FlsTotalSize not used. Unclear purpose
SWS_Fls_00145	N/I	If possible, e.g. with interrupt controlled implementations, the FLS module shall start the first round of the erase job directly within the function Fls_Erase to reduce overall runtime.	Not applicable. Not supported by hardware. Hardware has no related interrupts.
SWS_Fls_00146	N/I	If possible, e.g. with interrupt controlled implementations, the FLS module shall start the first round of the write job directly within the function Fls_Write to reduce overall runtime.	Not applicable. Not supported by hardware. Hardware has no related interrupts.
SWS_Fls_00247	N/F	If source code for caller and callee of the function Fls_GetVersionInfo is available, the FLS module should realize this function as a macro. The FLS module should define this macro in the module's header file.	The function will be implemented as function, not as a macro.
SWS_Fls_00040	N/F	The function Fls_MainFunction 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 FLS module's operating mode (normal, fast).	For Erase job not applicable as whole sector(s) is(are) erased.
SWS_Fls_00022	I/D	If development error detection for the module Fls is enabled: After a flash block has been erased, the function Fls_MainFunction shall compare <continue>	Functionality available if both FlsDevErrorDetect and FlsEraseBlankCheck configured to true. If only FlsEraseBlankCheck configured to true the DET error is not reported but Fls job ends with MEMIF_JOB_FAILED.
SWS_Fls_00055	I/D	If development error detection for the module Fls is enabled: Before writing a flash block, the function Fls_MainFunction shall compare <continue>	Functionality available if both FlsDevErrorDetect and FlsWriteBlankCheck configured to true. If only FlsWriteBlankCheck configured to true the DET error is not reported but Fls job ends with MEMIF_JOB_FAILED.
SWS_Fls_00056	I/D	If development error detection for the module Fls is enabled:: After writing a flash block, the function Fls_MainFunction shall compare <continue>	Functionality available if both FlsDevErrorDetect and FlsWriteVerifyCheck configured to true. If only FlsWriteVerifyCheck configured to true the

Table continues on the next page...

Table 3-5. Driver Deviations Table (continued)

SW Requirement ID	Status	Description	Notes
			DET error is not reported but Fls job ends with MEMIF_JOB_FAILED.
SWS_Fls_00232	N/I	The configuration parameter FlsUseInterrupts shall switch between interrupt and polling controlled job processing if this is supported by the flash memory hardware.	Not applicable. Not supported by hardware.
SWS_Fls_00233	N/I	The FLS module's implementer shall locate the interrupt service routine in Fls_Irq.c.	Not applicable. Not supported by hardware.
SWS_Fls_00234	N/I	If interrupt controlled job processing is supported and enabled with the configuration parameter FlsUseInterrupts, the interrupt service routine shall <continue>	Not applicable. Not supported by hardware.
ECUC_Fls_00292	N/I	FlsUseInterrupts	Not applicable. Not supported by hardware.
SWS_Fls_00196	N/I	The function Fls_MainFunction shall at the most issue one sector erase command (to the hardware) in each cycle.	Implementation now erases only one sector per cycle but the HW allows erasing of more physical sectors in parallel (but the final erase time is not reduced).
ECUC_Fls_00306	N/I	FlsCallCycle	FlsCallCycle not used, unclear purpose.
ECUC_Fls_00280	N/I	FlsNumberOfSectors	FlsNumberOfSectors not used, unclear purpose.
ECUC_Fls_00279	N/I	FlsProtection {FLS_PROTECTION} Erase/write protection settings. Only relevant if supported by hardware. (see Table TableConf_d2e40923.html)	FlsProtection: not used. Replaced by Vendor specific parameter (see PR-MCAL-3158)
SWS_Fls_00302	N/I	The module's status, mode and the job result shall be made available for debugging (reading)	Support for Debugging shall not be implemented according to PR-MCAL-3330.fls.

As a deviation from standard:

Fls_[VariantName]_PBcfg.c files will contain the definition for all parameters that are variant aware, independent of the configuration class that will be selected (PC, LT, PB) Fls_Cfg.c, Fls_Cfg.h file will contain the definition for all parameters that are not variant aware

3.5 Driver limitations

None

3.6 Driver usage and configuration tips

Async/sync mode

3.6.1 Introduction

It's possible to modify the behavior of sector erase / page write using two configuration parameters (FlsSectorEraseAsynch, FlsPageWriteAsynch) in FlsSector TAB.

If FlsSectorEraseAsynch/FlsPageWriteAsynch are enabled sector erase / page write job in the Fls_MainFunction are executed asynchronously, it means that Fls_MainFunction will not wait (not blocking) for completion of high voltage operation.

If FlsSectorEraseAsynch/FlsPageWriteAsynch are disabled sector erase / page write job are executed synchronously, which means sector erase / page write job are blocking and any high voltage operation will be completed during one Fls_Mainfunction.

3.6.2 Avoiding RWW problem

To avoid RWW (Read While Write) problems the flash driver provide the FlsAcLoadOnJobStart configuration parameter. If it is set to true the Fls driver will load the flash access code routine to RAM whenever an erase or write job is started and unload (overwrite) it after that job has been finished or cancelled.

FlsAcLoadOnJobStart functionality can be used only in case of Sync Mode, in which case the flash access code is loaded to RAM and therefore the flash driver shouldn't have RWW problems; if FlsAcLoadOnJobStart is set to false the sector erased / page written must belong to flash array / partition different from flash array / partition the application is executing from.

In case of Async operations it is only possible to erase / write to flash array different from flash array the application is executing from.

Note:

1. The flash driver use the sector erase / page write access code to set the MCR:EHV bit and wait for completion of high voltage operation (and therefore incompatible with Async operation).

2. The flash module is further divided into eight partitions that determine locations for valid read-while-write (RWW) operations. While the embedded flash memory is performing a 'write' (program or erase) to a given partition, it can simultaneously perform a read from any other partition.

3. FlsAcCallback should be in RAM if FlsACLoadOnJob is true to avoid RWW problem.

3.7 Runtime Errors

The driver supports runtime generation of the errors listed in the Table Runtime Errors. The runtime error reporting can be disabled globally (see Form FlsGeneral).

Table 3-6. Runtime Errors

Function	Error Code	Condition triggering the error
Fls_Flash_AbortSuspended()	FLS_E_ERASE_FAILED	Abort of the ongoing erase operation failed (in case of detected timeout event).
Fls_Flash_AbortSuspended()	FLS_E_WRITE_FAILED	Abort of the ongoing write operation failed (in case of detected timeout event).
Fls_Flash_Init()	FLS_E_ERASE_FAILED	Abort of the ongoing erase operation failed (in case of detected timeout event).
Fls_Flash_Init()	FLS_E_ERASE_FAILED	Resuming the suspended erase operation failed.
Fls_Flash_Init()	FLS_E_WRITE_FAILED	Abort of the ongoing write operation failed (in case of detected timeout event).
Fls_Flash_Init()	FLS_E_WRITE_FAILED	Resuming the suspended write operation failed.
Fls_Flash_MainFunction()	FLS_E_ERASE_FAILED	Async Erase operation failed.
Fls_Flash_MainFunction()	FLS_E_ERASE_FAILED	Async Erase operation failed (in case of interleaved blocks).
Fls_Flash_MainFunction()	FLS_E_ERASE_FAILED	Async Erase operation failed (in case of detected timeout event).
Fls_Flash_MainFunction()	FLS_E_WRITE_FAILED	Async Write operation failed.
Fls_Flash_MainFunction()	FLS_E_WRITE_FAILED	Async Write operation failed (in case of detected timeout event).
Fls_Flash_SectorErase()	FLS_E_ERASE_FAILED	Erase operation cannot be executed.
Fls_Flash_SectorErase()	FLS_E_ERASE_FAILED	Erase operation cannot be executed (in case of interleaved blocks).
Fls_Flash_SectorErase()	FLS_E_ERASE_FAILED	Erase operation cannot be executed. Sector is locked, must be unlocked before an HV operation can be set
Fls_Flash_SectorErase()	FLS_E_ERASE_FAILED	Erase operation cannot be executed (in case of interleaved blocks).
Fls_Flash_SectorErase()	FLS_E_ERASE_FAILED	Sync Erase operation failed.
Fls_Flash_SectorErase()	FLS_E_ERASE_FAILED	Sync Erase operation failed (in case of interleaved blocks).

Table continues on the next page...

Table 3-6. Runtime Errors (continued)

Function	Error Code	Condition triggering the error
Fls_Flash_SectorErase()	FLS_E_ERASE_FAILED	Sync Erase operation failed (and previous two cases).
Fls_Flash_SectorWrite()	FLS_E_WRITE_FAILED	Write operation cannot be executed.
Fls_Flash_SectorWrite()	FLS_E_WRITE_FAILED	Write operation cannot be executed (in case of interleaved blocks).
Fls_Flash_SectorWrite()	FLS_E_WRITE_FAILED	Write operation cannot be executed. Sector is locked, must be unlocked before an HV operation can be set
Fls_Flash_SectorWrite()	FLS_E_WRITE_FAILED	Write operation cannot be executed (in case of interleaved blocks). Sector is locked, must be unlocked before an HV operation can be set
Fls_Flash_SectorWrite()	FLS_E_WRITE_FAILED	Sync Write operation failed.
Fls_Flash_SectorWrite()	FLS_E_WRITE_FAILED	Sync Write operation failed (in case of interleaved blocks).
Fls_DoJobRead()	FLS_E_READ_FAILED	A non correctable ECC error is present at read location.
Fls_DoJobCompare()	FLS_E_COMPARE_FAILED	A non correctable ECC error is present at read location.

3.8 Development Error Description

Table 3-7. Development Error Description

Error Code	Value	Condition triggering the error
FLS_E_PARAM_CONFIG	1	API service called with wrong parameter
FLS_E_PARAM_ADDRESS	2	u32TargetAddress is not in range and aligned to first byte of flash sector
FLS_E_PARAM_LENGTH	3	u32TargetAddress is not in range and aligned to last byte of flash sector
FLS_E_PARAM_DATA	4	NULL_PTR == SourceAddressPtr
FLS_E_UNINIT	5	API service called without module initialization
FLS_E_BUSY	6	API service called while driver still busy
FLS_E_VERIFY_ERASE_FAILED	7	Erase verification (blank check) failed
FLS_E_VERIFY_WRITE_FAILED	8	Write verification (compare) failed
FLS_E_PARAM_POINTER	10	NULL_PTR passed

3.9 Software specification

The following sections contains driver software specifications.

3.9.1 Define Reference

Constants supported by the driver are as per AUTOSAR FLS Driver software specification Version 4.2 Rev0002 .

3.9.2 Enum Reference

Enumeration of all constants supported by the driver are as per AUTOSAR FLS Driver software specification Version 4.2 Rev0002 .

3.9.3 Function Reference

Functions of all functions supported by the driver are as per AUTOSAR FLS Driver software specification Version 4.2 Rev0002 .

3.9.3.1 Function Fls_Cancel

Cancel an ongoing flash read, write, erase or compare job.

Details:

Abort a running job synchronously so that directly after returning from this function a new job can be started.

Pre: The module must be initialized.

Post: Fls_Cancelchanges module status andFls_eJobResultinternal variable.

Prototype: `void Fls_Cancel(void);`

3.9.3.2 Function Fls_Compare

Compares a flash memory area with an application data buffer.

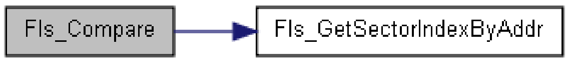


Figure 3-3. Function Fls_Compare References.

Details:

Starts a compare job asynchronously. The actual job is performed byFls_MainFunction.

Return: Std_ReturnType.

Pre: The module has to be initialized and not busy.

Post: Fls_Readchanges module status and some internal variables (Fls_u32JobSectorIt,Fls_u32JobAddrIt,Fls_u32JobAddrEnd,Fls_pJobDataSrcPtr,Fls_eJob,Fls_eJobResult).

Prototype: `Std_ReturnType Fls_Compare(Fls_AddressType u32SourceAddress, const uint8 *pTargetAddressPtr, Fls_LengthType u32Length);`

Table 3-8. Fls_Compare Arguments

Type	Name	Direction	Description
`Fls_AddressType`	u32SourceAddress	**input**	Source address in flash memory.
const uint8 *	pTargetAddressPtr	**input**	Pointer to source data buffer.
`Fls_LengthType`	u32Length	**input**	Number of bytes to compare.

Table 3-9. Fls_Compare Return Values

Name	Description
E_OK	Compare command has been accepted.
E_NOT_OK	Compare command has not been accepted.

3.9.3.3 Function Fls_BlankCheck

Verifies if a given memory area is erased and not yet programmed. The API is intended to be used on platforms on which reading an erased location is not supported or there is no fixed erased value.

Details:

Starts a check job asynchronously. The actual job is performed by Fls_MainFunction.

Return: Std_ReturnType.

Pre: The module has to be initialized and not busy.

Post: Fls_BlankCheck changes module status and some internal variables (Fls_u32JobSectorIt,Fls_u32JobAddrIt,Fls_u32JobAddrEnd,Fls_pJobDataSrcPtr,Fls_eJob,Fls_eJobResult).

Prototype: `Std_ReturnType Fls_BlankCheck(Fls_AddressType u32TargetAddress, Fls_LengthType u32Length);`

Table 3-10. Fls_BlankCheck Arguments

Type	Name	Direction	Description
Fls_AddressType	u32TargetAddress	**input**	Address in flash memory from which the blank check should be started.
`Fls_LengthType`	u32Length	**input**	Number of bytes to check.

Table 3-11. Fls_BlankCheck Return Values

Name	Description
E_OK	Check command has been accepted.
E_NOT_OK	Check command has not been accepted.

3.9.3.4 Function Fls_Erase

Erase one or more complete flash sectors.

Details:

Starts an erase job asynchronously. The actual job is performed by theFls_MainFunction.

Return: Std_ReturnType.

Pre: The module has to be initialized and not busy.

Post: Fls_Erasechanges module status and some internal variables (Fls_u32JobSectorIt,Fls_u32JobSectorEnd,Fls_eJob,Fls_eJobResult).

Prototype: `Std_ReturnType Fls_Erase(Fls_AddressType u32TargetAddress, Fls_LengthType u32Length);`

Table 3-12. Fls_Erase Arguments

Type	Name	Direction	Description
`Fls_AddressType`	u32TargetAddress	**input**	Target address in flash memory.
`Fls_LengthType`	u32Length	**input**	Number of bytes to erase.

Table 3-13. Fls_Erase Return Values

Name	Description
E_OK	Erase command has been accepted.
E_NOT_OK	Erase command has not been accepted.

3.9.3.5 Function Fls_GetJobResult

Returns the result of the last job.

Details:

Returns synchronously the result of the last job.

Return: MemIf_JobResultType.

Prototype: `MemIf_JobResultType Fls_GetJobResult(void);`

Table 3-14. Fls_GetJobResult Return Values

Name	Description
MEMIF_JOB_OK	Successfully completed job.
MEMIF_JOB_FAILED	Not successfully completed job.
MEMIF_JOB_PENDING	Still pending job (not yet completed).
MEMIF_JOB_CANCELED	Job has been canceled.
MEMIF_BLOCK_INCONSISTENT	Inconsistent block requested, it may contains corrupted data.
MEMIF_BLOCK_INVALID	Invalid block requested.

3.9.3.6 Function Fls_GetStatus

Returns the FLS module status.

Details:

Returns the FLS module status synchronously.

Return: MemIf_StatusType.

Prototype: `MemIf_StatusType Fls_GetStatus(void);`

Table 3-15. Fls_GetStatus Return Values

Name	Description
MEMIF_UNINIT	Module has not been initialized (yet).
MEMIF_IDLE	Module is currently idle.
MEMIF_BUSY	Module is currently busy.

3.9.3.7 Function Fls_GetVersionInfo

Returns version information about FLS module.

Details:

Version information includes:

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

Prototype: `void Fls_GetVersionInfo(Std_VersionInfoType *pVersionInfoPtr);`

Table 3-16. Fls_GetVersionInfo Arguments

Type	Name	Direction	Description
Std_VersionInfoType *	pVersionInfoPtr	**input, output**	Pointer to where to store the version information of this module.

3.9.3.8 Function Fls_Init

The function initializes Fls module.

Details:

The function sets the internal module variables according to given configuration set.

Pre: pConfigPtr must be NULL_PTR for PreCompile with 0 or 1 variant and must not be NULL_PTR for any other variants(PreCompile with more than one variant or PostBuild).

Pre: The module status must not be MEMIF_BUSY.

Prototype: `void Fls_Init(const Fls_ConfigType *pConfigPtr);`

Table 3-17. Fls_Init Arguments

Type	Name	Direction	Description
const`Fls_ConfigType`*	pConfigPtr	**input**	Pointer to flash driver configuration set.

3.9.3.9 Function Fls_MainFunction

Performs actual flash read, write, erase and compare jobs.

Details:

Bytes number processed per cycle depends by job type (erase, write, read, compare) current FLS module's operating mode (normal, fast) and write, erase Mode of Execution (sync, async).

Pre: The module has to be initialized.

Note

This function have to be called ciclically by the Basic Software Module; it will do nothing if there aren't pending job.

Prototype: `void Fls_MainFunction(void);`

3.9.3.10 Function Fls_Read

Reads from flash memory.

Details:

Starts a read job asynchronously. The actual job is performed byFls_MainFunction.

Return: MemIf_JobResultType.

Pre: The module has to be initialized and not busy.

Post: Fls_Readchanges module status and some internal variables (Fls_u32JobSectorIt,Fls_u32JobAddrIt,Fls_u32JobAddrEnd,Fls_pJobDataDestPtr,Fls_eJob,Fls_eJobResult).

Prototype: `Std_ReturnType Fls_Read(Fls_AddressType u32SourceAddress, uint8 *pTargetAddressPtr, Fls_LengthType u32Length);`

Table 3-18. Fls_Read Arguments

Type	Name	Direction	Description
`Fls_AddressType`	u32SourceAddress	**input**	Source address in flash memory.
`Fls_LengthType`	u32Length	**input**	Number of bytes to read.
uint8 *	pTargetAddressPtr	**output**	Pointer to target data buffer.

Table 3-19. Fls_Read Return Values

Name	Description
MEMIF_JOB_OK	Successfully completed job.
MEMIF_JOB_FAILED	Not successfully completed job.
MEMIF_JOB_PENDING	Still pending job (not yet completed).
MEMIF_JOB_CANCELED	Job has been canceled.
MEMIF_BLOCK_INCONSISTENT	Inconsistent block requested, it may contains corrupted data.
MEMIF_BLOCK_INVALID	Invalid block requested.

3.9.3.11 Function Fls_SetMode

Sets the FLS module's operation mode to the given Mode.

Details:

Every given mode determinates maximum bytes for read/write operations. Every mode has a set of pre-configured values.

Pre: The module has to be initialized and not busy.

Post: Fls_SetModechanges internal variablesFls_u32MaxReadandFls_u32MaxWrite.

Prototype: `void Fls_SetMode(MemIf_ModeType Mode);`

Table 3-20. Fls_SetMode Arguments

Type	Name	Direction	Description
MemIf_ModeType	Mode	**input**	MEMIF_MODE_FAST or MEMIF_MODE_SLOW.

3.9.3.12 Function Fls_Write

Write one or more complete flash pages to the flash device.

Details:

Starts a write job asynchronously. The actual job is performed byFls_MainFunction.

Return: Std_ReturnType.

Pre: The module has to be initialized and not busy.

Post: Fls_Writechanges module status and some internal variables (Fls_u32JobSectorIt,Fls_u32JobAddrIt,Fls_u32JobAddrEnd,Fls_pJobDataSrcPtr,Fls_eJob,Fls_eJobResult).

Prototype: `Std_ReturnType Fls_Write(Fls_AddressType u32TargetAddress, const uint8 *pSourceAddressPtr, Fls_LengthType u32Length);`

Table 3-21. Fls_Write Arguments

Type	Name	Direction	Description
`Fls_AddressType`	u32TargetAddress	**input**	Target address in flash memory.
const uint8 *	pSourceAddressPtr	**input**	Pointer to source data buffer.
`Fls_LengthType`	u32Length	**input**	Number of bytes to write.

Table 3-22. Fls_Write Return Values

Name	Description
E_OK	Write command has been accepted.
E_NOT_OK	Write command has not been accepted.

3.9.4 Structs Reference

Data structures supported by the driver are as per AUTOSAR FLS Driver software specification Version 4.2 Rev0002 .

3.9.5 Types Reference

Types supported by the driver are as per AUTOSAR FLS Driver software specification Version 4.2 Rev0002 .

3.10 Symbolic Names DISCLAIMER

All containers having the symbolic name tag set as true in the Autosar schema will generate defines like

#define <Container_Short_Name> <Container_ID>

For this reason it is forbidden to duplicate the name of such containers across the MCAL configuration, or to use names that may trigger other compile issues (e.g. match existing #ifdefs arguments).

Chapter 4
Tresos Configuration Plug-in

This chapter describes the Tresos configuration plug-in for the FLS Driver. The most of the parameters are described below.

4.1 Configuration elements of Fls

Included forms :

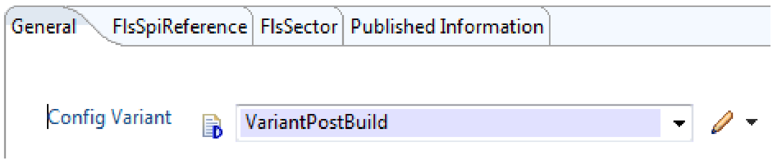
- IMPLEMENTATION_CONFIG_VARIANT
- NonAutosar
- FlsGeneral
- FlsPublishedInformation
- CommonPublishedInformation
- FlsConfigSet

Table 4-1. Revision table

Revision	Date
4.1.0	2010-12-03

4.2 Form IMPLEMENTATION_CONFIG_VARIANT

VariantPostBuild: Mix of precompile and postbuild time configuration parameters.



Config Variant VariantPostBuild

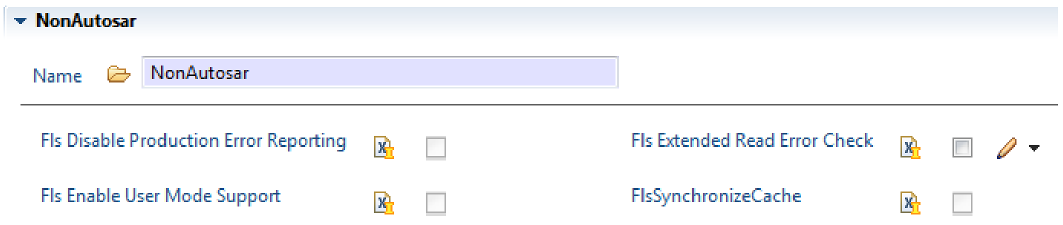
Figure 4-1. Tresos Plugin snapshot for IMPLEMENTATION_CONFIG_VARIANT form.

Table 4-2. Attribute IMPLEMENTATION_CONFIG_VARIANT detailed description

Property	Value
Label	Config Variant
Default	VariantPostBuild
Range	VariantPostBuild VariantPreCompile

4.3 Form NonAutosar

Container for general non-Autosar parameters of the flash driver. These parameters are always pre-compile.



Name	NonAutosar		
Fls Disable Production Error Reporting	☐	Fls Extended Read Error Check	☐
Fls Enable User Mode Support	☐	FlsSynchronizeCache	☐

Figure 4-2. Tresos Plugin snapshot for NonAutosar form.

4.3.1 FlsDisableDemReportErrorStatus (NonAutosar)

Compile switch to enable and disable the Diagnostic Error Reporting and Notification.

true: Diagnostic Error Reporting and Notification disabled.

false: Diagnostic Error Reporting and Notification enabled.

NOTE: DEM error reporting is marked as obsolete in FLS module starting from AUTOSAR 4.2.2, in consequence, this parameter is disabled.

Table 4-3. Attribute FlsDisableDemReportErrorStatus (NonAutosar) detailed description

Property	Value
Label	Fls Disable Production Error Reporting
Type	BOOLEAN
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	false

4.3.2 FlsExtendedReadErrorCheck (NonAutosar)

Compile switch to enable and disable the extended error read check

true: Every flash read will also verify RRE and RVE bits as a redundant error check

false: Flash read do not verify RRE and RVE bits for redundant error check.

Table 4-4. Attribute FlsExtendedReadErrorCheck (NonAutosar) detailed description

Property	Value
Label	Fls Extended Read Error Check
Type	BOOLEAN
Origin	Freescale
Symbolic Name	false
Default	false

4.3.3 FlsEnableUserModeSupport (NonAutosar)

When this parameter is enabled, the FLS module will adapt to run from User Mode, with the following measure : configuring REG_PROT for Fls IPs so that the registers under protection can be accessed from user mode by setting UAA bit in REG_PROT_GCR to 1 for more information and availability on this platform, please see chapter "User Mode Support" in IM.

true: FLS module will adapt to run from User Mode

false: FLS module will adapt to run from User Mode

Table 4-5. Attribute FlsEnableUserModeSupport (NonAutosar) detailed description

Property	Value
Label	Fls Enable User Mode Support
Type	BOOLEAN
Origin	Freescale
Symbolic Name	false
Default	false

4.3.4 FlsSynchronizeCache(NonAutosar)

Note: For the current platform, this feature is not supported. Only options 1. and 2. are available.

Synchronize the memory by invalidating the cache after each flash hardware operation.

The FLS driver needs to maintain the memory coherency by means of three methods:

1. Disable data cache, or

2. Configure the flash region upon which the driver operates, as non-cacheable, or

3. Enable the FlsSynchronizeCache feature.

Depending on the application configuration, one option may be more beneficial than other.

Enabled: The FLS driver will call Mcl cache API functions in order to invalidate the cache after each high voltage operation(write,erase) and before each read operation, in order to ensure that the cache and the modified flash memory are in sync.If enabled, the driver will attempt to invalidate only the modified lines from the cache.If the size of the region to be invalidated is greater than half of the cache size, then the entire cache is invalidated.

Note: If enabled, the MclLmemEnableCacheApi parameter has to be enabled and the MCL plugin included as a dependency.

Disabled: The upper layers have to ensure that the flash region upon which the driver operates is not cached. This can be obtained by either disabling the data cache or by configuring the memory region as non-cacheable.

true: Fls Synchronize Cache enabled.

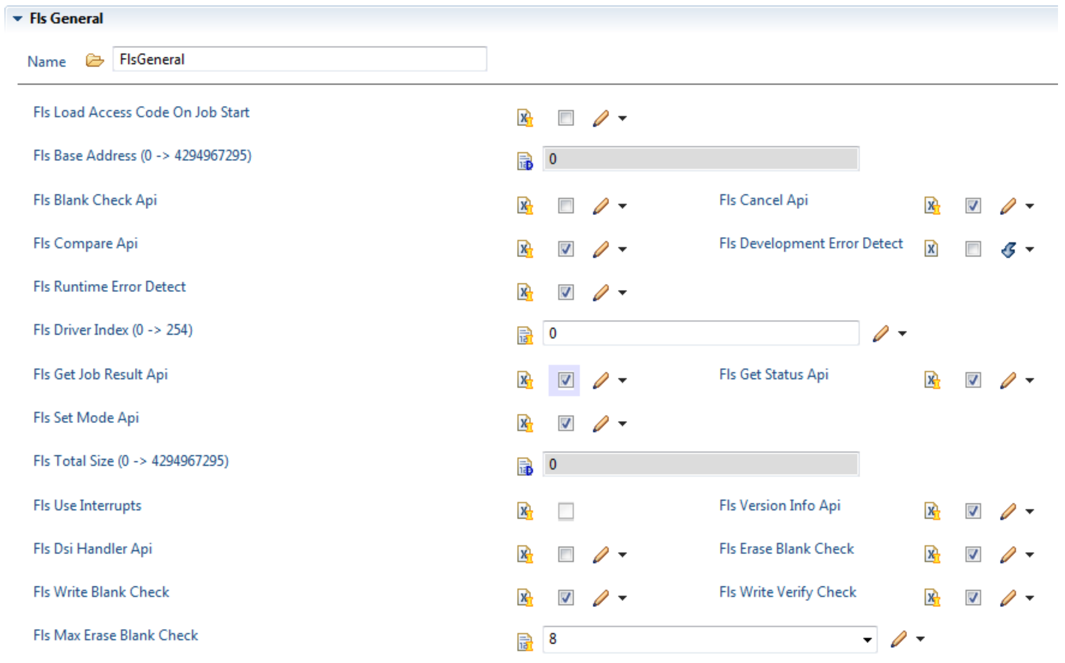
false: Fls Synchronize Cache disabled.

Table 4-6. Attribute FlsSynchronizeCache (NonAutosar) detailed description

Property	Value
Label	Fls Synchronize Cache
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	false

4.4 Form FlsGeneral

Container for general parameters of the flash driver. These parameters are always pre-compile.



Fls General

Name	FlsGeneral		
Fls Load Access Code On Job Start	☐		
Fls Base Address (0 -> 4294967295)	0		
Fls Blank Check Api	☐	Fls Cancel Api	✓
Fls Compare Api	✓	Fls Development Error Detect	☐
Fls Runtime Error Detect	✓		
Fls Driver Index (0 -> 254)	0		
Fls Get Job Result Api	✓	Fls Get Status Api	✓
Fls Set Mode Api	✓		
Fls Total Size (0 -> 4294967295)	0		
Fls Use Interrupts	☐	Fls Version Info Api	✓
Fls Dsi Handler Api	☐	Fls Erase Blank Check	✓
Fls Write Blank Check	✓	Fls Write Verify Check	✓
Fls Max Erase Blank Check	8		

Figure 4-3. Tresos Plugin snapshot for FlsGeneral form.

4.4.1 FlsAcLoadOnJobStart (FlsGeneral)

The flash driver shall load the flash access code to RAM whenever an erase or write job is started and unload (overwrite) it after that job has been finished or canceled.

true: Flash access code loaded on job start / unloaded on job end or error.

false: Flash access code not loaded to / unloaded from RAM at all.

Table 4-7. Attribute FlsAcLoadOnJobStart (FlsGeneral) detailed description

Property	Value
Label	Fls Load Access Code On Job Start
Type	BOOLEAN
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	false

4.4.2 FlsBaseAddress (FlsGeneral)

The flash memory start address (see also FLS118).

FLS169: This parameter defines the lower boundary for read / write / erase and compare jobs.

Note

Not needed / supported by the driver.

Table 4-8. Attribute FlsBaseAddress (FlsGeneral) detailed description

Property	Value
Label	Fls Base Address
Type	INTEGER_LABEL
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	0
Invalid	Range <=4294967295 >=0

4.4.3 FlsCancelApi (FlsGeneral)

Compile switch to enable and disable the Fls_Cancel function.

true: API supported / function provided.

false: API not supported / function not provided

Table 4-9. Attribute FlsCancelApi (FlsGeneral) detailed description

Property	Value
Label	Fls Cancel Api
Type	BOOLEAN
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	true

4.4.4 FlsCompareApi (FlsGeneral)

Compile switch to enable and disable the Fls_Compare function.

true: API supported / function provided.

false: API not supported / function not provided

Table 4-10. Attribute FlsCompareApi (FlsGeneral) detailed description

Property	Value
Label	Fls Compare Api
Type	BOOLEAN
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	true

4.4.5 FlsBlankCheckApi (FlsGeneral)

Compile switch to enable and disable the Fls_BlankCheck function.

true: API supported / function provided.

false: API not supported / function not provided.

Table 4-11. Attribute FlsBlankCheckApi (FlsGeneral) detailed description

Property	Value
Label	Fls Blank Check Api
Type	BOOLEAN
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	false

4.4.6 FlsRuntimeErrorDetect (FlsGeneral)

Pre-processor switch to enable and disable runtime error detection.

true: Runtime error detection enabled.

false: Runtime error detection disabled.

Table 4-12. Attribute FlsRuntimeErrorDetect (FlsGeneral) detailed description

Property	Value
Label	Fls Runtime Error Detect
Type	BOOLEAN

Table continues on the next page...

Table 4-12. Attribute FlsRuntimeErrorDetect (FlsGeneral) detailed description (continued)

Property	Value
Origin	Custom
Symbolic Name	false
Default	true

4.4.7 FlsDevErrorDetect (FlsGeneral)

Pre-processor switch to enable and disable development error detection (see FLS077).

true: Development error detection enabled.

false: Development error detection disabled.

Table 4-13. Attribute FlsDevErrorDetect (FlsGeneral) detailed description

Property	Value
Label	Fls Development Error Detect
Type	BOOLEAN
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	true

4.4.8 FlsDriverIndex (FlsGeneral)

Index of the driver, used by FEE.

Table 4-14. Attribute FlsDriverIndex (FlsGeneral) detailed description

Property	Value
Label	Fls Driver Index
Type	INTEGER
Origin	AUTOSAR_ECUC
Symbolic Name	true
Default	0
Invalid	Range <=254 >=0

4.4.9 FlsGetJobResultApi (FlsGeneral)

Compile switch to enable and disable the Fls_GetJobResult function.

true: API supported / function provided.

false: API not supported / function not provided

Table 4-15. Attribute FlsGetJobResultApi (FlsGeneral) detailed description

Property	Value
Label	Fls Get Job Result Api
Type	BOOLEAN
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	true

4.4.10 FlsGetStatusApi (FlsGeneral)

Compile switch to enable and disable the Fls_GetStatus function.

true: API supported / function provided.

false: API not supported / function not provided

Table 4-16. Attribute FlsGetStatusApi (FlsGeneral) detailed description

Property	Value
Label	Fls Get Status Api
Type	BOOLEAN
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	true

4.4.11 FlsSetModeApi (FlsGeneral)

Compile switch to enable and disable the Fls_SetMode function.

true: API supported / function provided.

false: API not supported / function not provided

Table 4-17. Attribute FlsSetModeApi (FlsGeneral) detailed description

Property	Value
Label	Fls Set Mode Api
Type	BOOLEAN
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	true

4.4.12 FlsTotalSize (FlsGeneral)

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.

Note

Not needed / supported by the driver.

Table 4-18. Attribute FlsTotalSize (FlsGeneral) detailed description

Property	Value
Label	Fls Total Size
Type	INTEGER_LABEL
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	0
Invalid	Range <=4294967295 >=0

4.4.13 FlsUseInterrupts (FlsGeneral)

Job processing triggered by hardware interrupt.

true: Job processing triggered by interrupt (hardware controlled)

false: Job processing not triggered by interrupt (software controlled)

Note

Not supported by hardware.

Table 4-19. Attribute FlsUseInterrupts (FlsGeneral) detailed description

Property	Value
Label	Fls Use Interrupts
Type	BOOLEAN
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	false

4.4.14 FlsVersionInfoApi (FlsGeneral)

Pre-processor switch to enable / disable the API to read out the modules version information.

true: Version info API enabled.

false: Version info API disabled.

Table 4-20. Attribute FlsVersionInfoApi (FlsGeneral) detailed description

Property	Value
Label	Fls Version Info Api
Type	BOOLEAN
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	true

4.4.15 FlsDsiHandlerApi (FlsGeneral)

Pre-processor switch to enable / disable the API to report data storage (ECC) errors to the flash driver.

true: Data storage handler API enabled.

false: Data storage handler API disabled.

Note

Vendor specific parameter

Table 4-21. Attribute FlsDsiHandlerApi (FlsGeneral) detailed description

Property	Value
Label	Fls Dsi Handler Api
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	false

4.4.16 FlsEraseBlankCheck (FlsGeneral)

Pre-processor switch to enable / disable the erase blank check. After a flash block has been erased, the erase blank check compares the contents of the addressed memory area against the value of an erased flash cell to check that the block has been completely erased.

true: Erase blank check enabled.

false: Erase blank check disabled.

Note

Vendor specific parameter

Table 4-22. Attribute FlsEraseBlankCheck (FlsGeneral) detailed description

Property	Value
Label	Fls Erase Blank Check
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	true

4.4.17 FlsWriteBlankCheck (FlsGeneral)

Pre-processor switch to enable / disable the write blank check. Before writing a flash block, the write blank check compares the contents of the addressed memory area against the value of an erased flash cell to check that the block has been completely erased.

true: Write blank check enabled.

false: Write blank check disabled.

Note

Vendor specific parameter

Table 4-23. Attribute FlsWriteBlankCheck (FlsGeneral) detailed description

Property	Value
Label	Fls Write Blank Check
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	true

4.4.18 FlsWriteVerifyCheck (FlsGeneral)

Pre-processor switch to enable / disable the write verify check. After writing a flash block, the write verify check compares the contents of the reprogrammed memory area against the contents of the provided application buffer to check that the block has been completely reprogrammed.

true: Write verify check enabled.

false: Write verify check disabled.

Note

Vendor specific parameter

Table 4-24. Attribute FlsWriteVerifyCheck (FlsGeneral) detailed description

Property	Value
Label	Fls Write Verify Check
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	true

4.4.19 FlsMaxEraseBlankCheck (FlsGeneral)

The maximum number of bytes to blank check in one cycle of the flash driver's job processing function. Affects only the flash blocks that have enabled asynchronous execution of the erase job (FlsSectorEraseAsynch=true).

Note

Vendor specific parameter

Table 4-25. Attribute FlsMaxEraseBlankCheck (FlsGeneral) detailed description

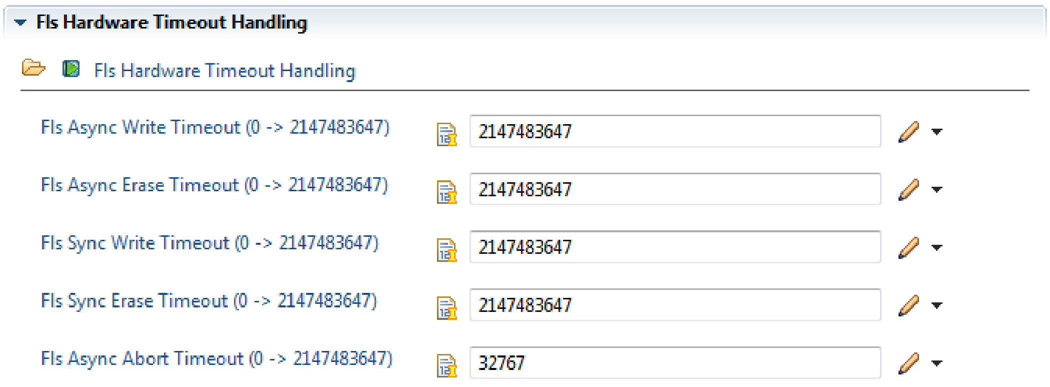
Property	Value
Label	Fls Max Erase Blank Check
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	256
Invalid	Range <=65536 >=8

4.5 Form FlsTimeouts

Container for timeout parameters of the flash driver. The implemented timeout check functionality provides monitoring and deadline supervision of the currently running HW job.

NOTE

Disabling/deleting the container suppresses compilation of the code providing the timeout check functionality.



Fls Async Write Timeout (0 -> 2147483647)	2147483647
Fls Async Erase Timeout (0 -> 2147483647)	2147483647
Fls Sync Write Timeout (0 -> 2147483647)	2147483647
Fls Sync Erase Timeout (0 -> 2147483647)	2147483647
Fls Async Abort Timeout (0 -> 2147483647)	32767

Figure 4-4. Tresos Plugin snapshot for FlsTimeouts form.

4.5.1 FlsAsyncWriteTimeout (FlsTimeouts)

The timeout value applied for the HW write operations handled in asynchronous mode. Each time the Fls_MainFunction() starts the flash program sequence, its internal timeout counter is initialized with the configured value. The counter is decremented per each of the next Fls_MainFunction() calls. Once it reaches 0, the ongoing program operation is aborted and, if enabled, "Flash write failed (HW)" DEM event is reported.

Table 4-26. Attribute FlsAsyncWriteTimeout (FlsTimeouts) detailed description

Property	Value
Label	Fls Async Write Timeout
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	2147483647
INVALID	Range <=2147483647 >=0

4.5.2 FlsAsyncEraseTimeout (FlsTimeouts)

The timeout value applied for the HW erase operations handled in asynchronous mode. Each time the Fls_MainFunction() starts the flash erase sequence, its internal timeout counter is initialized with the configured value. The counter is decremented per each of the next Fls_MainFunction() calls. Once it reaches 0, the ongoing erase operation is aborted and, if enabled, "Flash erase failed (HW)" DEM event is reported.

Table 4-27. Attribute FlsAsyncEraseTimeout (FlsTimeouts) detailed description

Property	Value
Label	Fls Async Erase Timeout
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	2147483647
INVALID	Range <=2147483647 >=0

4.5.3 FlsSyncWriteTimeout (FlsTimeouts)

The timeout value applied for the HW write operations handled in synchronous mode. Inside Fls_MainFunction() a SW loop is implemented waiting for the ongoing program operation to finish. Each time the Fls_MainFunction() starts the flash program sequence, its internal timeout counter is initialized with the configured value. The counter is decremented per each execution of the SW loop. Once it reaches 0, the ongoing program operation is aborted and, if enabled, "Flash write failed (HW)" DEM event is reported.

One SW loop execution takes 10 or 12 machine instructions depending on whether FlsACCallback is a NULL_PTR or not, respectively, plus additional instructions to execute FlsACCallback (if not a NULL_PTR).

Table 4-28. Attribute FlsSyncWriteTimeout (FlsTimeouts) detailed description

Property	Value
Label	Fls Sync Write Timeout
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	2147483647
INVALID	Range <=2147483647 >=0

4.5.4 FlsSyncEraseTimeout (FlsTimeouts)

The timeout value applied for the HW erase operations handled in synchronous mode. Inside Fls_MainFunction() a SW loop is implemented waiting for the ongoing erase operation to finish. Each time the Fls_MainFunction() starts the flash erase sequence, its internal timeout counter is initialized with the configured value. The counter is decremented per each execution of the SW loop. Once it reaches 0, the ongoing erase operation is aborted and, if enabled, "Flash erase failed (HW)" DEM event is reported.

One SW loop execution takes 10 or 12 machine instructions depending on whether FlsACCallback is a NULL_PTR or not, respectively, plus additional instructions to execute FlsACCallback (if not a NULL_PTR).

Table 4-29. Attribute FlsSyncEraseTimeout (FlsTimeouts) detailed description

Property	Value
Label	Fls Sync Erase Timeout

Table continues on the next page...

Table 4-29. Attribute FlsSyncEraseTimeout (FlsTimeouts) detailed description (continued)

Property	Value
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	2147483647
INVALID	Range <=2147483647 >=0

4.5.5 FlsAbortTimeout (FlsTimeouts)

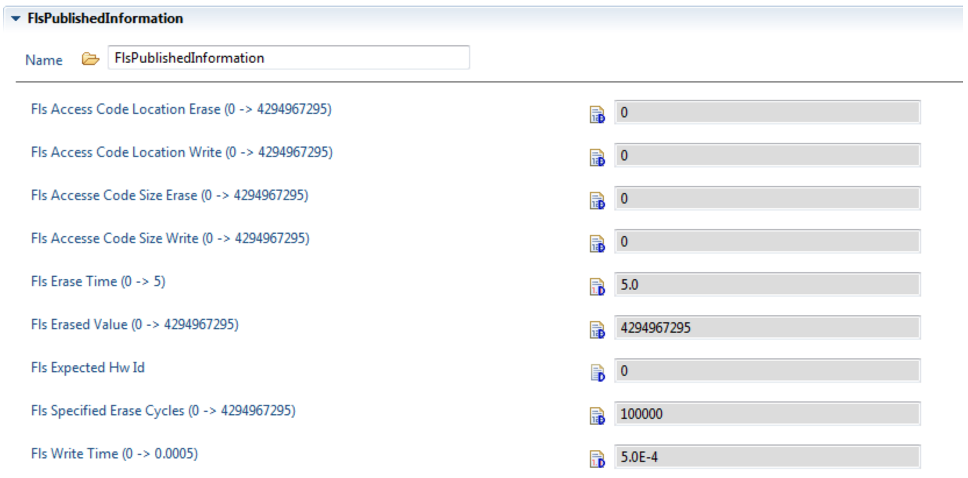
The timeout value applied for the abort of the HW write or erase operation. Inside Fls_MainFunction() a SW loop is implemented waiting for the ongoing abort operation to finish. When the Fls_MainFunction() starts the abort, its internal timeout counter is initialized with the configured value. The counter is decremented per each execution of the SW loop. Once it reaches 0, the SW loop is escaped and, if enabled, "Flash erase failed (HW)" or "Flash write failed (HW)" DEM event is reported depending on the type of the aborted HW job.

Table 4-30. Attribute FlsAbortTimeout (FlsTimeouts) detailed description

Property	Value
Label	Fls Async Abort Timeout
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	2147483647
INVALID	Range <=2147483647 >=0

4.6 Form FlsPublishedInformation

Additional published parameters not covered by CommonPublishedInformation container. Note that these parameters do not have any configuration class setting, since they are published information.



FlsPublishedInformation

Name FlsPublishedInformation

Fls Access Code Location Erase (0 -> 4294967295)	0
Fls Access Code Location Write (0 -> 4294967295)	0
Fls Accesse Code Size Erase (0 -> 4294967295)	0
Fls Accesse Code Size Write (0 -> 4294967295)	0
Fls Erase Time (0 -> 5)	5.0
Fls Erased Value (0 -> 4294967295)	4294967295
Fls Expected Hw Id	0
Fls Specified Erase Cycles (0 -> 4294967295)	100000
Fls Write Time (0 -> 0.0005)	5.0E-4

Figure 4-5. Tresos Plugin snapshot for FlsPublishedInformation form.

4.6.1 FlsAcLocationErase (FlsPublishedInformation)

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 (or the value is zero) it is assumed that the erase flash access code is position independent and therefore the RAM position can be freely configured.

Table 4-31. Attribute FlsAcLocationErase (FlsPublishedInformation) detailed description

Property	Value
Label	Fls Access Code Location Erase
Type	INTEGER_LABEL
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	0
Invalid	Range <=4294967295 >=0

4.6.2 FlsAcLocationWrite (FlsPublishedInformation)

Position in RAM, to which the write flash access code has to be loaded. Only relevant if the write flash access code is not position independent. If this information is not provided (or the value is zero) it is assumed that the write flash access code is position independent and therefore the RAM position can be freely configured.

Table 4-32. Attribute FlsAcLocationWrite (FlsPublishedInformation) detailed description

Property	Value
Label	Fls Access Code Location Write
Type	INTEGER_LABEL
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	0
Invalid	Range <=4294967295 >=0

4.6.3 FlsAcSizeErase (FlsPublishedInformation)

Number of bytes in RAM needed for the erase flash access code. If this information is not provided (or the value is zero) it is assumed that the access code is delivered not as a precompiled but C source code and its size is not known before linking. In such a case a size of the RAM memory reserved for the access code needs to determined dynamically by the linker (see the Integration Manual for more information).

Table 4-33. Attribute FlsAcSizeErase (FlsPublishedInformation) detailed description

Property	Value
Label	Fls Access Code Size Erase
Type	INTEGER_LABEL
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	0
Invalid	Range <=4294967295 >=0

4.6.4 FlsAcSizeWrite (FlsPublishedInformation)

Number of bytes in RAM needed for the write flash access code. If this information is not provided (or the value is zero) it is assumed that the access code is delivered not as a precompiled but C source code and its size is not known before linking. In such a case a size of the RAM memory reserved for the access code needs to determined dynamically by the linker (see the Integration Manual for more information).

Table 4-34. Attribute FlsAcSizeWrite (FlsPublishedInformation) detailed description

Property	Value
Label	Fls Access Code Size Write
Type	INTEGER_LABEL
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	0
Invalid	Range <=4294967295 >=0

4.6.5 FlsEraseTime (FlsPublishedInformation)

Maximum time to erase one complete flash sector [sec].

Note

This value can be found on DS as the maximum erase time occurs after the specified number of program/erase cycles .

Table 4-35. Attribute FlsEraseTime (FlsPublishedInformation) detailed description

Property	Value
Label	Fls Erase Time
Type	FLOAT_LABEL
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	5.0
Invalid	Range <=5.0 >=0

4.6.6 FlsErasedValue (FlsPublishedInformation)

The contents of an erased flash memory cell.

Table 4-36. Attribute FlsErasedValue (FlsPublishedInformation) detailed description

Property	Value
Label	Fls Erased Value
Type	INTEGER_LABEL
Origin	AUTOSAR_ECUC

Table continues on the next page...

Table 4-36. Attribute FlsErasedValue (FlsPublishedInformation) detailed description (continued)

Property	Value
Symbolic Name	false
Default	4294967295
Invalid	Range <=4294967295 >=0

4.6.7 FlsExpectedHwId (FlsPublishedInformation)

Unique identifier of the hardware device that is expected by this driver (the device for which this driver has been implemented). Only relevant for external flash drivers.

Table 4-37. Attribute FlsExpectedHwId (FlsPublishedInformation) detailed description

Property	Value
Label	Fls Expected Hw Id
Type	STRING_LABEL
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	0

4.6.8 FlsSpecifiedEraseCycles (FlsPublishedInformation)

Number of erase cycles specified for the flash device (usually given in the device data sheet). FLS198: If the number of specified erase cycles depends on the operating environment (temperature, voltage, ...) during reprogramming of the flash device, the minimum number for which a data retention of at least 15 years over the temperature range from -40C .. +125C can be guaranteed shall be given.

Note

If there are different numbers of specified erase cycles for different flash sectors of the device this parameter has to be extended to a parameter list (similar to the sector list above).

Table 4-38. Attribute FlsSpecifiedEraseCycles (FlsPublishedInformation) detailed description

Property	Value
Label	Fls Specified Erase Cycles
Type	INTEGER_LABEL
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	100000
Invalid	Range <=4294967295 >=0

4.6.9 FlsWriteTime (FlsPublishedInformation)

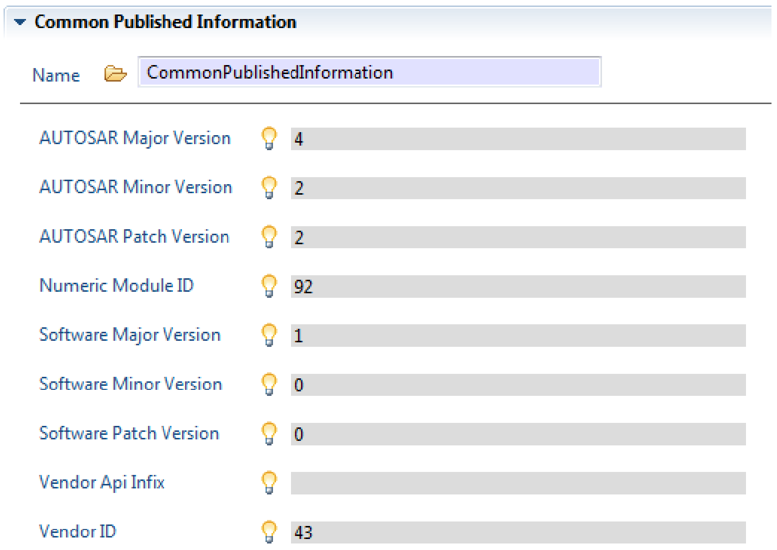
Maximum time to program one complete flash page [sec].

Table 4-39. Attribute FlsWriteTime (FlsPublishedInformation) detailed description

Property	Value
Label	Fls Write Time
Type	FLOAT_LABEL
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	0.0005
Invalid	Range <=0.0005 >=0

4.7 Form CommonPublishedInformation

Common container, aggregated by all modules. It contains published information about vendor and versions.



Name	CommonPublishedInformation
AUTOSAR Major Version	4
AUTOSAR Minor Version	2
AUTOSAR Patch Version	2
Numeric Module ID	92
Software Major Version	1
Software Minor Version	0
Software Patch Version	0
Vendor Api Infix	
Vendor ID	43

Figure 4-6. Tresos Plugin snapshot for CommonPublishedInformation form.

4.7.1 ArReleaseMajorVersion (CommonPublishedInformation)

Major version number of AUTOSAR specification on which the appropriate implementation is based on.

Table 4-40. Attribute ArReleaseMajorVersion (CommonPublishedInformation) detailed description

Property	Value
Label	AUTOSAR Major Version
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	4
Invalid	Range >=4 <=4

4.7.2 ArReleaseMinorVersion (CommonPublishedInformation)

Minor version number of AUTOSAR specification on which the appropriate implementation is based on.

Table 4-41. Attribute ArReleaseMinorVersion (CommonPublishedInformation) detailed description

Property	Value
Label	AUTOSAR Minor Version
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	2
Invalid	Range >=2 <=2

4.7.3 ArReleaseRevisionVersion (CommonPublishedInformation)

Revision version number of AUTOSAR specification on which the appropriate implementation is based on.

Table 4-42. Attribute ArReleaseRevisionVersion (CommonPublishedInformation) detailed description

Property	Value
Label	AUTOSAR Release Revision Version
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	2
Invalid	Range >=2 <=2

4.7.4 ModuleId (CommonPublishedInformation)

Module ID of this module from Module List.

Table 4-43. Attribute ModuleId (CommonPublishedInformation) detailed description

Property	Value
Label	Module Id
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false

Table continues on the next page...

Table 4-43. Attribute ModuleId (CommonPublishedInformation) detailed description (continued)

Property	Value
Default	92
Invalid	Range >=92 <=92

4.7.5 SwMajorVersion (CommonPublishedInformation)

Major version number of the vendor specific implementation of the module. The numbering is vendor specific.

Table 4-44. Attribute SwMajorVersion (CommonPublishedInformation) detailed description

Property	Value
Label	Software Major Version
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	1
Invalid	Range >=1 <=1

4.7.6 SwMinorVersion (CommonPublishedInformation)

Minor version number of the vendor specific implementation of the module. The numbering is vendor specific.

Table 4-45. Attribute SwMinorVersion (CommonPublishedInformation) detailed description

Property	Value
Label	Software Minor Version
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	0
Invalid	Range >=0 <=0

4.7.7 SwPatchVersion (CommonPublishedInformation)

Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific.

Table 4-46. Attribute SwPatchVersion (CommonPublishedInformation) detailed description

Property	Value
Label	Software Patch Version
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	0
Invalid	Range >=0 <=0

4.7.8 VendorApiInfix (CommonPublishedInformation)

In driver modules which can be instantiated several times on a single ECU, BSW00347 requires that the name of APIs is extended by the VendorId and a vendor specific name. This parameter is used to specify the vendor specific name. In total, the implementation specific name is generated as follows: <ModuleName>_>VendorId>_<VendorApiInfix><Api name from SWS>. E.g. assuming that the VendorId of the implementor is 123 and the implementer chose a VendorApiInfix of "v11r456" a api name Can_Write defined in the SWS will translate to Can_123_v11r456Write. This parameter is mandatory for all modules with upper multiplicity > 1. It shall not be used for modules with upper multiplicity =1.

Table 4-47. Attribute VendorApiInfix (CommonPublishedInformation) detailed description

Property	Value
Label	Vendor Api Infix
Type	STRING_LABEL
Origin	Custom
Symbolic Name	false
Default	
Enable	false

4.7.9 VendorId (CommonPublishedInformation)

Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list.

Table 4-48. Attribute VendorId (CommonPublishedInformation) detailed description

Property	Value
Label	Vendor Id
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	43
Invalid	Range >=43 <=43

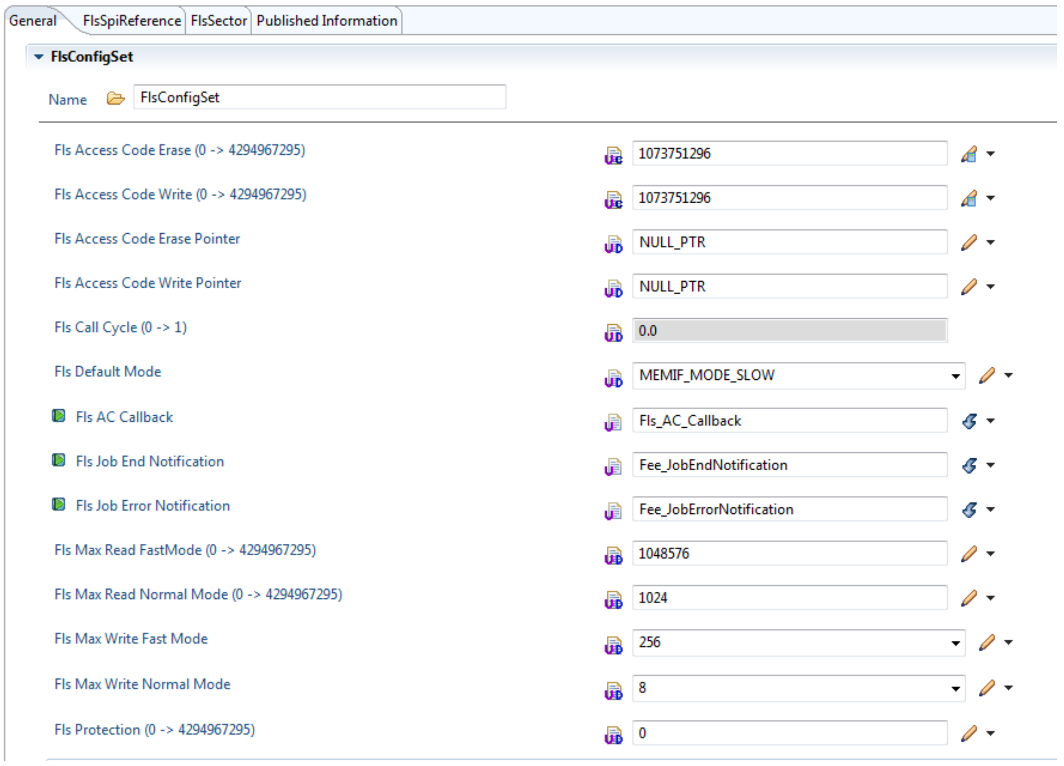
4.8 Form FlsConfigSet

Container for runtime configuration parameters of the flash driver.

Implementation Type: Fls_ConfigType.

Included forms :

- Form FlsDemEventParameterRefs
- Form FlsSectorList



General | FlsSpiReference | FlsSector | Published Information

FlsConfigSet

Name: FlsConfigSet

Fls Access Code Erase (0 -> 4294967295)	1073751296
Fls Access Code Write (0 -> 4294967295)	1073751296
Fls Access Code Erase Pointer	NULL_PTR
Fls Access Code Write Pointer	NULL_PTR
Fls Call Cycle (0 -> 1)	0.0
Fls Default Mode	MEMIF_MODE_SLOW
Fls AC Callback	Fls_AC_Callback
Fls Job End Notification	Fee_JobEndNotification
Fls Job Error Notification	Fee_JobErrorNotification
Fls Max Read FastMode (0 -> 4294967295)	1048576
Fls Max Read Normal Mode (0 -> 4294967295)	1024
Fls Max Write Fast Mode	256
Fls Max Write Normal Mode	8
Fls Protection (0 -> 4294967295)	0

Figure 4-7. Tresos Plugin snapshot for FlsConfigSet form.

4.8.1 FlsAcErase (FlsConfigSet)

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.

Note: To use Fls Access Code Erase be sure Fls Access Code Erase Pointer is NULL or NULL_PTR.

Table 4-49. Attribute FlsAcErase (FlsConfigSet) detailed description

Property	Value
Label	Fls Access Code Erase
Type	INTEGER
Origin	AUTOSAR_ECUC
Symbolic Name	false
Invalid	Range <=4294967295 >=0

4.8.2 FlsAcWrite (FlsConfigSet)

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.

Note: To use Fls Access Code Write be sure Fls Access Code Write Pointer is NULL or NULL_PTR.

Table 4-50. Attribute FlsAcWrite (FlsConfigSet) detailed description

Property	Value
Label	Fls Access Code Write
Type	INTEGER
Origin	AUTOSAR_ECUC
Symbolic Name	false
Invalid	Range <=4294967295 >=0

4.8.3 FlsAcErasePointer (FlsConfigSet)

Pointer in RAM to which the erase flash access code shall be loaded.

Table 4-51. Attribute FlsAcErasePointer (FlsConfigSet) detailed description

Property	Value
Label	Fls Access Code Erase Pointer
Type	FUNCTION-NAME
Origin	Custom
Symbolic Name	false
Default	NULL_PTR

4.8.4 FlsAcWritePointer (FlsConfigSet)

Pointer in RAM to which the write flash access code shall be loaded. Used as function pointer to access the write flash access code.

Table 4-52. Attribute FlsAcWritePointer (FlsConfigSet) detailed description

Property	Value
Label	Fls Access Code Write Pointer

Table continues on the next page...

Table 4-52. Attribute FlsAcWritePointer (FlsConfigSet) detailed description (continued)

Property	Value
Type	FUNCTION-NAME
Origin	Custom
Symbolic Name	false
Default	NULL_PTR

4.8.5 FlsCallCycle (FlsConfigSet)

Cycle time of calls of the flash driver main function

Note

Not supported by the driver.

Table 4-53. Attribute FlsCallCycle (FlsConfigSet) detailed description

Property	Value
Label	Fls Call Cycle
Type	FLOAT_LABEL
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	0.0
Invalid	Range <=1.0 >=0.0

4.8.6 FlsDefaultMode (FlsConfigSet)

This parameter is the default FLS device mode after initialization.

Table 4-54. Attribute FlsDefaultMode (FlsConfigSet) detailed description

Property	Value
Label	Fls Default Mode
Type	ENUMERATION
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	MEMIF_MODE_SLOW
Range	MEMIF_MODE_FAST MEMIF_MODE_SLOW

4.8.7 FlsACCallback (FlsConfigSet)

Mapped to the Access Code Callback provided by some upper layer module, typically the Wdg module.

Note: Disable/delete (depending on the used configuration tool) the parameter to have it set as NULL_PTR.

Table 4-55. Attribute FlsACCallback (FlsConfigSet) detailed description

Property	Value
Label	Fls AC Callback
Type	FUNCTION-NAME
Origin	Custom
Symbolic Name	false
Default	Fls_AC_Callback
Enable	false

4.8.8 FlsJobEndNotification (FlsConfigSet)

Mapped to the job end notification routine provided by some upper layer module, typically the Fee module.

Note: Disable/delete (depending on the used configuration tool) the parameter to have it set as NULL_PTR.

Table 4-56. Attribute FlsJobEndNotification (FlsConfigSet) detailed description

Property	Value
Label	Fls Job End Notification
Type	FUNCTION-NAME
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	Fee_JobEndNotification
Enable	false

4.8.9 FlsJobErrorNotification (FlsConfigSet)

Mapped to the job error notification routine provided by some upper layer module, typically the Fee module.

Note: Disable/delete (depending on the used configuration tool) the parameter to have it set as NULL_PTR.

Table 4-57. Attribute FlsJobErrorNotification (FlsConfigSet) detailed description

Property	Value
Label	Fls Job Error Notification
Type	FUNCTION-NAME
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	Fee_JobErrorNotification
Enable	false

4.8.10 FlsStartFlashAccessNotif (FlsConfigSet)

Start flash access. If configured, this notification will be called before any flash memory access. It is called before flash memory read accesses(in read, compare, verify write, verify erase jobs) and before flash memory program operations(in write and erase jobs). The purpose of this notification together with FlsFinishedFlashAccess, is to ensure that, if needed, no other executed code(other tasks, cores, masters) will access the affected flash area simultaneously with the access initiated by the driver. For more details, see also Integration manual, chapter 5. Module requirements.

Note: Disable/delete (depending on the used configuration tool) the parameter to have it set as NULL_PTR.

Table 4-58. Attribute FlsStartFlashAccessNotif (FlsConfigSet) detailed description

Property	Value
Label	Fls Start Flash Access Notification
Type	FUNCTION-NAME
Origin	Custom
Symbolic Name	false
Default	Fls_StartFlashAccessNotif
Enable	false

4.8.11 FlsFinishedFlashAccessNotif (FlsConfigSet)

Finished flash access. If configured, this notification will be called after any flash memory access. It is called after flash memory read accesses(in read, compare, verify write, verify erase jobs) and after flash memory program operations(in write and erase jobs). The purpose of this notification together with FlsStartFlashAccess, is to ensure that, if needed, no other executed code(other tasks, cores, masters) will access the affected flash area simultaneously with the access initiated by the driver. For more details, see also Integration manual, chapter 5. Module requirements.

Note: Disable/delete (depending on the used configuration tool) the parameter to have it set as NULL_PTR.

Table 4-59. Attribute FlsFinishedFlashAccessNotif (FlsConfigSet) detailed description

Property	Value
Label	Fls Finished Flash Access Notification
Type	FUNCTION-NAME
Origin	Custom
Symbolic Name	false
Default	Fls_FinishedFlashAccessNotif
Enable	false

4.8.12 FlsMaxReadFastMode (FlsConfigSet)

The maximum number of bytes to read or compare in one cycle of the flash driver's job processing function in fast mode.

Table 4-60. Attribute FlsMaxReadFastMode (FlsConfigSet) detailed description

Property	Value
Label	Fls Max Read FastMode
Type	INTEGER
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	1048576
Invalid	Range <=4294967295 >=0

4.8.13 FlsMaxReadNormalMode (FlsConfigSet)

The maximum number of bytes to read or compare in one cycle of the flash driver's job processing function in normal mode.

Table 4-61. Attribute FlsMaxReadNormalMode (FlsConfigSet) detailed description

Property	Value
Label	Fls Max Read Normal Mode
Type	INTEGER
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	1024
Invalid	Range <=4294967295 >=0

4.8.14 FlsMaxWriteFastMode (FlsConfigSet)

The maximum number of bytes to write in one cycle of the flash driver's job processing function in fast mode.

Table 4-62. Attribute FlsMaxWriteFastMode (FlsConfigSet) detailed description

Property	Value
Label	Fls Max Write Fast Mode
Type	INTEGER
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	256
Invalid	Range <=4294967295 >=0

4.8.15 FlsMaxWriteNormalMode (FlsConfigSet)

The maximum number of bytes to write in one cycle of the flash driver's job processing function in normal mode.

Table 4-63. Attribute FlsMaxWriteNormalMode (FlsConfigSet) detailed description

Property	Value
Label	Fls Max Write Normal Mode
Type	INTEGER
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	8
Invalid	Range <=4294967295 >=0

4.8.16 FlsProtection (FlsConfigSet)

Erase/write protection settings. Only relevant if supported by hardware.

Note

Not supported by the driver.

Table 4-64. Attribute FlsProtection (FlsConfigSet) detailed description

Property	Value
Label	Fls Protection
Type	INTEGER
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	0
Invalid	Range <=4294967295 >=0

4.8.17 Form FlsDemEventParameterRefs

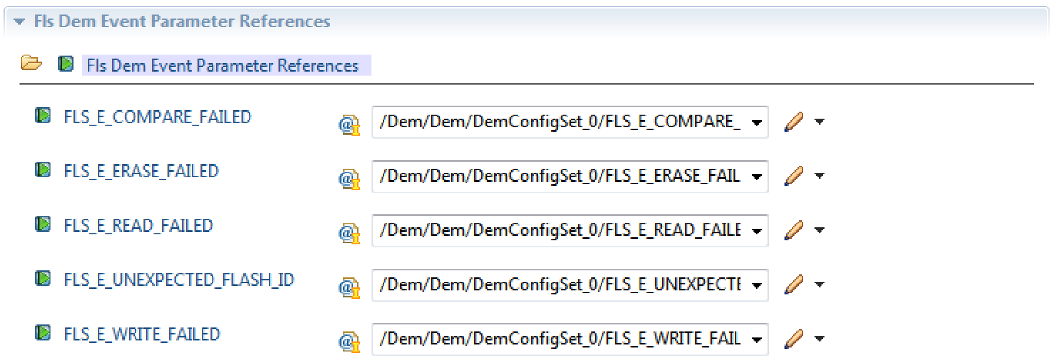
NOTE: DEM error reporting is marked as obsolete in FLS module starting from AUTOSAR 4.2.2, in consequence, this container and all subsequent parameters are disabled. The associated errors are reclassified as runtime errors and reported to DET.

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

Note

Disabling/deleting the container suppresses calling the Dem_ReportErrorStatus API.

Is included by form: Form FlsConfigSet


Fls Dem Event Parameter References
Fls Dem Event Parameter References

FLS_E_COMPARE_FAILED	/Dem/Dem/DemConfigSet_0/FLS_E_COMPARE_
FLS_E_ERASE_FAILED	/Dem/Dem/DemConfigSet_0/FLS_E_ERASE_FAIL
FLS_E_READ_FAILED	/Dem/Dem/DemConfigSet_0/FLS_E_READ_FAILE
FLS_E_UNEXPECTED_FLASH_ID	/Dem/Dem/DemConfigSet_0/FLS_E_UNEXPECTE
FLS_E_WRITE_FAILED	/Dem/Dem/DemConfigSet_0/FLS_E_WRITE_FAIL

Figure 4-8. Tresos Plugin snapshot for FlsDemEventParameterRefs form.

4.8.17.1 FLS_E_COMPARE_FAILED (FlsDemEventParameterRefs)

Reference to the DemEventParameter which shall be issued when the error "Flash compare failed (HW)" has occurred.

Note

Disabling/deleting the container suppresses calling the Dem_ReportErrorStatus API in case the corresponding error occurs.

Table 4-65. Attribute FLS_E_COMPARE_FAILED (FlsDemEventParameterRefs) detailed description

Property	Value
Type	SYMBOLIC-NAME-REFERENCE
Origin	AUTOSAR_ECUC
Enable	true

4.8.17.2 FLS_E_ERASE_FAILED (FlsDemEventParameterRefs)

Reference to the DemEventParameter which shall be issued when the error "Flash erase failed (HW)" has occurred.

Note

Disabling/deleting the container suppresses calling the Dem_ReportErrorStatus API in case the corresponding error occurs.

Table 4-66. Attribute FLS_E_ERASE_FAILED (FlsDemEventParameterRefs) detailed description

Property	Value
Type	SYMBOLIC-NAME-REFERENCE
Origin	AUTOSAR_ECUC
Enable	true

4.8.17.3 FLS_E_READ_FAILED (FlsDemEventParameterRefs)

Reference to the DemEventParameter which shall be issued when the error "Flash read failed (HW)" has occurred.

Note

Disabling/deleting the container suppresses calling the Dem_ReportErrorStatus API in case the corresponding error occurs.

Table 4-67. Attribute FLS_E_READ_FAILED (FlsDemEventParameterRefs) detailed description

Property	Value
Type	SYMBOLIC-NAME-REFERENCE
Origin	AUTOSAR_ECUC
Enable	true

4.8.17.4 FLS_E_UNEXPECTED_FLASH_ID (FlsDemEventParameterRefs)

Reference to the DemEventParameter which shall be issued when the error "Expected hardware ID not matched" has occurred.

Note

Disabling/deleting the container suppresses calling the Dem_ReportErrorStatus API in case the corresponding error occurs.

Table 4-68. Attribute FLS_E_UNEXPECTED_FLASH_ID (FlsDemEventParameterRefs) detailed description

Property	Value
Type	SYMBOLIC-NAME-REFERENCE
Origin	AUTOSAR_ECUC
Enable	true

4.8.17.5 FLS_E_WRITE_FAILED (FlsDemEventParameterRefs)

Reference to the DemEventParameter which shall be issued when the error "Flash write failed (HW)" has occurred.

Note

Disabling/deleting the container suppresses calling the Dem_ReportErrorStatus API in case the corresponding error occurs.

Table 4-69. Attribute FLS_E_WRITE_FAILED (FlsDemEventParameterRefs) detailed description

Property	Value
Type	SYMBOLIC-NAME-REFERENCE
Origin	AUTOSAR_ECUC
Enable	true

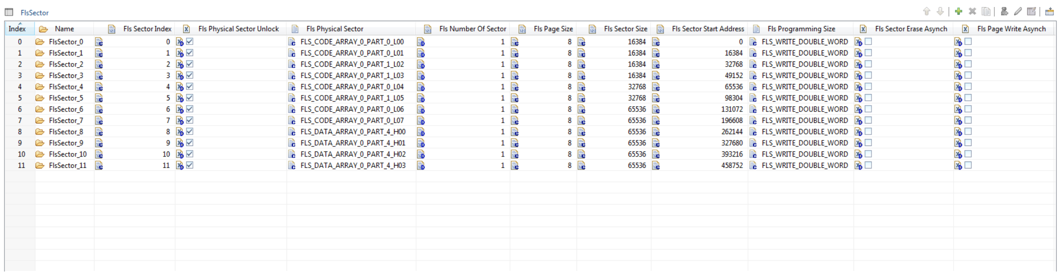
4.8.18 Form FlsSectorList

List of flashable sectors and pages.

Is included by form : Form FlsConfigSet

Included forms :

- Form FlsSector



FlsSector

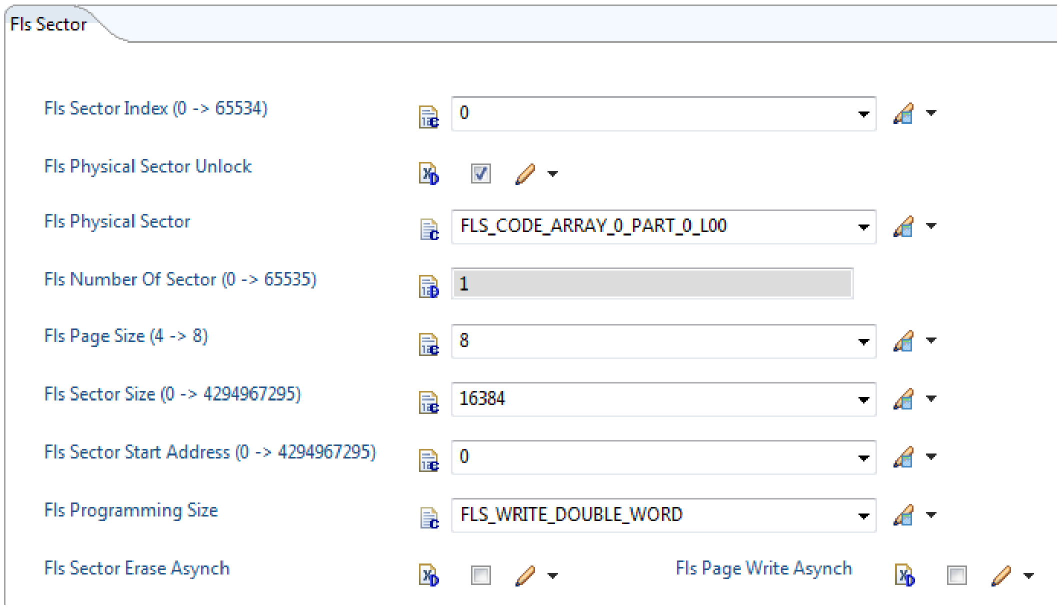
Index	Name	Fls Sector Index	Fls Physical Sector Unlock	Fls Physical Sector	Fls Number Of Sector	Fls Page Size	Fls Sector Size	Fls Sector Start Address	Fls Programming Size	Fls Sector Erase Asynch	Fls Page Write Asynch
0	FlsSector_0	0	✓	FLS_CODE_ARRAY_0_PART_0_L00	1	8	16384	0	FLS_WRITE_DOUBLE_WORD	☐	☐
1	FlsSector_1	1	✓	FLS_CODE_ARRAY_0_PART_0_L01	1	8	16384	16384	FLS_WRITE_DOUBLE_WORD	☐	☐
2	FlsSector_2	2	✓	FLS_CODE_ARRAY_0_PART_1_L02	1	8	16384	32768	FLS_WRITE_DOUBLE_WORD	☐	☐
3	FlsSector_3	3	✓	FLS_CODE_ARRAY_0_PART_1_L03	1	8	16384	49152	FLS_WRITE_DOUBLE_WORD	☐	☐
4	FlsSector_4	4	✓	FLS_CODE_ARRAY_0_PART_0_L04	1	8	32768	65536	FLS_WRITE_DOUBLE_WORD	☐	☐
5	FlsSector_5	5	✓	FLS_CODE_ARRAY_0_PART_1_L05	1	8	32768	98304	FLS_WRITE_DOUBLE_WORD	☐	☐
6	FlsSector_6	6	✓	FLS_CODE_ARRAY_0_PART_0_L06	1	8	65536	131072	FLS_WRITE_DOUBLE_WORD	☐	☐
7	FlsSector_7	7	✓	FLS_CODE_ARRAY_0_PART_0_L07	1	8	65536	196608	FLS_WRITE_DOUBLE_WORD	☐	☐
8	FlsSector_8	8	✓	FLS_DATA_ARRAY_0_PART_4_H00	1	8	65536	262144	FLS_WRITE_DOUBLE_WORD	☐	☐
9	FlsSector_9	9	✓	FLS_DATA_ARRAY_0_PART_4_H01	1	8	65536	327680	FLS_WRITE_DOUBLE_WORD	☐	☐
10	FlsSector_10	10	✓	FLS_DATA_ARRAY_0_PART_4_H02	1	8	65536	393216	FLS_WRITE_DOUBLE_WORD	☐	☐
11	FlsSector_11	11	✓	FLS_DATA_ARRAY_0_PART_4_H03	1	8	65536	458752	FLS_WRITE_DOUBLE_WORD	☐	☐

Figure 4-9. Tresos Plugin snapshot for FlsSectorList form.

4.8.18.1 Form FlsSector

Configuration description of a flashable sector

Is included by form : Form FlsSectorList



Fls Sector

Fls Sector Index (0 -> 65534)	0
Fls Physical Sector Unlock	✓
Fls Physical Sector	FLS_CODE_ARRAY_0_PART_0_L00
Fls Number Of Sector (0 -> 65535)	1
Fls Page Size (4 -> 8)	8
Fls Sector Size (0 -> 4294967295)	16384
Fls Sector Start Address (0 -> 4294967295)	0
Fls Programming Size	FLS_WRITE_DOUBLE_WORD
Fls Sector Erase Asynch	☐
Fls Page Write Asynch	☐

Figure 4-10. Tresos Plugin snapshot for FlsSector form.

4.8.18.1.1 FlsSectorIndex (FlsSector)

Fls Sector Index is an invariant index, used to order flash sectors and loop over them in the correct, configured order. Its value should be equal with the position of the configured sector inside the configured sector list (the same value as the shown index). Rationale: The generated .epc configuration might reorder the flash sectors(alphabetically), thus the index parameter changes, becoming out of sync with the real intended order (for example: Fls Sector Start Addresses).

Note

When generating a new configuration, the value of FlsSectorIndex is intended to be kept default, the same as the sector index.

Note

Vendor specific parameter

Table 4-70. Attribute FlsSectorIndex (FlsSector) detailed description

Property	Value
Label	Fls Sector Index
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	index

4.8.18.1.2 FlsPhysicalSectorUnlock (FlsSector)

Fls_Init ensures unlock modify operation for this Flash Physical Sector, it is not possible to lock until a new reset.

Note

Vendor specific parameter

Table 4-71. Attribute FlsPhysicalSectorUnlock (FlsSector) detailed description

Property	Value
Label	Fls Physical Sector Unlock
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	true

4.8.18.1.3 FlsPhysicalSector (FlsSector)

Physical flash device sector.

Note

Vendor specific parameter

Table 4-72. Attribute FlsPhysicalSector (FlsSector) detailed description

Property	Value
Label	Fls Physical Sector
Type	ENUMERATION
Origin	Custom
Symbolic Name	false

4.8.18.1.4 FlsNumberOfSectors (FlsSector)

Number of continuous sectors with the above characteristics.

Note

Not supported by the driver.

Table 4-73. Attribute FlsNumberOfSectors (FlsSector) detailed description

Property	Value
Label	Fls Number Of Sector
Type	INTEGER_LABEL
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	1
Invalid	Range <=65535 >=0

4.8.18.1.5 FlsPageSize (FlsSector)

Size of one page of this sector. Implementation Type: Fls_LengthType.

- For Code Flash page size is 8 (or 8/32 write/read page size, see the Note below)
- For Data Flash page size is 8 (or 8/32 write/read page size, see the Note below)

Table 4-74. Attribute FlsPageSize (FlsSector) detailed description

Property	Value
Label	Fls Page Size
Type	INTEGER
Origin	AUTOSAR_ECUC
Symbolic Name	false
Invalid	Range <=8 >=4

4.8.18.1.6 FlsSectorSize (FlsSector)

Size of this sector.

Implementation Type: Fls_LengthType.

Table 4-75. Attribute FlsSectorSize (FlsSector) detailed description

Property	Value
Label	Fls Sector Size
Type	INTEGER
Origin	AUTOSAR_ECUC
Symbolic Name	false
Invalid	Range <=4294967295 >=0

4.8.18.1.7 FlsSectorStartaddress (FlsSector)

Start address of this sector.

Implementation Type: Fls_AddressType.

Table 4-76. Attribute FlsSectorStartaddress (FlsSector) detailed description

Property	Value
Label	Fls Sector Start Address
Type	INTEGER
Origin	AUTOSAR_ECUC
Symbolic Name	false
Invalid	Range <=4294967295 >=0

4.8.18.1.8 FlsProgrammingSize (FlsSector)

This is the size to program Flash memory during one High voltage operation.

Table 4-77. Attribute FlsProgrammingSize (FlsSector) detailed description

Property	Value
Label	Fls Programming Size
Type	ENUMERATION
Origin	Custom
Symbolic Name	false

4.8.18.1.9 FlsSectorEraseAsynch (FlsSector)

Enable asynchronous execution of the erase job in the Fls_MainFunction function which doesn't wait (block) for completion of the sector erase operation. The flash driver doesn't use the erase access code to the erase flash sector in asynchronous mode so it can be used only on flash sectors which belong to flash array different from flash array the application is executing from.

Note

Vendor specific parameter

Table 4-78. Attribute FlsSectorEraseAsynch (FlsSector) detailed description

Property	Value
Label	Fls Sector Erase Asynch
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	false

4.8.18.1.10 FlsPageWriteAsynch (FlsSector)

Enable asynchronous execution of the write job in the Fls_MainFunction function which doesn't wait (block) for completion of the page write operation(s). The flash driver doesn't use the write access code to the write flash page(s) in asynchronous mode so it can be used only on flash sectors which belong to flash array different from flash array the application is executing from.

Note

Vendor specific parameter

Table 4-79. Attribute FlsPageWriteAsynch (FlsSector) detailed description

Property	Value
Label	Fls Page Write Asynch
Type	BOOLEAN
Origin	Custom
Symbolic Name	false
Default	false

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer’s technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, C-5, CodeTest, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, and UMEMS are trademarks of NXP B.V. All other product or service names are the property of their respective owners. ARM, AMBA, ARM Powered, Artisan, Cortex, Jazelle, Keil, SecurCore, Thumb, TrustZone, and μVision are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. ARM7, ARM9, ARM11, big.LITTLE, CoreLink, CoreSight, DesignStart, Mali, mbed, NEON, POP, Sensinode, Socrates, ULINK and Versatile are trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2017 NXP B.V.

Document Number UM35FLSASR4.2 Rev0002R1.0.0
Revision 1.0.0