User Manual

for MPC5634M FLS Driver

Document Number: UM14FLSASR3.0R2.0.0
Rev. 2.6

freescale™

Contents

Chapter 1
Revision History

Table 1-1. Revision History

Revision	Date	Author	Description
2.5	03-Feb-2011	Gaetano Stabile	Update for Monaco automatic documentation
2.6	21-Dec-2011	Khanindra Jyoti Deka	Update for Monaco RTM 2.0.0

Chapter 2 Introduction

This User Manual describes Freescale Semiconductor AUTOSAR Flash (Fls) for MPC5634M .

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 Freescale Semiconductor .

Table 2-1. MPC5634M Derivatives

Freescale Semiconductor	mpc5634m_bga208, mpc5634m_qfp144, mpc5634m_qfp176

All of the above microcontroller devices are collectively named as MPC5634M .

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	Development Error Tracer
ECU	Electronic Control Unit
EEPROM	Electrically Erasable Programmable Read-Only Memory
FEE	Flash EEPROM Emulation
FLS	Flash
MCU	Micro Controller Unit
VLE	Variable Length Encoding
XML	Extensible Markup Language

2.5 Reference List

Table 2-3. Reference List

#	Title	Version
1	AUTOSAR 3.0Fls Driver Software Specification Document.	V2.2.0 R3.0 Rev 0001
2	MPC5634M Reference Manual	Rev. 6, 4 October 2011

Chapter 3
Driver

3.1 Requirements

Requirements for this driver are detailed in the AUTOSAR 3.0Fls Driver Software Specification document (See Table Reference List).

3.2 Driver Design Summary

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

Note: If the FLS driver is used in user mode, be sure that the Flash memory controller registers are accessible and that accessed Flash memory partition is not protected.

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

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: 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.

- **Driver Limitations:** See also [Async/sync mode]
  a. When Fls_Cancel API is enabled and Fee driver is used, FlsMaxWriteFastMode / FlsMaxWriteNormalMode cannot be less than Fee Block header (16 bytes).
  b. When Fls_Cancel API is enabled and Fee driver is used, Page Write in Async mode cannot be used (FlsPageWriteAsynch = false).

3.3 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 Table 3-1 provides Status column description.

Table 3-1. Deviations Status Column Description

Term	Definition
N/A	Not available
N/T	Not testable
N/S	Out of scope
N/R	Unclear Requirement
N/I	Not implemented
N/F	Not fully implemented
I/D	Implemented with Deviations

Table Table 3-2 identifies the Fls AUTOSAR requirements that are not fully implemented, implemented differently, or out of scope for the driver.

Table 3-2. Driver Deviations Table

Requirement	Status	Description	Notes
FLS004	N/I	The FLS module shall be able to detect the following errors and exceptions depending on its configuration (development/production): <continue>	FLS_E_UNEXPECTED_FLASH_ID is applicable only for external Fls driver
FLS144	N/I	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

Table continues on the next page...

Table 3-2. Driver Deviations Table (continued)

Requirement	Status	Description	Notes
FLS215	N/R	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).	Additionaly there is possibility to alter default behaviour and have Erase/Write jobs asynchronous, i.e. Fls_MainFunction function doesn't wait (block) for completion of the erase sector/page write operation(s).
FLS217	N/R	The FLS module shall add a device specific base address to the address type Fls_AddressType if necessary.	Unclear concept: Not used
FLS015	N/R	If development error detection for the module Fls is enabled: the function Fls_Init shall check the (hardware specific) contents of the given configuration set for being within the allowed range. If this is not the case, it shall raise the development error FLS_E_PARAM_CONFIG.	Not applicable. The FLS module shall check static configuration parameters statically. See FLS205
FLS221	N/R	The job of the function Fls_Erase shall erase a flash memory block starting from FlsBaseAddress + TargetAddress of size Length.	FlsBaseAddress not used, unclear purpose.
FLS020	N/R	If development error detection for the module Fls is enabled: the function Fls_Erase shall check that the erase start address <continue>	FlsBaseAddress not used, unclear purpose.
FLS145	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.
FLS226	N/R	The job of the function Fls_Write shall program a flash memory block with data provided via SourceAddressPtr starting from FlsBaseAddress + TargetAddress of size Length.	FlsBaseAddress not used, unclear purpose.
FLS026	N/R	If development error detection for the module Fls is enabled: the function Fls_Write shall check that the write start address (FlsBaseAddress + TargetAddress) is aligned <continue>	FlsBaseAddress not used, unclear purpose.
FLS146	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.
FLS032	N/R	The function Fls_Cancel shall reset the FLS module's internal job processing variables (like address, length and data pointer) and set the FLS module state to FLS_IDLE.	FLS_IDLE(doesn't exist...) or MEMIF_IDLE is meant?

Table continues on the next page...

Table 3-2. Driver Deviations Table (continued)

Requirement	Status	Description	Notes
FLS239	N/I	The read job of the function Fls_Read shall copy a continuous flash memory block starting from FlsBaseAddress + SourceAddress of size Length to the buffer pointed to by TargetAddressPtr.	FlsBaseAddress not used, unclear purpose.
FLS097	N/R	If development error detection for the module Fls is enabled: the function Fls_Read shall check that the read start address (FlsBaseAddress + SourceAddress) lies <continue>	FlsBaseAddress not used, unclear purpose.
FLS244	N/R	The job of the function Fls_Compare shall compare a continuous flash memory block starting from FlsBaseAddress + SourceAddress of size Length with the buffer pointed to by TargetAddressPtr.	FlsBaseAddress not used, unclear purpose.
FLS150	N/R	If development error detection for the module Fls is enabled: the function Fls_Compare shall check that the compare start address (FlsBaseAddress + SourceAddress) lies within <continue>	FlsBaseAddress not used, unclear purpose.
FLS165	N/R	The function Fls_GetVersionInfo shall return the version information of the FLS module. The version information includes: <continue>	implementation also fills VersionInfoPtr->instanceID
FLS247	I/D	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.
FLS040	I/D	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.
FLS022	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.
FLS055	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.

Table continues on the next page...

Table 3-2. Driver Deviations Table (continued)

Requirement	Status	Description	Notes
FLS056	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 DET error is not reported but Fls job ends with MEMIF_JOB_FAILED.
FLS232	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.
FLS233	N/I	The FLS module's implementer shall locate the interrupt service routine in Fls_Irq.c.	Not applicable. Not supported by hardware.
FLS234	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.
FLS272	N/I	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.	Unclear how to implement this functionality. The hardware has no direct support for this.
FLS196	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).
FLS169	N/R	FlsBaseAddress	FlsBaseAddress not used, unclear purpose.
FLS170	N/R	FlsTotalSize	FlsTotalSize not used, unclear purpose.
FLS272	N/R	FlsCallCycle	FlsCallCycle not used, unclear purpose.
FLS177	N/I	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. <continue>	We don't provide the module description file (we have no experience with that topic).
FLS176	N/R	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.	FLS_PAGE_LIST?
FLS182	N/R	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.	FLS_PAGE_LIST?

3.4 Runtime Errors

The driver generates the following DEM errors at runtime.

Table 3-3. Runtime Errors

Function	Error Code	Condition triggering the error
Fls_LLD_MainFunction()	FLS_E_ERASE_FAILED	Async Erase operation failed.
Fls_LLD_MainFunction()	FLS_E_ERASE_FAILED	Async Erase operation failed (in case of interleaved blocks).
Fls_LLD_MainFunction()	FLS_E_WRITE_FAILED	Async Write operation failed.
Fls_LLD_SectorErase()	FLS_E_ERASE_FAILED	Erase operation cannot be executed.
Fls_LLD_SectorErase()	FLS_E_ERASE_FAILED	Erase operation cannot be executed (in case of interleaved blocks).
Fls_LLD_SectorErase()	FLS_E_ERASE_FAILED	Erase operation cannot be executed. Sector is locked, must be unlocked before an HV operation can be set
Fls_LLD_SectorErase()	FLS_E_ERASE_FAILED	Erase operation cannot be executed (in case of interleaved blocks).
Fls_LLD_SectorErase()	FLS_E_ERASE_FAILED	Sync Erase operation failed.
Fls_LLD_SectorErase()	FLS_E_ERASE_FAILED	Sync Erase operation failed (in case of interleaved blocks).
Fls_LLD_SectorErase()	FLS_E_ERASE_FAILED	Sync Erase operation failed (and previous two cases).
Fls_LLD_SectorWrite()	FLS_E_WRITE_FAILED	Write operation cannot be executed.
Fls_LLD_SectorWrite()	FLS_E_WRITE_FAILED	Write operation cannot be executed (in case of interleaved blocks).
Fls_LLD_SectorWrite()	FLS_E_WRITE_FAILED	Write operation cannot be executed. Sector is locked, must be unlocked before an HV operation can be set
Fls_LLD_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_LLD_SectorWrite()	FLS_E_WRITE_FAILED	Sync Write operation failed.
Fls_LLD_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.
Fls_LLD_Init	FLS_E_TIMEOUT	Timeout happens while Erase, Write or Abort operation
Fls_LLD_Cancel	FLS_E_TIMEOUT	Timeout happens while Abort operation
Fls_LLD_SectorErase	FLS_E_TIMEOUT	Timeout happens while Erase operation
Fls_LLD_SectorWrite	FLS_E_TIMEOUT	Timeout happens while Write operation

3.5 Software specification

The following sections contains driver software specifications.

3.5.1 Define Reference

Constants supported by the driver are as per AUTOSAR Fls Driver software specification Version 3.0 .

3.5.2 Enum Reference

Enumeration of all constants supported by the driver are as per AUTOSAR Fls Driver software specification Version 3.0 .

3.5.2.1 Enumeration Fls_JobType

Type of job currently executed by Fls_MainFunction.

Table 3-4. Enumeration Fls_JobType Values

Name	Initializer	Description
FLS_JOB_ERASE	0	Erase one or more complete flash sectors .
FLS_JOB_WRITE		Write one or more complete flash pages .
FLS_JOB_READ		Read one or more bytes from flash memory .
FLS_JOB_COMPARE		Compare data buffer with content of flash memory .

3.5.3 Function Reference

Functions of all functions supported by the driver are as per AUTOSAR Fls Driver software specification Version 3.0 .

3.5.3.1 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_Cancel` changes module status and `Fls_JobResult` internal variable.

Note

Violates MISRA-C:2004 Advisory Rule 8.10: See `Fls_c_REF_19`

Prototype: `void Fls_Cancel(void);`

3.5.3.2 Function Fls_Compare

Compares a flash memory area with an application data buffer.

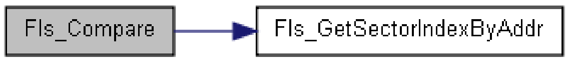


Figure 3-1. Function Fls_Compare References.

Details:

Starts a compare job asynchronously. The actual job is performed by `Fls_MainFunction`.

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

Post: `Fls_Read` changes module status and some internal variables (`Fls_JobSectorIt`, `Fls_JobAddrIt`, `Fls_JobAddrEnd`, `Fls_JobDataSrcPtr`, `Fls_Job`, `Fls_JobResult`).

Return: Std_ReturnType .

Note

Violates MISRA-C:2004 Advisory Rule 8.10: See `Fls_c_REF_19`

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

Table 3-5. Fls_Compare Arguments

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

Table 3-6. Fls_Compare Return Values

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

3.5.3.3 Function Fls_Erase

Erase one or more complete flash sectors.

Details:

Starts an erase job asynchronously. The actual job is performed by the `Fls_MainFunction`.

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

Post: `Fls_Erase` changes module status and some internal variables (`Fls_JobSectorIt`, `Fls_JobSectorEnd`, `Fls_Job`, `Fls_JobResult`).

Return: Std_ReturnType .

Note

Violates MISRA-C:2004 Advisory Rule 8.10: See `Fls_c_REF_19`

Prototype: `Std_ReturnType Fls_Erase(Fls_AddressType TargetAddress, Fls_LengthType Length);`

Table 3-7. Fls_Erase Arguments

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

Table 3-8. Fls_Erase Return Values

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

3.5.3.4 Function Fls_GetJobResult

Returns last job result.

Details:

Returns synchronously the result of the last job.

Return: MemIf_JobResultType .

Note

Violates MISRA-C:2004 Advisory Rule 8.10: See `Fls_c_REF_19`

Prototype: `MemIf_JobResultType Fls_GetJobResult(void);`

Table 3-9. 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_CANCELLED	Job has been cancelled.
MEMIF_BLOCK_INCONSISTENT	Inconsistent block requested, it may contains corrupted data.
MEMIF_BLOCK_INVALID	Invalid block requested.

3.5.3.5 Function Fls_GetStatus

Returns the FLS module status.

Details:

Returns the FLS module status synchronously.

Return: MemIf_StatusType .

Note

Violates MISRA-C:2004 Advisory Rule 8.10: See `Fls_c_REF_19`

Prototype: `MemIf_StatusType Fls_GetStatus(void);`

Table 3-10. 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.

3.5.3.6 Function Fls_GetVersionInfo

Returns version information about FLS module.

Details:

Version information includes:

- Module Id,
- Vendor Id,
- Vendor specific version numbers.

Note

Violates MISRA-C:2004 Advisory Rule 8.10: See `Fls_c_REF_19`Violates MISRA-C:2004 Advisory Rule 16.4, The identifiers used in the declaration and definition of a function shall be identical: See `Fls_c_REF_18`

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

Table 3-11. Fls_GetVersionInfo Arguments

Type	Name	Direction	Description
Std_VersionInfoType *	VersionInfoPtr	**input, output**	Where to store module version informations.

3.5.3.7 Function Fls_Init

The function initializes Fls module.

Details:

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

Pre: `ConfigPtr` must not be `NULL_PTR` and the module status must not be `MEMIF_BUSY`.

Note

Violates MISRA-C:2004 Advisory Rule 8.10: See `Fls_c_REF_19`Violates MISRA-C:2004 Advisory Rule 19.15, Repeated include: See `Fls_Api_h_REF_2`

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

Table 3-12. Fls_Init Arguments

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

3.5.3.8 Function Fls_MainFunction

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

Details:

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

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.

Note

Violates MISRA-C:2004 Advisory Rule 8.10: See `Fls_c_REF_19`

Prototype: `void Fls_MainFunction(void);`

3.5.3.9 Function Fls_Read

Reads from flash memory.

Details:

Starts a read job asynchronously. The actual job is performed by `Fls_MainFunction`.

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

Post: `Fls_Read` changes module status and some internal variables (`Fls_JobSectorIt`, `Fls_JobAddrIt`, `Fls_JobAddrEnd`, `Fls_JobDataDestPtr`, `Fls_Job`, `Fls_JobResult`).

Return: Std_ReturnType .

Note

Violates MISRA-C:2004 Advisory Rule 8.10: See `Fls_c_REF_19`

Prototype: `Std_ReturnType Fls_Read(Fls_AddressType SourceAddress, uint8 *TargetAddressPtr, Fls_LengthType Length);`

Table 3-13. Fls_Read Arguments

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

Table 3-14. Fls_Read Return Values

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

3.5.3.10 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_SetMode` changes internal variables `Fls_MaxRead` and `Fls_MaxWrite`.

Note

Violates MISRA-C:2004 Advisory Rule 8.10: See `Fls_c_REF_19`

Prototype: `void Fls_SetMode(MemIf_ModeType Mode);`

Table 3-15. Fls_SetMode Arguments

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

3.5.3.11 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 by `Fls_MainFunction`.

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

Post: `Fls_Write` changes module status and some internal variables (`Fls_JobSectorIt`, `Fls_JobAddrIt`, `Fls_JobAddrEnd`, `Fls_JobDataSrcPtr`, `Fls_Job`, `Fls_JobResult`).

Return: Std_ReturnType .

Note

Violates MISRA-C:2004 Advisory Rule 8.10: See `Fls_c_REF_19`

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

Table 3-16. Fls_Write Arguments

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

Table 3-17. Fls_Write Return Values

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

3.5.4 Structs Reference

Data structures supported by the driver are as per AUTOSAR Fls Driver software specification Version 3.0 .

3.5.4.1 Structure Fls_ConfigType

Fls module initialization data structure.

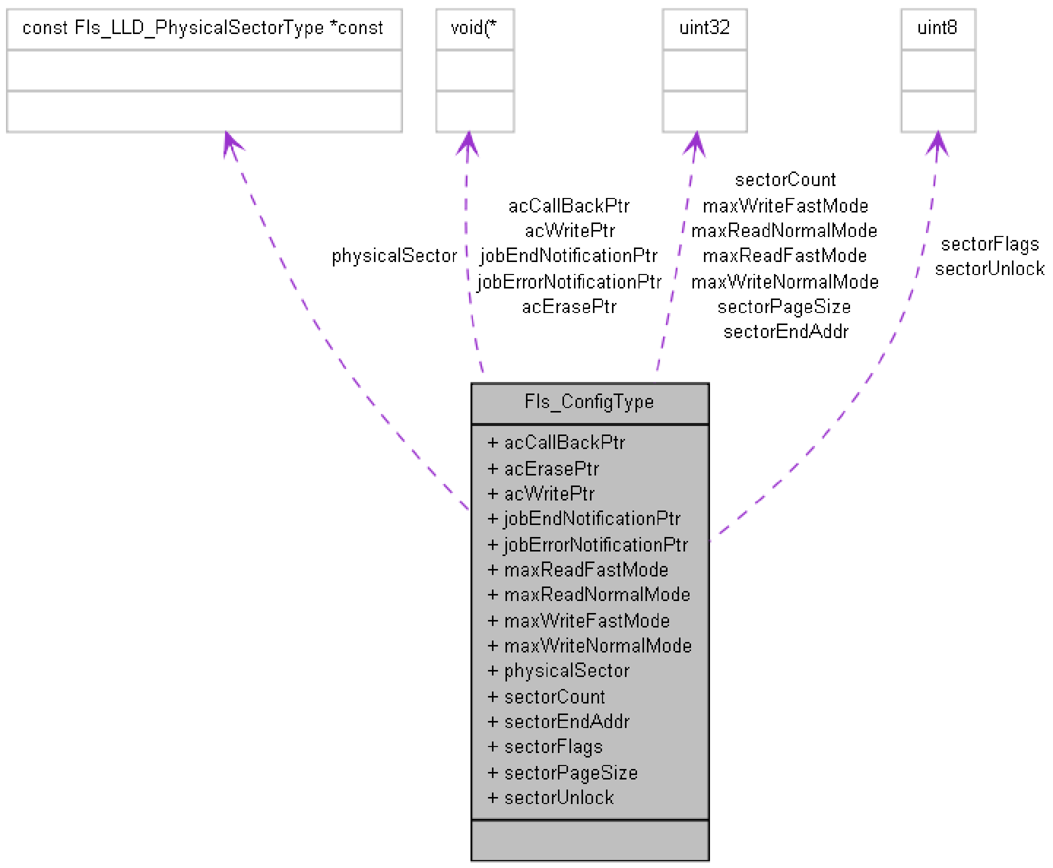


Figure 3-2. Struct Fls_ConfigType

Declaration:

```
typedef struct
                           {
                             Fls_ACCallbackPtrType acCallBackPtr,
                                     Fls_LLD_AcErasePtrType acErasePtr,
                                     Fls_LLD_AcWritePtrType acWritePtr,
                                     Fls_JobEndNotificationPtrType jobEndNotificationPtr,
                                     Fls_JobErrorNotificationPtrType jobErrorNotificationPtr,
                                     Fls_LengthType maxReadFastMode,
                                     Fls_LengthType maxReadNormalMode,
                                     Fls_LengthType maxWriteFastMode,
                                     Fls_LengthType maxWriteNormalMode,
                                     const Fls_LLD_PhysicalSectorType *const physicalSector,
                                     Fls_SectorCountType sectorCount,
                                     const Fls_AddressType *const sectorEndAddr,
                                     const uint8 *const sectorFlags,
                                     const Fls_LengthType *const sectorPageSize,
                                     const uint8 *const sectorUnlock
                           } Fls_ConfigType;
```

Table 3-18. Structure Fls_ConfigType member description

Member	Description
acCallBackPtr	pointer to ac callback function
acErasePtr	pointer to erase access code function in RAM or ROM
acWritePtr	pointer to write access code function in RAM or ROM
jobEndNotificationPtr	pointer to job end notification function
jobErrorNotificationPtr	pointer to job error notification function
maxReadFastMode	max number of bytes to read in one cycle of Fls_MainFunction (fast mode)
maxReadNormalMode	max number of bytes to read in one cycle of Fls_MainFunction (normal mode)
maxWriteFastMode	max number of bytes to write in one cycle of Fls_MainFunction (fast mode)
maxWriteNormalMode	max number of bytes to write in one cycle of Fls_MainFunction (normal mode)
physicalSector	pointer to array containing physical sector ID of each configured sector
sectorCount	number of configured logical sectors
sectorEndAddr	pointer to array containing last logical address of each configured sector
sectorFlags	pointer to array containing flags set of each configured sector
sectorPageSize	pointer to array containing page size of each configured sector
sectorUnlock	pointer to array containing Unlock information of each configured sector

3.5.5 Types Reference

Types supported by the driver are as per AUTOSAR Fls Driver software specification Version 3.0 .

3.5.5.1 Typedef Fls_SectorIndexType

Logical sector index.

Type: uint32

3.5.5.2 Typedef Fls_ACCallbackPtrType

Pointer type of Fls_AC_Callback function.

Type: void(*

3.5.5.3 Typedef Fls_AddressType

Address offset from the configured flash base address to access a certain flash memory area.

Type: uint32

3.5.5.4 Typedef Fls_JobEndNotificationPtrType

Pointer type of Fls_JobEndNotification function.

Type: void(*

3.5.5.5 Typedef Fls_JobErrorNotificationPtrType

Pointer type of Fls_JobErrorNotification function.

Type: void(*

3.5.5.6 Typedef Fls_LengthType

Number of bytes to read/write/erase/compare.

Type: uint32

3.5.5.7 Typedef Fls_SectorCountType

Number of configured sectors.

Type: uint32

3.5.6 Variables Reference

Variables supported by the driver are as per AUTOSAR Fls Driver software specification Version 3.0 .

3.6 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 Platform

4.1 Flash Banks/Arrays, Sectors details

The Flash memory comprises a platform Flash controller interface and three Flash memory array of 512KB for Code Flash.

Table 4-1. Sectors details

Type	Array	Partition	Sector Name	Sector Size (KB)
CODE_ARRAY	0	N/A	L0	16
CODE_ARRAY	0	N/A	L1	16
CODE_ARRAY	0	N/A	L2	32
CODE_ARRAY	0	N/A	L3	32
CODE_ARRAY	0	N/A	L4	16
CODE_ARRAY	0	N/A	L5	16
CODE_ARRAY	0	N/A	L6	64
CODE_ARRAY	0	N/A	L7	64
CODE_ARRAY	0	N/A	M0	128
CODE_ARRAY	0	N/A	M1	128
CODE_ARRAY	1	N/A	H0	128
CODE_ARRAY	1	N/A	H1	128
CODE_ARRAY	1	N/A	H2	128
CODE_ARRAY	1	N/A	H3	128
CODE_ARRAY	2	N/A	H4	128
CODE_ARRAY	2	N/A	H5	128
CODE_ARRAY	2	N/A	H6	128
CODE_ARRAY	2	N/A	H7	128

4.2 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*)(0xC3F88004)) = 0xA1A11111; /* FLASH_LML */
    *((volatile uint32*)(0xC3F88004)) = 0x00000000; /* FLASH_LML */
    *((volatile uint32*)(0xC3F88008)) = 0xB2B22222; /* FLASH_HBL */
    *((volatile uint32*)(0xC3F88008)) = 0x00000000; /* FLASH_HBL */
    *((volatile uint32*)(0xC3F8800C)) = 0xC3C33333; /* FLASH_SLL */
    *((volatile uint32*)(0xC3F8800C)) = 0x00000000; /* FLASH_SLL */

    /* FLASHMEM0_CF0_B unprotect/unlock */
    *((volatile uint32*)(0xC3FB0004)) = 0xA1A11111; /* FLASH_LML */
    *((volatile uint32*)(0xC3FB0004)) = 0x00000000; /* FLASH_LML */
    *((volatile uint32*)(0xC3FB0008)) = 0xB2B22222; /* FLASH_HBL */
    *((volatile uint32*)(0xC3FB0008)) = 0x00000000; /* FLASH_HBL */
    *((volatile uint32*)(0xC3FB000C)) = 0xC3C33333; /* FLASH_SLL */
    *((volatile uint32*)(0xC3FB000C)) = 0x00000000; /* FLASH_SLL */

    /* FLASHMEM0_DF0_A unprotect/unlock */
    *((volatile uint32*)(0xC3F8C004)) = 0xA1A11111; /* FLASH_LML */
    *((volatile uint32*)(0xC3F8C004)) = 0x00000000; /* FLASH_LML */
    *((volatile uint32*)(0xC3F8C00C)) = 0xC3C33333; /* FLASH_SLL */
    *((volatile uint32*)(0xC3F8C00C)) = 0x00000000; /* FLASH_SLL */
}
```

Chapter 5
Async/sync mode

5.1 Introduction

It's possible to modify the behavior of sector erase / page write using two configuration parameters (FlsPageEraseAsync, FlsPageWriteAsync) into 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.

The following outcome has to be shown after robustness analysis for the Fls/Fee drivers:

- Fls write in sync mode is similar to the async mode - in fact the sync mode consumes less time to complete the FEE job (because less Fee/Fls_MainFunctions are needed).
- if Fls async write is used, then the Fls_Cancel API can trigger some robustness issues hence it became prohibited to use the Fls_Cancel API for async writes.
- Note that none of the above applies to async erase (where it is ok to have the Fls_Cancel API, and it also make sense to have an async mode for better performance).

5.2 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: 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).

Chapter 6
Tresos Configuration Plug-in

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

6.1 Configuration elements of Fls

Included forms :

- IMPLEMENTATION_CONFIG_VARIANT
- FlsGeneral
- CommonPublishedInformation
- FlsPublishedInformation
- FlsConfigSet

Table 6-1. Revision table

Revision	Date
revision1.0.0	2009-05-14T17:00:00

6.2 Form IMPLEMENTATION_CONFIG_VARIANT

VariantPostBuild: Mix of precompile and postbuild time configuration parameters. If Config Variant = VariantPostBuild, the files Fls_Cfg.h and Fls_PBcfg.c should be used.

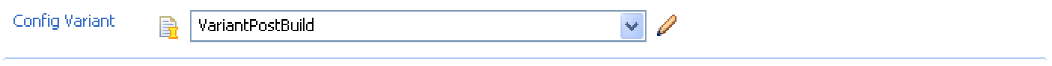

Config Variant
VariantPostBuild

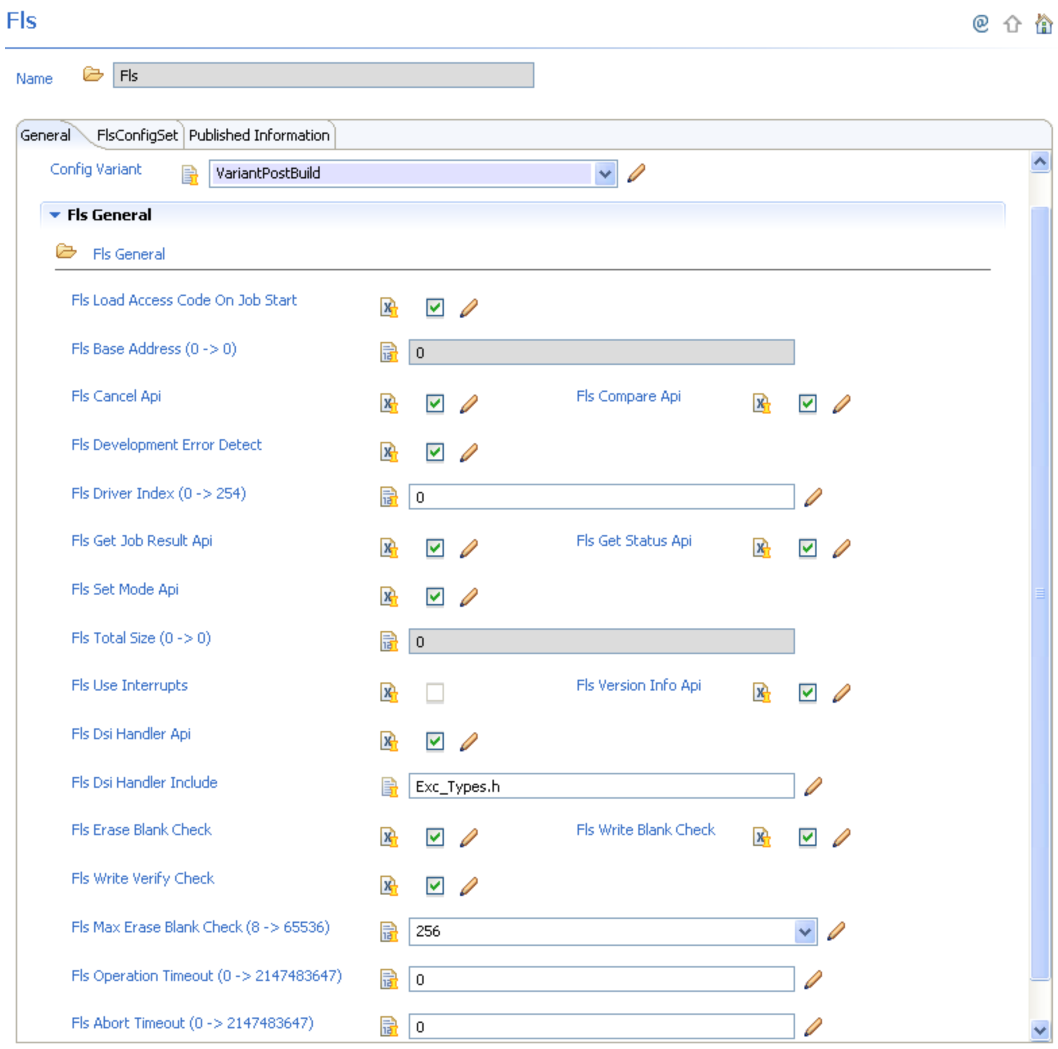
Figure 6-1. Tresos Plugin snapshot for IMPLEMENTATION_CONFIG_VARIANT form.

Table 6-2. Attribute IMPLEMENTATION_CONFIG_VARIANT detailed description

Property	Value
Label	Config Variant
Default	VariantPostBuild
Range	VariantPostBuild

6.3 Form FlsGeneral

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



Fls

Name: Fls

General | FlsConfigSet | Published Information

Config Variant: VariantPostBuild

Fls General

Fls Load Access Code On Job Start	✓		
Fls Base Address (0 -> 0)	0		
Fls Cancel Api	✓	Fls Compare Api	✓
Fls Development 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 -> 0)	0		
Fls Use Interrupts	☐	Fls Version Info Api	✓
Fls Dsi Handler Api	✓		
Fls Dsi Handler Include	Exc_Types.h		
Fls Erase Blank Check	✓	Fls Write Blank Check	✓
Fls Write Verify Check	✓		
Fls Max Erase Blank Check (8 -> 65536)	256		
Fls Operation Timeout (0 -> 2147483647)	0		
Fls Abort Timeout (0 -> 2147483647)	0		

Figure 6-2. Tresos Plugin snapshot for FlsGeneral form.

6.3.1 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 6-3. 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

6.3.2 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 6-4. Attribute FlsBaseAddress (FlsGeneral) detailed description

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

6.3.3 FlsCancelApi (FlsGeneral)

Compile switch to enable and disable the Fls_Cancel function.

true: API supported / function provided.

false: API not supported / function not provided

Note

when FlsCancelApi is enabled, only synchronous write mode can be set

Table 6-5. Attribute FlsCancelApi (FlsGeneral) detailed description

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

6.3.4 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 6-6. Attribute FlsCompareApi (FlsGeneral) detailed description

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

6.3.5 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 6-7. Attribute FlsDevErrorDetect (FlsGeneral) detailed description

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

6.3.6 FlsDriverIndex (FlsGeneral)

Index of the driver, used by FEE.

Table 6-8. 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

6.3.7 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 6-9. Attribute FlsGetJobResultApi (FlsGeneral) detailed description

Property	Value
Label	Fls Get Job Result Api
Type	BOOLEAN

Table continues on the next page...

Table 6-9. Attribute FlsGetJobResultApi (FlsGeneral) detailed description (continued)

Property	Value
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	true

6.3.8 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 6-10. Attribute FlsGetStatusApi (FlsGeneral) detailed description

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

6.3.9 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 6-11. Attribute FlsSetModeApi (FlsGeneral) detailed description

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

6.3.10 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 6-12. Attribute FlsTotalSize (FlsGeneral) detailed description

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

6.3.11 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 6-13. Attribute FlsUseInterrupts (FlsGeneral) detailed description

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

6.3.12 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 6-14. Attribute FlsVersionInfoApi (FlsGeneral) detailed description

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

6.3.13 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.

Table 6-15. Attribute FlsDsiHandlerApi (FlsGeneral) detailed description

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

6.3.14 FlsDsiHandlerInclude (FlsGeneral)

Data storage exception handler include file.

Table 6-16. Attribute FlsDsiHandlerInclude (FlsGeneral) detailed description

Property	Value
Label	Fls Dsi Handler Include
Type	STRING
Origin	Custom
Symbolic Name	false
Default	Exc_Types.h

6.3.15 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 6-17. Attribute FlsEraseBlankCheck (FlsGeneral) detailed description

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

6.3.16 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 6-18. Attribute FlsWriteBlankCheck (FlsGeneral) detailed description

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

6.3.17 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 6-19. Attribute FlsWriteVerifyCheck (FlsGeneral) detailed description

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

6.3.18 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 6-20. Attribute FlsMaxEraseBlankCheck (FlsGeneral) detailed description

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

6.3.19 FlsOperationTimeout (FlsGeneral)

Timeout value for Erase and Write operation (also used fot resumed oparation)

Note

Vendor specific parameter

Table 6-21. Attribute FlsOperationTimeout (FlsGeneral) detailed description

Property	Value
Label	Fls Operation Timeout
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	2147483647
Invalid	Range >=0 <=2147483647

6.3.20 FlsAbortTimeout (FlsGeneral)

Timeout value for Abort operation.

Note

Vendor specific parameter

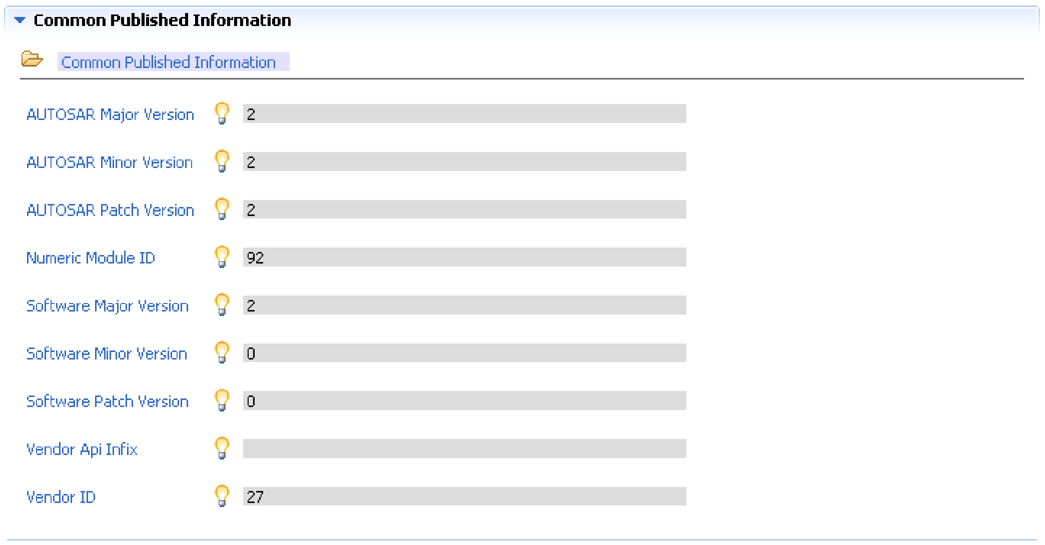
Table 6-22. Attribute FlsAbortTimeout (FlsGeneral) detailed description

Property	Value
Label	Fls Abort Timeout
Type	INTEGER
Origin	Custom
Symbolic Name	false
Default	32767
Invalid	Range >=0 <=2147483647

6.4 Form CommonPublishedInformation

CommonPublishedInformation

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



AUTOSAR Major Version	2
AUTOSAR Minor Version	2
AUTOSAR Patch Version	2
Numeric Module ID	92
Software Major Version	2
Software Minor Version	0
Software Patch Version	0
Vendor Api Infix	
Vendor ID	27

Figure 6-3. Tresos Plugin snapshot for CommonPublishedInformation form.

6.4.1 ArMajorVersion (CommonPublishedInformation)

AUTOSAR Major Version

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

Table 6-23. Attribute ArMajorVersion (CommonPublishedInformation) detailed description

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

6.4.2 ArMinorVersion (CommonPublishedInformation)

AUTOSAR Minor Version

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

Table 6-24. Attribute ArMinorVersion (CommonPublishedInformation) detailed description

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

6.4.3 ArPatchVersion (CommonPublishedInformation)

AUTOSAR Patch Version

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

Table 6-25. Attribute ArPatchVersion (CommonPublishedInformation) detailed description

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

6.4.4 ModuleId (CommonPublishedInformation)

Module ID

Module ID of this module.

Table 6-26. Attribute ModuleId (CommonPublishedInformation) detailed description

Property	Value
Label	Numeric Module ID
Origin	Custom
Symbolic Name	false
Default	92
Invalid	Range >=92 <=92

6.4.5 SwMajorVersion (CommonPublishedInformation)

Software Major Version

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

Table 6-27. Attribute SwMajorVersion (CommonPublishedInformation) detailed description

Property	Value
Label	Software Major Version

Table continues on the next page...

Table 6-27. Attribute SwMajorVersion (CommonPublishedInformation) detailed description (continued)

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

6.4.6 SwMinorVersion (CommonPublishedInformation)

Software Minor Version

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

Table 6-28. Attribute SwMinorVersion (CommonPublishedInformation) detailed description

Property	Value
Label	Software Minor Version
Origin	Custom
Symbolic Name	false
Default	9
Invalid	Range >=9 <=9

6.4.7 SwPatchVersion (CommonPublishedInformation)

Software Patch Version

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

Table 6-29. Attribute SwPatchVersion (CommonPublishedInformation) detailed description

Property	Value
Label	Software Patch Version

Table continues on the next page...

Table 6-29. Attribute SwPatchVersion (CommonPublishedInformation) detailed description (continued)

Property	Value
Origin	Custom
Symbolic Name	false
Default	1
Invalid	Range >=1 <=1

6.4.8 VendorApiInfix (CommonPublishedInformation)

Vendor Api Infix

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 6-30. Attribute VendorApiInfix (CommonPublishedInformation) detailed description

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

6.4.9 VendorId (CommonPublishedInformation)

Vendor ID

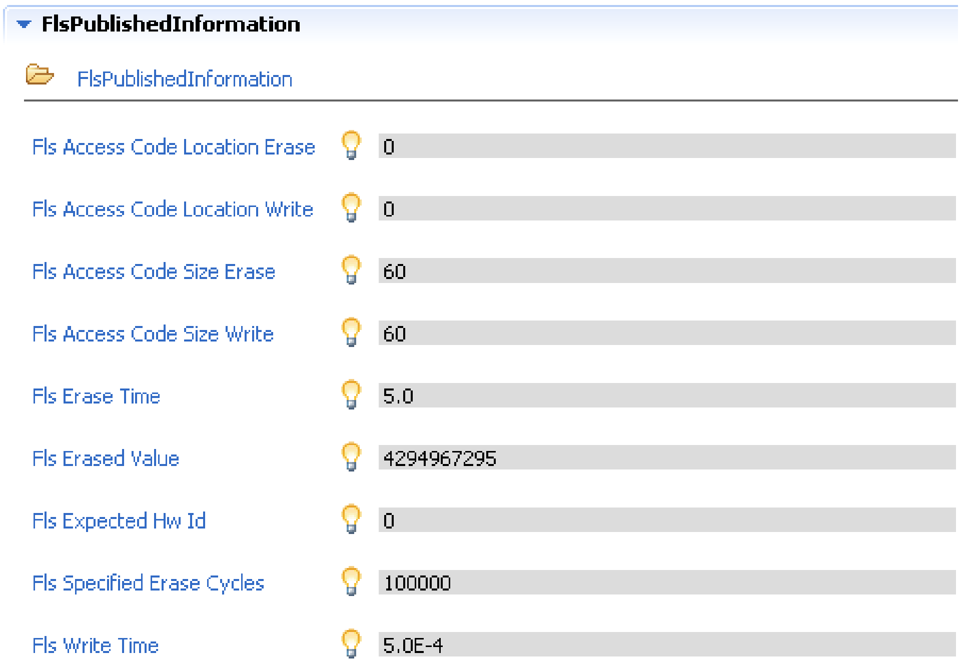
Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list.

Table 6-31. Attribute VendorId (CommonPublishedInformation) detailed description

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

6.5 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.



Fls Access Code Location Erase	0
Fls Access Code Location Write	0
Fls Access Code Size Erase	60
Fls Access Code Size Write	60
Fls Erase Time	5.0
Fls Erased Value	4294967295
Fls Expected Hw Id	0
Fls Specified Erase Cycles	100000
Fls Write Time	5.0E-4

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

6.5.1 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 it is assumed that the erase flash access code is position independent and that therefore the RAM position can be freely configured.

Table 6-32. 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 <=0 >=0

6.5.2 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 it is assumed that the write flash access code is position independent and that therefore the RAM position can be freely configured.

Table 6-33. 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 <=0 >=0

6.5.3 FlsAcSizeErase (FlsPublishedInformation)

Number of bytes in RAM needed for the erase flash access code.

Table 6-34. Attribute FlsAcSizeErase (FlsPublishedInformation) detailed description

Property	Value
Label	Fls Access Code Size Erase
Type	INTEGER_LABEL
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	60
Invalid	Range <=84 >=84

6.5.4 FlsAcSizeWrite (FlsPublishedInformation)

Number of bytes in RAM needed for the write flash access code.

Table 6-35. Attribute FlsAcSizeWrite (FlsPublishedInformation) detailed description

Property	Value
Label	Fls Access Code Size Write
Type	INTEGER_LABEL
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	60
Invalid	Range <=84 >=84

6.5.5 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 6-36. 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 >=5.0

6.5.6 FlsErasedValue (FlsPublishedInformation)

The contents of an erased flash memory cell.

Table 6-37. Attribute FlsErasedValue (FlsPublishedInformation) detailed description

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

6.5.7 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 6-38. Attribute FlsExpectedHwId (FlsPublishedInformation) detailed description

Property	Value
Label	Fls Expected Hw Id
Type	STRING_LABEL

Table continues on the next page...

Table 6-38. Attribute FlsExpectedHwId (FlsPublishedInformation) detailed description (continued)

Property	Value
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	0

6.5.8 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 6-39. 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 <=100000 >=100000

6.5.9 FlsWriteTime (FlsPublishedInformation)

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

Table 6-40. 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 >=0.0005

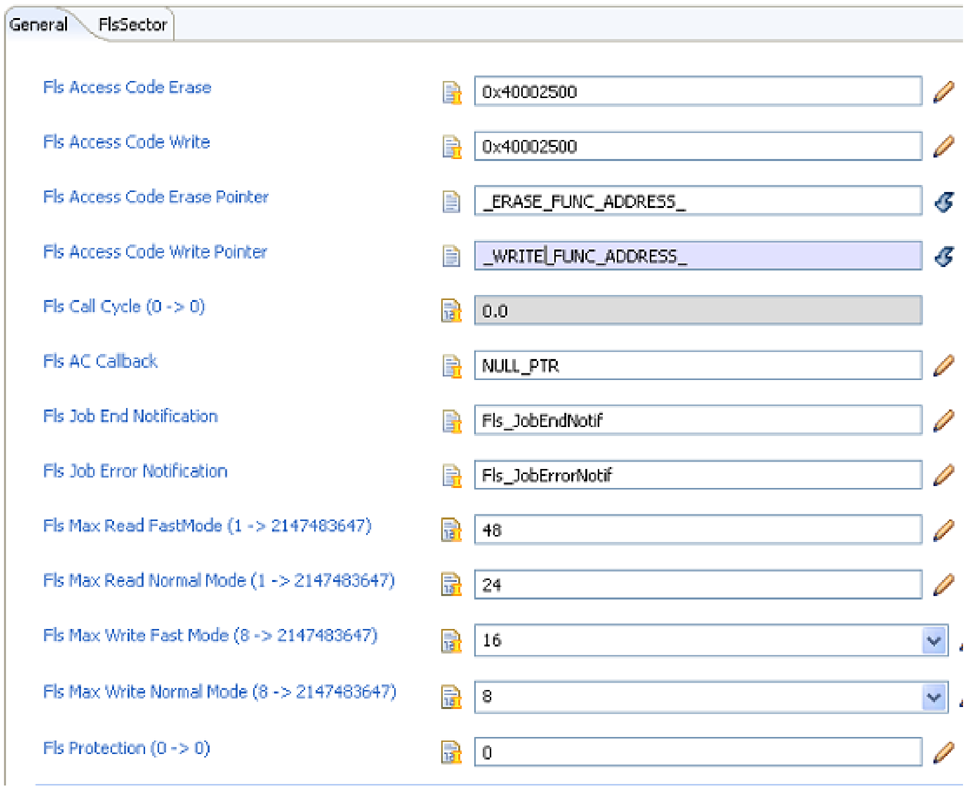
6.6 Form FlsConfigSet

Container for runtime configuration parameters of the flash driver.

Implementation Type: Fls_ConfigType.

Included forms :

- Form FlsSectorList



Fls Access Code Erase	0x40002500
Fls Access Code Write	0x40002500
Fls Access Code Erase Pointer	_ERASE_FUNC_ADDRESS_
Fls Access Code Write Pointer	_WRITE_FUNC_ADDRESS_
Fls Call Cycle (0 -> 0)	0.0
Fls AC Callback	NULL_PTR
Fls Job End Notification	Fls_JobEndNotif
Fls Job Error Notification	Fls_JobErrorNotif
Fls Max Read FastMode (1 -> 2147483647)	48
Fls Max Read Normal Mode (1 -> 2147483647)	24
Fls Max Write Fast Mode (8 -> 2147483647)	16
Fls Max Write Normal Mode (8 -> 2147483647)	8
Fls Protection (0 -> 0)	0

Figure 6-5. Tresos Plugin snapshot for FlsConfigSet form.

6.6.1 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 6-41. Attribute FlsAcErase (FlsConfigSet) detailed description

Property	Value
Label	Fls Access Code Erase
Type	FUNCTION-NAME
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	0x40002500

6.6.2 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 6-42. Attribute FlsAcWrite (FlsConfigSet) detailed description

Property	Value
Label	Fls Access Code Write
Type	FUNCTION-NAME
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	0x40002500

6.6.3 FlsAcErasePointer (FlsConfigSet)

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

Table 6-43. Attribute FlsAcErasePointer (FlsConfigSet) detailed description

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

6.6.4 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 6-44. Attribute FlsAcWritePointer (FlsConfigSet) detailed description

Property	Value
Label	Fls Access Code Write Pointer
Type	FUNCTION-NAME

Table continues on the next page...

Table 6-44. Attribute FlsAcWritePointer (FlsConfigSet) detailed description (continued)

Property	Value
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	NULL_PTR

6.6.5 FlsCallCycle (FlsConfigSet)

Cycle time of calls of the flash driver main function

Note

Not supported by the driver.

Table 6-45. Attribute FlsCallCycle (FlsConfigSet) detailed description

Property	Value
Label	Fls Call Cycle
Type	FLOAT
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	0.0
Invalid	Range <=0 >=0

6.6.6 FlsACCallback (FlsConfigSet)

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

Table 6-46. Attribute FlsACCallback (FlsConfigSet) detailed description

Property	Value
Label	Fls AC Callback
Type	FUNCTION-NAME
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	NULL_PTR

6.6.7 FlsJobEndNotification (FlsConfigSet)

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

Table 6-47. Attribute FlsJobEndNotification (FlsConfigSet) detailed description

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

6.6.8 FlsJobErrorNotification (FlsConfigSet)

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

Table 6-48. Attribute FlsJobErrorNotification (FlsConfigSet) detailed description

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

6.6.9 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 6-49. Attribute FlsMaxReadFastMode (FlsConfigSet) detailed description

Property	Value
Label	Fls Max Read FastMode
Type	INTEGER
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	1048576
Invalid	Range <=2147483647 >=1

6.6.10 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 6-50. 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 <=2147483647 >=1

6.6.11 FlsMaxWriteFastMode (FlsConfigSet)

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

Table 6-51. Attribute FlsMaxWriteFastMode (FlsConfigSet) detailed description

Property	Value
Label	Fls Max Write Fast Mode

Table continues on the next page...

Table 6-51. Attribute FlsMaxWriteFastMode (FlsConfigSet) detailed description (continued)

Property	Value
Type	INTEGER
Origin	AUTOSAR_ECUC
Symbolic Name	false
Default	256
Invalid	Range <=2147483647 >=8

6.6.12 FlsMaxWriteNormalMode (FlsConfigSet)

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

Table 6-52. 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 <=2147483647 >=8

6.6.13 FlsProtection (FlsConfigSet)

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

Note

Not supported by the driver.

Table 6-53. Attribute FlsProtection (FlsConfigSet) detailed description

Property	Value
Label	Fls Protection
Type	INTEGER

Table continues on the next page...

Table 6-53. Attribute FlsProtection (FlsConfigSet) detailed description (continued)

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

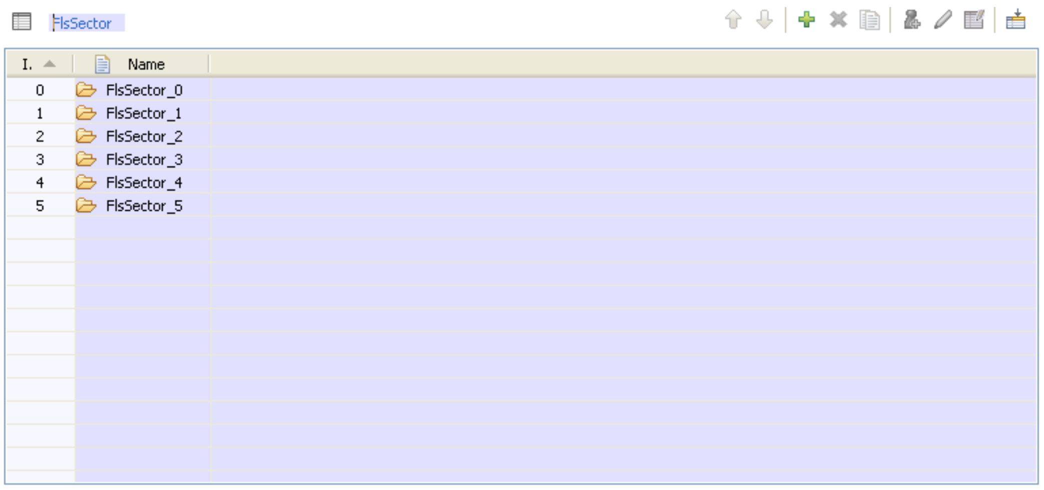
6.6.14 Form FlsSectorList

List of flashable sectors and pages.

Is included by form : Form FlsConfigSet

Included forms :

- Form FlsSector



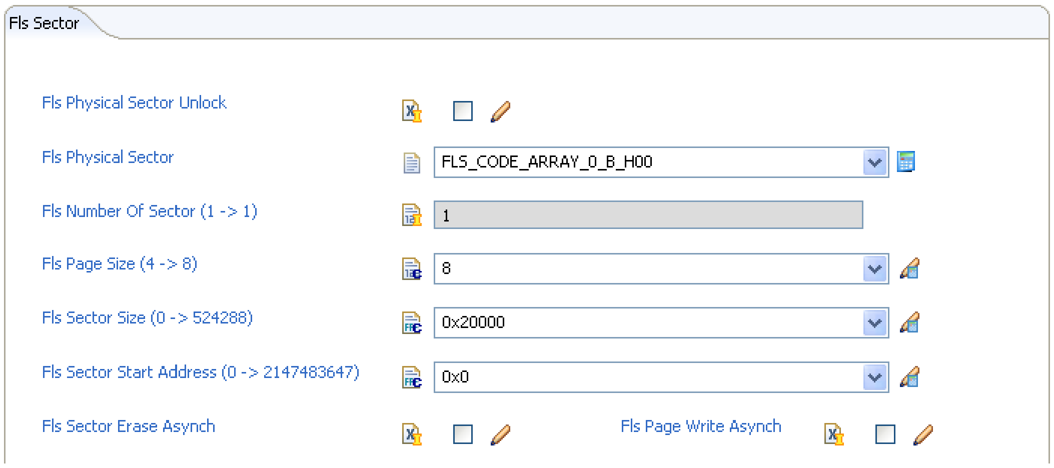
I.	Name
0	FlsSector_0
1	FlsSector_1
2	FlsSector_2
3	FlsSector_3
4	FlsSector_4
5	FlsSector_5

Figure 6-6. Tresos Plugin snapshot for FlsSectorList form.

6.6.14.1 Form FlsSector

Configuration description of a flashable sector

Is included by form : Form FlsSectorList



Fls Physical Sector Unlock ☐
Fls Physical Sector FLS_CODE_ARRAY_0_B_H00
Fls Number Of Sector (1 -> 1) 1
Fls Page Size (4 -> 8) 8
Fls Sector Size (0 -> 524288) 0x20000
Fls Sector Start Address (0 -> 2147483647) 0x0
Fls Sector Erase Asynch ☐
Fls Page Write Asynch ☐

Figure 6-7. Tresos Plugin snapshot for FlsSector form.

6.6.14.1.1 FlsPhysicalSectorUnlock (FlsSector)

FlsPhysicalSector the 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 6-54. Attribute FlsPhysicalSectorUnlock (FlsSector) detailed description

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

6.6.14.1.2 FlsPhysicalSector (FlsSector)

Physical flash device sector.

Table 6-55. Attribute FlsPhysicalSector (FlsSector) detailed description

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

6.6.14.1.3 FlsNumberOfSectors (FlsSector)

Number of continuous sectors with the above characteristics.

Note

Not supported by the driver.

Table 6-56. Attribute FlsNumberOfSectors (FlsSector) detailed description

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

6.6.14.1.4 FlsPageSize (FlsSector)

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

- For Code Flash page size is 8 byte
- For Data Flash page size is 8 byte
- For Data Flash Optimized (DFO) page size is 4 byte

Table 6-57. Attribute FlsPageSize (FlsSector) detailed description

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

6.6.14.1.5 FlsSectorSize (FlsSector)

Size of this sector.

Implementation Type: Fls_LengthType.

Table 6-58. Attribute FlsSectorSize (FlsSector) detailed description

Property	Value
Label	Fls Sector Size
Type	INTEGER
Origin	AUTOSAR_ECUC
Symbolic Name	false
Invalid	Range <=524288 >=0

6.6.14.1.6 FlsSectorStartaddress (FlsSector)

Start address of this sector.

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

Example:

Suppose user want to configure

FLS_DATA_ARRAY_0_L00 which has physical address 0x800000 and size 0x4000
FLS_DATA_ARRAY_0_L02 which has physical address 0x808000 and size 0x4000

When user configure sectors as:

Fls sector 1 - FLS_DATA_ARRAY_0_L00

Fls sector 2 - FLS_DATA_ARRAY_0_L02,

"Fls Sector Start Address" for FlsSector_0 will be 0 and "Fls Sector Start Address" for FlsSector_1 will be 0x4000

If user want to write FLS_DATA_ARRAY_0_L02, user need to write to the logical address 0x4000 - 0x7FFF.

If user want to erase it, user need to erase sector from address 0x4000 with size 0x4000.
Note: If the user do not need to set the "Fls Sector Start Address" and "Fls Sector Size" let it be automatically computed.

Implementation Type: Fls_AddressType.

Table 6-59. Attribute FlsSectorStartaddress (FlsSector) detailed description

Property	Value
Label	Fls Sector Start Address
Type	INTEGER
Origin	AUTOSAR_ECUC
Symbolic Name	false
Invalid	Range <=2147483647 >=0

6.6.14.1.7 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 6-60. Attribute FlsSectorEraseAsynch (FlsSector) detailed description

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

6.6.14.1.8 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 6-61. 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:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: UM14FLSASR3.0R2.0.0
Rev. 2.6