SCOPE OF APPLICATION All Project/Engineering	HYUNDAI AutoEver	SHT/SHTS 1 / 43
Responsibility: Classic Autosar Team	AUTOSAR Fota User Manual	DOC. NO

AUTOSAR Fota User Manual

Document Change History			
Date (YYYY-MM-DD)	Ver.	Editor	Content
2022-12-24	0.1.0.0	AnhPNT2	• Initial Version
2023-04-26	0.2.0.0	VuPH6	• Self-programming feature
2023-08-07	0.3.0.0	AnhPNT2	• Update of PFlash interfaces and user-callouts • OTA Dual memory feature
2023-10-27	1.0.0.0	AnhPNT2	• Modify realease notes • Modify functionalities description • Modify API specification • Modify configuration guide
2023-11-08	1.0.1.0	DJLee	• Multiple SwUnit Update feature add • Comment add
2023-12-20	1.0.1.0	AnhPNT2	• Limitation Item add • Add explanation and dependencies for SwUnit configuration
2024-03-26	1.1.0.0	HJOh	• Add explanation · User-callout and MCU specific settings

Edition Date: 2024-03-26	File Name Fota_UM.pdf	Creation **Hyunjung Oh** 2024-03-26	Check **Myeongkwon Choi** 2024-03-26	Approval **Myeongkwon Choi** 2024-03-26
Document Management System				

Table of Contents

1. Overview

This document is created based on AUTOSAR standard SRS/SWS and AUTOEVER vendor specific requirement.
For details functional description, please refer to the Reference Documents.

The following terms mean:

- Changeable : Can configure by user
- Fixed : Can not change this configuration by user
- Not Supported : Can not use this configuration

2. Reference

Sl. No.	Title	Version
1.	AUTOSAR_EXP_FirmwareOverTheAir.pdf	R21-11

3. Limitations and Deviations

3.1 Limitations

When using SecurBoot and FBL update function
Use Hae HSM V2.8.0 or higher

When using Traveo-II memory-swap update
Use Hae HSM V2.6.2 or higher

In principle, the versions of each module of FBL and RTSW must be the same.

3.2 Deviations

None

4. Functionality

Fota module, along with Program Flash Memory driver, are considered as core module in the lite AUTOSAR platform achieving the functionalitiy of a flashing bootloader. Its main functionality includes: boot management, firmware update and support features for OTA campaign.

4.1 Boot Manager (BM)

Fota module includes a module-embedded boot manager or a separated boot manager, depend on users' preference. In the case of FBL self reprogramming, the separated BM usage is mandatory to allow the replacement of current FBL with the updated one through operation of the FBL-updater.

The BM main responsibility is to help decide the entry point for ECU software in different scenarios:

- **Check for previous Application running cycles' requests to boot into FBL for software updates/FBL self update/OTA Application swapping.**
- **Check the validity of firmware images and select software entry point base on jumping priority.**
- **If there is not any valid application software present, the BM will perform jump to the current FBL by default.**

4.1.1 Separated Boot manager

The separated BM is built as an independent firmware and is always run firstly on each ECU power cycles.

In the scenario of OTA mode, the BM recognizes the application programming request and jump to FBL. After FBL finished programming of new Application, the control is given back and BM is capable to analyze the application jumping priorities then performs the jump appropriately.

In the scenario of FBL self reprogramming, the BM recognizes the FBL re-programming request and jump to FBL. After FBL finished programming of the FBL-updater, control is given back and the BM perform jumping to FBL-updater.

4.2 Firmware reprogramming

Fota module exists as a complex device driver in BSW platform, between the Diagnostics modules and Code flash memory driver in the workflow and functions as the core handling module for firmware updating. It supports firmware reprogramming feature by handling firmware data chunks received from Dcm module, processing the data and finally triggering memory driver to write firmware data into program flash.

4.2.1 Flexible configuration of firmware instances

The configuration of Fota module allows users to configure:

- **Multiple software instances and each instance structure flexibly.**
- **Multiple metadata blocks and handling of firmware metadata flexibly.**
- **Multiple physical firmware blocks and separated processing rule for each firmware block (decryption, multiple pre-/post-processing call-outs).**

- Multiple signature block and separated signature verification method, verification target for checking firmware image integrity.

4.2.2 Interaction with other modules in reprogramming workflow

Since the UDS services and routines involving in the reprogramming process are realized as callout function, Fota provided such functions receiving Dcm data buffer for further processing. When receiving diagnostics request from update master ECU, Dcm module will trigger Fota following each steps in the reprogramming sequence by provoking according interfaces. This process ends and result is provided when a final response (OK/NOK) was provided from Fota module.

4.2.2.1 Code flash area erase

To prepare for new firmware flashing, the area of old firmware shall be erased. The update-master indicates the erase memory request and provide through erase area routine of diagnostics stack. Fota module receives the request and calculate necessary information about the firmware instance to be erased. The information about area address and erase length is then sent to Code flash driver for actual memory erase execution.

4.2.2.2 Firmware data transfer

The sequence to transfer firmware data include 3 Diag services: Request Download ($34), Transfer Data ($36) and Transfer Exit ($37). Fota provide according APIs for each service handling purpose and the operation state of Fota will alter accordingly with each of API calls during reprogramming flow. Fota will provoke Csm interfaces to further process the data block, then Code flash driver interfaces to finally write the firmware to permanent memory.

4.2.2.3 Checking the integrity of new firmware

After finishing the new firmware download, check programming dependencies (CPD) routine is provided to verify the firmware integrity. Fota also uses Csm services based on the verification configuration to ensure the validity of new firmware, then triggers writing of the partition flag or erasing of firmware depending on the integrity checking process result.

4.2.3 Handling of firmware data

4.2.3.1 Metadata blocks

In case secured flashing is supported and encrypted firmware image is used, metadata block presents to provide necessary elements cryptographic activities. Fota supports metadata block delimiter and size validation. The metadata info is then extracted and shall be processed by selected SHA hashing algorithm. The index of pre-shared secret key is determined by result of modulo operation of the hash digest to N, where N is the number of pre-shared secret keys stored in secured flash compliant ECU.

The cipher key for decryption of firmware blocks is generated through key derivation phase. The integrity verification of derived cipher key is also ensured by MAC verification algorithm. The derived keys are then stored in configured slots for usage in firmware block processing rules.

4.2.3.2 Firmware blocks

Fota handles firmware blocks, which contain the new software to be updated, by physical bl ocks and mapped block processing rules. In case of encrypted firmware, either call-outs from user defined library or Csm jobs from Crypto stack could be configured and stored cipher keys from metadata block processing shall be used to decrypt firmware data. For more options in the process of firmware block, multiple user call-outs before and after each block processing are also allowed. Firmware data buffer is forwarded to code flash driver for permanent write in code flash memory.

4.2.3.3 Signature blocks

Secured flash compliant firmware signatures are created by asymetric algorithm with OEM's private key and flashed along to ensure the authenticity of firmware image. Multiple verification entities with selected algorithm and targeted firmware blocks could be configured. On receiving request for verification, Fota module shall process each verification entities and trigger Csm services to use securely stored public key to verify the targeted firmware blocks.

4.2.3.4 Partition flags

A specific address in code flash memory is designated for partition flag. Content of this memory location is to be updated when the verification phase finished. Depend on the verification result, a specific value for valid new firmware indication and its calculated boot priority shall be written. The updated firmware could be considered for activation in next boot sequence.

4.3 OTA feature

In this feature, Fota module's presence is also required in the application firmware. This allows OTA Master to, even while ECU is running an application firmware carrying out its designated functionalities, trigger firmware update sequence on the inactive memory bank.

To support the feature, firmware data processing mechanism and configuration of Fota shall include alternative address processing capability. Additional routines for OTA feature are also provided: Erase Target Area, Check Active Area, and Swap Active Area so the OTA Master can query currently active memory bank and perform update to the other one then trigger the swap request so the ECU start to use the new updated application on next power cycle.

4.4 Support of ES98765-02

Fota provides support for both 1st and 2nd generation reprogramming specification regulated in ES98765-01 and ES98765-02, respectively. In case of 2nd generation specification, Fota provide additional routines such as Check Memory and Finish Update.

4.5 Multiple SwUnit Update feature

Fota supports split updates in the SwUnit format specified in ES98765-01 and ES98765-02.
However, it is supported if the HW can be divided according to ES.
Supported targets are listed in the table below

	Single Type	MMU Type	Non-MMU
ES98765-01	O	X	X
ES98765-02	O	O	X

O: Multiple SwUnit Support / X: Only Single SwUnit Support

When using multiple SwUnits, all SwUnit and Block areas except the management area (Partition Flag block) must be flashed before reprogramming

In particular, the Signature Block area must have a signature that can be authenticated in that area even during initial flashing. This is because when using redundancy, the corresponding area is synchronized between banks, and in the case of the FBL area, it is used to send to the HSM in preparation for SecureBoot.

※ If there is no signature during rollback-rule or Activation-rule, SecureBoot lock occurs.

Even if the entire SwUnit is not updated in the processing rule, the entire SwUnit must complete CheckMemory. (including previous Procssing rules)
If not, activation rules can not be performed.

4.6 User callout function that must be checked

4.6.1 FUNC(Std_ReturnType, FOTA_CODE) Fota_DualMemDownGradeChk_Callout(void)

This function is a redundancy downgrade check function.
In the case of the MMU method, E_OK is returned only when the version information of the Inactive Bank location is greater than or equal to the version information of the Active Bank.
If not, it returns E_NOT_OK.

Although it is an MMU type, the 2nd generation OTA specification supports multiple SwUnits. If multiple SwUnits are set, all multiple SwUnits must be compared in this function.

In the case of the Non MMU method, the Inactive partition must be checked and the version address found accordingly. If the inactive partition is greater than or equal to the active partition, E_OK is returned. Otherwise, E_NOT_OK is returned.

However, in the case of OEUK Security Level, E_OK is returned regardless of version comparison.

4.6.2 FUNC(Std_ReturnType, FOTA_CODE) Fota_DualMemSwUnitsVerDependChk_Callout(void)

Dependency check between SwUnits must be performed. When versions cannot be combined, NG processing must be performed.
In case of a single SwUnit, it returns E_OK. (In case of Non MMU Swap, it is a single SwUnit)

You must check the version dependency between SwUnits in the Inactive area.

4.6.3 FUNC(Std_ReturnType, FOTA_CODE) Fota_DualMemSwUnitsVerDependChk_Callout(void)

Defines ResetOperTime, a response parameter for the SwapActiveArea command.

4.6.4 FUNC(Fota_JobResultType_CallOut, FOTA_CODE) Fota_SecureBootMacUpdate(void)

If FBL_Type is included in the SwUnit setting, the function is called during CPD.

If FBL SwUnit is set and SecureBoot is enabled, this function must be filled.
In that case, the Hyundai AutoEver HSM version must be a higher version, including version V2.7.0.

Warning) If this function does not work properly, it may cause SecureBoot locking. If MacUpdate does not work properly and E_OK returns, it may cause SecureBoot locking.

4.7 User callout function for self-development

There may be various types of update targets, such as sub-controllers or external Flash. This module deals only with standard MCU internal flash.
However, custom development is possible through the provided callout.

The callout below is a callout when a reprogramming-only command is received with a SwUnit N

umber that is not set in Fota SwUnit.
If RTSW uses the internal flash and needs to update the external flash, the external flash update function is
You can implement it in Callout.

This callout is a re-branched form of DCM callout and therefore follows its characteristics. (Execution time, stack size, etc.)

When receiving ReadActiveArea Commend

```
FUNC (Std_ReturnType, FOTA_CODE) Fota_ProcessReadActiveArea_UserCallout
(
  VAR(uint16, AUTOMATIC) InEcuSwUnit,
  VAR(uint8, AUTOMATIC) OpStatus,
  P2VAR(uint8, AUTOMATIC, FOTA_PRIVATE_DATA) LpOut_MemoryArea,
  P2VAR(uint8, AUTOMATIC, FOTA_PRIVATE_DATA) LpErrorCode
)
```

When receiving CheckProgramDependency Commend of 2nd Gen OTA

```
FUNC (Std_ReturnType, FOTA_CODE) Fota_ProcessActivate_UserCallout
(
  VAR(uint8, AUTOMATIC) InMemArea,
  VAR(uint16, AUTOMATIC) InEcuSwUnit,
  VAR(uint8, AUTOMATIC) OpStatus,
  P2VAR(uint8, AUTOMATIC, FOTA_PRIVATE_DATA) LpErrorCode
)
```

When receiving EraseMemory Commend or EraseTargetArea Commend

```
FUNC (Std_ReturnType, FOTA_CODE) Fota_ProcessEraseTargetArea_UserCallout
(
  VAR(uint8, AUTOMATIC) InMemArea,
  VAR(uint16, AUTOMATIC) InEcuSwUnit,
  VAR(uint8, AUTOMATIC) OpStatus,
  P2VAR(uint8, AUTOMATIC, FOTA_PRIVATE_DATA) LpOut_MemoryArea,
  P2VAR(uint8, AUTOMATIC, FOTA_PRIVATE_DATA) LpErrorCode
)
```

When receiving RequestDownload Commend

```
FUNC(Std_ReturnType, FOTA_CODE) Fota_ProcessRequestDownload_UserCallout
(
  Dcm_OpStatusType OpStatus,
  uint8 DataFormatIdentifier,
  uint32 MemoryAddress,
  uint32 MemorySize,
  P2VAR(uint32, AUTOMATIC, FOTA_PRIVATE_DATA) LpBlockLength,
  P2VAR(uint8, AUTOMATIC, FOTA_PRIVATE_DATA) LpErrorCode
)
```

When receiving TransferData Commend

```
FUNC(Dcm_ReturnWriteMemoryType, FOTA_CODE) Fota_ProcessTransferDataWrite_UserCallout
(
  Dcm_OpStatusType OpStatus,
  uint8 MemoryIdentifier /* Not Supported Argument */,
  uint32 MemoryAddress,
  uint32 MemoryWriteLen,
  P2CONST(uint8, AUTOMATIC, FOTA_PRIVATE_DATA) pWriteData
)
```

```
FUNC (Std_ReturnType, FOTA_CODE) Fota_ProcessRequestTransferExit_UserCallout
(
  Dcm_OpStatusType OpStatus,
  P2VAR(uint8, AUTOMATIC, DCM_APPL_DATA) LpMemoryData,
  uint32* LulParameterRecordSize,
  P2VAR(Dcm_NegativeResponseCodeType, AUTOMATIC, FOTA_PRIVATE_DATA)LpErrorCode
)
```

When receiving CheckingProgramDependency commend of 1st gen OTA
When receiving CheckMemory commend of 2nd gen OTA

```
FUNC (Std_ReturnType, FOTA_CODE) Fota_ProcessVerify_UserCallout
(
  VAR(uint8, AUTOMATIC) InMemArea,
  VAR(uint16, AUTOMATIC) InEcuSwUnit,
  VAR(uint8, AUTOMATIC) OpStatus,
  P2VAR(uint8, AUTOMATIC, FOTA_PRIVATE_DATA) LpOut_MemoryArea,
  P2VAR(uint8, AUTOMATIC, FOTA_PRIVATE_DATA) LpErrorCode
)
```

4.8 DataSync between Inactive Bank and Active Bank

Memory-Swap controller which use multi-SwUnit update is supported only when using 2nd generation OTA and MMU Swap type.
In this case, if multiple SwUnit settings are made, DataSync may occur when receiving the Finish Update command or EraseMemory.

DataSync is an operation to copy data from ActiveBank to Inactive Bank

For a single SwUnit, the DataSync feature is disabled. (Including Non-MMU)

In this case, the following items must be observed.

All SwUnits that are synchronized must be written in a valid format.

Blocks within SwUnit must be written as much as the block setting size.
In other words, the block within the SwUnit that is synced corresponds to TargetBlock in the SignatureBlock setting.
It also includes the Signature Block itself.

When flashing with T32 or Gang, the area must be in a format that allows signature authenticati on according to Fota settings. (Used when Mac-updating)

the signature area is not included when flashing the sre file. Therefore, signature authentication is not possible.

However, if the sre file is derived through asims in Secureflash 1.0 format, booting becomes impossible because there is no partition_flag value.

Hint)
The initial Flash image (T32/Gang) can be generated in Secure Flash 1.0 format through Asims. However, it must include SECTION 'SK'.

5. Configuration Guide

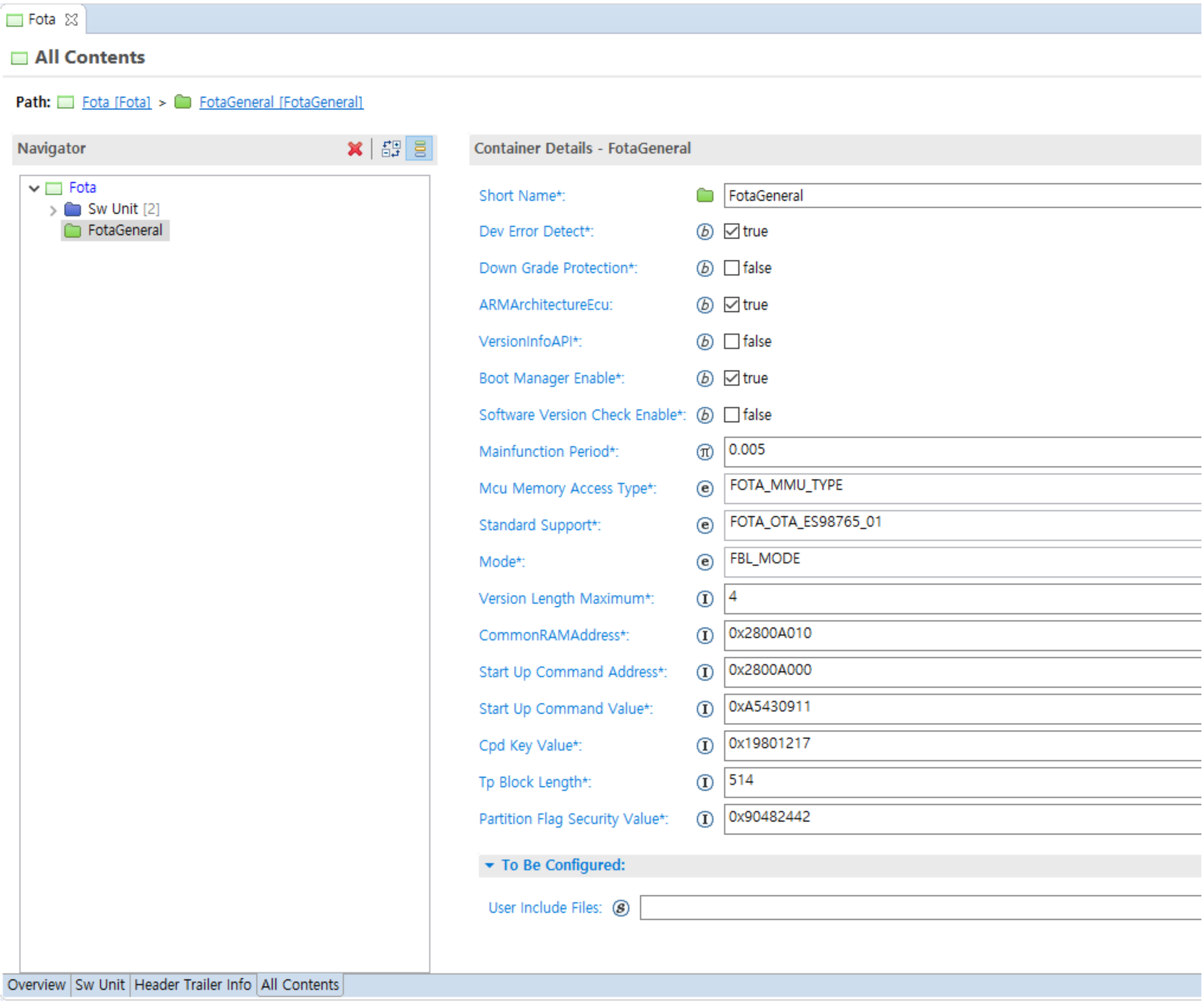
5.1 Fota Container

Container Name	Value	Category
FotaGeneral	[1] General configuration for Fota module	F
FotaSwUnit	[1..255] Specific configuration for Firmware instances	C

5.2 FotaGeneral

This section contains general configurations for operation of Fota module.

Parameters Name	Value	Category
Dev Error Detect	[boolean] Enable DET report feature	C
Down Grade Protection	[boolean] Enable Downgrade prevention feature	C
ARM Architecture ECU	[boolean] Specify whether current ECU platform is ARM-architecture based for supporting SW jump feature	C
Version Info API	[boolean] Enable of supporting interface for version info	C
Boot Manager Enable	[boolean] Specify whether the embedded BM or a separated BM firmware shall be used [boolean]	C
Software Version Check Enable	[boolean] Enable software version check feature	C
User Include Files	[string] Specify user code files need to be included	C
Mainfunction Period	[float] The period of Fota mainfunction in Os workflow	C
Mcu Memory Access Type	Specify the memory access type of Mcu platform FOTA_MMU_TYPE/ FOTA_NON_MMU_TYPE/ FOTA_SINGLE_TYPE	C
Standard Support	Specify the reprogramming standard FOTA_OTA_ES98765_01/ FOTA_OTA_ES98765_02	C
Mode	Specify operation mode of Fota module FBL_MODE/ APP_MODE/ UPDATER_MODE	C
Version Length Maximum	[Integer] Specify maximum length of firmware version block	C
Common RAM Address	Specify the assigned address for shared RAM section between FBL and Application SW to support retaining operation context between Diagnostics sessions [Integer]	C
Start Up Command Address	[Integer] Specify the assigned address for Start-up command to support retaining operation context between power cycles	C
Start Up Command Value	[Integer] The specific value for indicating the Start-up command	C
Cpd Key Value	[Integer] The specific value of the CPD key	C
Tp Block Length	[Integer] Length of each Tp block used in data transfer to support allocating of data buffers	C
Partition Flag Security Value	[Integer] The specific value for indicating a valid SW instance	C



Short Name*: FotaGeneral
Dev Error Detect*: ☑ true
Down Grade Protection*: □ false
ARMArchitectureEcu: ☑ true
VersionInfoAPI*: □ false
Boot Manager Enable*: ☑ true
Software Version Check Enable*: □ false
Mainfunction Period*: 0.005
Mcu Memory Access Type*: FOTA_MMU_TYPE
Standard Support*: FOTA_OTA_ES98765_01
Mode*: FBL_MODE
Version Length Maximum*: 4
CommonRAMAddress*: 0x2800A010
Start Up Command Address*: 0x2800A000
Start Up Command Value*: 0xA5430911
Cpd Key Value*: 0x19801217
Tp Block Length*: 514
Partition Flag Security Value*: 0x90482442
User Include Files:

Figure 1. Sample FotaGeneral configuration

5.3 FotaSwUnit

5.3.1 Fota Sw Unit

This section contains specific configurations related to the firmware instances to be installed/updated. Multiple FotaSWUnit can also be configured in concept of memory swap.

The SwUnit set in Fota is limited to the update target. The SwUnit set in Fota depends much on the Mcu Memory Access Type and Standard Support configuration.

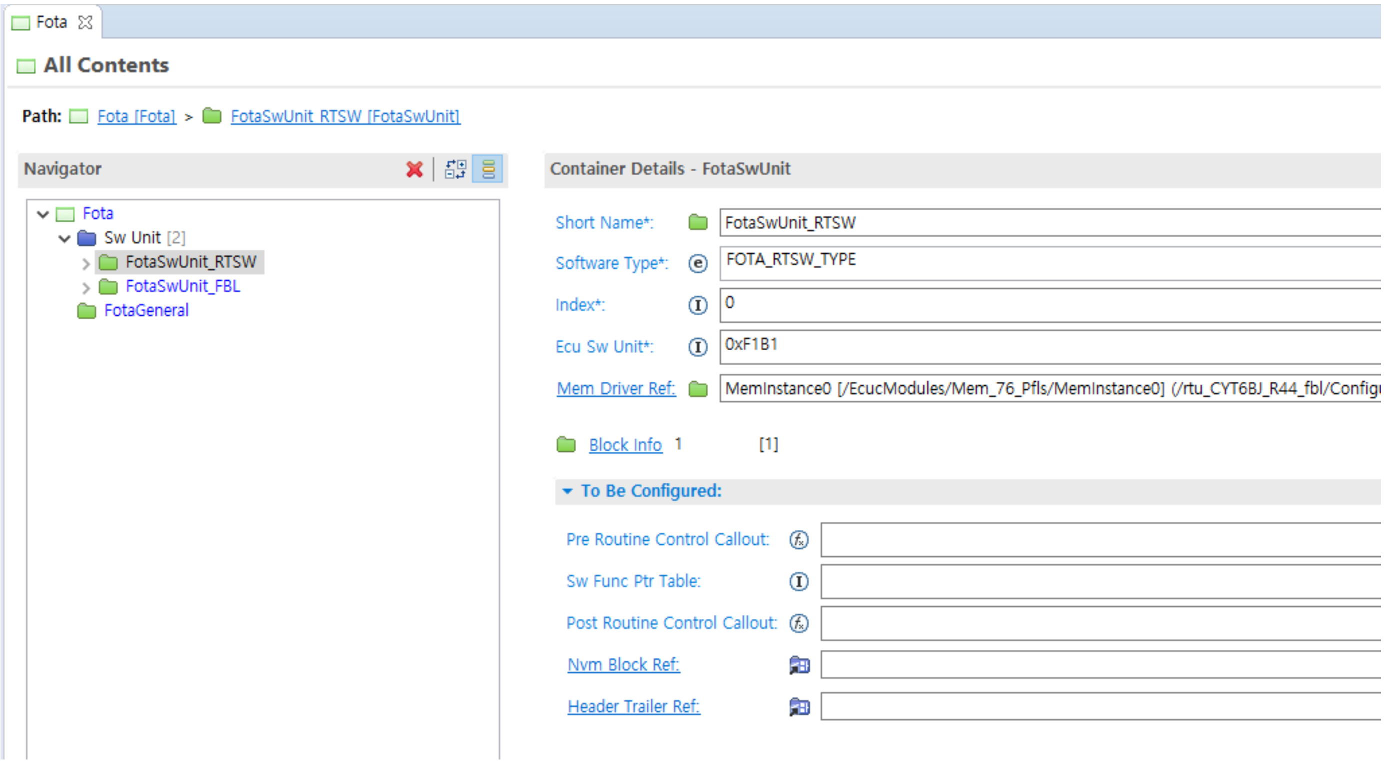
- In case of Single type, memory swap feature is not available. Only a single instance of each SwUnit should be configured.

- In case of MMU type, memory swap feature is enabled and depend on which reprogramming standard is being supported, multiple SwUnit is allowed or not. For each SwUnit, only one instance with the address information of Active Area should be set. The information of SwUnit on Inactive Area could be proceeded automatically by hardware support.

- In case of Non-MMU type, the memory swap concept is still supported, but by software method, not hardware supported. Therefore, SwUnit must be set to two SwUnitTypes corresponding to both Partitions. Multiple Sw Unit is not supported in this case, therefore, other SwUnits cannot be configured.

Parameters Name	Value	Category
Software Type	Specify the software type of firmware instance FOTA_FBL_TYPE/ FOTA_UPDATER_TYPE/ FOTA_NEW_FBL_TYPE/ FOTA_RTSW_TYPE/ FOTA_RTSW_PARTA_TYPE/ FOTA_RTSW_PARTB_TYPE/ FOTA_RTSW_DATA_TYPE/ FOTA_RTSW_USER_TYPE	C
Index	[integer] Index of SW instance for features that operate by index	C
ECU Sw Unit	[integer] Identification of SW instance for features that operate by Sw Unit Id	C
Pre Routine Control Callout	[string] Specify the callout function to be provoke before process routine control	C
Sw Func Ptr Table	[string] Specify the address of function pointer table for reprogramming of this Sw unit	C
Post Routine Control Callout	[string] Specify the callout function to be provoke after process routine control	C
Mem Driver Ref	Reference to the assigned memory instance in Mem driver for this Sw unit	C
Nvm Block Ref	Reference to the assigned memory instance in Mem driver for this Sw unit	C
Header Trailer Ref	Reference to the header/trailer information of this Sw unit	C

Container Name	Value	Category
Fota Block Info	Configuration for firmware blocks in this Sw unit	C



Fota

All Contents

Path: Fota [Fota] > FotaSwUnit_RTSW [FotaSwUnit]

Navigator

- Fota
  - Sw Unit [2]
    - FotaSwUnit_RTSW
    - FotaSwUnit_FBL
  - FotaGeneral

Container Details - FotaSwUnit

Short Name*: FotaSwUnit_RTSW

Software Type*: FOTA_RTSW_TYPE

Index*: 0

Ecu Sw Unit*: 0xF1B1

Mem Driver Ref: MemInstance0 [/EcucModules/Mem_76_Pfls/MemInstance0] (/rtu_CYT6BJ_R44_fbl/Configu

Block Info 1 [1]

To Be Configured:

Pre Routine Control Callout:

Sw Func Ptr Table:

Post Routine Control Callout:

Nvm Block Ref:

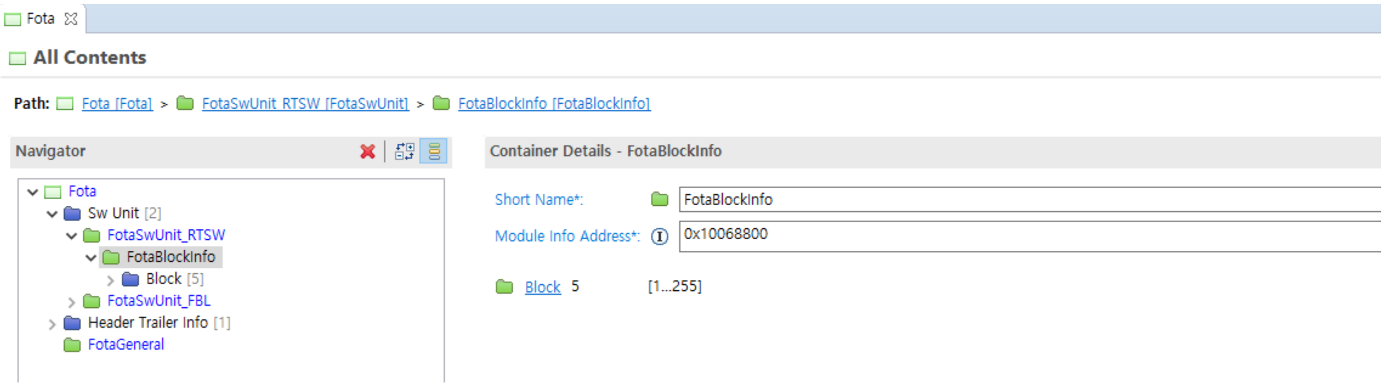
Header Trailer Ref:

Figure 2 Sample FotaSwUnit configuration

5.3.2 Fota Block Info

Parameters Name	Value	Category
Module Info Address	[Integer] Specify the addres of Sw unit information	C

Container Name	Value	Category
FotaBlock	[1..255] Configuration for firmware blocks in this Sw unit	C



Fota

All Contents

Path: Fota [Fota] > FotaSwUnit_RTSW [FotaSwUnit] > FotaBlockInfo [FotaBlockInfo]

Navigator

- Fota
  - Sw Unit [2]
    - FotaSwUnit_RTSW
      - FotaBlockInfo
        - Block [5]
    - FotaSwUnit_FBL
  - Header Trailer Info [1]
  - FotaGeneral

Container Details - FotaBlockInfo

Short Name*: FotaBlockInfo

Module Info Address*: 0x10068800

Block 5 [1...255]

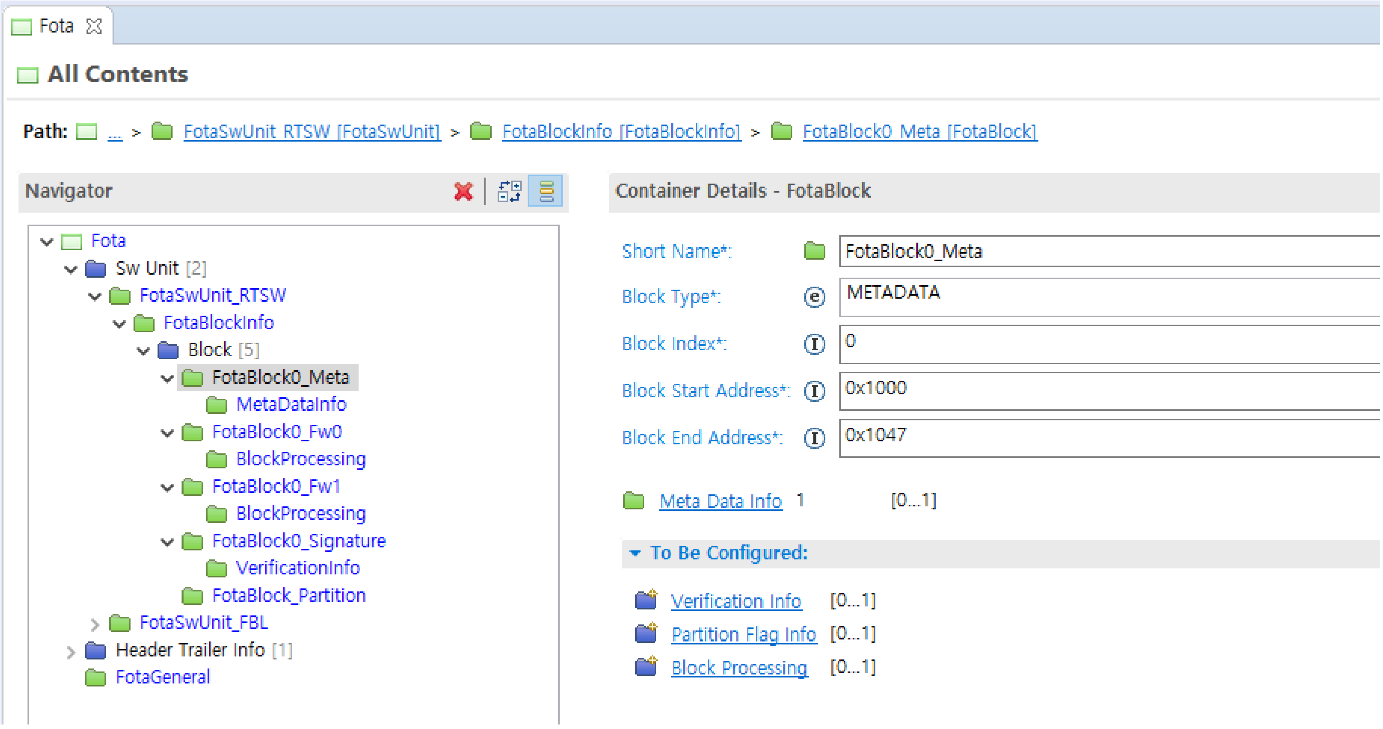
Figure 3 Sample FotaBlockInfo configuration

5.3.3 Fota Block

Parameters Name	Value	Category
Block Type	Specify the type of firmware data blocks METADATA/ FIRMWARE/ SIGNATURE/ PARTITION_FLAG/ CRC	C
Block Index	[integer] Specify the index of firmware data blocks	C

Block Start Address	[Integer] Specify the start address of firmware data blocks	C
Block End Address	[integer] Specify the end address of firmware data blocks	C

Container Name	Value	Category
MetaDataInfo	Configuration for metadata processing info of this Sw unit	C
BlockProcessing	Configuration for firmware data block processing info of this Sw unit	C
VerificationInfo	Configuration for verification processing info of this Sw unit	C



Fota

All Contents

Path: ... > FotaSwUnit_RTSW [FotaSwUnit] > FotaBlockInfo [FotaBlockInfo] > FotaBlock0_Meta [FotaBlock]

Navigator

```
Fota
  Sw Unit [2]
    FotaSwUnit_RTSW
      FotaBlockInfo
        Block [5]
          FotaBlock0_Meta
            MetaDataInfo
          FotaBlock0_Fw0
            BlockProcessing
          FotaBlock0_Fw1
            BlockProcessing
          FotaBlock0_Signature
            VerificationInfo
          FotaBlock_Partition
    FotaSwUnit_FBL
  Header Trailer Info [1]
  FotaGeneral
```

Container Details - FotaBlock

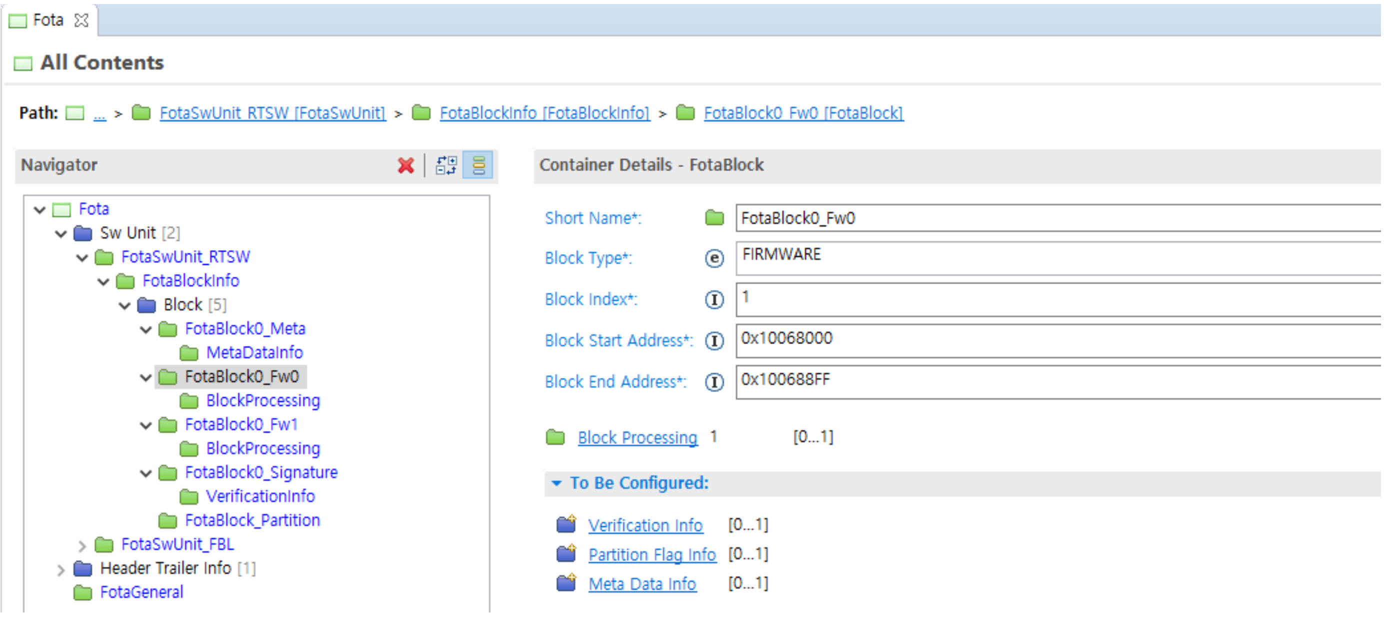
Short Name*: FotaBlock0_Meta
Block Type*: METADATA
Block Index*: 0
Block Start Address*: 0x1000
Block End Address*: 0x1047

Meta Data Info 1 [0...1]

To Be Configured:

Verification Info [0...1]
Partition Flag Info [0...1]
Block Processing [0...1]

Figure 4 Sample FotaBlock configuration for Metadata block



Fota

All Contents

Path: ... > FotaSwUnit_RTSW [FotaSwUnit] > FotaBlockInfo [FotaBlockInfo] > FotaBlock0_Fw0 [FotaBlock]

Navigator

```
Fota
  Sw Unit [2]
    FotaSwUnit_RTSW
      FotaBlockInfo
        Block [5]
          FotaBlock0_Meta
            MetaDataInfo
          FotaBlock0_Fw0
            BlockProcessing
          FotaBlock0_Fw1
            BlockProcessing
          FotaBlock0_Signature
            VerificationInfo
          FotaBlock_Partition
    FotaSwUnit_FBL
  Header Trailer Info [1]
  FotaGeneral
```

Container Details - FotaBlock

Short Name*: FotaBlock0_Fw0
Block Type*: FIRMWARE
Block Index*: 1
Block Start Address*: 0x10068000
Block End Address*: 0x100688FF

Block Processing 1 [0...1]

To Be Configured:

Verification Info [0...1]
Partition Flag Info [0...1]
Meta Data Info [0...1]

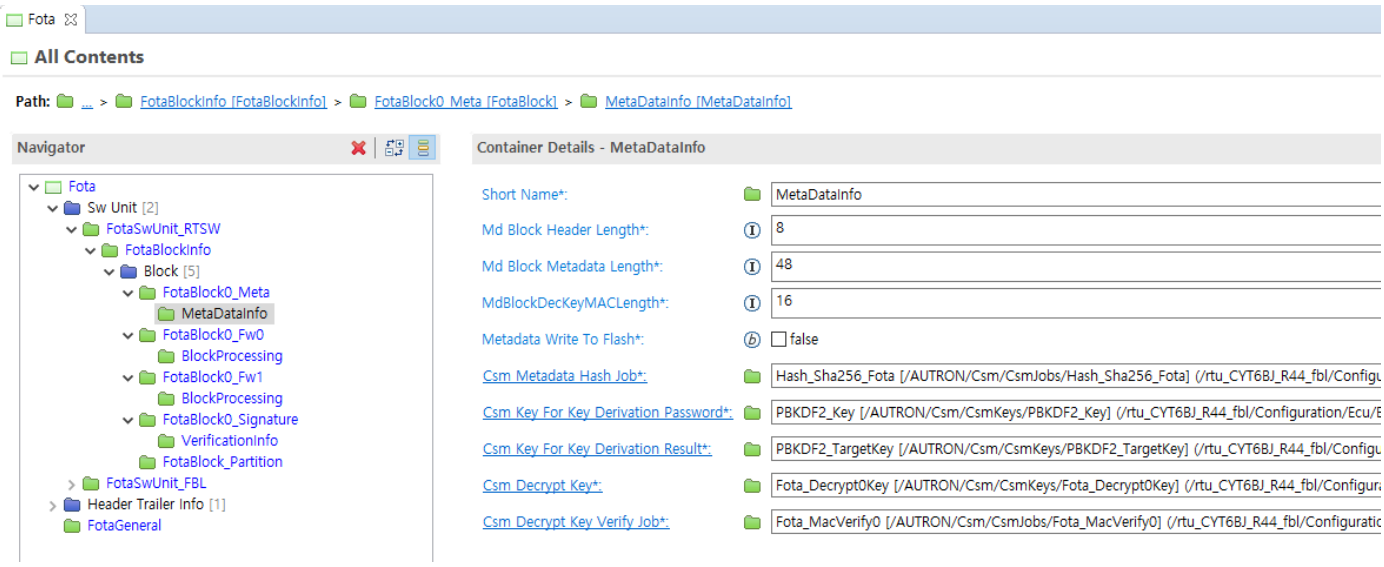
Figure 5 Sample FotaBlock configuration for Firmware block

Each firmware instance shall be constructed from firmware blocks. The handling rules and workflow of Fota module significantly altered (for example, handling of decryption and integrity verification) accordingly to the characteristics of each configured firmware blocks. In details, each Fota Block must contain its block info in one of 4 types: MetadataInfo, BlockProcessing, PartitionFlagInfo, and VerificationInfo.

5.3.3.1 MetaDataInfo

MetadataInfo represent the metadata sector in a firmware image which shall contain imp ortant information the decryption of firmware data. Therefore, the block is configured in case of an encrypted firmware is used for updating, and relevant references to configuration for Crypto stack CsmJobs and CsmKeys are needed.

Parameter Name	Value	Category
Md Block Header Length	Length of the header of metadata sector	C
Md Block Metadata Length	Length of the actual metadata in metadata sector	C
Md Block Dec Key MAC Length	Length of the MAC value of cipher key in metadata sector	C
Write Md To Flash	Specify whether the metadata block shall be written into Code Flash	C
Csm Metadata Process Job	Reference to CsmJob that shall be used to pre-process the met adata to prepare the cryptographic materials for sub-sequent steps	C
Csm Secret Key For Key Derive	Reference to CsmKey that shall be used for storing the secret password to derive the decryption key & MAC verification key	C
Csm Target Key For Key Derive	Reference to CsmKey that shall be used for storing the result of decryption key & MAC verification key derivation process.	C
Csm Decrypt Key	Reference to CsmKey that shall be used for storing the decryption key	C
Csm Decrypt Key Verify Job	Reference to CsmKey that shall be used for storing the MAC verification key for verify the integrity of decryption key	C



Path: ... > FotaBlockInfo [FotaBlockInfo] > FotaBlock0_Meta [FotaBlock] > MetaDataInfo [MetaDataInfo]

```
Fota
  Sw Unit [2]
    FotaSwUnit_RTSW
      FotaBlockInfo
        Block [5]
          FotaBlock0_Meta
            MetaDataInfo
          FotaBlock0_Fw0
            BlockProcessing
          FotaBlock0_Fw1
            BlockProcessing
          FotaBlock0_Signature
            VerificationInfo
          FotaBlock_Partition
    FotaSwUnit_FBL
  Header Trailer Info [1]
  FotaGeneral
```

Short Name*:	MetaDataInfo
Md Block Header Length*:	8
Md Block Metadata Length*:	48
MdBlockDecKeyMACLength*:	16
Metadata Write To Flash*:	□ false
Csm Metadata Hash Job*:	Hash_Sha256_Fota [/AUTRON/Csm/CsmJobs/Hash_Sha256_Fota] (/rtu_CYT6BJ_R44_fbl/Configu
Csm Key For Key Derivation Password*:	PBKDF2_Key [/AUTRON/Csm/CsmKeys/PBKDF2_Key] (/rtu_CYT6BJ_R44_fbl/Configuration/Ecu/E
Csm Key For Key Derivation Result*:	PBKDF2_TargetKey [/AUTRON/Csm/CsmKeys/PBKDF2_TargetKey] (/rtu_CYT6BJ_R44_fbl/Configu
Csm Decrypt Key*:	Fota_Decrypt0Key [/AUTRON/Csm/CsmKeys/Fota_Decrypt0Key] (/rtu_CYT6BJ_R44_fbl/Configura
Csm Decrypt Key Verify Job*:	Fota_MacVerify0 [/AUTRON/Csm/CsmJobs/Fota_MacVerify0] (/rtu_CYT6BJ_R44_fbl/Configuratio

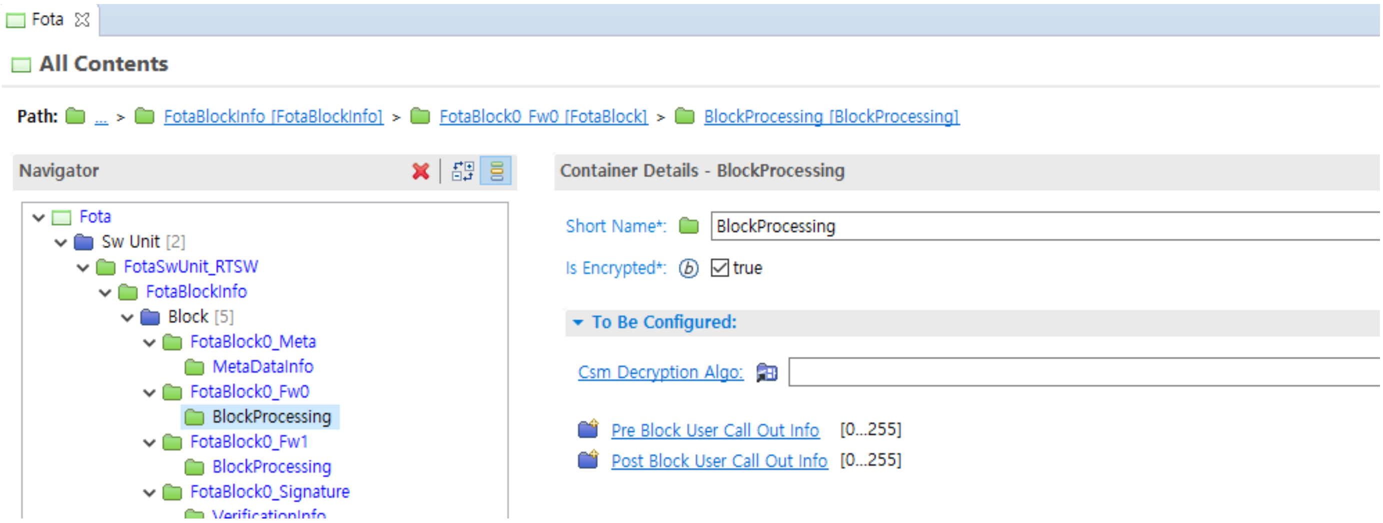
Figure 6 Sample MetadataInfo configuration

5.3.3.2 BlockProcessingInfo

BlockProcessingInfo represents physical firmware data sectors which shall be processed for possibly decryption/decompression before then be written into code flash. Therefore, the block is always configured and relevant references to configuration for Crypto stack CsmJobs and CsmKeys are needed.

Parameter Name	Description	Category
Is Encrypted	Specify whether the firmware block is encrypted	C
Csm Decryption Algo	Reference to CsmJob that shall be used to decrypt this firmware block	C

Container Name	Description	Category
Pre Block User Call Out Info	Container of User callouts to be used before processing this firmware block	C
Post Block User Call Out Info	Container of User callouts to be used after processing this firmware block	C



Short Name*: BlockProcessing

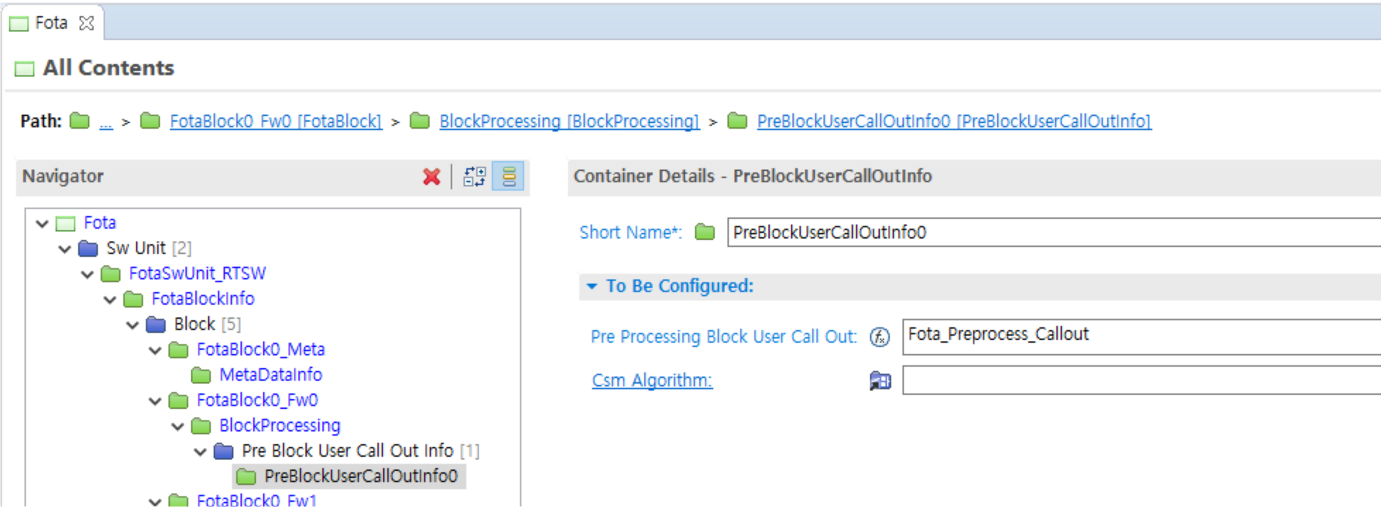
Is Encrypted*: ☑ true

Csm Decryption Algo:

Pre Block User Call Out Info [0...255]

Post Block User Call Out Info [0...255]

Figure 7 Sample BlockProessing configuration



Path: ... > FotaBlock0_Fw0 [FotaBlock] > BlockProcessing [BlockProcessing] > PreBlockUserCallOutInfo0 [PreBlockUserCallOutInfo]

Container Details - PreBlockUserCallOutInfo

Short Name*: PreBlockUserCallOutInfo0

To Be Configured:

Pre Processing Block User Call Out: Fota_Preprocess_Callout

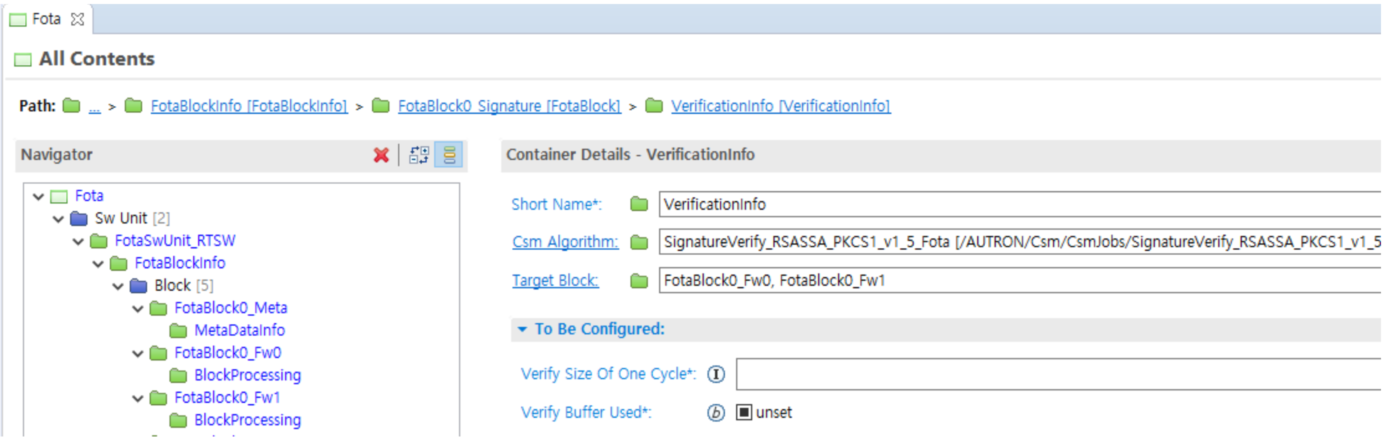
Csm Algorithm:

Figure 8 Sample PreBlockUserCallOutInfo configuratio

5.3.3.3 VerificationInfo

VerificationInfo represent the signature sectors of a firmware image which shall contain information for the integrity and authenticity verification. Therefore, the block needs to be configured in case signature verification method is used, and relevant references to configuration for Crypto stack CsmJobs are needed.

Parameter Name	Description	Category
Csm Algorithm	Reference to CsmJob that shall be used to verify firmware blocks	C
Target Block	Reference to the specific firmware blocks that shall be the target of this verification process	C
Verify Buffer Used	Enable buffer mode for verification data block	C
Verify Size Of One Cycle	Specify the size in bytes of data buffer when buffer mode for verification enabled	C



Path: ... > FotaBlockInfo [FotaBlockInfo] > FotaBlock0_Signature [FotaBlock] > VerificationInfo [VerificationInfo]

Container Details - VerificationInfo

Short Name*: VerificationInfo

Csm Algorithm: SignatureVerify_RSASSA_PKCS1_v1_5_Fota [/AUTRON/Csm/CsmJobs/SignatureVerify_RSASSA_PKCS1_v1_5

Target Block: FotaBlock0_Fw0, FotaBlock0_Fw1

To Be Configured:

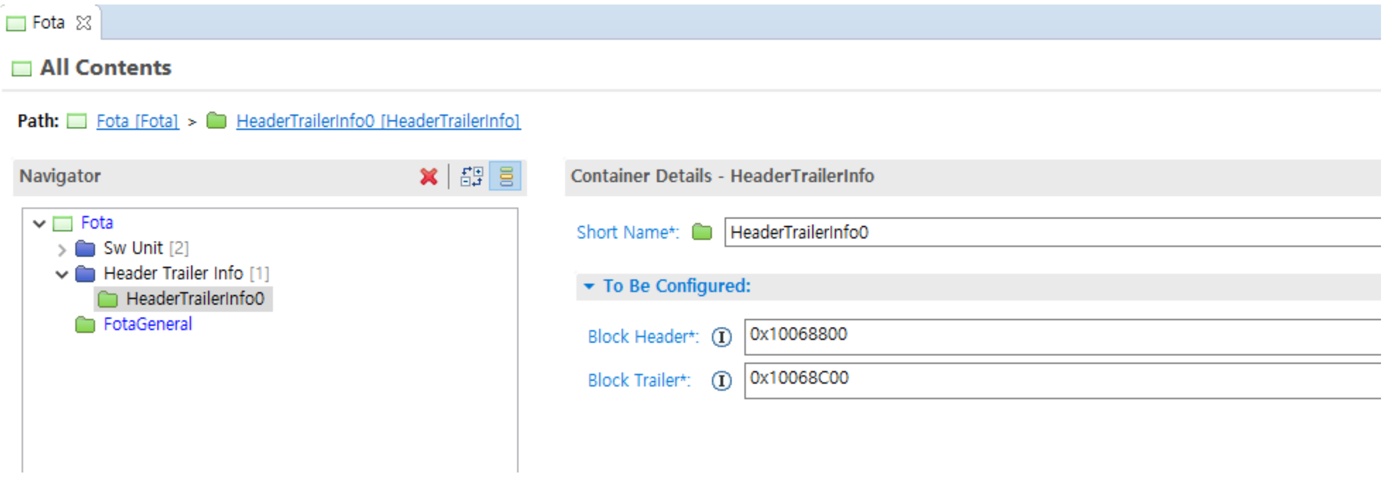
Verify Size Of One Cycle*:

Verify Buffer Used*: ■ unset

Figure 9 Sample VerificationInfo configuration

5.4 Fota Header Trailer Info

Parameters Name	Type	Category
Block Header	[Integer] Specify the address of header info for software version check feature	C
Block Trailer	[integer] Specify the address of trailer info for software version check feature	C



Short Name*: HeaderTrailerInfo0

Block Header*: 0x10068800

Block Trailer*: 0x10068C00

Figure 10 Sample HeaderTrailerInfo configuration

5.5 Prerequisites of Crypto stack configuration

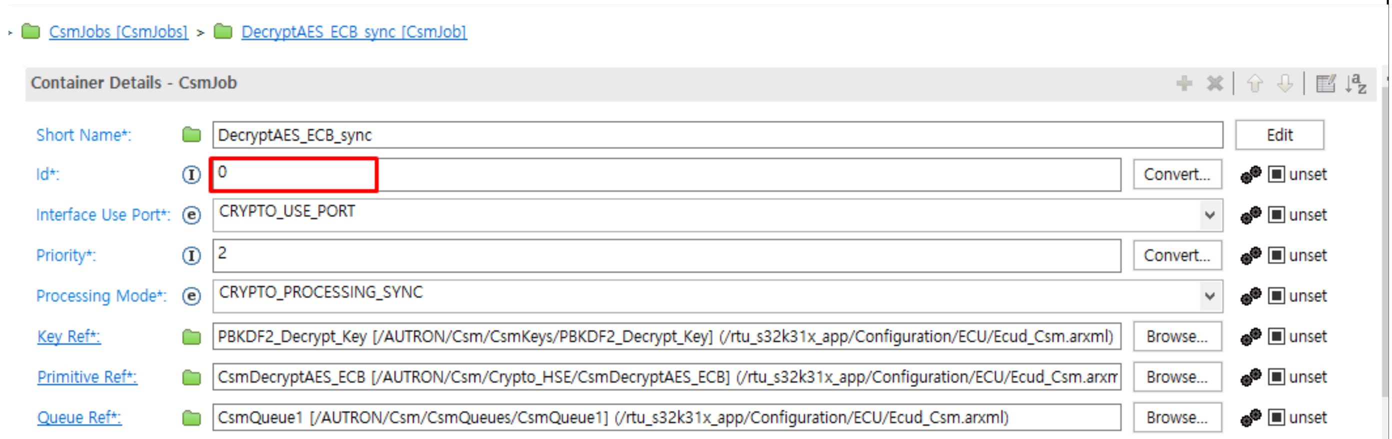
5.5.1 Metadata processing and firmware decryption:

A CsmJob with correct algorithm configured for processing the metadata sector.
A CsmKey to represent the secret key used for key derivation.
A CsmKey to represent the target key which stores the output of key derivation algorithm.
A CsmJob with correct algorithm configured for MAC verification of the decrypt key.
A CsmKey to represent the key for MAC verification job.
A CsmJob with correct algorithm configured for decrypting firmware sectors.
A CsmKey to represent the decrypt key used for decryption job.
Note: Multiple decryption flows are supported, though, users must ensure one separated pair of Csm Decryption Job and Csm Decryption Key configuration is used for each flow.

5.5.2 Signature verification:

A CsmJob with correct algorithm configured for verifying the signature.
A CsmKey to represent the key used for signature verification job.

5.5.3 Csm setting in S32K31X



CsmJobs [CsmJobs] > DecryptAES_ECB_sync [CsmJob]

Container Details - CsmJob

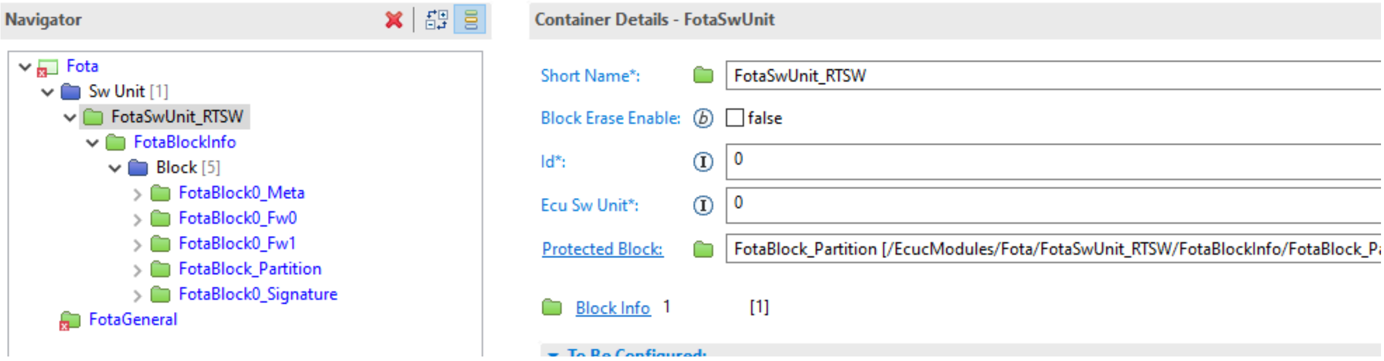
Short Name*:	DecryptAES_ECB_sync	Edit	
Id*:	0	Convert...	unset
Interface Use Port*:	CRYPTO_USE_PORT		unset
Priority*:	2	Convert...	unset
Processing Mode*:	CRYPTO_PROCESSING_SYNC		unset
Key Ref*:	PBKDF2_Decrypt_Key [/AUTRON/Csm/CsmKeys/PBKDF2_Decrypt_Key] (/rtu_s32k31x_app/Configuration/ECU/Ecud_Csm.arxml)	Browse...	unset
Primitive Ref*:	CsmDecryptAES_ECB [/AUTRON/Csm/Crypto_HSE/CsmDecryptAES_ECB] (/rtu_s32k31x_app/Configuration/ECU/Ecud_Csm.arxm	Browse...	unset
Queue Ref*:	CsmQueue1 [/AUTRON/Csm/CsmQueues/CsmQueue1] (/rtu_s32k31x_app/Configuration/ECU/Ecud_Csm.arxml)	Browse...	unset

Note: In case of S32K31x, using HSE and AES-ECB algotirhm for PBKDF2 decryption. Internally use fixed Csm Job ID value. User must set the decrypting Csm Job with ID 0.

5.6 Sample configurations

5.6.1 Application reprogramming with encrypted image and signature verification

- Firmware instance general configuration:



Container Details - FotaSwUnit

Short Name*: FotaSwUnit_RTSW

Block Erase Enable: □ false

Id*: 0

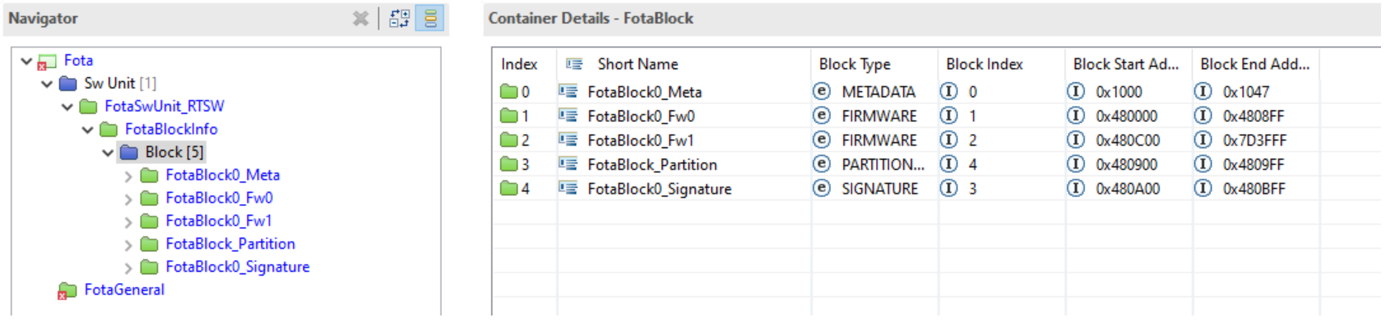
Ecu Sw Unit*: 0

Protected Block: FotaBlock_Partition [/EcucModules/Fota/FotaSwUnit_RTSW/FotaBlockInfo/FotaBlock_P

Block Info 1 [1]

To Be Configured:

- Firmware instance specific configuration:

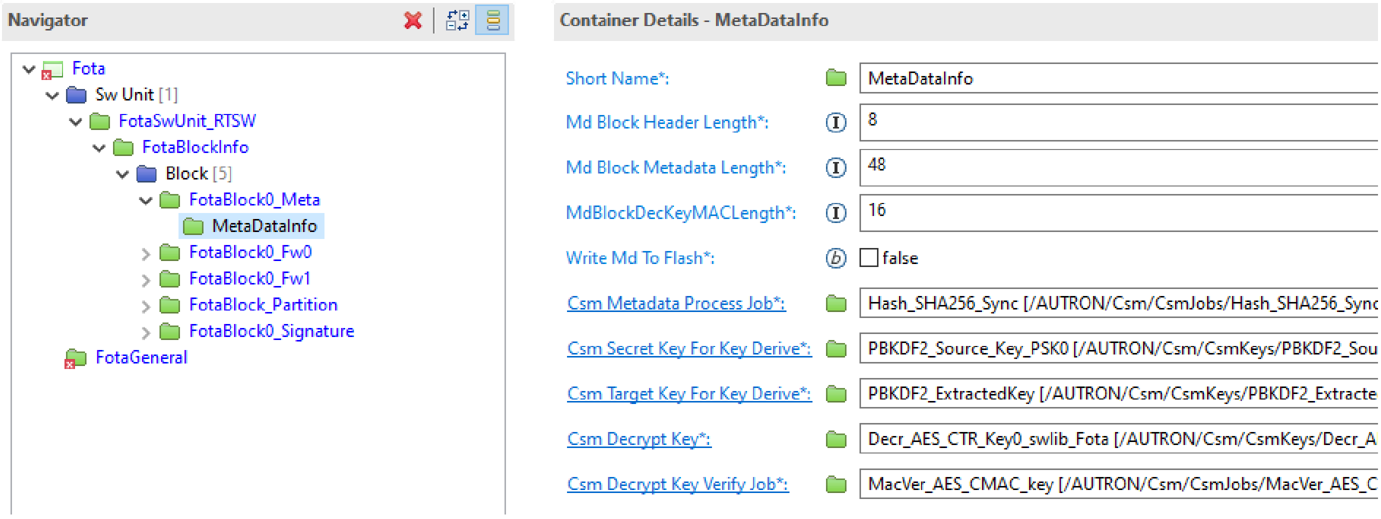


Container Details - FotaBlock

Index	Short Name	Block Type	Block Index	Block Start Ad...	Block End Add...
0	FotaBlock0_Meta	METADATA	0	0x1000	0x1047
1	FotaBlock0_Fw0	FIRMWARE	1	0x480000	0x4808FF
2	FotaBlock0_Fw1	FIRMWARE	2	0x480C00	0x7D3FFF
3	FotaBlock_Partition	PARTITION...	4	0x480900	0x4809FF
4	FotaBlock0_Signature	SIGNATURE	3	0x480A00	0x480BFF

- **Block Type:** all component sectors of this image like METADATA, FIRMWARE, SIGNATURE blocks present according to firmware instance structure. The PARTITION_FLAG address is also defined.
- **Block Start Address:** designated start address of sector in code flash.
- **Block End Address:** designated end address of sector in code flash.

- Metadata block configuration



Container Details - MetaDataInfo

Short Name*: MetaDataInfo

Md Block Header Length*: 8

Md Block Metadata Length*: 48

MdBlockDecKeyMACLength*: 16

Write Md To Flash*: □ false

Csm Metadata Process Job*: Hash_SHA256_Sync [/AUTRON/Csm/CsmJobs/Hash_SHA256_Sync

Csm Secret Key For Key Derive*: PBKDF2_Source_Key_PSK0 [/AUTRON/Csm/CsmKeys/PBKDF2_Sou

Csm Target Key For Key Derive*: PBKDF2_ExtractedKey [/AUTRON/Csm/CsmKeys/PBKDF2_Extracte

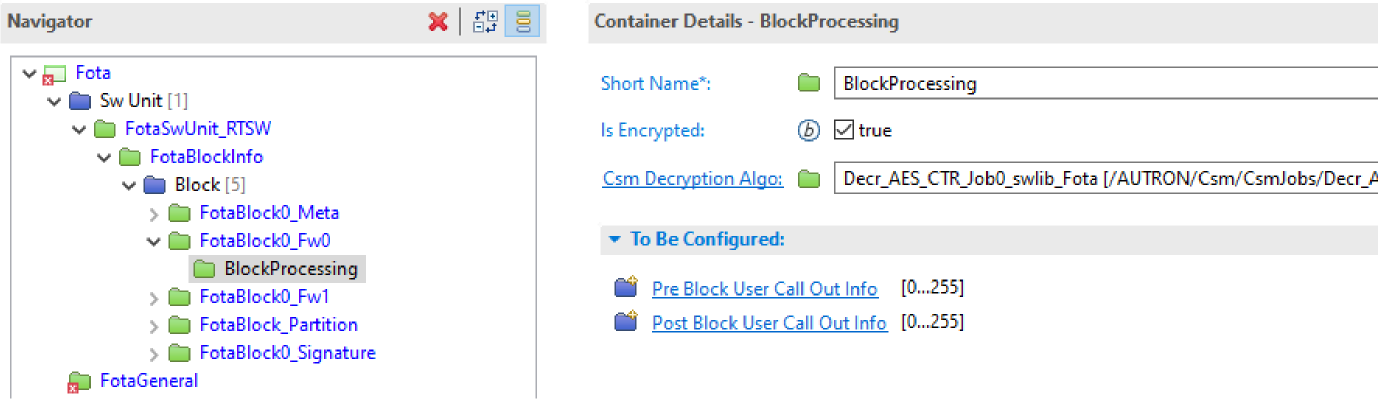
Csm Decrypt Key*: Decr_AES_CTR_Key0_swlib_Fota [/AUTRON/Csm/CsmKeys/Decr_A

Csm Decrypt Key Verify Job*: MacVer_AES_CMAC_key [/AUTRON/Csm/CsmJobs/MacVer_AES_C

- **Md Block Header/Metadata/DecKeyMAC Length:** configure based on the using structure of metadata sector.

- **Write Md To Flash:** is checked if users want to store Metadata in code flash and must have properly defined sector address for it.
- **CsmJobs and CsmKeys:** need to select the according Crypto stack Csm's jobs and keys for key derivation and decryption activities. **(Refer to 6.3.1)**

- Firmware data block configuration: the block processing rule is configured for each physical block. However, the rule shall be identical between physical blocks that belong to the same logical block.



Navigator

- Fota
  - Sw Unit [1]
    - FotaSwUnit_RTSW
      - FotaBlockInfo
        - Block [5]
          - FotaBlock0_Meta
          - FotaBlock0_Fw0
            - BlockProcessing
          - FotaBlock0_Fw1
          - FotaBlock_Partition
          - FotaBlock0_Signature
  - FotaGeneral

Container Details - BlockProcessing

Short Name*: BlockProcessing

Is Encrypted: ☑ true

Csm Decryption Algo: Decr_AES_CTR_Job0_swlib_Fota [/AUTRON/Csm/CsmJobs/Decr_A

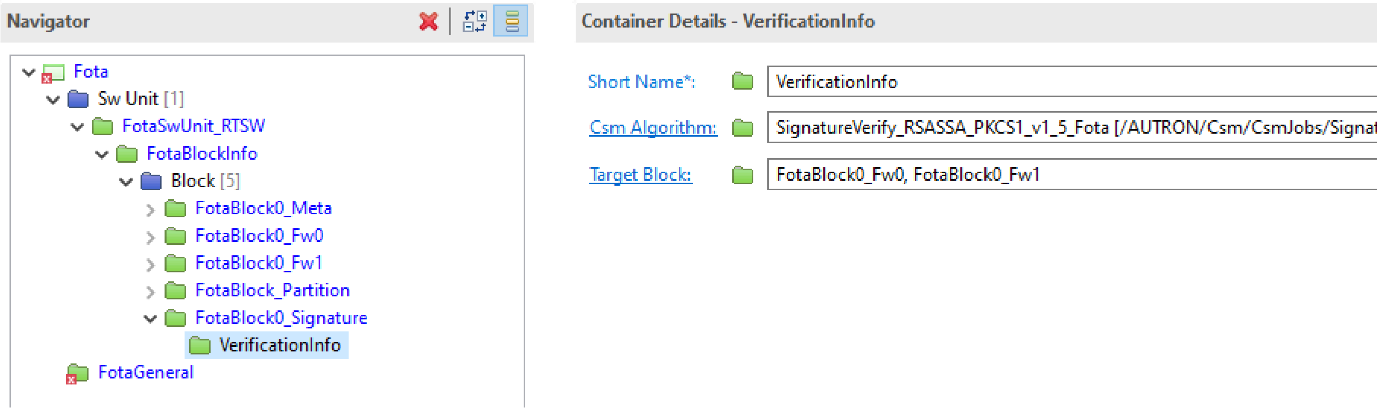
To Be Configured:

Pre Block User Call Out Info [0...255]

Post Block User Call Out Info [0...255]

- **Is Encrypted:** set to TRUE for enable encryption for the block.
- **Csm Decryption Algo:** leave blank in case user defined decryption library is used or map to the Csm decryption job planned for this block. **(Refer to 6.3.1)**

- Verification block configuration



Short Name*: VerificationInfo

Csm Algorithm: SignatureVerify_RSASSA_PKCS1_v1_5_Fota [/AUTRON/Csm/CsmJobs/Signat

Target Block: FotaBlock0_Fw0, FotaBlock0_Fw1

- **Csm Algorithm:** map to the Csm signature verification job shall be used verify this signature.
- **Target Block:** select the blocks that this signature was created on.

6. Application Programming Interface (API)

6.1 Imported Types

This section explains the Data types imported by the Fota Module and lists its dependency on other modules.

6.1.1 Standard Types

The following list shows all types of Std_Types.h that are used by the Fota Module

- Std_ReturnType
- Std_VersionInfoType

6.1.2 Rte_Type

The following list shows all types of Rte_Type.h that are used by the Fota Module

- Dcm_OpStatusType
- Dcm_NegativeResponseCodeType
- Dcm_ConfirmationStatusType
- Dcm_ProtocolType
- Crypto_OperationModeType
- Crypto_VerifyResultType

6.1.3 Dcm_Types

The following list shows all types of Dcm_Type.h that are used by the Fota Module

- Dcm_ReturnWriteMemoryType

6.1.4 Mem_76_Pfls

The following list shows all types of Mem_76_Pfls.h that are used by the Fota Module

- Mem_76_Pfls_JobResultType
- Mem_76_Pfls_FuncPrtTableType

6.2 Type definitions

None

6.3 Provided interfaces

6.3.1 General

Fota module takes the main responsibility for handling firmware updating workflow. In fact, firmware update master communicates with target ECU through diagnostics protocols which is handled by Dcm module. Fota provides the necessary interfaces for target ECU Diagnostics stack to invoke accordingly to their workflow while communicate with update master.

6.3.2 Data Types

N/A

6.3.3 Functions

This section describes the APIs that includes functionalities of Code Flash reprogramming in Fota module.

6.3.3.1 Fota_Init

Service name:	Fota_Init
Syntax:	FUNC(void, FOTA_CODE) Fota_Init(void)
Service ID[hex]:	0x01
Sync/Async:	Synchronous
Reentrancy:	Non reentrant
Parameters (in):	N/A
Parameters (inout):	N/A
Parameters (out):	N/A
Return Value	N/A
Description:	Initialization function - initializes all variables and sets the module state to initialized. This function is used by BSW.
Available via:	Fota.h

6.3.3.2 Fota_DeInit

Service name:	Fota_DeInit
Syntax:	FUNC(void, FOTA_CODE) Fota_DeInit(void)
Service ID[hex]:	0x0B
Sync/Async:	Synchronous
Reentrancy:	Non reentrant
Parameters (in):	N/A
Parameters (inout):	N/A
Parameters (out):	N/A
Return Value	N/A
Description:	De-initialize module. If there is still an access job pending, it is immediately terminated (using hardware cancel operation) and the Mem driver module state is set to unitialized. Therefore, Mem must be re-initialized before it will accept any new job requests after this service is processed. This function is used by BSW.
Available via:	Fota.h

6.3.3.3 Fota_Start_EraseMemory

Service name:	Fota_Start_EraseMemory	
Syntax:	FUNC (Std_ReturnType, DCM_CALLOUT_CODE) Fota_Start_EraseMemory (P2VAR(uint8, AUTOMATIC, FOTA_PRIVATE_DATA) pRoutineDataIn, VAR(uint8, AUTOMATIC) OpStatus, P2VAR(uint8, AUTOMATIC, DCM_APPL_DATA) pRoutineDataOut, P2VAR(uint16, AUTOMATIC, DCM_APPL_DATA) LpCur_DataLen, P2VAR(uint8, AUTOMATIC, DCM_APPL_DATA) LpErrorCode)	
Service ID[hex]:	N/A	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	pRoutineDataIn	Dcm input routine signal value
	OpStatus	Dcm operation status Value
Parameters (inout):	LpCur_DataLen	Dcm current routine data length
Parameters (out):	pRoutineDataOut	Dcm output routine data value
	LpErrorCode	Dcm Negative Response Code (Error Code)
Return Value	Std_ReturnType	Fota process internal functions return code
Description:	Server Function in the Client-Server Port Comm DCM Call this to erase memory (RID = FF00). This service indicates a request for Code Flash area erasing. This function is used by user. But it needs configuration. (It cannot be called directly by user).	
Available via:	Fota_Diag.h	

6.3.3.4 Fota_Start_EraseTargetArea

Service name:	Fota_Start_EraseTargetArea	
Syntax:	FUNC (Std_ReturnType, DCM_CALLOUT_CODE) Fota_Start_EraseTargetArea (P2VAR(uint8, AUTOMATIC, DCM_APPL_DATA) pRoutineDataIn, VAR(uint8, AUTOMATIC) OpStatus, P2VAR(uint8, AUTOMATIC, DCM_APPL_DATA) pRoutineDataOut, P2VAR(uint16, AUTOMATIC, DCM_APPL_DATA) LpCur_DataLen, P2VAR(uint8, AUTOMATIC, DCM_APPL_DATA) LpErrorCode)	
Service ID[hex]:	N/A	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	pRoutineDataIn	Dcm input routine signal value
	OpStatus	Dcm operation status Value

Parameters (inout):	LpCur_DataLen	Dcm current routine data length
Parameters (out):	pRoutineDataOut	Dcm output routine data value
	LpErrorCode	Dcm Negative Response Code (Error Code)
Return Value	Std_ReturnType	Fota process internal function's return code
Description:	Server Function in the DCM Call this to erase target area. This service indicate a request for Code Flash area erasing. This function is used by user. But it needs configuration. (It cannot be called directly by user).	
Available via:	Fota_Diag.h	

6.3.3.5 Fota_RequestDownload

Service name:	Fota_RequestDownload	
Syntax:	FUNC(Std_ReturnType, DCM_CALLOUT_CODE) Fota_RequestDownload (Dcm_OpStatusType OpStatus, uint8 DataFormatIdentifier, uint32 MemoryAddress, uint32 MemorySize, P2VAR(uint32, AUTOMATIC, DCM_APPL_DATA) LpBlockLength, P2VAR(uint8, AUTOMATIC, DCM_APPL_DATA) LpErrorCode)	
Service ID[hex]:	N/A	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	OpStatus	Dcm operation status Value
	MemoryAddress	Memory address value from Dcm memory service
	MemorySize	Memory Size value from Dcm memory service
	DataFormatIdentifier	Data format ID from Dcm memory service
Parameters (inout):	N/A	
Parameters (out):	LpBlockLength	Block length value from Dcm memory service
	LpErrorCode	Dcm Negative Response Code (Error Code)
Return Value	Std_ReturnType	Fota process internal function's return code
Description:	DCM CallOut function call this to request download. This service indicates a request for Code Flash area data write preparing. This function is used by user. But it needs configuration. (It cannot be called directly by user)	
Available via:	Fota_Diag.h	

6.3.3.6 Fota_DataTransfer

Service name:	Fota_DataTransfer
Syntax:	FUNC(Dcm_ReturnWriteMemoryType, DCM_CALLOUT_CODE) Fota_DataTransf

	er (Dcm_OpStatusType OpStatus, uint8 MemoryIdentifier, uint32 MemoryAddress, uint32 MemoryWriteLen, P2VAR(uint8, AUTOMATIC, DCM_APPL_DATA) pWriteData)	
Service ID[hex]:	N/A	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	OpStatus	Dcm operation status Value
	MemoryAddress	Memory address value from Dcm write memory service
	MemoryIdentifier	Memory identifier value from Dcm write memory service /* Not Supported Argument */
	MemoryWriteLen	Memory size value from Dcm write memory service
	pWriteData	Data from Dcm write memory service
Parameters (inout):	N/A	
Parameters (out):	N/A	
Return Value	Dcm_ReturnWriteMemoryType	Dcm return type for write operation.
Description:	DCM CallOut function call this to write memory. This service indicates a request for Code Flash area data write executing. This function is used by user. But it needs configuration. (It cannot be called directly by user).	
Available via:	Fota_Diag.h	

6.3.3.7 Fota_RequestTransferExit

Service name:	Fota_RequestTransferExit	
Syntax:	FUNC(Std_ReturnType, DCM_CALLOUT_CODE) Fota_RequestTransferExit (Dcm_OpStatusType OpStatus, P2VAR(uint8, AUTOMATIC, DCM_APPL_DATA) LpMemoryData, uint32* LulParameterRecordSize, P2VAR(Dcm_NegativeResponseCodeType, AUTOMATIC, DCM_APPL_DATA) LpErrorCode)	
Service ID[hex]:	N/A	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	OpStatus	Dcm operation status Value

Parameters (inout):	N/A	
Parameters (out):	LpErrorCode	Dcm Negative Response Code (Error Code)
Return Value	Std_ReturnType	Fota process internal function's return code
Description:	DCM CallOut function call this to exit Transfer. This service indicates a request for Code Flash area data write finishing. This function is used by user. But it needs configuration. (It cannot be called directly by user)	
Available via:	Fota_Diag.h	

6.3.3.8 Fota_Start_CheckMemory

Service name:	Fota_Start_CheckMemory	
Syntax:	FUNC(Std_ReturnType, DCM_CALLOUT_CODE) Fota_Start_CheckMemory (VAR(uint8, AUTOMATIC) OpStatus, P2VAR(uint8, AUTOMATIC, DCM_APPL_DATA) LpErrorCode)	
Service ID[hex]:	N/A	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	OpStatus	Dcm operation status Value
Parameters (inout):	N/A	
Parameters (out):	LpErrorCode	Dcm Negative Response Code (Error Code)
Return Value	Std_ReturnType	Fota process internal function's return code
Description:	Server Function in the DCM Call this at Check Memory (RID=0200). This service indicates a request for Flash image integrity verification in ES98765-02 support scheme. This function is used by user. But it needs configuration. (It cannot be called directly by user)	
Available via:	Fota_Diag.h	

6.3.3.9 Fota_Start_CheckProgrammingDependency

Service name:	Fota_Start_CheckProgrammingDependency
Syntax:	FUNC(Std_ReturnType, DCM_CALLOUT_CODE) Fota_Start_CheckProgrammingDependency (P2VAR(uint8, AUTOMATIC, FOTA_PRIVATE_DATA) pRoutineDataIn, VAR(uint8, AUTOMATIC) OpStatus, P2VAR(uint8, AUTOMATIC, DCM_APPL_DATA) pRoutineDataOut, P2VAR(uint16, AUTOMATIC, DCM_APPL_DATA) LpCur_DataLen, P2VAR(uint8, AUTOMATIC, DCM_APPL_DATA) LpErrorCode)
Service ID[hex]:	N/A

Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	pRoutineDataIn	Dcm input routine data value
	OpStatus	Dcm operation status Value
Parameters (inout):	LpCur_DataLen	Dcm current routine data length
Parameters (out):	pRoutineDataOut	Dcm output routine data value
	LpErrorCode	Dcm Negative Response Code (Error Code)
Return Value	Std_ReturnType	Fota process internal function's return code
Description:	Server Function in the DCM Call this at check programming dependency (RID=FF01). This service indicates a request for Flash image integrity verification in ES98765-01 support scheme. This function is used by user. But it needs configuration. (It cannot be called directly by user).	
Available via:	Fota_Diag.h	

6.3.3.10 Fota_Start_ReadActiveArea

Service name:	Fota_Start_ReadActiveArea	
Syntax:	FUNC (Std_ReturnType, DCM_CALLOUT_CODE) Fota_Start_ReadActiveArea (P2VAR(uint8, AUTOMATIC, DCM_APPL_DATA) pRoutineDataIn, VAR(uint8, AUTOMATIC) OpStatus, P2VAR(uint8, AUTOMATIC, DCM_APPL_DATA) pRoutineDataOut, P2VAR(uint16, AUTOMATIC, DCM_APPL_DATA) LpCur_DataLen, P2VAR(uint8, AUTOMATIC, DCM_APPL_DATA) LpErrorCode)	
Service ID[hex]:	N/A	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	pRoutineDataIn	Dcm input routine data value
	OpStatus	Dcm operation status Value
Parameters (inout):	LpCur_DataLen	Dcm current routine data length
Parameters (out):	pRoutineDataOut	Dcm output routine data value
	LpErrorCode	Dcm Negative Response Code (Error Code)
Return Value	Std_ReturnType	Fota process internal function's return code
Description:	Server Function in the DCM Call this at read active area (RID=0210). This service provides information about the current active memory bank to requester in OTA dual memory programming scheme. This function is used by user. But it needs configuration. (It cannot be called directly by user).	
Available via:	Fota_Diag.h	

6.3.3.11 Fota_Start_SwapActiveArea

Service name:	Fota_Start_SwapActiveArea	
Syntax:	FUNC (Std_ReturnType, DCM_CALLOUT_CODE) Fota_Start_SwapActiveArea (P2VAR(uint8, AUTOMATIC, DCM_APPL_DATA) pRoutineDataIn, VAR(uint8, AUTOMATIC) OpStatus, P2VAR(uint8, AUTOMATIC, DCM_APPL_DATA) pRoutineDataOut, P2VAR(uint16, AUTOMATIC, DCM_APPL_DATA) LpCur_DataLen, P2VAR(uint8, AUTOMATIC, DCM_APPL_DATA) LpErrorCode)	
Service ID[hex]:	N/A	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	pRoutineDataIn	Dcm input routine data value
	OpStatus	Dcm operation status Value
Parameters (inout):	LpCur_DataLen	Dcm current routine data length
Parameters (out):	pRoutineDataOut	Dcm output routine data value
	LpErrorCode	Dcm Negative Response Code (Error Code)
Return Value	Std_ReturnType	Fota process internal function's return code
Description:	Server Function in the DCM Call this at swap active area (RID=0213). This service is used to indicate the request to swap the running address of application to the other (inactive) bank between two presenting memory bank in OTA dual memory scheme. This function is used by user. But it needs configuration. (It cannot be called directly by user).	
Available via:	Fota_Diag.h	

6.3.3.12 Fota_Start_FinishUpdate

Service name:	Fota_Start_FinishUpdate	
Syntax:	FUNC (Std_ReturnType, DCM_CALLOUT_CODE) Fota_Start_FinishUpdate (VAR(uint8, AUTOMATIC) OpStatus, P2VAR(uint8, AUTOMATIC, DCM_APPL_DATA) LpErrorCode)	
Service ID[hex]:	N/A	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	OpStatus	Dcm operation status Value
Parameters (inout):	N/A	
Parameters (out):	LpErrorCode	Dcm Negative Response Code (Error Code)
Return Value	Std_ReturnType	Fota process internal function's return code

Description:	Server Function in the DCM Call this at Finish Update (RID=0211). This service is used to indicate the request to process area data synchronization. This function is used by user. But it needs configuration. (It cannot be called directly by user).
Available via:	Fota_Diag.h

6.4 Scheduled functions

Service name:	Fota_MainFunction
Syntax:	FUNC(void, FOTA_CODE) Fota_MainFunction(void)
Service ID[hex]:	0x03
Description:	Service for performing the processing of the Fota functionalities. This function is used by BSW.
Available via:	Fota.h

6.5 Expected interfaces

6.5.1 Mem_76_Pfls

Fota module collaborates closely with Mem Driver to perform firmware data writes to code flash after having processed the data blocks transferred from firmware update master.

Interface	Functionality
Fota_PflsInit	This service is used to indicate a request for Code flash driver initialization.
Fota_PflsDeinit	This service is used to indicate a request for Code flash driver de-initialization.
Fota_PflsCancelReq	This service is used to indicate a request for Code flash driver job cancelation.
Fota_PflsEraseRequest	This service is used to indicate a request for Code flash data erasing.
Fota_PflsWriteRequest	This service is used to indicate a request for Code flash data writing.
Fota_PflsGetJobResult	This service is used to indicate a request for Code flash executing job result.
Mem_76_Pfls_HwSpecificService	This interface is used to provoke hardware-specific services that are handled by Pflash to support specific operations such as: • Fota_PflsSwapBankRequest • Fota_PflsGetActiveBank • Fota_PflsGetCovAddr • Fota_PflsTgtAreaSet • Fota_PflsGetFlashAlignment • Fota_PflsGetSectorSize

6.5.2 Csm

Fota module collaborates closely with Csm module to perform cryptographic activities involving encrypted data blocks such as processing of crypto materials, derivation of secret keys, and decryption of firmware data. The verification of new firmware integrity and authenticity also required capabilities of Crypto stack.

Interface	Functionality
Csm_KeyElementSet	This service is used to define cryptography materials in the form of AUTOSAR Crypto Key Elements.
Csm_KeyElementGet	This service is used to derive cryptography materials from AUTOSAR Crypto definition.
Csm_KeySetValid	This service is used to define a combination of cryptography materials in form of Csm Key.
Csm_Hash	This service is used to request to perform a hash calculation on firmware image's metadata as input materials for further cryptographic execution.
Csm_MacVerify	This service is used to request to perform a MAC verification of cipher key.
Csm_Decrypt	This service is used to request to perform a decryption on image data chunk received.
Csm_KeyDerive	This service is used to request to perform a key derivation.
Csm_SignatureVerify	This service is used to request to perform a signature verification in Programming dependencies check.

6.5.3 Optional Interfaces

Some optional Fota user callouts are provided in integration_Fota_F for usage of FBL and integration_Fota for usage of Application software.

Interface	Functionality
FUNC(Fota_SF_ReturnType, FOTA_CODE) Fota_DecryptStart_Callout	These callouts are used to allow users to specify their own implementation of firmware data decryption process.
FUNC(Fota_SF_ReturnType, FOTA_CODE) Fota_DecryptUpdate_Callout	
FUNC(Fota_SF_ReturnType, FOTA_CODE) Fota_DecryptFinish_Callout	
FUNC(Std_ReturnType, FOTA_CODE) Fota_DeriveKeyRequest_Callout	These callouts are used to allow users to specify their own implementation of secret key derivation process.
FUNC(Std_ReturnType, FOTA_CODE) Fota_IsWarmReset	This callout is used to check the lastest reset reason of MCUs for further specific process at start-up phase.
FUNC(void, ECUM_CALLOUT_CODE) Fota_RequestReset	This callout is used to issue an MCU reset request for FBL specific operations.

6.6 Service Interfaces

6.6.1 Client-Server-Interfaces

6.6.1.1 ServiceRequestNotification

Name:	ServiceRequestNotification		
Comment:	-		
IsService	true		
Variation:	Service for performing the processing of the Fota functionalities		
Possible Errors:	0	E_OK	
	1	E_NOT_OK	

Operation:	Confirmation	
Comment:	Server Function in the Client-Server Port Comm DCM Call this to Service Request confirmation	
Mapped to API	Fota_SupplierNotification_ServiceRequest_Confirmation	
Variation:	-	
Parameters	ConfirmationStatus	
	Type	Dcm_ConfirmationStatusType
	Direction	IN
	Comment	- DCM_RES_POS_OK: Transmission of positive response was successful - DCM_RES_POS_NOT_OK: Transmission of positive response failled - DCM_RES_NEG_OK: Transmission of negative response was successful - DCM_RES_NEG_NOT_OK: Transmission of negative response failled
	SID	
	Type	uint8
	Direction	IN
	Comment	Service ID
	ReqType	
	Type	uint8
	Direction	IN
	Comment	Rx message address type
	ConnectionId	
	Type	uint16
	Direction	IN
	Comment	-

	ProtocolType	
	Type	Dcm_ProtocolType
	Direction	IN
	Comment	-
	TesterSourceAddress	
	Type	uint16
	Direction	IN
	Comment	Source address
	LddRetVal	
	Type	Std_ReturnType
	Direction	RETURN
	Comment	- RTE_E_OK : Request was successful - RTE_E_ServiceRequestNotification_E_NOT_OK : Request was not successful
Possible Errors:	E_OK	
	E_NOT_OK	

Operation:	Indication	
Comment:	Server Function in the Client-Server Port Comm DCM Call this to Service Request indication	
Mapped to API	Fota_SupplierNotification_ServiceRequest_Indication	
Variation:	-	
Parameters	ErrorCode	
	Type	Dcm_NegativeResponseCodeType
	Direction	OUT
	Comment	If this operation returns value E_NOT_OK, the Dcm module shall send a negative response with NRC code equal to the parameter ErrorCode parameter value. (Refer to the Rte_Dcm_Type.h)
	SID	
	Type	uint8
	Direction	IN
	Comment	Service ID
	ReqType	
	Type	uint8
	Direction	IN
	Comment	Rx message address type 1: Functional Address 0: Physical Address
	ConnectionId	

	Type	uint16
	Direction	IN
	Comment	-
	RequestData	
	Type	P2CONST(uint8, AUTOMATIC, RTE_APPL_CONST)
	Direction	IN
	Comment	Pointer to received data
	TesterSourceAddress	
	Type	uint16
	Direction	IN
	Comment	Source address (Refer to configureation DcmDslProtocolRx TesterSourceAddr)
	ProtocolType	
	Type	Dcm_ProtocolType
	Direction	IN
	Comment	-
	LddRetVal	
	Type	Std_ReturnType
	Direction	RETURN
	Comment	- RTE_E_OK : Request was successful - RTE_E_ServiceRequestNotification_E_NOT_OK : Request was not successful
Possible Errors:	E_OK	
	E_NOT_OK	

6.6.2 Implementation Data Types

None

6.6.3 Ports

6.6.3.1 Fota_StateRequest

Name	Fota_StateRequest		
Kind	RequiredPort	**Interface**	EcuM_StateRequest (Client Com specs)
Description	-		
Port Defined Argument Value(s)	**Type**	EcuM_UserType	
	Value	RequestReset	
Variation	-		

6.6.3.2 Swap_ServiceRequestNotification_{Operation}

Name	Swap_ServiceRequestNotification		
Kind	ProvidedPort	**Interface**	ServiceRequestNotification (Server Com spec)
Description	-		
Port Defined Argument Value(s)	**Type**	Confirmation, Indication	
	Value	-	
Variation	-		

7. Bswmd

7.1 BSW MDT PARAMETER CONFIGURATION

This section explains about the elements and valid values

Element Name	BSW-IMPLEMENTATION
SW-VERSION	Software version of this implementation. The numbering contains three levels (like major, minor, patch), its values are vendor specific. Example: 1.0.0
VENDOR-ID	This parameter specifies vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list. Example: 76
AR-RELEASE-VERSION	Version of the AUTOSAR Release on which this implementation is based. The numbering contains three levels (major, minor, revision) which are defined by AUTOSAR. **Example: 4.4.0**
BEHAVIOR-REF	This parameter contains reference to a corresponding BSW-INTERNAL-BEHAVIOR. Example: /Bsw_Os/Os/BswInternalBehavior_Os
VENDOR-API-INFIX	In driver modules which can be instantiated several times on a single ECU, SRS_BSW_00413 requires that the names of files, APIs, published parameters and memory allocation keywords are extended by the vendorId and a vendor specific name. This parameter is used to specify the vendor specific name. In total, the implementation specific API name is generated as follows: <ModuleName>_<vendorId>_<vendorApiInfix>_<API name from SWS>. E.g. assuming that the vendorId of the implementer is 123 and the implementer chose a vendorApiInfix of "v11r456" an API name Can_Write defined in the SWS will translate to Can_123_v11r456_Write. This attribute is mandatory for all modules with upper multiplicity > 1. It shall not be used for modules with upper multiplicity =1.

Element Name	BSW-MODULE-DESCRIPTION
MODULE-ID	This parameter specifies Module ID of this Module from AUTOSAR Module List. Example: “1” for Os

7.2 Exclusive Areas

N/A

8. APPENDIX

N/A