SCOPE OF APPLICATION All Project/Engineering	HYUNDAI AutoEver	SHT/SHTS 1 / 52
Responsibility: Classic AUTOSAR Team	AUTOSAR NvM User Manual	DOC. NO

AUTOSAR NvM User Manual

Document Change History			
Date (YYYY-MM-DD)	Ver.	Editor	Content (before revision-> after revision)
2021-03-10	1.0.0.0	YJ.Yun	Initial version
2021-07-29	1.0.1.0	JH.Lim	Update module release note
2021-10-29	1.0.2.0	JH.Lim	Update module release note Update configuration guide for NvMNvramDeviceId
2022-03-01	1.1.0.0	JH.Lim	Update module release note Update Limitations Add error message (ERR020052)
2022-05-26	1.1.1.0	SH.Park	Update module release note Update configuration guide
2022-07-06	1.1.1.1	ThuanLD5	Update module release note
2022-08-12	1.1.2.0	ThuanLD5	Update module release note
2022-11-03	1.2.0.0	ThuanLD5	Update module release note Update container Update configuration Update error messages Update function setting
2022-12-21	1.3.0.0	ThuanLD5	Update module release note Update container Add User Defined Functions
2023-04-28	1.3.1.0	YJ.Yun	Update module release note
2023-05-13	1.3.2.0	MK.Choi	Support for U2A

Edition Date: 2022-04-28	File Name NvM_UM.pdf	Creation 2023-04-28 YoungJin Yun	Check 2023-04-28 InWon Kang	Approval 2023-04-28 JongHyun Baek
Document Management System				

Table of Contents

1. Overview

This document is based on the Autosar standard SRS/SWS. For more detailed functional description when using the module, refer to the reference document below. Also, refer to UserManual and Integration Manual supported by each MCU for detailed settings of Fee and Fls.

The interpretation of the category related to setting is as follows

- Changeable (C) : Items that can be set by the user
- Fixed (F) : Items that cannot be changed by user
- NotSupported (N) : Items that not supported

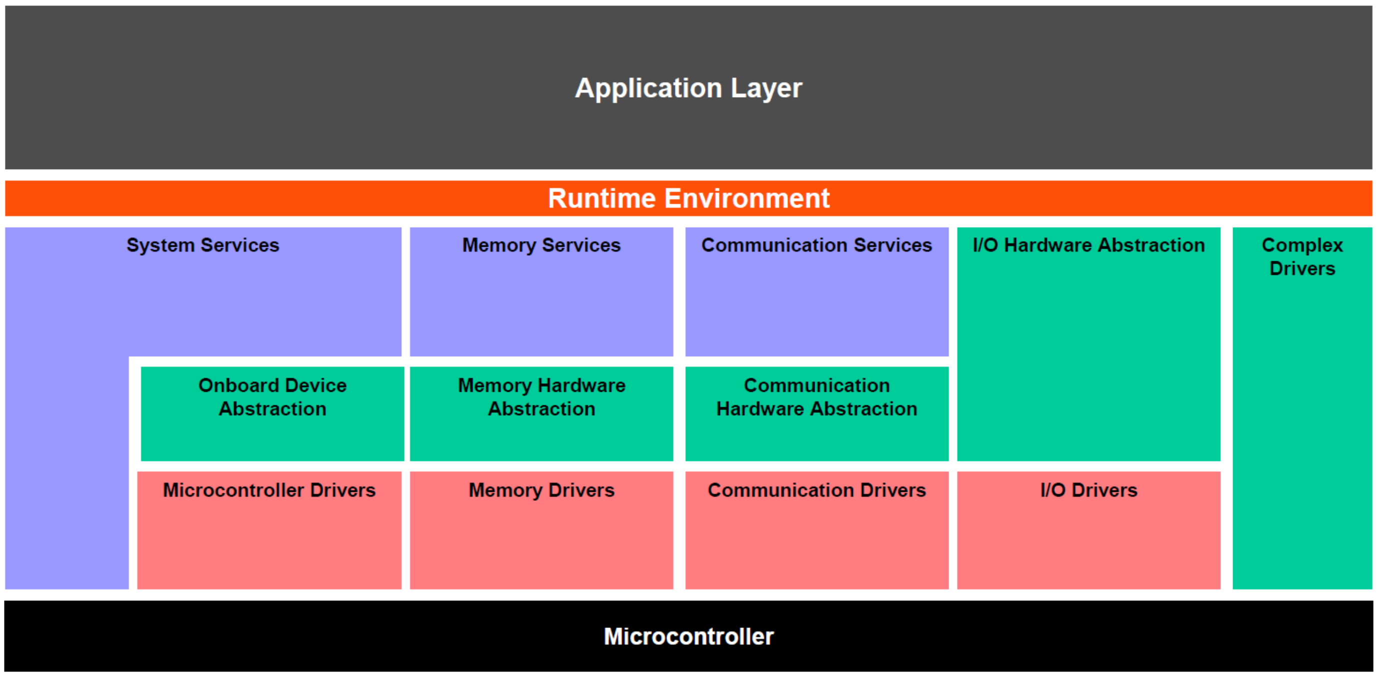
2. Reference

Sl. No.	Title	Version
1.	AUTOSAR_SWS_BSWGeneral.doc	4.4.0
2.	AUTOSAR_SWS_NVRAMManager.pdf	4.4.0
3.	AUTOSAR_SWS_EEPROMAbstraction.pdf	4.4.0
4.	AUTOSAR_SWS_EEPROMDriver.pdf	4.4.0
5.	AUTOSAR_SWS_FlashDriver.pdf	4.4.0
6.	AUTOSAR_SWS_FlashEEPROMEmulation.pdf	4.4.0
7.	AUTOSAR_SWS_CRCLibrary.pdf	4.4.0
8.	AUTOSAR_SWS_MemoryAbstractionInterface.pdf	4.4.0

3. AUTOSAR System

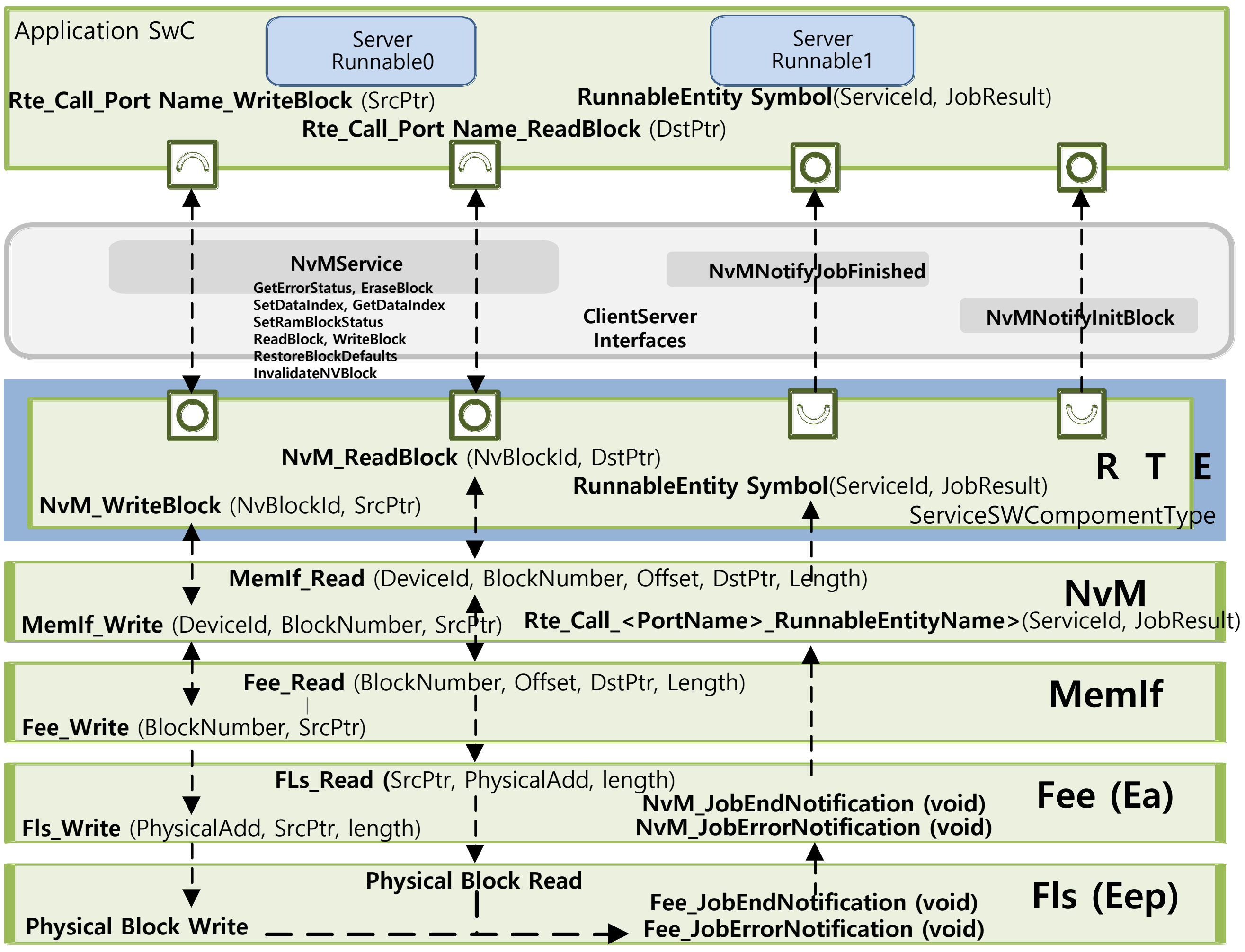
3.1 Overview of Software Layers

The Layered Architecture of the AUTOSAR platform is as follows. The AUTOSAR platform can be divided into Service Layer, ECU Abstraction Layer, Complex Device Drivers and Microcontroller Abstraction Layer.



3.2 AUTOSAR Memory Stack

Memory Stack module means NvM / MemIf / Ea / Eep / Fee / Fls module. Here, Fee and Fls modules are modules belonging to Mcal. For basic information, refer to the User Manual and Integration Manual of each MCU manufacturer. In order to use EEPROM, NvM and MemIf modules are basically required, and Fee / Fls module is used to use internal EEPROM, and Ea / Eep modules are additionally needed to use external EEPROM. The interface between Autosar layer and each module to use EEPROM is as follows.


Application SwC
Server Runnable0
Server Runnable1
Rte_Call_Port Name_WriteBlock (SrcPtr)
RunnableEntity Symbol(ServiceId, JobResult)
Rte_Call_Port Name_ReadBlock (DstPtr)
NvMService
GetErrorStatus, EraseBlock
SetDataIndex, GetDataIndex
SetRamBlockStatus
ReadBlock, WriteBlock
RestoreBlockDefaults
InvalidateNVBlock
NvMNotifyJobFinished
ClientServer Interfaces
NvMNotifyInitBlock
NvM_ReadBlock (NvBlockId, DstPtr)
RunnableEntity Symbol(ServiceId, JobResult)
R T E
NvM_WriteBlock (NvBlockId, SrcPtr)
ServiceSWCompomentType
MemIf_Read (DeviceId, BlockNumber, Offset, DstPtr, Length)
NvM
MemIf_Write (DeviceId, BlockNumber, SrcPtr)
Rte_Call_<PortName>_RunnableEntityName>(ServiceId, JobResult)
Fee_Read (BlockNumber, Offset, DstPtr, Length)
MemIf
Fee_Write (BlockNumber, SrcPtr)
FLs_Read (SrcPtr, PhysicalAdd, length)
NvM_JobEndNotification (void)
NvM_JobErrorNotification (void)
Fee (Ea)
Fls_Write (PhysicalAdd, SrcPtr, length)
Physical Block Read
Fee_JobEndNotification (void)
Fls (Eep)
Physical Block Write
Fee_JobErrorNotification (void)

Write requests (From SWS_NvM_00698)
Applications have to adhere to the following rules during write request for implicit synchronization between application and NVRAM manager:
1) The application fills a RAM block with the data that has to be written by the NvM module.
2) The application issues the NvM_WriteBlock or NvM_WritePRAMBlock request which transfers control to the NvM module.
3) From now on the application must not modify the RAM block until success or failure of the request is signaled or derived via polling. In the meantime the contents of the RAM block may be read.
4) An application can use polling to get the status of the request or can be informed via a callback function asynchronously.
5) After completion of the NvM module operation, the RAM block is reusable for modifications.

Read requests (From SWS_NvM_00699)
Applications have to adhere to the following rules during read request for implicit synchronization between application and NVRAM manager:
1) The application provides a RAM block that has to be filled with NVRAM data from the NvM module's side.
2) The application issues the NvM_ReadBlock request which transfers control to the NvM module.
3) From now on the application must not read or write to the RAM block until success or failure of the request is signaled or derived via polling.
4) An application can use polling to get the status of the request or can be informed via a callback function.
5) After completion of the NvM module operation, the RAM block is available with new data for use by the application.

4. Limitations and Deviations

4.1 Limitations

User do not take care about Bswmd_NvM.arxml.
Limitations are given mainly by the finite number of "Block Management Types" and their individual treatment of NV data. These limits can be reduced by an enhanced user defined management information, which can be stored as a structured part of the real NV data. In this case the user defined management information has to be interpreted and handled by the application at least.

4.2 Deviations

When creating Swcd, PortName is created according to 8.2.3 chapter in AUTOSAR_SWS_NVRAMManager.pdf.

5. Configuration Guide

5.1 NvM Common Container

Refer to the following settings.

Parameter Name	Value	Category
NvMApiConfigClass	NVM_API_CONFIG_CLASS_3	F
NvMBswMMultiBlockJobStatus Information	True	F
NvMCompiledConfigId[5)]	51	C
NvMCrcNumOfBytes	4	F
NvMDatasetSelectionBits[1)]	User Defined (From SRS)	C
NvMDevErrorDetect	True	F
NvMDrvModeSwitch	true	C
NvMDynamicConfiguration[5)]	User Defined	C
NvMJobPrioritization[2)]	User Defined (From SRS)	F
NvMMultiBlockCallback[6)]	User Defined	C
NvMPollingMode	False	C
NvMRepeatMirrorOperations	0	F
NvMSetRamBlockStatusApi	False	F
NvMSizeImmediateJobQueue[3)]	User Defined	C
NvMSizeStandardJobQueue[3)]	User Defined	C
NvMVersionInfoApi	User Defined	C
NvMMainfunctionTriggerRef[8)]	Integrator Defined	F
NvMUserJobFunction[7)]	Integrator Defined	F
NvMMainFunctionPeriod	Integrator Defined	F
NvMUserIncludeFiles[4)] (AUTOEVER specific)	User Defined	C
NvMReadAllOrderSupport[9)]	User Defined	C
NvMWriteAllOrderSupport[9)]	User Defined	C
NvMUserWdgToggleFunction[10)]	User Defined	C

1) When DataSet Type Block Usage in SRS is No: 1
When DataSet Type Block Usage in SRS is Yes: It depends on the maximum value among NvBlockNum values of DataSet Type Blocks, and the number of NvBlockNums must be designed in the application.
Depending on the value of NvMDatasetSelectionBits, the maximum number of NvBlock(Fee/Ea Block) that can be set in the block set as DataSet Type is different. (All DataSet Type Block can set up to 2n NvBlocks. n: NvMDatasetSelectionBits)
- Ex) If this value is 2, the maximum number of NvBlocks that all DataSet Blocks can have is 4
- Ex) If this value is 3, the maximum number of NvBlocks a DataSet Block can have is 8

BlockNumber value of all Fee/Ea Block is changed according to this setting value. (Chap 5.2.2)

Basic formula : 2^n * NvMNvBlockBaseNumber + Index
(n : NvMDatasetSelectionBits, Index : sequential from 0, as many as NvBlock(Fee/Ea Block))

Ex) The configuration parameter NvMDatasetSelectionBits is configured to be 2.

- For a native NVRAM block with NvMNvBlockBaseNumber = 2:
  - NV block is accessed with FEE/EA_BLOCK_NUMBER = 8
- For a redundant NVRAM block with NvMNvBlockBaseNumber = 3:
  - 1st NV block with data index 0 is accessed with FEE/EA_BLOCK_NUMBER = 12
  - 2nd NV block with data index 1 is accessed with FEE/EA_BLOCK_NUMBER = 13
- For a dataset NVRAM block with NvMNvBlockBaseNumber = 4, NvMNvBlockNum = 3:
  - NV block #0 with data index 0 is accessed with FEE/EA_BLOCK_NUMBER = 16
  - NV block #1 with data index 1 is accessed with FEE/EA_BLOCK_NUMBER = 17
  - NV block #2 with data index 2 is accessed with FEE/EA_BLOCK_NUMBER = 18

2) If Immediate Block Usage of SRS is Yes (If Immediate Block is used in Application), set NvMJobPrioritization item to True.
If Immediate Block is used, the NvM module shall use two queues, one for immediate write jobs (crash data) another for all other jobs (including immediate read/erase jobs, standar d jobs). Otherwise the NvM Module shall use one queue and processes all jobs in FCFS ord er.

3) NvMSizeImmediateJobQueue / NvMSizeStandardJobQueue
- The number of requests (Read/Write job, etc.) requested by NvM Manger.
- QueueOverFlow Dem Error occurs when Request(Read/Write) is accumulated in the queue based on the set value. When you need to increase the value.
- Immeidate Block and Standard Block are set in the application as many as the set number of blocks.

4) If RamBlockDataAddress/RomBlockDataAddress is set in each block, the address is referenced in the NvM generation file, and the address (buffer) is not declared as extern, and a compilation error occurs. Therefore, when setting RamBlockDataAddress/RomBlockDataAddress, the header file whose address (buffer) is declared as extern must be included in NvM, and the header file name is added to NvMUserIncludeFiles.

5) Set according to DynamicConfiguration Usage item of SRS Information.
Note: If there is no corresponding item in SRS, the MCU does not support DynamicConfiguration. The description of DynamicConfiguration is as follows.
The job of the function NvM_ReadAll shall process an extended runtime preparation for all blocks which are configured with NvMResistantToChangedSw == FALSE and NvMDynamicConfiguration == TRUE and configuration ID mismatch occurs.
The job of the function NvM_ReadAll shall process the normal runtime preparation of all NVRAM blocks when they are configured with NvMResistantToChangedSw == TRUE and NvMDynamicConfiguration == TRUE and if a configuration ID mismatch occurs.
The job of the function NvM_ReadAll shall update the configuration ID from the RAM block assigned to the reserved NVRAM block with ID 1 according to the new (compiled) configuration ID, mark the NVRAM block to be written during NvM_WriteAll and request a CRC recalculation if a configuration ID mismatch occurs and if the NVRAM block is configured with

NvMDynamicConfiguration == TRUE. (See 7.2.2.16 regarding Extended/normal runtime)

6) When the Job is finished (ex. ReadAll completed) after MultiBlock Request (ex, ReadAll) call, the set Callback is called.
7) NvMUserJobFunction: User defined function which used in Memory task
8) NvMMainfunctionTriggerRef: Reference to an OsAlarm for the Memory servicing routine implementation
9) This function allow the user to set the order of blocks processed during ReadAll/WriteAll opearations, when disable, it operates based on AUTOSAR Specification.
10) NvMUserWdgToggleFunction: User defined function in Memory task to allow External Watchdog triggering

    If the user configures NvMUserWdgToggleFunction, user functions are repeatedly performed in mem_EaInitPerform and Mem_FeeInitPerform until initialization is completed.

    This function should be used only when Reset occurs before memory initialization is completed because External Watchdog is used and Timer configuration time is short. In general, it is recommended to use the Watchdog function provided by the platform.

5.2 NvMBlockDescriptor Container

Refer to the following settings.

Parameter Name	Value	Category
NvMBlockCrcType[1)]	User Defined	C
NvMBlockHeaderInclude	User Defined	C
NvMBlockJobPriority[2)]	User Defined	C
NvMBlockManagementType[3)]	User Defined	C
NvMBlockUseAutoValidation	User Defined	C
NvMBlockUseCrc[1)]	User Defined	C
NvMBlockUseCRCCompMechanism	User Defined	C
NvMBlockUseSetRamBlockStatus	User Defined	C
NvMBlockUseSyncMechanism	User Defined	C
NvMBlockWriteProt	False	F
NvMBswMBlockStatusInformation	False	F
NvMCalcRamBlockCrc[1)]	User Defined	C
NvMInitBlockCallback[14)]	User Defined	C
NvMMaxNumOfReadRetries	1	C
NvMMaxNumOfWriteRetries	1	C
NvMNvBlockBaseNumber[4)]	User Defined	C
NvMNvBlockLength[5)]	User Defined	C
NvMNvBlockNum[6)]	User Defined	C
NvMNvramBlockIdentifier[7)]	User Defined	C
NvMNvramDeviceId[8)]	User Defined	C
NvMRamBlockDataAddress[9)]	User Defined	C
NvMReadRamBlockFromNvCallback[17)]	User Defined	C
NvMResistantToChangedSw[15)]	User Defined	C
NvMRomBlockDataAddress	User Defined	C
NvMRomBlockNum[16)]	0	C
NvMSelectBlockForFirstInitAll	User Defined	C
NvMSelectBlockForReadAll[10)]	User Defined	C
NvMSelectBlockForWriteAll[10)]	User Defined	C
NvMSingleBlockCallback[11)]	User Defined	C
NvMStaticBlockIDCheck	False	F
NvMWriteBlockOnce	False	F
NvMWriteRamBlockToNvCallback [18)]	User Defined	C
NvMWriteVerification	False	C
NvMWriteVerificationDataSize	1	C
NvMDefaultRomCRCEnabled (AUTOEVER specific)	False	F
NvMTargetBlockReference[12)]	User Defined	C
NvMNameOfFeeBlock/ NvMNameOfEaBlock [13)]	User Defined	C
NvMReadAllOrder[19)]	User Defined	C
NvMWriteAllOrder[19)]	User Defined	C

1) When using CRC
   - NvMBlockCrcType: Set as desired in the App.
   - NvMBlockUseCrc, NvMCalcRamBlockCrc: Set to True. (Set to False when not in use)
   - The size of Fee / Ea Block should be set to NvMNvBlockLength + CRC Byte.
2) NvMBlockJobPriority
   - Immediate Block: Set to 0
   - For Standard Block: Set to a value other than 0 (considering priority among blocks)
3) NvMBlockManagementType
   - NVM_BLOCK_NATIVE : one copy
   - NVM_BLOCK_REDUNDANT : two copy
   - NVM_BLOCK_DATASET : an array of equally sized data blocks.
4) NvMNvBlockBaseNumber
   - Same setting as NvMNvramBlockIdentifier
5) NvMNvBlockLength
   - Set according to the design of the App (sub module Block-connected Fee/Ea Block -Length must be the same)
6) NvMNvBlockNum
   - NVM_BLOCK_NATIVE: set to 1
   - NVM_BLOCK_REDUNDANT: set to 2
   - NVM_BLOCK_DATASET: Set according to the design of the App.
7) NvMNvramBlockIdentifier
   - It should be set to Sequential with other Block ID. (From Autosar specification)
8) NvMNvramDeviceId
   - Device ID of the connected sub-module (Fls/Eep)
   - Set to 0 when using Ea Block (External EEPROM), set to 1 when using Fee block(Internal EEPROM).
   - When writing Int / Ext EEPROM at the same time, Block saved in Internal EEPROM is set to 1, Block stored in External EEPROM is set to 0.
9) NvMRamBlockDataAddress
   - Must be set when using ReadAll / WriteAll
   - Reads the NvBlock value in the RamBlock set in ReadAll, and writes the RamBlock value in WriteAll in the NvBlock.
   - The file declared as extern must be added to UserIncludeFile of NvM Common Container so that it can be included in the configured RamBlock NvM_Cfg.c.
   - Since the NvM module updates only with the start address and the set length of the set RamBlock, when the length is smaller than the Ramblock NvBlock, the ram area of other variables can be invaded. Therefore, the length of Ramblock and NvBlock are exactly the same.
10) NvMSelectBlockForReadAll / NvMSelectBlockForWriteAll
   - ReadAll: The value stored in EEPROM is read as Ram when StartUp.
   - WriteAll: Writes Ram value to EEPROM during ShutDowm.
   - When using WriteAll, WriteAll Time should be measured and reflected in the time from WdgDeInit to WdgReset.
11) NvMSingleBlockCallback
   - When using, set to “Rte_Call_NvM_PNJF_{Block}_JobFinished” (see Chapter 8.2.3)
     Block = {ecuc(NvM/NvMBlockDescriptor.SHORT-NAME)}
12) NvMTargetBlockReference

- If it is a block used in the internal EEPROM, select Fee
- Select Ea for Block used in External EEPROM

13) NvMNameOfFeeBlock/ NvMNameOfEaBlock
- After creating Fee / Ea Block according to NvM setting, connect the first Block.

14) NvMInitBlockCallback
- If InitBlockCallback is set during Read Fail, Callback is called. In the application, the Ram Block value is processed according to the design intention within the callback. Since RamBlock is considered to be processed by Application, NvM sets the block status to NVM_REQ_OK after InitBlockCallback is called.
- When using, set to "Rte_Call_NvM_PNIB_{Block}_InitBlock" (see Chapter 8.2.3) Block = {ecuc(NvM/NvMBlockDescriptor.SHORT-NAME)}

15) NvMResistantToChangedSw
- When NvMDynamicConfiguration setting is TRUE, it is meaningful.
- The job of the function NvM_ReadAll shall process an extended runtime preparation for all blocks which are configured with NvMResistantToChangedSw == FALSE and NvMDynamicConfiguration == TRUE and configuration ID mismatch occurs.
- The job of the function NvM_ReadAll shall process the normal runtime preparation of all NVRAM blocks when they are configured with NvMResistantToChangedSw == TRUE and NvMDynamicConfiguration == TRUE and if a configuration ID mismatch occurs.

16) NvMRomBlockNum
- If RomBlock is not set, the value should always be set to 0.

17) NvMReadRamBlockFromNvCallback
- When using, set to "Rte_Call_NvM_PM_{Block}_ ReadRamBlockFromNvM" (see Chapter 8.2.3)
  Block = {ecuc(NvM/NvMBlockDescriptor.SHORT-NAME)}

18) NvMSingleBlockCallback
- When using, set to "Rte_Call_NvM_P_{Block}_ WriteRamBlockToNvM" (see Chapter 8.2.3) Block = {ecuc(NvM/NvMBlockDescriptor.SHORT-NAME)}

19) NvMReadAllOrder/NvMWriteAllOrder
- For blocks that do not require order setting, it's not necessary to set the order.
- For blocks that require sequencing, the order must start from value <1> and be sequential with other orders.
- If the Block ID is <0> or <1>, order should not be set.

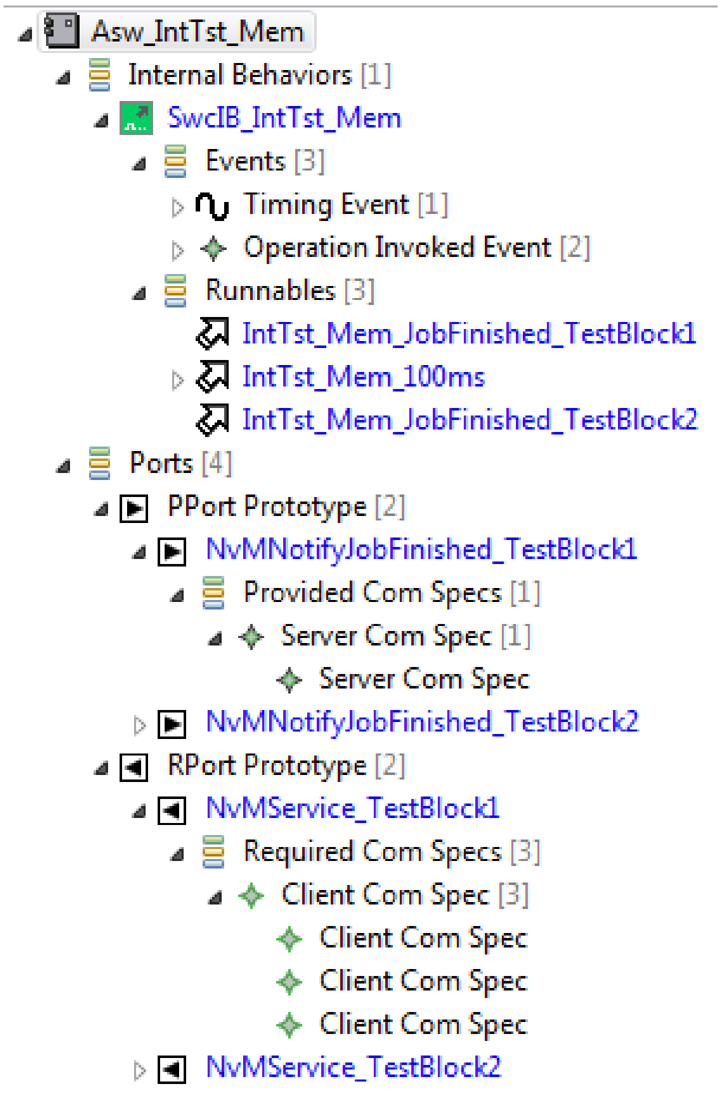
5.3 NvMDemEventParameterRefs Container

Refer to the following settings.

Parameter Name	Value	Category
NVM_E_INTEGRITY_FAILED	NVM_E_INTEGRITY_FAILED	F
NVM_E_LOSS_OF_REDUNDANCY	NVM_E_LOSS_OF_REDUNDANCY	F
NVM_E_HARDWARE	NVM_E_QUEUE_OVERFLOW	F
NVM_E_REQ_FAILED	NVM_E_REQ_FAILED	F
NVM_E_VERIFY_FAILED	NVM_E_VERIFY_FAILED	F
NVM_E_WRITE_PROTECTED	NVM_E_WRITE_PROTECTED	F
NVM_E_WRONG_BLOCK_ID	NVM_E_WRONG_BLOCK_ID	F

5.4 System Configuration

5.4.1 ApplicationSwComponentType setting


```
Asw_IntTst_Mem
  Internal Behaviors [1]
    SwcIB_IntTst_Mem
      Events [3]
        Timing Event [1]
        Operation Invoked Event [2]
      Runnables [3]
        IntTst_Mem_JobFinished_TestBlock1
        IntTst_Mem_100ms
        IntTst_Mem_JobFinished_TestBlock2
  Ports [4]
    PPort Prototype [2]
      NvMNotifyJobFinished_TestBlock1
        Provided Com Specs [1]
          Server Com Spec [1]
            Server Com Spec
      NvMNotifyJobFinished_TestBlock2
    RPort Prototype [2]
      NvMService_TestBlock1
        Required Com Specs [3]
          Client Com Spec [3]
            Client Com Spec
            Client Com Spec
            Client Com Spec
      NvMService_TestBlock2
```

- **Pport and Rport are created one by one**
  - PPort
    - short name : NvMNotifyJobFinished_<Block ID> (Port for Notification)
    - Map NvMNotifyJobFinished to Provided Interface
    - Create ServerComSpec and Map JobFinished to Operation
    - Write 1 to Queue
  - RPort
    - short name : NvMService<Block ID> (Port for Service)
    - Map NvMService to Required Interface
    - Generate ClientComSpec
    - API Mapping required for each Operation item (GetErrorStatus , ReadBlock, WriteBlock, etc.)
- **SwcInternalBehavior**
  - Create Runnable Entity
    - Short name : IntTst_Mem_JobFinished_Block<Block ID>

- Symbol : IntTst_Mem_JobFinished_Block<Block ID>
- CanBeInvokedConcurrently = false

▪ **Events / OperationInvokedEvent creation**

• Short name :

- OIE_IntTst_Mem_JobFinished_<Block ID>
- Map IntTst_Mem_JobFinished_<Block ID> to StartOnEvent Item

• Create Operation / POperation In Atomic Swc

- Instance Ref
  - Context P Port : NvMNotifyJobFinished_<Block ID>
  - Target Provided Operation : JobFinished

▪ **Ex) Service Interface Runnables registration (GetErrorStatus , ReadBlock, WriteBlock)**

• Short name: IntTst_Mem_100ms (Function to call NvM Api)

• Symbol: IntTst_Mem_100ms

• Server call points/3 SynchronousServerCallPoint creation

- short name: SynchronousServerCallPoint_GetErrorStatus_<Block ID>

• Create Operation / ROperation In Atomic Swc Instance Ref

- Context R Port: NvMService<Block ID>
- Target Required Operation : GetErrorStatus
- short name: SynchronousServerCallPoint_ReadBlock_<Block ID>

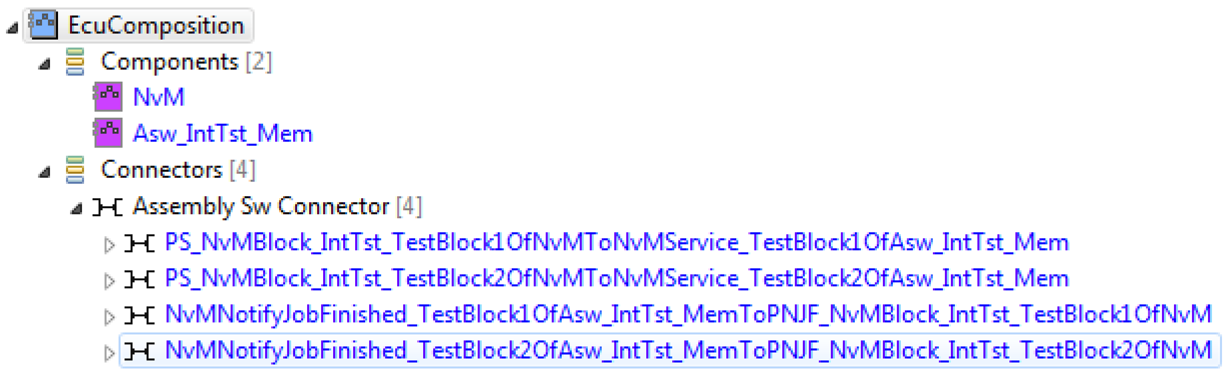
• Create Operation / ROperation In Atomic Swc Instance Ref

- Context R Port: NvMService<Block ID>
- Target Required Operation : ReadBlock
- short name: SynchronousServerCallPoint_WriteBlock_<Block ID>

• Create Operation / ROperation In Atomic Swc Instance Ref

- Context R Port : NvMService<Block ID>
- Target Required Operation : WriteBlock

5.4.2 EcuComposition settings



```
EcuComposition
  Components [2]
    NvM
    Asw_IntTst_Mem
  Connectors [4]
    Assembly Sw Connector [4]
      PS_NvMBlock_IntTst_TestBlock1OfNvMToNvMService_TestBlock1OfAsw_IntTst_Mem
      PS_NvMBlock_IntTst_TestBlock2OfNvMToNvMService_TestBlock2OfAsw_IntTst_Mem
      NvMNotifyJobFinished_TestBlock1OfAsw_IntTst_MemToPNJF_NvMBlock_IntTst_TestBlock1OfNvM
      NvMNotifyJobFinished_TestBlock2OfAsw_IntTst_MemToPNJF_NvMBlock_IntTst_TestBlock2OfNvM
```

▪ **Create AssemblySwConnector**

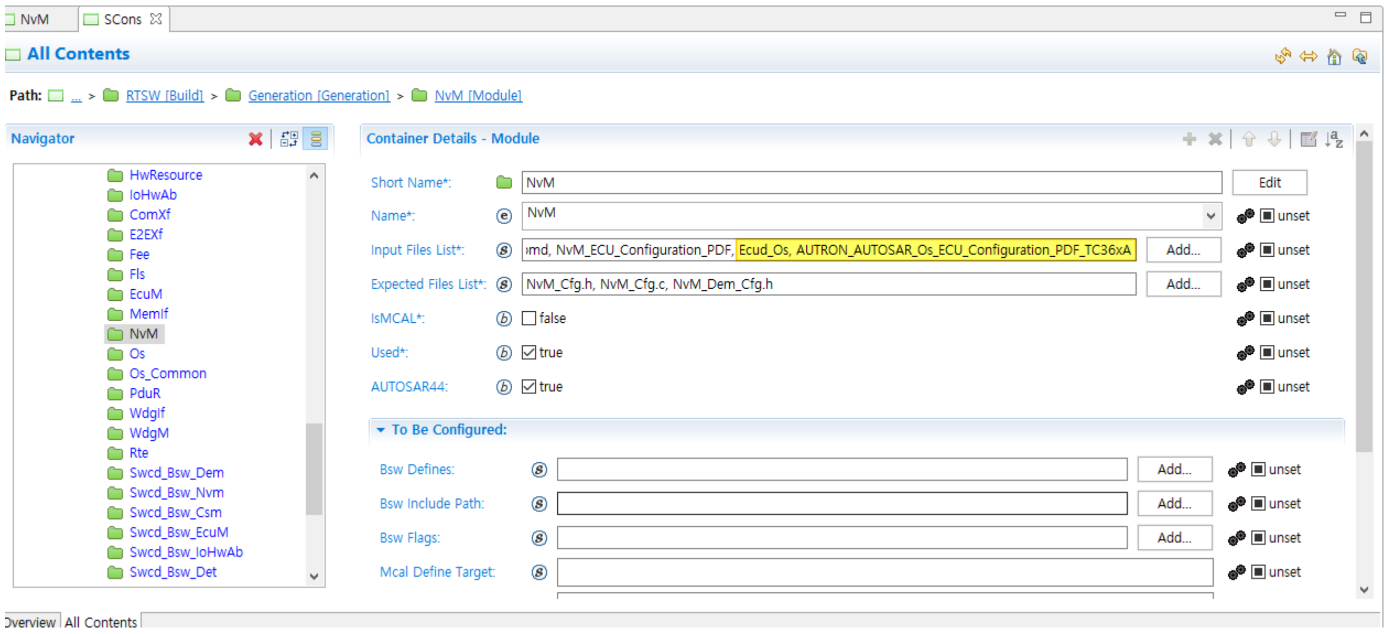
• Each Provider in AssemblySwConnector. Create Requester one by one

- Short name : PS_<Block ID>ofNvMToNvMService_<Block ID>ofAsw_IntTst_Mem
  - PPort In Composition Instance Ref
    - ♦ Context Component : NvM
    - ♦ Target P Port : PS_<Block ID>
    - ♦ Base : EcuComposition

    - RPort In Composition Instance Ref
        - Context Component : Asw_IntTst_Mem
        - Target R Port : NvMService_<Block ID>
        - Base : EcuComposition
- Short name : NvMNotifyJobFinished_<Block ID>ofAsw_IntTst_MemToPNJF_<Block ID>
    - PPort In Composition Instance Ref
        - Context Component : Asw_IntTst_Mem
        - Target P Port : NvMNotifyJobFinished_<Block ID>
        - Base : EcuComposition
    - RPort In Composition Instance Ref
        - Context Component : NvM
        - Target R Port : PNJF_<Block ID>
        - Base : EcuComposition

5.5 SCons.arxml Configuration

Input File List	- Ecud_Os - AUTRON_AUTOSAR_Os_ECU_Configuration_PDF_TC39xA_R4X (in case TC39x) In case of Os_ECU_Configuration_PDF, add PDF for your MCU environment - Dem_ECU_Configuration_PDF - Ea_ECU_Configuration_PDF - Eep_ECU_Configuration_PDF - NvM_ECU_Configuration_PDF - Bswmd_NvM - ECUConfigurationParameters - Ecud_Ea - Ecud_Eep - Ecud_NvM - Ecud_Dem - Fls_17_Dmu_TC397.bmd - Fee_TC397.bmd - Ecud_Fee - Ecud_Fls


Path: ... > RTSW [Build] > Generation [Generation] > NvM [Module]

Container Details - Module

Short Name*: NvM
Name*: NvM
Input Files List*: ›md, NvM_ECU_Configuration_PDF, Ecud_Os, AUTRON_AUTOSAR_Os_ECU_Configuration_PDF_TC36xA
Expected Files List*: NvM_Cfg.h, NvM_Cfg.c, NvM_Dem_Cfg.h
IsMCAL*: false
Used*: true
AUTOSAR44: true

To Be Configured:
Bsw Defines:
Bsw Include Path:
Bsw Flags:
Mcal Define Target:

6. Application Programming Interface (API)

6.1 Type Definitions

6.1.1 NvM_ConfigType

Name:	NvM_ConfigType	
Type:	Structure	
Range:	implementation specific	--
Description:	Configuration data structure of the NvM module.	
Available via:	NvM.h	

6.1.2 NvM_MultiBlockRequestType

Name:	NvM_MultiBlockRequestType		
Type:	Enumeration		
Range:	NVM_READ_ALL	0x00	NvM_ReadAll was performed
	NVM_WRITE_ALL	0x01	NvM_WriteAll was performed
	NVM_VALIDATE_ALL	0x02	NvM_ValidateAll was performed
	NVM_FIRST_INIT_ALL	0x03	NvM_FirstInitAll was performed
	NVM_CANCEL_WRITE_ALL	0x04	NvM_CancelWriteAll was performed
Description:	Identifies the type of request performed on multi block when signaled via the callback function or when reporting to BswM		
Available via:	NvM.h		

6.2 Macro Constants

None

6.3 Functions

6.3.1 Synchronous Requests

6.3.1.1 NvM_Init

Service name:	NvM_Init
Syntax:	void NvM_Init(const NvM_ConfigType* ConfigPtr)
Service ID[hex]:	0x00

Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ConfigPtr	Pointer to the selected configuration set.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Service for basic NVRAM Manager Initialization by resetting all internal variables. This function is used by BSW.	
Available via:	NvM.h	

6.3.1.2 NvM_SetDataIndex

Service name:	NvM_SetDataIndex	

Syntax:	Std_ReturnType NvM_SetDataIndex(
	    NvM_BlockIdType BlockId,
	    uint8 DataIndex
	)	

Service ID[hex]:	0x01	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	BlockId	The block identifier uniquely identifies one NVRAM block descriptor. A NVRAM block descriptor contains all needed information about a single NVRAM block.
	DataIndex	Index position (association) of a NV/ROM block.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The index position was set successfully. E_NOT_OK: An error occurred.
Description:	The function sets the association of Dataset NV block with its corresponding RAM block by storing the 8 bit DataIndex passed by the application to the index field of the RAM block. This function is used by user. But it needs configuration. (It cannot be called directly by user)	

6.3.1.3 NvM_GetDataIndex

Service name:	NvM_GetDataIndex

Syntax:	Std_ReturnType NvM_GetDataIndex(
	    NvM_BlockIdType BlockId,
	    uint8* DataIndexPtr
	)

Service ID[hex]:	0x02
Sync/Async:	Synchronous
Reentrancy:	Reentrant

Parameters (in):	BlockId	The block identifier uniquely identifies one NVRAM block descriptor. A NVRAM block descriptor contains all needed information about a single NVRAM block.
Parameters (inout):	None	
Parameters (out):	DataIndexPtr	Pointer to where to store the current dataset index (0..255)
Return value:	Std_ReturnType	E_OK: The index position has been retrieved successfully. E_NOT_OK: An error occurred.
Description:	This function reads the index (association of NV block with its corresponding RAM block) from the RAM block index field. This function is used by user. But it needs configuration. (It cannot be called directly by user)	

6.3.1.4 NvM_SetBlockProtection

Service name:	NvM_SetBlockProtection	
Syntax:	Std_ReturnType NvM_SetBlockProtection(NvM_BlockIdType BlockId, boolean ProtectionEnabled)	
Service ID[hex]:	0x03	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	BlockId	The block identifier uniquely identifies one NVRAM block descriptor. A NVRAM block descriptor contains all needed information about a single NVRAM block.
	ProtectionEnabled	TRUE: Write protection shall be enabled FALSE: Write protection shall be disabled
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The block was enabled/disabled as requested E_NOT_OK: An error occurred.
Description:	This function enables/disables the write block protection bit in the RAM block attribute/error/status field. This function is available only if API Configuration Class 3 is enabled. This function is used by user. But it needs configuration. (It cannot be called directly by user)	

6.3.1.5 NvM_GetErrorStatus

Service name:	NvM_GetErrorStatus
Syntax:	Std_ReturnType NvM_GetErrorStatus(NvM_BlockIdType BlockId, NvM_RequestResultType* RequestResultPtr)
Service ID[hex]:	0x04
Sync/Async:	Synchronous
Reentrancy:	Reentrant

Parameters (in):	BlockId	The block identifier uniquely identifies one NVRAM block descriptor. A NVRAM block descriptor contains all needed information about a single NVRAM block.
Parameters (inout):	None	
Parameters (out):	RequestResultPtr	Pointer to where to store the request result. See NvM_RequestResultType .
Return value:	Std_ReturnType	E_OK: The block dependent error/status information was read successfully. E_NOT_OK: An error occurred.
Description:	This API reads the attribute field bits from the RAM block attribute/ error/ status field and returns the error/status information to the application. . This function is used by user. But it needs configuration. (It cannot be called directly by user)	
Available via:	NvM.h	

6.3.1.6 NvM_GetVersionInfo

Service name:	NvM_GetVersionInfo	
Syntax:	void NvM_GetVersionInfo(Std_VersionInfoType* versioninfo)	
Service ID[hex]:	0x0f	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	versioninfo	Pointer to where to store the version information of this module.
Return value:	None	
Description:	Service to get the version information of the NvM module. This function is used by user. But it needs configuration. (It cannot be called directly by user)	
Available via:	NvM.h	

6.3.1.7 NvM_SetRamBlockStatus

Service name:	NvM_SetRamBlockStatus	

Syntax:	Std_ReturnType
	    NvM_SetRamBlockStatus(
	    NvM_BlockIdType BlockId,
	    boolean BlockChanged
	)	

Service ID[hex]:	0x05	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
	BlockId	The block identifier uniquely identifies one NVRAM block descriptor. A NVRAM block descriptor contains all needed information about a single NVRAM block.

Parameters (in):	BlockChanged	TRUE: Validate the permanent RAM block or the explicit synchronization and mark block as changed. FALSE: Invalidate the permanent RAM block or the explicit synchronization and mark block as unchanged.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The status of the permanent RAM block or the explicit synchronization was changed as requested. E_NOT_OK: An error occurred.
Description:	This API recalculates the CRC (if configured) for the RAM block data and sets the state of the RAM block to valid/invalid. This function is used by user. But it needs configuration. (It cannot be called directly by user)	
Available via:	NvM.h	

6.3.1.8 NvM_SetBlockLockStatus

Service name:	NvM_SetBlockLockStatus	

Syntax:	void NvM_SetBlockLockStatus(
	    NvM_BlockIdType BlockId,
	    boolean BlockLocked
	)	

Service ID[hex]:	0x13	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	BlockId	The block identifier uniquely identifies one NVRAM block descriptor. A NVRAM block descriptor contains all needed information about a single NVRAM block.
	BlockLocked	TRUE: Mark the RAM.block as locked FALSE: Mark the RAM.block as unlocked
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Service for setting the lock status of a permanent RAM block or of the explicit synchronization of a NVRAM block. This function is used by user. But it needs configuration. (It cannot be called directly by user)	
Available via:	NvM.h	

6.3.1.9 NvM_CancelJobs

Service name:	NvM_CancelJobs
Syntax:	Std_ReturnType NvM_CancelJobs(NvM_BlockIdType BlockId)
Service ID[hex]:	0x10
Sync/Async:	Synchronous

Reentrancy:	Reentrant	
Parameters (in):	BlockId	The block identifier uniquely identifies one NVRAM block descriptor. A NVRAM block descriptor contains all needed information about a single NVRAM block.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The job was successfully removed from queue. E_NOT_OK: The job could not be found in the queue.
Description:	Service to cancel all jobs pending for a NV block. This function is used by user. But it needs configuration. (It cannot be called directly by user)	
Available via:	NvM.h	

6.3.2 Asynchronous Single Block Requests

6.3.2.1 NvM_ReadBlock

Service name:	NvM_ReadBlock	
Syntax:	Std_ReturnType NvM_ReadBlock(NvM_BlockIdType BlockId, void* NvM_DstPtr)	
Service ID[hex]:	0x06	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	BlockId	The block identifier uniquely identifies one NVRAM block descriptor. A NVRAM block descriptor contains all needed information about a single NVRAM block.
Parameters (inout):	None	
Parameters (out):	NvM_DstPtr	Pointer to the RAM data block.
Return value:	Std_ReturnType	E_OK: request has been accepted E_NOT_OK: request has not been accepted
Description:	Request updates the job queue with the BlockId and NvM_DstPtr and Service Id to 'Read' the NV/ROM block data to its corresponding RAM block. This function is used by user. But it needs configuration. (It cannot be called directly by user)	
Available via:	NvM.h	

6.3.2.2 NvM_WriteBlock

Service name:	NvM_WriteBlock
Syntax:	Std_ReturnType NvM_WriteBlock(NvM_BlockIdType BlockId, const void* NvM_SrcPtr)
Service ID[hex]:	0x07

Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	BlockId	The block identifier uniquely identifies one NVRAM block descriptor. A NVRAM block descriptor contains all needed information about a single NVRAM block.
	NvM_SrcPtr	Pointer to the RAM data block.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: request has been accepted E_NOT_OK: request has not been accepted
Description:	Request updates the job queue with the BlockId, NvM_SrcPtr and Service Id to 'Write' the RAM block data to its corresponding NV block. This function is used by user. But it needs configuration. (It cannot be called directly by user)	
Available via:	NvM.h	

6.3.2.3 NvM_RestoreBlockDefaults

Service name:	NvM_RestoreBlockDefaults	

Syntax:	Std_ReturnType
	   NvM_RestoreBlockDefaults(
	   NvM_BlockIdType BlockId,
	   void* NvM_DestPtr
	)	

Service ID[hex]:	0x08	
Sync/Async:	Asynchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	BlockId	The block identifier uniquely identifies one NVRAM block descriptor. A NVRAM block descriptor contains all needed information about a single NVRAM block.
Parameters (inout):	None	
Parameters (out):	NvM_DestPtr	Pointer to the RAM data block.
Return value:	Std_ReturnType	E_OK: request has been accepted E_NOT_OK: request has not been accepted
Description:	Request updates the job queue with the BlockId, NvM_DestPtr and Service Id to 'Restore' the ROM block default data to its corresponding RAM block. This function is used by user. But it needs configuration. (It cannot be called directly by user)	
Available via:	NvM.h	

6.3.2.4 NvM_EraseNvBlock

Service name:	NvM_EraseNvBlock
Syntax:	Std_ReturnType NvM_EraseNvBlock(NvM_BlockIdType BlockId)

Service ID[hex]:	0x09	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	BlockId	The block identifier uniquely identifies one NVRAM block descriptor. A NVRAM block descriptor contains all needed information about a single NVRAM block.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: request has been accepted E_NOT_OK: request has not been accepted
Description:	Request updates the job queue with the BlockId and Service Id to 'Erase' the NV block data. This function is used by user. But it needs configuration. (It cannot be called directly by user)	
Available via:	NvM.h	

6.3.2.5 NvM_InvalidateNvBlock

Service name:	NvM_InvalidateNvBlock	

Syntax:	Std_ReturnType
	    NvM_InvalidateNvBlock(
	    NvM_BlockIdType BlockId
	)	

Service ID[hex]:	0x0b	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	BlockId	The block identifier uniquely identifies one NVRAM block descriptor. A NVRAM block descriptor contains all needed information about a single NVRAM block.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: request has been accepted E_NOT_OK: request has not been accepted
Description:	Request updates the job queue with the BlockId and Service Id to 'Invalidate' the NV block data. This function is used by user. But it needs configuration. (It cannot be called directly by user)	
Available via:	NvM.h	

6.3.2.6 NvM_ReadPRAMBlock

Service name:	NvM_ReadPRAMBlock
Syntax:	Std_ReturnType NvM_ReadPRAMBlock(NvM_BlockIdType BlockId)
Service ID[hex]:	0x16

Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	BlockId	The block identifier uniquely identifies one NVRAM block descriptor. A NVRAM block descriptor contains all needed information about a single NVRAM block.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: request has been accepted E_NOT_OK: request has not been accepted
Description:	Request updates the job queue with the BlockId and Service Id to 'Read' the NV/ROM block data to its corresponding PRAM block. This function is used by user. But it needs configuration. (It cannot be called directly by user)	
Available via:	NvM.h	

6.3.2.7 NvM_WritePRAMBlock

Service name:	NvM_WritePRAMBlock	
Syntax:	Std_ReturnType NvM_WritePRAMBlock(NvM_BlockIdType BlockId)	
Service ID[hex]:	0x17	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	BlockId	The block identifier uniquely identifies one NVRAM block descriptor. A NVRAM block descriptor contains all needed information about a single NVRAM block.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: request has been accepted E_NOT_OK: request has not been accepted
Description:	Service to copy the data of the RAM block to its corresponding permanent RAM block. This function is used by user. But it needs configuration. (It cannot be called directly by user)	
Available via:	NvM.h	

6.3.2.8 NvM_RestorePRAMBlockDefaults

Service name:	NvM_RestorePRAMBlockDefaults
Syntax:	Std_ReturnType NvM_RestorePRAMBlockDefaults(NvM_BlockIdType BlockId)
Service ID[hex]:	0x18

Sync/Async:	Asynchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	BlockId	The block identifier uniquely identifies one NVRAM block descriptor. A NVRAM block descriptor contains all needed information about a single NVRAM block.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: request has been accepted E_NOT_OK: request has not been accepted
Description:	Request updates the job queue with the BlockId and Service Id to 'Restore' the ROM block default data to its corresponding RAM block. This function is used by user. But it needs configuration. (It cannot be called directly by user)	
Available via:	NvM.h	

6.3.3 Asynchronous Multi Block Requests

6.3.3.1 NvM_ReadAll

Service name:	NvM_ReadAll

Syntax:	void
	   NvM_ReadAll(
	   void
	)

Service ID[hex]:	0x0c
Sync/Async:	Asynchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Initiates a multi block read request. This function is used by user. But it needs configuration. (It cannot be called directly by user)
Available via:	NvM.h

6.3.3.2 NvM_WriteAll

Service name:	NvM_WriteAll
Syntax:	void NvM_WriteAll(void)
Service ID[hex]:	0x0d
Sync/Async:	Asynchronous

Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Request to synchronize the contents of RAM blocks to their corresponding NV blocks during shutdown.This function is used by user. But it needs configuration. (It cannot be called directly by user)
Available via:	NvM.h

6.3.3.3 NvM_CancelWriteAll

Service name:	NvM_CancelWriteAll
Syntax:	void NvM_CancelWriteAll(void)
Service ID[hex]:	0x0a
Sync/Async:	Asynchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Service to cancel a running NvM_WriteAll request. This function is used by user. But it needs configuration. (It cannot be called directly by user)
Available via:	NvM.h

6.3.3.4 NvM_ValidateAll

Service name:	NvM_ValidateAll
Syntax:	void NvM_ValidateAll(void)
Service ID[hex]:	0x19
Sync/Async:	Asynchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None

Return value:	None
Description:	Initiates a multi block validation request. This function is used by user. But it needs configuration. (It cannot be called directly by user)
Available via:	NvM.h

6.3.3.5 NvM_FirstInitAll

Service name:	NvM_FirstInitAll
Syntax:	void NvM_FirstInitAll(void)
Service ID[hex]:	0x14
Sync/Async:	Asynchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	The function initiates a multi block first initialization request. The job of the function does not care if a block exists in the non-volatile memory or not OR if it is valid (i.e. not corrupted) or not, when processing it. This function is used by user. But it needs configuration. (It cannot be called directly by user)
Available via:	NvM.h

6.3.4 Callback Notifications

6.3.4.1 NvM_JobEndNotification

Service name:	NvM_JobEndNotification
Syntax:	void NvM_JobEndNotification(void)
Service ID[hex]:	0x11
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Function to be used by the underlying memory abstraction to signal end of job without error. This function is used by user. But it needs configuration. (It

	cannot be called directly by user)
Available via:	NvM_MemIf.h

6.3.4.2 NvM_JobErrorNotification

Service name:	NvM_JobErrorNotification
Syntax:	void NvM_JobErrorNotification(void)
Service ID[hex]:	0x12
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Function to be used by the underlying memory abstraction to signal end of job with error. This function is used by user. But it needs configuration. (It cannot be called directly by user)
Available via:	NvM_MemIf.h

6.3.5 Scheduled Functions

6.3.5.1 NvM_Mainfunction

Service name:	NvM_MainFunction
Syntax:	void NvM_MainFunction(void)
Service ID[hex]:	0x0e
Description:	Service for performing the processing of the NvM jobs. This function is used by BSW.
Available via:	SchM_NvM.h

6.3.6 User Defined Function

6.3.6.1 NvM_UserJobFunction

Function Name	NvM_UserJobFunction
Syntax:	FUNC(NvM_OpStatusType, NVM_CODE) NvM_UserJobFunction (void)
Service ID	None
Sync/Async	Synchronous

Reentrancy	Non Reentrant
Parameters (In)	None
Parameters (Inout)	None
Parameters (Out)	None
Return Value	NvM_OpStatusType: NVM_OPSTATUS_IDLE: user job is idle NVM_OPSTATUS_BUSY: user job is busy
Description	When users set NvMUserJobFunction, UserJob function is driven in OsTask_BSW_Mem_Process Task. NVM_OPSTATUS_IDLE must be returned when there is nothing to handle in UserJobFunction. NVM_OPSTATUS_BUSY is also returned when UserJob function needs to be executed. And mainfunctions of memory stacks are not driven. Therefore, UserJob processing must be completed and NVM_OPSTATUS_IDLE needs to be returned as soon as possible. Note: UserJob must be started when the return value of NvM_CddGetStatus is NVM_OPSTATUS_IDLE.
Preconditions	None
Configuration Dependency	This function is available only if configuration parameter NvMUserJobFunction is configured.

6.3.6.2 NvM_UserWdgToggleFunction

Function Name	NvM_UserWdgToggleFunction
Syntax:	void NvM_UserWdgToggleFunction (void)
Service ID	None
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (In)	None
Parameters (Inout)	None
Parameters (Out)	None
Return Value	None
Description	If the user configures NvMUserWdgToggleFunction, user functions are repeatedly performed in mem_EaInitPerform and Mem_FeeInitPerform until initialization is completed. This function should be used only when Reset occurs before memory initialization is completed because External Watchdog is

	used and Timer configuration time is short. In general, it is recommended to use the Watchdog function provided by the platform.
Preconditions	None
Configuration Dependency	This function is available only if configuration parameter NvMUserWdgToggleFunction is configured.

6.4 Noted

6.4.1 In Communication with application SW-C

For the prototype of the RTE-based generated function, see the AUTOSAR_EXP_NVDataHandling.pdf document.

7. Generator

7.1 Generator Option

7.1.1 NvM

Option	Description
S	Mode Switch Interface for ECU Mode, Client Server Interface and P-Port for each service are created in Swcd_Bsw_EcuM.arxml.
P	Create Port Name according to Autosar NvM SWS 4.2.2 spec.

7.2 Generator Error Message

7.2.1 NvM

7.2.1.1 Error Messages

1) ERR020005: Value of NvMNvBlockBaseNumber must equals NvMTargetBlockReference.[Fee/Ea]BlockConfiguration.[Fee/Ea]BlockNumber >> NvMDatasetSelectionBits.
   - With NvMBlockDescriptor, value of /AUTOEVER/NvM/NvMBlockDescriptor/NvMNvBlockBaseNumber = NvMTargetBlockReference.[Fee/Ea]BlockConfiguration.[Fee/Ea]BlockNumber >> /AUTOEVER/NvM/NvMCommon/NvMDatasetSelectionBits
2) ERR020006: NvMNvramBlockIdentifier must be unique.
   - With NvMBlockDescriptor, /AUTOEVER/NvM/NvMBlockDescriptor/NvMNvramBlockIdentifier must be unique
3) ERR020011: The value configured for the parameter "Parameter Name" in the container "Container Name" should follow the pattern: "Pattern".
   - This error occurs, When the parameter "Parameter Name" in "NvMBlockDescriptor" is not configured as per the "Pattern".

Parameter Name	Container Name	Pattern	Example
SW-VERSION	BSW-IMPLEMENTATION	1.[0-9]+.[0-9]+	1.0.0
NvMRamBlockDataAdress	NvMBlockDescriptor short name	[a-zA-Z][a-zA-Z0-9₩_]*	RamBlockDataAddress_1
NvMRomBlockDataAddress			RomBlockDataAddress_1

4) ERR020015: Value of the parameter "NvMBlockUseSyncMechanism" should be configured as <false/0>, when the value of the parameter "NvMWriteVerification" is configured as <true/1> in the container "<NvMBlockDescriptor>".
   - This error occurs, if the value configured for the parameter NvMBlockUseSyncMechanism is <true/1> when the value of the parameter NvMWriteVerification is configured as <true/1> in the container NvMBlockDescriptor. With <NvMBlockDescriptor> is short name of NvMBlockDescriptor

5) ERR020016: The sum of parameters "NvMRomBlockNum" and "NvMNvBlockNum" should be less than or equal to <255> when the value of the parameter "NvMBlockManagementType" is configured as <NVM_BLOCK_DATASET> in the container "<NvMBlockDescriptor>".
   - This error occurs, if the sum of the value of the parameters NvMRomBlockNum and NvMNvBlockNum is greater than 255 when the value of the parameter NvMBlockManagementType is configured as NVM_BLOCK_DATASET in the container NvMBlockDescriptor for each configured block. With <NvMBlockDescriptor> is short name of NvMBlockDescriptor
6) ERR020017: Value of the parameters "NvMReadRamBlockFromNvCallback" and "NvMWriteRamBlockToNvCallback" should be configured since the value of the parameter "NvMBlockUseSyncMechanism" in the container is configured in the container "<NvMBlockDescriptor>".
   - This error occurs, if the parameters NvMReadRamBlockFromNvCallback or NvMWriteRamBlockToNvCallback are not configured when the value of the parameter NvMBlockUseSyncMechanism is configured as <true/1> in the container NvMBlockDescriptor. With <NvMBlockDescriptor> is short name of NvMBlockDescriptor
7) ERR020020: Value of the parameter "NvMNvBlockNum" in the container "<NvMBlockDescriptor>" should be less than or equal to 2^NvMDatasetSelectionBits in the container "NvMCommon".
   - This error occurs, if the value of the parameter NvMNvBlockNum in the container <NvMBlockDescriptor> is greater than 2^NvMDatasetSelectionBits in the container NvMCommon. With <NvMBlockDescriptor> is short name of NvMBlockDescriptor
8) ERR020021: Value of the parameter "NvMWriteVerificationDataSize" should be less than or equal to "NvMNvBlockLength" and "NvMNvBlockLength" should be completely divisible by "NvMWriteVerificationDataSize" in the container "<NvMBlockDescriptor>".
   - This error occurs, if the value of the parameter NvMWriteVerificationDataSize is greater than NvMNvBlockLength and NvMNvBlockLength is not completely divisible by NvMWriteVerificationDataSize. With <NvMBlockDescriptor> is short name of NvMBlockDescriptor
9) ERR020022: Value of the parameter "NvMBlockManagementType" should be <NVM_BLOCK_REDUNDANT> and the value of the parameter "NvMRamBlockDataAddress" should be configured in the container "<NvMBlockDescriptor>" for the block in which "NvMNvramBlockIdentifier" is configured as <1>.
   - This error occurs, if the value configured for the parameter NvMBlockManagementType is
     configured other than <NVM_BLOCK_REDUNDANT> or the parameter NvMRamBlockDataAddress is not configured in the container NvMBlockDescriptor for the block in which NvMNvramBlockIdentifier is configured as <1>. With <NvMBlockDescriptor> is short name of NvMBlockDescriptor
10) ERR020024: Value of the parameter "NvMRamBlockDataAddress" or "NvMBlockUseSyncMechanism" should be configured for the block that has "NvMSelectBlockForReadAll/NvMSelectBlockForWriteAll" is configured as <true/1>.
   - This error occurs, if the parameter 'NvMRamBlockDataAddress' is not configured and 'NvMBlockUseSyncMechanism' is <false/0> for the block that has "NvMSelectBlockForReadAll/NvMSelectBlockForWriteAll" is configured as <true/1>.
11) ERR020026: Value configured for the parameter "Parameter Name" should be unique in the container "Container Name".
   - This error occurs, if the value configured for the parameters is listed in table below that is not unique.

Parameter Name	Container Name
Value of NvMNameOfEaBlock	NvMEaRef
Value of NvMNameOfFeeBlock	NvMFeeRef

12) ERR020027: The value configured for the parameter "NvMNvramBlockIdentifier" in the container "NvMBlockDescriptor" should be sequential.
    - This error occurs, if the value configured for the parameter NvMNvramBlockIdentifier in the container NvMBlockDescriptor is not sequential.
13) ERR020028: The value of the parameter "NvMNvramBlockIdentifier" in the container "NvMBlockDescriptor" should start with <1>.
    - This error occurs, if the value configured for the parameter NvMNvramBlockIdentifier in the container NvMBlockDescriptor should start with 1.
14) ERR020029: Value of the parameter "NvMCalcRamBlockCrc" should be configured as <true/1>, when the value of the parameter "NvMBlockUseCrc" is configured as <true/1> in the container "<NvMBlockDescriptor>".
    - This error occurs, if the value configured for the parameter NvMCalcRamBlockCrc is <false/0> when the value configured for the parameter NvMBlockuseCrc is <true/1> in the container NvMBlockDescriptor. With <NvMBlockDescriptor> is short name of NvMBlockDescriptor
15) ERR020030: Value of the parameter "NvMBlockCrcType" should be configured, when the value of the parameter "NvMBlockUseCrc" is configured as <true/1> in the container "<NvMBlockDescriptor>".
    - This error occurs, if the parameter NvMBlockCrcType is not configured, when the value of the parameter NvMBlockUseCrc is configured as <true/1> in the container NvMBlockDescriptor. With <NvMBlockDescriptor> is short name of NvMBlockDescriptor
16) ERR020031: Value of the parameter "NvMBlockUseCrc" should be configured as <true/1>, when the value of the parameter "NvMWriteBlockOnce" is configured as <true/1> in the container "<NvMBlockDescriptor>".
    - This error occurs, if the value of the parameter NvMBlockUseCrc is configured as <false/0>, when the value of the parameter NvMWriteBlockOnce is configured as <true/1> in the container NvMBlockDescriptor. With <NvMBlockDescriptor> is short name of NvMBlockDescriptor
17) ERR020032: Value of the parameter "NvMBlockWriteProt" should not be configured as <true/1>, when the value of the parameter "NvMWriteBlockOnce" is configured as <true/1> in the container "<NvMBlockDescriptor>".
    - This error occurs, if the value of the parameter NvMBlockWriteProt is configured as <true/1>, when the value of the parameter NvMWriteBlockOnce is configured as <true/1> in the container NvMBlockDescriptor. With <NvMBlockDescriptor> is short name of NvMBlockDescriptor
18) ERR020046: Value of the parameter "NvMBlockUseSyncMechanism" should be <true/1> or the parameter "NvMRamBlockDataAddress" should be configured when "NvMBlockUseAutoValidation" is configured as <true/1>.
    - This error occurs, if NvMBlockUseSyncMechanism is configured as <false/0> and NvMRamBlockDataAddress is not configured when NvMBlockUseAutoValidation is configured as <true/1>
19) ERR020047: Value of the parameter "NvMBlockManagementType" in the container "<NvMBlockDescriptor>" should not be configured as <NVM_BLOCK_DATASET>, since value of

the parameter "Parameter Name" in the container "NvMCommon" is configured as <Value>.

- This error occurs, if the value of the parameter NvMBlockManagementType in the container NvMBlockDescriptor is configured as <NVM_BLOCK_DATASET>, when one of the below mentioned parameters "Parameter Name" are configured as <value>. (Parameter is listed in below table). With <NvMBlockDescriptor> is short name of NvMBlockDescriptor

Parameter Name	Value
NvMApiConfigClass	NVM_API_CONFIG_CLASS_1
NvMDatasetSelectionBits	0

20) ERR020039: AUTOSAR Release version <version> configured for the parameter 'AR-RELEASEVERSION' in provided MDT file is not correct. AUTOSAR Release version should be one of the following: 4.4.0
    This information occurs, if the value of the element AR-RELEASE-VERSION present in the
    BSW Module Description template is configured other than 4.4.0, with version is value configured for "AR-RELEASEVERSION" in provided MDT file
21) ERR020048: Sum of (NvMNvBlockLength + CRC size (if configured) + Static Id size (if configured)) greater than EaBlockSize is referenced.
    - For all NvMBlockDescriptor, this error occurs, if NvMNvBlockLength + CRC size (if configured) + Static Id size (if configured) greater than EaBlockSize
22) ERR020036: Value of the parameter "NvMSelectBlockForReadAll" is considered as <false/0>, when the value of the parameter "NvMRamBlockDataAddress" is not configured and the value of the parameter "NvMBlockUseSyncMechanism" is not <true/1> in the container "<NvMBlockDescriptor>".
    - This error occurs, if the value configured for the parameter NvMSelectBlockForReadAll is <true/1>, when the parameter NvMRamBlockDataAddress is not configured and NvMBlockUseSyncMechanism is not <true/1> in the container NvMBlockDescriptor. With <NvMBlockDescriptor> is short name of NvMBlockDescriptor
23) ERR020038: Value of the parameter "NvMSelectBlockForWriteAll" is considered as <false/0>, when the value of the parameter "NvMRamBlockDataAddress" is not configured and the value of "NvMBlockUseSyncMechanism" is not <true/1> in the container "<NvMBlockDescriptor>".
    - This error occurs, if the value configured for the parameter NvMSelectBlockForWriteAll is <true/1>, when the parameter NvMRamBlockDataAddress is not configured and NvmBlockUseSyncMechanism is not <true/1> in the container NvMBlockDescriptor. With <NvMBlockDescriptor> is short name of NvMBlockDescriptor
24) ERR020010: The value of "NvramDeviceId" must equals DeviceIndex of TargetBlockReference.
    - This error occurs, if the value configured for the parameter NvMNvramDeviceId in the container NvMBlockDescriptor is not equal to the value configured follow the value of NvMTargetBlockReference in the table below.

Shortname of NvMTargetBlockReference	Value
NvMEaRef	1
NvMFeeRef	0

25) ERR020052: "OsSecPerTick" must be defined when it was not be configured in OsCounter Container or the value of "OsSecPerTick" must be greater than zero.
    When "NvMCommon/NvMMainfunctionTriggerRef" parameter is configured. User needs to

define OsSecPerTick in 2 ways following:
- Configure Os/OsCounter/OsSecondsPerTick in container reference by "NvMCommon/NvMMainfunctionTriggerRef" >> Os/OsAlarm/OsAlarmCounterRef >> Os/OsCounter/OsSecondsPerTick
- Add module options with this syntax: "OsSecPerTick: number" with number is value of OsSecPerTick.

26) ERR020053: The value configured for the parameter "NvMReadAllOrder" in the container “NvMBlockDescriptor" should be blank when "NvMSelectBlockForReadAll" is configured as <false/0> or "NvMNvramBlockIdentifier" is <0> or <1>.
- This error occurs, if value of the parameter "NvMReadAllOrder" is configured when "NvMSelectBlockForReadAll" is configured as <false/0> or "NvMNvramBlockIdentifier" is 0 or 1.
27) ERR020054: The value configured for the parameter "NvMReadAllOrder" must be unique.
- This error occurs, if value of the parameter "NvMReadAllOrder" is configured with duplicate parameter.
28) ERR020055: The value of parameter "NvMReadAllOrder" should start with <1>.
- This error occurs, if values configured for the parameter "NvMReadAllOrder" do not start with 1.
29) ERR020056: The value configured for the parameter "NvMReadAllOrder" should be sequential.
- This error occurs, if values configured for the parameter "NvMReadAllOrder" are not sequential.
30) ERR020057: The value configured for the parameter "NvMWriteAllOrder" in the container “NvMBlockDescriptor" should be blank when "NvMSelectBlockForWriteAll" is configured as <false/0> or "NvMNvramBlockIdentifier" is <0> or <1>.
- This error occurs, if value of the parameter "NvMWriteAllOrder" is configured when "NvMSelectBlockForWriteAll" is configured as <false/0> or "NvMNvramBlockIdentifier" is 0 or 1.
31) ERR020058: The value configured for the parameter "NvMWriteAllOrder" must be unique.
- This error occurs, if value of the parameter "NvMWriteAllOrder" is configured with duplicate parameter.
32) ERR020059: The value of parameter "NvMWriteAllOrder" should start with <1>.
- This error occurs, if values configured for the parameter "NvMWriteAllOrder" do not start with 1.
33) ERR020060: The value configured for the parameter "NvMWriteAllOrder" should be sequential.
- This error occurs, if values configured for the parameter "NvMWriteAllOrder" are not sequential.

7.2.1.2 Warning Messages

34) WRN020034: Parameter "NvMRomBlockDataAddress" in the container "<NvMBlockDescriptor>" should not be configured, since value of the parameter "NvMRomBlockNum" in the container "<NvMBlockDescriptor>" is configured as <0>.
    - This warning occurs, if the value of the parameter NvMRomBlockDataAddress is configured when the value of the parameter NvMRomBlockNum is configured as <0> in the container NvMBlockDescriptor. With <NvMBlockDescriptor> is short name of NvMBlockDescriptor
35) WRN020035: Value of the parameter "NvMRomBlockDataAddress" should be configured since the value of the parameter "NvMRomBlockNum" is other than <0> in the container "<NvMBlockDescriptor>".

- This warning occurs, if the value of the parameter NvMRomBlockDataAddress is not configured when the value of the parameter NvMRomBlockNum is configured other than <0> in the container NvMBlockDescriptor. With <NvMBlockDescriptor> is short name of NvMBlockDescriptor

36) WRN020043: Value of the parameter "NvMBlockUseCrc" should not be configured as <true/1>, when the value of the parameter "NvMBlockJobPriority" is configured as <0> in the container "<NvMBlockDescriptor>".
  - This information occurs, if the value configured for the parameter NvMBlockUseCrc is <true/1>, when the value of the parameter NvMBlockJobPriority is configured as <0> . With <NvMBlockDescriptor> is short name of NvMBlockDescriptor

7.2.1.3 Information Message

- NvM has no information message

8. SWP Error Code

8.1 SWP Error Code List

8.1.1 NVM_E_HARDWARE

Error Name:	NVM_E_HARDWARE	
Short Description:	Reading from or writing to non volatile memory failed	
Long Description:	If read job (multi job or single job read) fails either because the MemIf reports MEMIF_JOB_FAILED, MEMIF_BLOCK_INCONSISTENT or a CRC mismatch occurs or if a write/invalidate/erase job fails because the MemIf reports MEMIF_JOB_FAILED, NvM shall report NVM_E_HARDWARE to the DEM.	
Detection Criteria:	Fail	MemIf reports MEMIF_JOB_FAILED, MEMIF_BLOCK_INCONSISTENT or a CRC mismatch occurs during read / write / invalidate / erase operation.
	Pass	Read / write / invalidate / erase is successful. (MemIf does not report MEMIF_JOB_FAILED , MEMIF_BLOCK_INCONSISTENT and no CRC mismatch occurs)
Secondary Parameters:	The condition under which the FAIL and/or PASS detection is active: Every time a read / write / invalidate / erase is requested for the block NvM shall report if the condition of the block changed.	
Time Required:	Not applicable. (there is no timeout monitoring in the NvM)	
Monitor Frequency	continuous	

8.1.2 NVM_E_INTEGRITY_FAILED

Error Name:	NVM_E_INTEGRITY_FAILED	
Short Description:	Processing of the read service detects an inconsistency.	
Long Description:	If the read for a block detects that the data and/or CRC are corrupted based on the CRC check performed after the read was finished successfully (JobEndNotification from underlying memory module). This only applies for blocks configured with CRC.	
Detection Criteria:	Fail	See SWS_NvM_00864
	Pass	See SWS_NvM_00872
Secondary Parameters:	The condition under which the FAIL or PASS detection is active: CRC checking is performed each time a block with CRC is read successfully by the underlying memory module and it will indicate failure or pass.	
Time Required:	Not applicable. There is no timeout monitoring or constraint for NvM.	
Monitor Frequency	continuous	

8.1.3 NVM_E_REQ_FAILED

Error Name:	NVM_E_REQ_FAILED	
Short Description:	Processing of the read service failed at a lower layer in the MemStack architecture, including all retries.	
Long Description:	If the underlying layer reports JobErrorNotification, indicating that the request failed, either after it was accepted by the underlying memory module or because the module refused the request. This is done after all retries also failed.	
Detection Criteria:	Fail	See SWS_NvM_00865
	Pass	See: SWS_NvM_00873
Secondary Parameters:	The condition under which the FAIL or PASS detection is active: check is performed to see if the job was accepted or not and, if accepted, to see if it finished successfully or not.	
Time Required:	Not applicable. There is no timeout monitoring or constraint for NvM.	
Monitor Frequency	continuous	

8.1.4 NVM_E_WRONG_BLOCK_ID

Error Name:	NVM_E_WRONG_BLOCK_ID	
Short Description:	Static block ID check, during read, indicates failure.	
Long Description:	If the read was successfully finished by the underlying memory module but the Static ID check failed (meaning the block ID that was read is not the same as the block ID for which the read was requested).	
Detection Criteria:	Fail	See SWS_NvM_00866
	Pass	See SWS_NvM_00874
Secondary Parameters:	The condition under which the FAIL or PASS detection is active: check is performed each time the reading of a block is finished successfully by the underlying memory module, if the block is configured to have the Static ID checking performed for it.	
Time Required:	Not applicable. There is no timeout monitoring or constraint for NvM.	
Monitor Frequency	continuous	

8.1.5 NVM_E_VERIFY_FAILED

Error Name:	NVM_E_VERIFY_FAILED	
Short Description:	The write verification failed.	
Long Description:	If, after a successfully finished write, the verification for the written data fails.	
Detection Criteria:	Fail	See SWS_NvM_00867

	Pass	See SWS_NvM_00875
Secondary Parameters:	The condition under which the FAIL or PASS detection is active: a check is performed each time a block that is configured to have write verification performed on it, has a write operation successfully finished.	
Time Required:	Not applicable. There is no timeout monitoring or constraint for NvM.	
Monitor Frequency	continuous	

8.1.6 NVM_E_LOSS_OF_REDUNDANCY

Error Name:	NVM_E_LOSS_OF_REDUNDANCY	
Short Description:	A redundant block has lost the redundancy.	
Long Description:	A redundant block has the same contents written in two different block instances – hence the redundancy. If the contents are different, if the first instance becomes corrupted or if the first instance cannot be read then NvM will report this fault.	
Detection Criteria:	Fail	See SWS_NvM_00868
	Pass	See SWS_NvM_00876
Secondary Parameters:	The condition under which the FAIL or PASS detection is active: checks are performed whenever a reading is requested for a redundant block.	
Time Required:	Not applicable. There is no timeout monitoring or constraint for NvM.	
Monitor Frequency	continuous	

8.1.7 NVM_E_WRITE_PROTECTED

Error Name:	NVM_E_WRITE_PROTECTED	
Short Description:	A write is requested for a write protected block.	
Long Description:	If a block has the write protection active (either configured or set by explicit request) and a write request is made for the block, NvM will detect this and report the fault to Dem.	
Detection Criteria:	Fail	See: SWS_NvM_00870
	Pass	See: SWS_NvM_00878
Secondary Parameters:	The condition under which the FAIL or PASS detection is active: check is performed for each write request.	
Time Required:	Not applicable. There is no timeout monitoring or constraint for NvM.	
Monitor Frequency	continuous	

9. Appendix

9.1 Setting Guide by Function

9.1.1 Redundant Block Setting (2 copies)

1) After creating a block in the NvM module.
2) Set Block Management Type to NVM_BLOCK_REDUNDANT and Nv Block Num to 2.
3) Create 2 Ea / Fee Blocks.
4) Length is the same, block Number is according to the formula (n: NvMDatasetSelectionBits)
5) First Ea/Fee Block Number: 2n * NvMNvBlockBaseNumber
6) Second Ea/Fee Block Number: 2n * NvMNvBlockBaseNumber+1
7) Mapping of NvM Block Reference to the first Ea / Fee Block

9.1.2 CRC Implement

1) Set NvMBlockUseCrc and NvMCalcRamBlockCrc of Block to be checked by CRC to True. NvMBlockCrcType is set according to the design intention of App among CRC8, CRC16, and CRC32.
2) The size of Underlayer Block (Fee/Ea Block) should be basically the same as the length of NvM Block, but if CRC is added, CRC size should be added to NvM Block Length.
3) For CRC8, set the Underlayer Block size to +1, for CRC16, the Length to +2, and for CRC32, the Length to +4.
4) Immediate Block does not support CRC. (From SWS_NvM_00721) That is, if BlockJobPriority is set to 0 in the NvM Block Descriptor setting, even if CRC is set, the Generator informs the following information and does not use the CRC function.(Regardless of JobPriorization of Common Container) Value of the parameter'NvMBlockUseCrc' should not be configured as <true/1>, when the value of the parameter'NvMBlockJobPriority' is configured as <0> in the container'NvMBlockDescriptor' and the Generation Tool resets the value of the parameter'NvMBlockUseCrc' 'to <false/0>.

9.1.3 Immediate Block Setting

1) Set NvMJobPrioritization item to True in NvM Common Container
2) Set NvMBlockJobPriority item of Block to be set to Immediate as 0
3) Modify the setting value so that OverFlow does not appear in NvMSizeImmediateJobQueue

※ If Immediate Block is used, the NvM module shall use two queues, one for immediate write jobs (crash data) another for all other jobs (including immediate read/erase jobs, standard jobs). Otherwise the NvM Module shall use one queue and processes all jobs in FCFS order. NVRAM blocks with immediate priority are not expected to be configured to have a CRC.

9.1.4 ReadAll / WriteAll settings

1) Set NvMSelectBlockForReadAll / NvMSelectBlockForWriteAll of Block to be set to

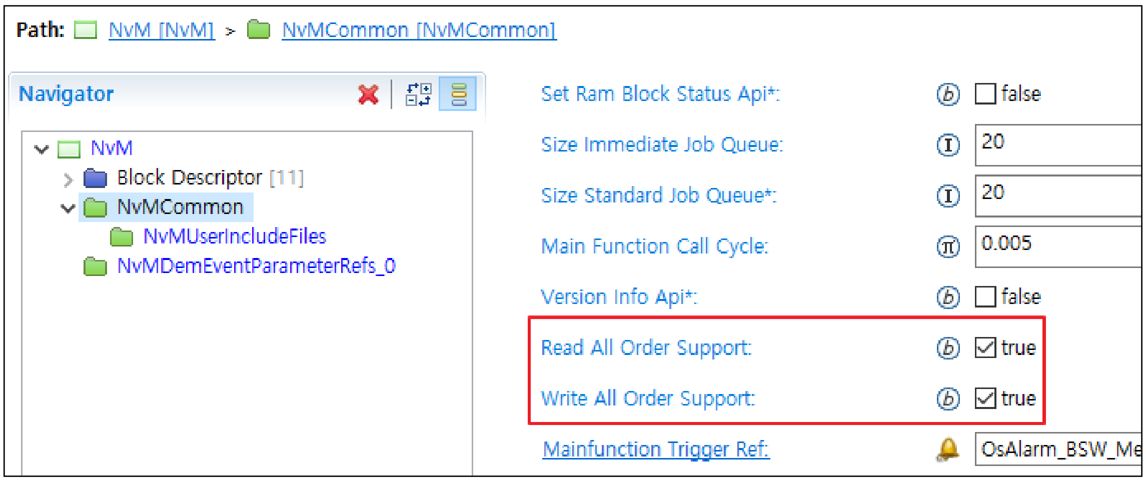
ReadAll/WriteAll to True

ReadAll: The value stored in EEPROM is read in Ram at StartUp.

WriteAll: Writes Ram value to EEPROM during ShutDown.

9.1.5 ReadAll/WriteAll Order Supporting setting

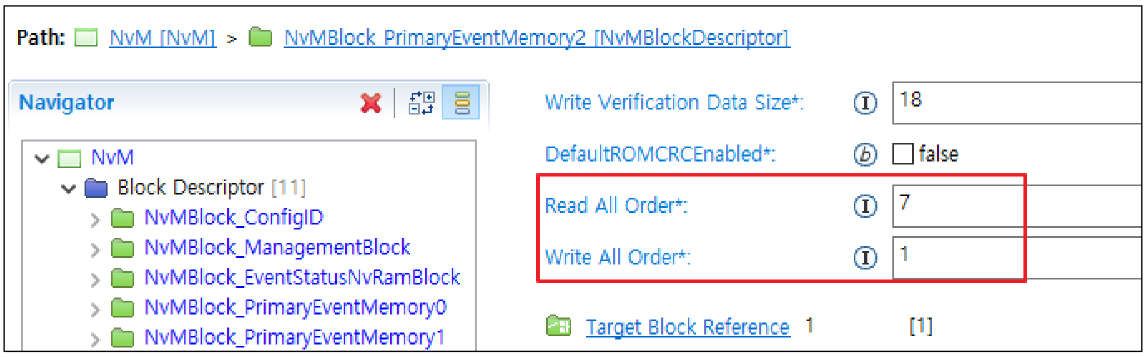
1) To set the Block processing order for the user during the ReadAll /WriteAll operation, set the NvMReadAllOrderSupport / NvMWriteAllOrderSupport checkbox to True


Path: NvM [NvM] > NvMCommon [NvMCommon]

Navigator
- NvM
  - Block Descriptor [11]
  - NvMCommon
    - NvMUserIncludeFiles
  - NvMDemEventParameterRefs_0

Set Ram Block Status Api*:	false
Size Immediate Job Queue:	20
Size Standard Job Queue*:	20
Main Function Call Cycle:	0.005
Version Info Api*:	false
Read All Order Support:	✓ true
Write All Order Support:	✓ true
Mainfunction Trigger Ref:	OsAlarm_BSW_Me

2) If NvMReadAllOrderSupport / NvMWriteAllOrderSupport is True, the Order value is set in the NvMReadAllOrder / NvMWriteAllOrder item of the blocks for which you want to set the order. However, a Block with a Block ID of "1" is a Config Block and the user cannot set the order, so Order is not set. For blocks that do not require order setting, you do not need to set Order. For blocks that require order setting, the order should start from "1" and be sequential with other orders.


Path: NvM [NvM] > NvMBlock_PrimaryEventMemory2 [NvMBlockDescriptor]

Navigator
- NvM
  - Block Descriptor [11]
    - NvMBlock_ConfigID
    - NvMBlock_ManagementBlock
    - NvMBlock_EventStatusNvRamBlock
    - NvMBlock_PrimaryEventMemory0
    - NvMBlock_PrimaryEventMemory1

Write Verification Data Size*:	18
DefaultROMCRCEnabled*:	false
Read All Order*:	7
Write All Order*:	1
Target Block Reference 1	[1]

9.2 Notes on design

9.2.1 NvM Block Identifier

The NvM Block Ids are Expected to be in a sequential order.

1) NvM Block Id 0, Block 1 is a block used by the NvM Module itself and should not be used by App or other Bsw modules.

2) Reserved NVRAM block IDs:
0 -> to derive multi block request results via NvM_GetErrorStatus

1 -> redundant NVRAM block which holds the configuration ID: NvMBlock_ConfigID
※ Application users do not need to be additionally set in the distributed project.

9.2.2 RamBlock Length

The length of RamBlock should be set equal to the length of NvBlock.

9.2.3 Immediate Block

When processing a job (i.e Write/Read) of a standard attribute block, requesting a write request of a block with an immediate attribute cancels the job being processed and executes the canceled job again after executing the immediate block write. In case of Standard Job Write being processed at the time, it may be canceled in a state where only a part is written, and Immediate Block Write is processed and then canceled Write is executed again. If the company is reset, the EEPROM data of the job being processed may remain partially written, and the error can be detected through CRC setting.

9.2.4 Whether to check return value and block status when requesting

When requesting requests such as Read/Write, the return value of the API should be checked. When the return value is E_OK, the request is successfully registered in NvM's queue, and in the case of E_NOT_OK, an error occurs and it is not registered in the queue. Most of errors are Block Pending status of Request or Queue is Full status. Therefore, the user should request the request after confirming that it is not pending by checking the status of the block before requesting. When Async Request such as Read/Write is requested, it should be confirmed that the job is completed through Polling or callback. If Reset occurs without request being completed, it cannot be guaranteed that the operation is completed.

9.2.5 ReadAll Time

In the standard platform of Hyundai AUTOEVER, after executing ReadAll function in the StartUpTwo stage, when it is finished, it moves to the StartUpThree stage and performs Can communication. Since the time taken until it depends on the number of ReadAll Blocks, if the time to Can communication is exceeded after power is applied, the number of ReadAlls should be reduced.

In addition, in the case of Internal EEPROM, StartUpTwo time in the Virgin state takes longer compared to the non-Virgin state. Is the time it takes to Init the EaBlock, and depends on the number of Ea Blocks. In the Virgin state, if the number of Fee Blocks is large, Wdg Reset may take place, and can be solved by changing the NvMMaxNumOfReadRetries setting or adjusting the Wdg Time until StartUpTwo.

9.2.6 WriteAll Time

AUTOEVER's Hyundai standard platform performs the WriteAll function after WdgM DeInit in the Shutdown Sequence phase. DeInitTimeOut can be set in WdgM setting, and Wdg Reset occurs when TimeOut occurs after WdgM DeInit. So, WriteAll must be completed in DeInitTimeOut value. Since the execution time of WriteAll depends on the type of EEPROM and the number of NvM blocks, the user should check the WriteAll time and reflect it in the DeInitTimeOut value.

9.2.7 NvM API Call-Context

You can call NvM's API through RTE from Application or directly call NvM's API from CDD. When, you shouldn't call it in the Interrupt Service Routine (ISR). This is because the execution time of the API may be flexible depending on the conditions. Also, it should not be called in Low Power Mode. This is because it is designed to operate only in High Power Mode.

9.2.8 Fee Init of Virgin Internal EEPROM

In case of Internal EEPROM, DFlash initialization time from Virgin state is longer compared to the non-Virgin state. When the Fee Init process (i.e. in factory) in the Internal EEPROM Virgin state, you should not receive an interruption (i.e reset). Therefore, it is necessary to guarantee the time for Fee Init in the initial process. The time can be known by measuring the DFlash to Virgin, and measuring from Power On to NvM_ReadAll.

9.2.9 Erase All when changing Memory Layout

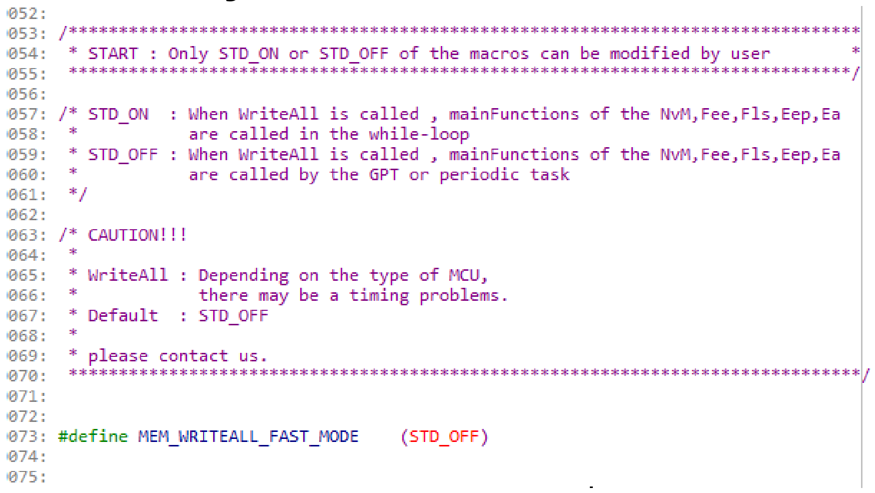
In order to prevent abnormal operation, internal/external EEPROM must be erased when the memory layout is changed (NvM Block addition/deletion/Length/CRC change, etc.).

9.2.10 Mem_Integration_User Implementation

Path: Static_Code₩Integration_Code₩integration_Mem₩usercode
File: Mem_Integration_User.h, Mem_Integration_User.c
Only the following functions can be modified by the user.

9.2.10.1 MEM_WRITEALL_FAST_MODE

File : Mem_Integration_User.h



```

/*******************************************************************************
 * START : Only STD_ON or STD_OFF of the macros can be modified by user       *
 *******************************************************************************/

/* STD_ON  : When WriteAll is called , mainFunctions of the NvM,Fee,Fls,Eep,Ea
 *           are called in the while-loop
 * STD_OFF : When WriteAll is called , mainFunctions of the NvM,Fee,Fls,Eep,Ea
 *           are called by the GPT or periodic task
 */

/* CAUTION!!!
 *
 * WriteAll : Depending on the type of MCU,
 *            there may be a timing problems.
 * Default  : STD_OFF
 *
 * please contact us.
 ********************************************************************************/


#define MEM_WRITEALL_FAST_MODE    (STD_OFF)


```

- **STD_ON:** You can reduce WriteAll execution time by calling NvM/Fee/Fls/Ea/Eep Mainfunction in a while-loop.
- **STD_OFF:** Calls NvM/Fee/Fls/Ea/Eep Mainfunction every 1ms like normal operation.

9.2.10.2 User Callbacks

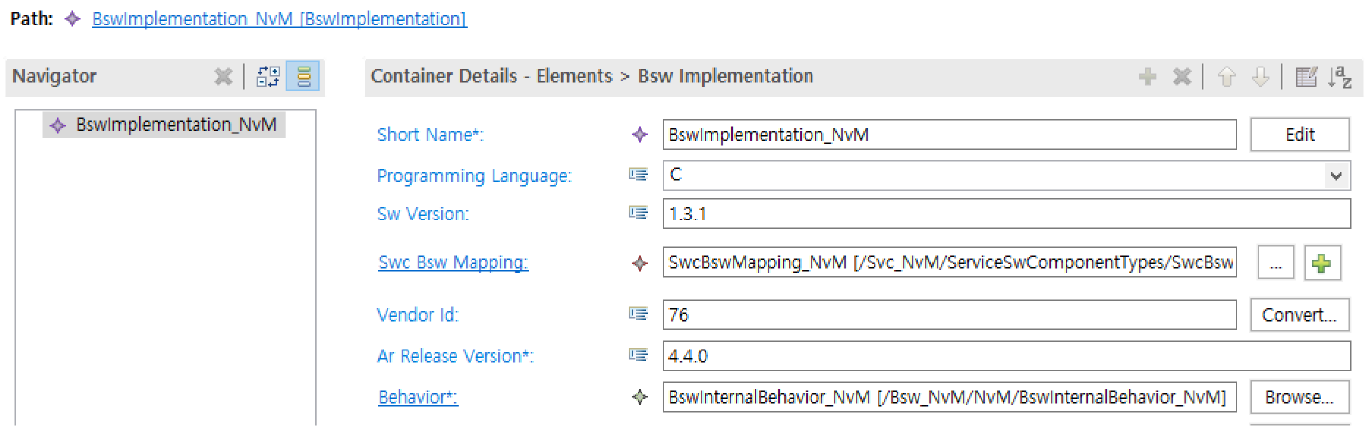
1) Mem_PostFeeInitCallback (Common)
   Callback that is called before Fee Init.

2) Mem_Cypress_IllegalStateCallback(Cypress amethyst/ artemis)
   Called when FEE initialization fails due to a change in memory layout, etc.
   There is no other recovery method other than "Flash erase all".
3) Mem_Infineon_IllegalStateCallback (Infineon Aurix)
   Called when Fee cannot operate normally. When Callback is called, Fee stops operating.
   You should respond to each case by referring to the Fee manual.

9.3 Bswmd (Bsw Module Description)

9.3.1 Bsw module version setting

When compiling each module, if version information does not match, an error is generated by Compile. At this time, the version information must be modified in the BswImplementation Container as follows of Bswmd.


Path: BswImplementation_NvM [BswImplementation]

Navigator: BswImplementation_NvM

Container Details - Elements > Bsw Implementation

Short Name*:	BswImplementation_NvM
Programming Language:	C
Sw Version:	1.3.1
Swc Bsw Mapping:	SwcBswMapping_NvM [/Svc_NvM/ServiceSwComponentTypes/SwcBsw
Vendor Id:	76
Ar Release Version*:	4.4.0
Behavior*:	BswInternalBehavior_NvM [/Bsw_NvM/NvM/BswInternalBehavior_NvM]

9.4 Exclusive Areas

9.4.1 SchM Module Apis

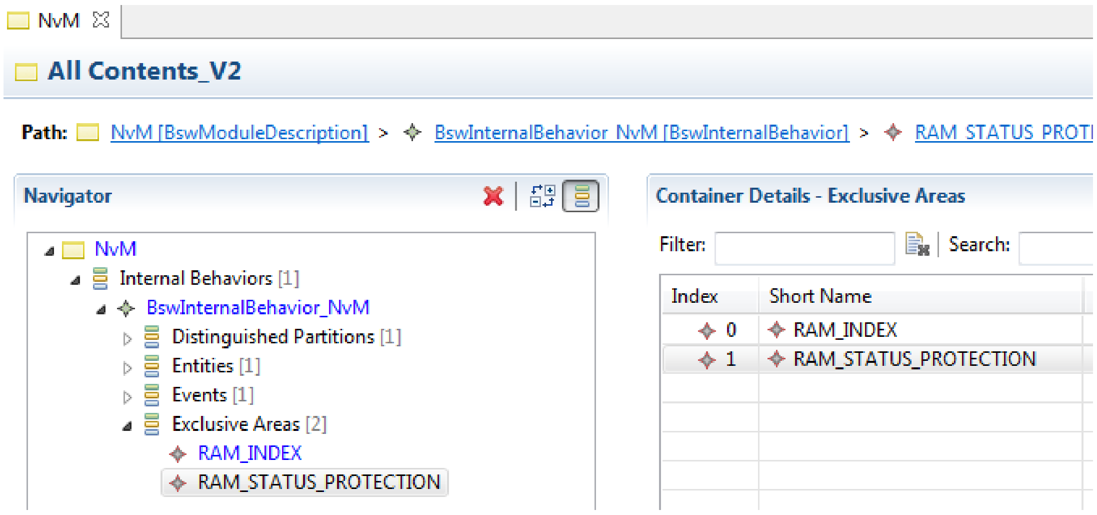
In order to provide data integrity of shared resources, Memory Module uses the scheduler service to enable and to disable data protection.
Following exclusive areas along with scheduler services are used to provide the protection:

Module	SchM APIs

NvM	SchM_Enter_NvM_RAM_INDEX()
	SchM_Exit_NvM_RAM_INDEX()
	SchM_Enter_NvM_RAM_STATUS_PROTECTION()
	SchM_Exit_NvM_RAM_STATUS_PROTECTION()

9.4.2 Setting method

Add the following to the Exclusive Areas of the module BswModuleDescription Container


NvM
All Contents_V2
Path: NvM [BswModuleDescription] > BswInternalBehavior_NvM [BswInternalBehavior] > RAM_STATUS_PROTI

Navigator
- NvM
  - Internal Behaviors [1]
    - BswInternalBehavior_NvM
      - Distinguished Partitions [1]
      - Entities [1]
      - Events [1]
      - Exclusive Areas [2]
        - RAM_INDEX
        - RAM_STATUS_PROTECTION

Container Details - Exclusive Areas
Filter: Search:

Index	Short Name
0	RAM_INDEX
1	RAM_STATUS_PROTECTION

9.5 Normal and extended runtime preparation of NVRAM blocks

This subchapter is supposed to provide a short summary of normal and extended runtime preparation
of NVRAM blocks. The detailed behavior regarding the handling of NVRAM blocks during start-up is specified in chapter 8.1.3.3.1. (AUTOSAR_SWS_NVRAMManager.pdf)

Depending on the two configuration parameters NvMDynamicConfiguration and NvMResistantToChangedSw the NVRAM Manager shall behave in different ways during start-up, i.e. while processing the request NvM_ReadAll().

If NvMDynamicConfiguration is set to FALSE, the NVRAM Manager shall ignore the stored configuration ID and continue with the normal runtime preparation of NVRAM blocks.
In this case the RAM block shall be checked for its validity. If the RAM block content is detected to be invalid the NV block shall be checked for its validity. A NV block which is detected to be valid shall be copied to its assigned RAM block. If an invalid NV Block is detected default data shall be loaded.

If NvMDynamicConfiguration is set to TRUE and a configuration ID mismatch is detected, the extended runtime preparation shall be performed for those NVRAM blocks which are configured with NvMResistantToChangedSw(FALSE). In this case default data shall be loaded independent of the validity of an assigned RAM or NV block.

That is, when DynamicConfiguraiton in the common container is true, BlockID 1 is read first in ReadAll, and ReadAll proceeds when the value and the set NvMCompiledConfigId match. If it does not match, if the setting value of NvMResistantToChangeSw that can be set for each block is false, it is treated as a failure regardless of whether or not the read was successful. (If there is Default Data, it is copied to RamBlock.) NvMResistantToChangeSw setting block is true regardless of whether NvMCompiledConfigId matches. Proceed with ReadAll. However, writing NvMCompiledConfigId in BlockID1 should be done in Application through Write Api or WriteAll.

9.6 Notification Interface with Application

9.6.1 SingleBlock Callback

The NvM module supports SingleBlock Callback for each block, and when used, is set as follows in SingleBlockCallback of Block Descriptor.

- Rte_Call_NvM_PNJF_{Block}_JobFinished
- Block = {ecuc(NvM/NvMBlockDescriptor.SHORT-NAME)}

※ To change the project with Naming rule Rte_Call_NvM_PNx_JobFinished as above, refer to **Chap 8.7**.

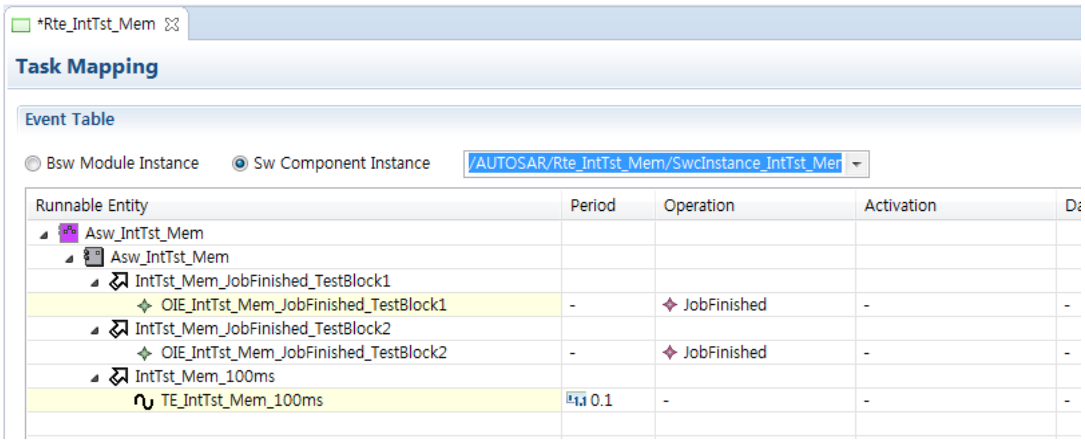
When configuring as above, the port and interface are created in the created NvM Swcd as follows.

```
PNJF_{Block}
Block = {ecuc(NvM/NvMBlockDescriptor.SHORT-NAME)}

ClientServerInterface NvMNotifyJobFinished {
    JobFinished( IN NvM_BlockRequestType BlockRequest, IN NvM_RequestResultType JobResult);
};
```

The request type (read, write, ... etc.) of the previous processed block job, and NvM_RequestResultType is the result of the requested request.
In Application, it can be used by connecting it to ApplicationSwComponent.
The callback-related runnable set in the Application Sw Component must be canBeInvokedConcurrently set to FALSE.
In order to prevent the simultaneous occurrence (canBeInvokedConcurrently), the task is activated and the task is called to callback. Therefore, one Task is required to use SingleBlockCallback, which must be directly created in the Application as follows.

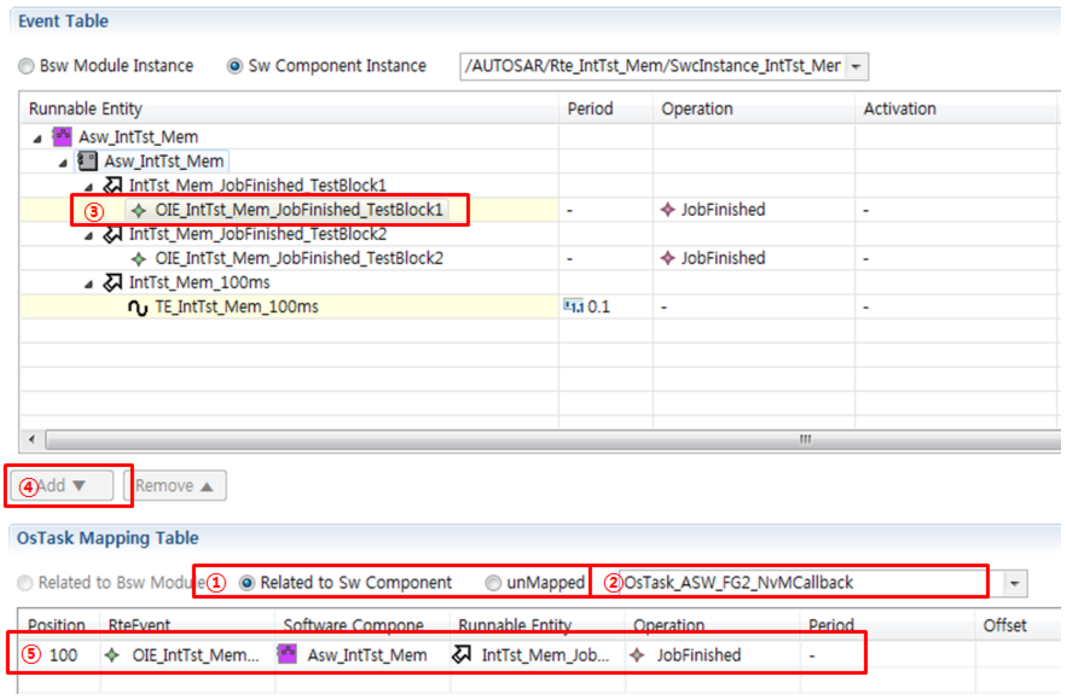
1. Create Application SW Component and register Runnable of SingleBlockCallback.
2. Task creation in OS
   ShortName: OsTask_ASW_FG2_NvMCallback
   Priority: Adjust greater than FG1 and less than FG3
   ResourceRef: FG2
3. On the RTE's Task Mapping tab, set SW Component Instance and select the configured App SW-C



Bsw Module Instance [✓ Sw Component Instance] /AUTOSAR/Rte_IntTst_Mem/SwcInstance_IntTst_Mem

Runnable Entity	Period	Operation	Activation	Da
Asw_IntTst_Mem				
Asw_IntTst_Mem				
IntTst_Mem_JobFinished_TestBlock1				
OIE_IntTst_Mem_JobFinished_TestBlock1	-	JobFinished	-	-
IntTst_Mem_JobFinished_TestBlock2				
OIE_IntTst_Mem_JobFinished_TestBlock2	-	JobFinished	-	-
IntTst_Mem_100ms				
TE_IntTst_Mem_100ms	0.1	-	-	-

4. Callback Runnable on Task Mapping



[✓ Sw Component Instance] /AUTOSAR/Rte_IntTst_Mem/SwcInstance_IntTst_Mer

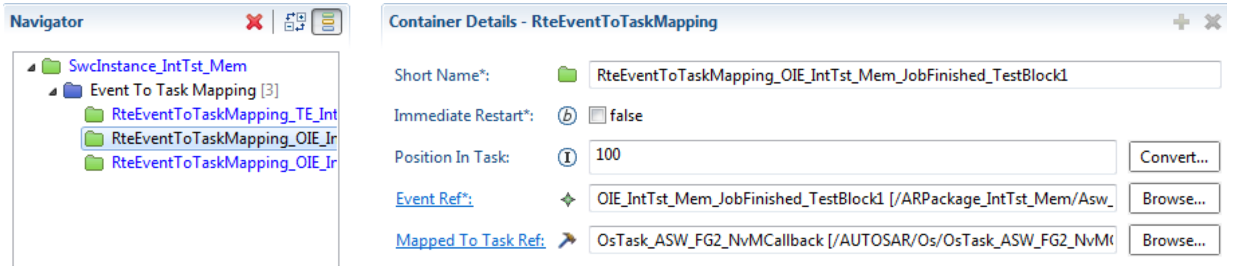
Runnable Entity	Period	Operation	Activation
Asw_IntTst_Mem			
Asw_IntTst_Mem			
IntTst_Mem_JobFinished_TestBlock1			
OIE_IntTst_Mem_JobFinished_TestBlock1	-	JobFinished	-
IntTst_Mem_JobFinished_TestBlock2			
OIE_IntTst_Mem_JobFinished_TestBlock2	-	JobFinished	-
IntTst_Mem_100ms			
TE_IntTst_Mem_100ms	0.1	-	-

[✓ Related to Sw Component] unMapped — OsTask_ASW_FG2_NvMCallback

Position	RteEvent	Software Compone	Runnable Entity	Operation	Period	Offset
100	OIE_IntTst_Mem...	Asw_IntTst_Mem	IntTst_Mem_Job...	JobFinished	-	

① When mapping for the first time, select unMapped, then select RelatedToSwComponent
② Select OsTask_ASW_FG2_NvMCallback
③ Select runnable to map
④ Add Click
⑤ Confirm creation in OsTaskMappingTable

When setting SingleBlockCallback related Event, set MappedToTaskRef as OsTask_ASW_FG2_NvMCallback as follows (RTE> Sw ComponentInstance> EventToTaskMapping).

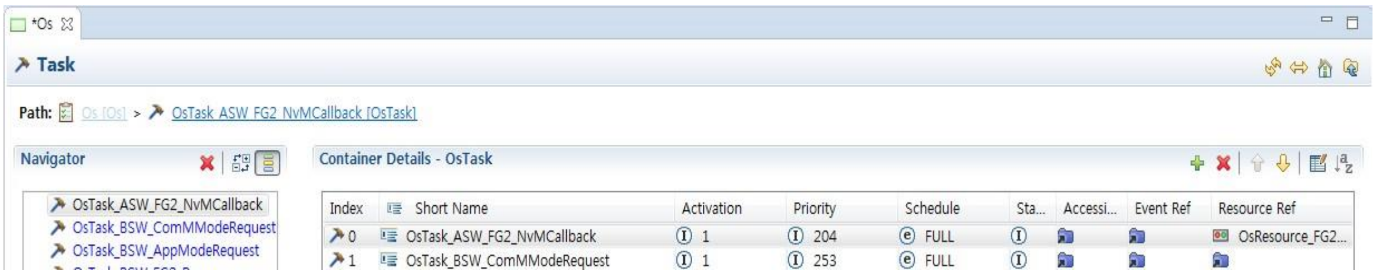


SwcInstance_IntTst_Mem
Event To Task Mapping [3]
RteEventToTaskMapping_TE_Int
RteEventToTaskMapping_OIE_Ir
RteEventToTaskMapping_OIE_Ir

Short Name*:	RteEventToTaskMapping_OIE_IntTst_Mem_JobFinished_TestBlock1
Immediate Restart*:	false
Position In Task:	100
Event Ref*:	OIE_IntTst_Mem_JobFinished_TestBlock1 [/ARPackage_IntTst_Mem/Asw_
Mapped To Task Ref:	OsTask_ASW_FG2_NvMCallback [/AUTOSAR/Os/OsTask_ASW_FG2_NvM(

※ Since SingleBlockCallback Runnable is called from the task of FG2, the execution time should be minimized.

9.6.2 InitBlock Callback


*Os

Task

Path: Os [Os] > OsTask_ASW_FG2_NvMCallback [OsTask]

Navigator

- OsTask_ASW_FG2_NvMCallback
- OsTask_BSW_ComMModeRequest
- OsTask_BSW_AppModeRequest

Container Details - OsTask

Index	Short Name	Activation	Priority	Schedule	Sta...	Accessi...	Event Ref	Resource Ref
0	OsTask_ASW_FG2_NvMCallback	1	204	FULL				OsResource_FG2...
1	OsTask_BSW_ComMModeRequest	1	253	FULL				

The NvM module supports InitBlockCallback for each block, and when used, set the following in the InitBlockCallback of Block Descriptor.

- Rte_Call_NvM_PNIB_{Block}_InitBlock
- Block = {ecuc(NvM/NvMBlockDescriptor.SHORT-NAME)}

When configuring as above, the port and interface are created in the created NvM Swcd as follows.

```
PNIB_{Block}
Block = {ecuc(NvM/NvMBlockDescriptor.SHORT-NAME)}

ClientServerInterface NvMNotifyInitBlock
    {InitBlock();
};
```

In Application, it can be used by connecting it to ApplicationSwComponent.