

SCOPE OF APPLICATION All Project/Engineering	<b>HYUNDAI</b> <b>AutoEver</b>	SHT/SHTS 1 / 53
Responsibility: Classic AUTOSAR Team	AUTOSAR NvM User Manual	DOC. NO
<h1>AUTOSAR NvM User Manual</h1>		

Document Change History				
Date (YYYY-MM-DD)	Ver.	Editor	Chap	Content (before revision-> after revision)
2021-03-10	1.0.0.0	YJ.Yun	All	Initial version
2021-07-29	1.0.1.0	JH.Lim	4.3	Update module release note
2021-10-29	1.0.2.0	JH.Lim	4.3 5.1.2	Update module release note Update configuration guide for NvMNvramDeviceId
2022-03-01	1.1.0.0	JH.Lim	4.3 4.4 7.2.1.1	Update module release note Update Limitations Add error message (ERR020052)
2022-05-26	1.1.1.0	SH.Park	4.3 5.1.1	Update module release note Update configuration guide
2022-07-06	1.1.1.1	ThuanLD5	4.3	Update module release note
2022-08-12	1.1.2.0	ThuanLD5	4.3	Update module release note
2022-11-03	1.2.0.0	ThuanLD5	4.3 5.1 5.5 7.2 9.1	Update module release note Update container Update configuration Update error messages Update function setting
2022-12-21	1.3.0.0	ThuanLD5	4.3.1 5.1.1 6.3.6	Update module release note Update container Add User Defined Functions
2023-04-28	1.3.1.0	YJ.Yun	4.3.1	Update module release note

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

<b>1. OVERVIEW .....</b>	<b>5</b>
<b>2. REFERENCE .....</b>	<b>5</b>
<b>3. AUTOSAR SYSTEM .....</b>	<b>6</b>
3.1 Overview of Software Layers .....	6
3.2 AUTOSAR Memory Stack .....	7
<b>4. PRODUCT RELEASE NOTES .....</b>	<b>9</b>
4.1 Overview.....	9
4.2 Scope of the release .....	9
4.3 Module release notes .....	9
4.3.1 Change log .....	9
4.4 Limitations .....	13
4.5 Deviation .....	13
<b>5. CONFIGURATION GUIDE .....</b>	<b>14</b>
5.1 NvM Module.....	14
5.1.1 Common Container.....	14
5.1.2 NvMBlockDescriptor Container .....	17
5.1.3 NvMDemEventParameterRefs Container.....	19
5.2 Ea Module (Internal/External Eeprom).....	19
5.2.1 EaGeneral Container .....	19
5.2.2 BlockConfiguration setting .....	20
5.3 Eep Module (Internal/External EEPROM) .....	20
5.3.1 EepGeneral Container .....	20
5.3.2 EepInitConfiguration .....	21
5.3.3 EepDemEventParameterRefs .....	21
5.3.4 EepExternalDriver.....	21
5.4 System Configuration.....	22
5.4.1 ApplicationSwComponentType setting.....	22
5.4.2 EcuComposition settings .....	23
<b>6. APPLICATION PROGRAMMING INTERFACE (API).....</b>	<b>25</b>
6.1 Type Definitions .....	25
6.1.1 NvM_ConfigType .....	25
6.1.2 NvM_MultiBlockRequestType .....	25

<b>6.2</b>	<b>Macro Constants.....</b>	<b>25</b>
<b>6.3</b>	<b>Functions .....</b>	<b>25</b>
6.3.1	Synchronous Requests.....	25
6.3.1.1	NvM_Init.....	25
6.3.1.2	NvM_SetDataIndex .....	25
6.3.1.3	NvM_GetDataIndex.....	26
6.3.1.4	NvM_SetBlockProtection .....	26
6.3.1.5	NvM_GetErrorStatus .....	27
6.3.1.6	NvM_GetVersionInfo.....	27
6.3.1.7	NvM_SetRamBlockStatus .....	27
6.3.1.8	NvM_SetBlockLockStatus.....	28
6.3.1.9	NvM_CancelJobs.....	28
6.3.2	Asynchronous Single Block Requests.....	30
6.3.2.1	NvM_ReadBlock .....	30
6.3.2.2	NvM_WriteBlock.....	30
6.3.2.3	NvM_RestoreBlockDefaults .....	30
6.3.2.4	NvM_EraseNvBlock.....	31
6.3.2.5	NvM_InvalidateNvBlock.....	31
6.3.2.6	NvM_ReadPRAMBlock .....	32
6.3.2.7	NvM_WritePRAMBlock .....	32
6.3.2.8	NvM_RestorePRAMBlockDefaults.....	32
6.3.3	Asynchronous Multi Block Requests .....	33
6.3.3.1	NvM_ReadAll.....	33
6.3.3.2	NvM_WriteAll .....	33
6.3.3.3	NvM_CancelWriteAll .....	33
6.3.3.4	NvM_ValidateAll .....	34
6.3.3.5	NvM_FirstInitAll .....	34
6.3.4	Callback Notifications.....	34
6.3.4.1	NvM_JobEndNotification .....	34
6.3.4.2	NvM_JobErrorNotification .....	35
6.3.5	Scheduled Functions .....	35
6.3.5.1	NvM_Mainfunction .....	35
<b>6.4</b>	<b>Noted .....</b>	<b>37</b>
6.4.1	In Communication with application SW-C .....	37
<b>7.</b>	<b>GENERATOR.....</b>	<b>38</b>
<b>7.1</b>	<b>Generator Option .....</b>	<b>38</b>
7.1.1	NvM .....	38
<b>7.2</b>	<b>Generator Error Message .....</b>	<b>38</b>
7.2.1	NvM .....	38
7.2.1.1	Error Messages.....	38
7.2.1.2	Warning Messages .....	42
7.2.1.3	Information Message.....	42
<b>8.</b>	<b>SWP ERROR CODE .....</b>	<b>43</b>
<b>8.1</b>	<b>SWP Error Code List.....</b>	<b>43</b>
8.1.1	NVM_E_HARDWARE.....	43
8.1.2	NVM_E_INTEGRITY_FAILED .....	43
8.1.3	NVM_E_REQ_FAILED .....	43
8.1.4	NVM_E_WRONG_BLOCK_ID .....	44
8.1.5	NVM_E_VERIFY_FAILED .....	44
8.1.6	NVM_E_LOSS_OF_REDUNDANCY .....	44

8.1.7	NVM_E_WRITE_PROTECTED .....	45
<b>9.</b>	<b>APPENDIX .....</b>	<b>46</b>
<b>9.1</b>	<b>Setting Guide by Function .....</b>	<b>46</b>
9.1.1	Redundant Block Setting (2 copies) .....	46
9.1.2	CRC Implement .....	46
9.1.3	Immediate Block Setting .....	46
9.1.4	ReadAll / WriteAll settings .....	46
<b>9.2</b>	<b>Notes on design .....</b>	<b>47</b>
9.2.1	NvM Block Identifier .....	47
9.2.2	RamBlock Length .....	47
9.2.3	Immediate Block .....	48
9.2.4	Whether to check return value and block status when requesting .....	48
9.2.5	ReadAll Time .....	48
9.2.6	WriteAll Time .....	48
9.2.7	NvM API Call-Context .....	48
9.2.8	Fee Init of Virgin Internal EEPROM .....	48
9.2.9	Erase All when changing Memory Layout .....	49
9.2.10	Mem_Integration_User Implementation .....	49
9.2.10.1	MEM_WRITEALL_FAST_MODE .....	49
9.2.10.2	User Callbacks .....	49
<b>9.3</b>	<b>Bswmd (Bsw Module Description) .....</b>	<b>49</b>
9.3.1	Bsw module version setting .....	49
<b>9.4</b>	<b>Exclusive Areas .....</b>	<b>50</b>
9.4.1	SchM Module Apis .....	50
9.4.2	Setting method .....	50
<b>9.5</b>	<b>Normal and extended runtime preparation of NVRAM blocks .....</b>	<b>51</b>
<b>9.6</b>	<b>Notification Interface with Application .....</b>	<b>51</b>
9.6.1	SingleBlock Callback .....	51
9.6.2	InitBlock Callback .....	53

## 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

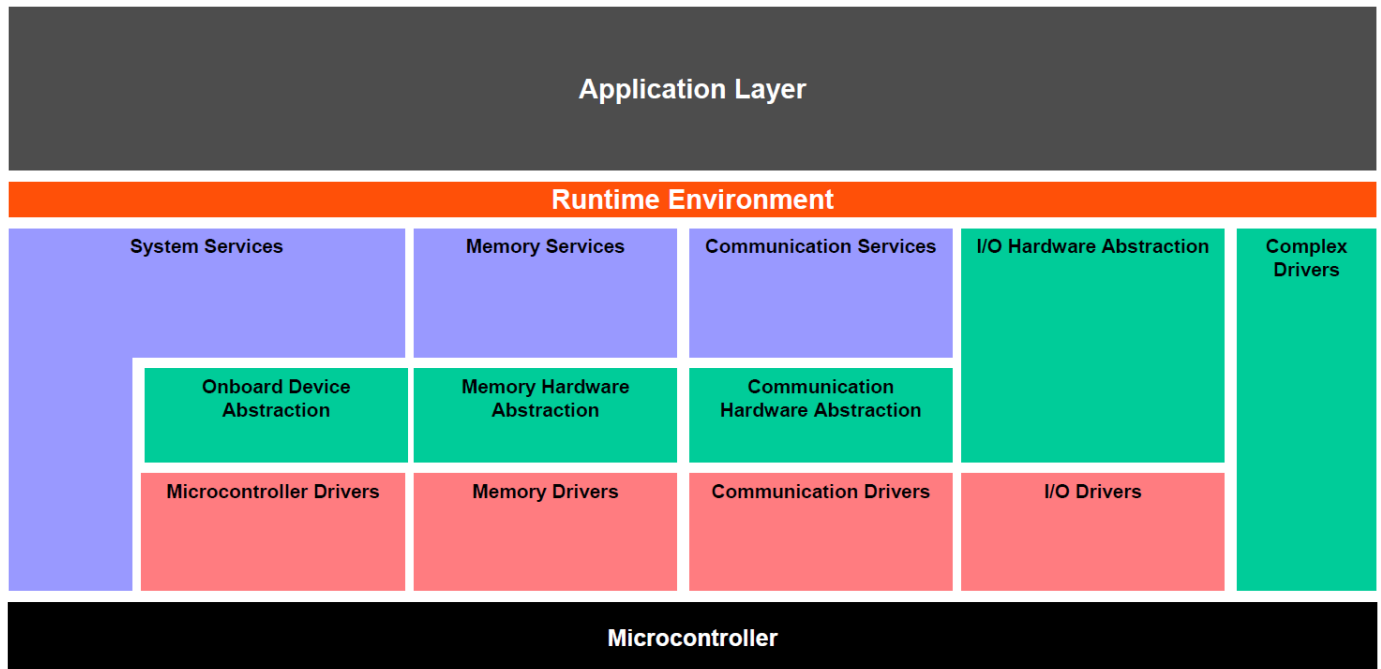
## 2. Reference

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

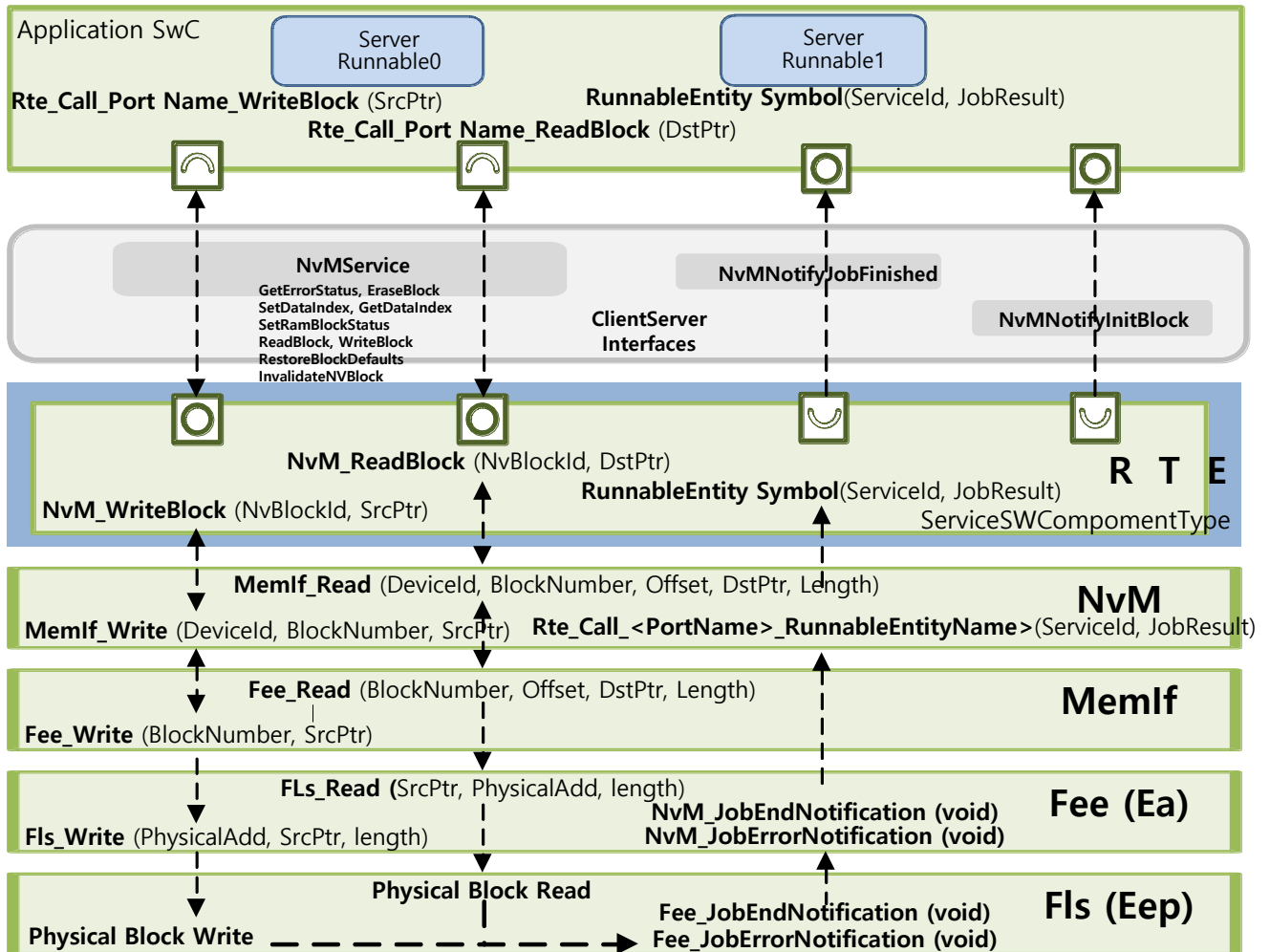
#### 3.1 Overview of Software Layers

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.



## **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. Product Release Notes

### 4.1 Overview

The purpose of this chapter is to provide release-related content for HYUNDAI AUTOEVER NvM Product and describes restrictions and specifics for NvM Software product release version.

### 4.2 Scope of the release

All contents of this document are limited to the following HYUNDAI AUTOEVER NvM modules.

Module	Autosar version	Module version
NvM	4.4.0	1.3.1

※ Module version means the Sw version of each module's BswModule Description (Bswmd) file.

### 4.3 Module release notes

#### 4.3.1 Change log

##### ➤ Version 1.3.1.0

##### - Defect

- Improve header-file-inclusion structure of NvM\_Cfg.c for Per Instance Memory functionality

<b>Cause</b>	Compilation error when using PerInstanceMemory as RamBlockAddress like below. #error Multiple Application Header Files included.  Error caused by inclusion of Rte_xxx.h twice.  Occurs when using R44 Rte and using PerInstanceMemory or ArtTypePerInstanceMemory as RamBlockAddress and configuring more than one Rte_xxx.h for NvMUserIncludeFiles
<b>Operation effect</b>	N/A
<b>Setting effect</b>	N/A
<b>ASW Action</b>	Remove the Rte_NvM.h in configuration NvMUserIncludeFiles

##### ➤ Version 1.3.0.0

##### - Feature

- Add callout function for User Watchdog Reset in initialization step

<b>Cause</b>	When External Watchdog is used, reset may occur when the memory size is large or the execution time of the initialization logic is long during memory initialization. To prevent this case, add User Watchdog Toggle callout so that the user can trigger the External Watchdog at an appropriate period.
<b>ASW Impact</b>	If the user configures NvMUserWdgToggleFunction, user functions are repeatedly performed in mem_EalnitPerform and Mem_FealnitPerform until initialization is completed.
<b>Configuration Impact</b>	NvMCommon/NvMUserWdgToggleFunction
<b>ASW Action</b>	When the user configured the NvMUserWdgToggleFunction, user should implement the user function.

※ 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.

- Task

■ Configuration Audit, Quality Gate Inspection result

<b>Cause</b>	Document and items need to be changed to meet Configuration Audit, Quality Gate Inspection result
<b>ASW Impact</b>	N/A
<b>Configuration Impact</b>	N/A
<b>ASW Action</b>	N/A

■ Final Inspection result

<b>Cause</b>	Document and items need to be changed to meet Final Inspection result
<b>ASW Impact</b>	N/A
<b>Configuration Impact</b>	N/A
<b>ASW Action</b>	N/A

➤ Version 1.2.0.0

- Feature

■ Development of block processing order setting function when requesting ReadAll / WriteAll service

<b>Cause</b>	NvM should support the user to set the order rather than the block number order.
<b>ASW Impact</b>	If the feature is enable, read/write functions are performed in order of block ID defined by user
<b>Configuration Impact</b>	NvMReadAllOrder, NvMWriteAllOrder
<b>ASW Action</b>	5.1.2 NvMBlockDescriptor Container

- Improvement

■ Change defined Device ID value

<b>Cause</b>	NvM_R44 need to maintain compability with NvM_R40 if it does not violate the AUTOSAR specification.
<b>ASW Impact</b>	MemIf_R44-1.0.3.0
<b>Configuration Impact</b>	N/A
<b>ASW Action</b>	N/A

- Improvement

■ Update InputFileList Guideline in the UM/IM

<b>Cause</b>	Missing configuration in UM/IM, if inputfilelist is not configured, unexpected error is occurred.
<b>ASW Impact</b>	N/A
<b>Configuration Impact</b>	N/A
<b>ASW Action</b>	5.5 SCons.arxml Configuration

➤ Version 1.1.2.0

- Improvement

■ Fix UNECE security coding rule violations

<b>Cause</b>	Code need to be changed to meet UNECE security coding rules
<b>ASW Impact</b>	N/A
<b>Configuration Impact</b>	N/A

<b>ASW Action</b>	N/A
-------------------	-----

➤ Version 1.1.1.1

- Improvement

■ Module release artifacts improvements

<b>Cause</b>	Artifacts of module need to be changed and improved to meet Autoever requirements about copyright contents and folder structure
<b>ASW Impact</b>	N/A
<b>Configuration Impact</b>	N/A
<b>ASW Action</b>	N/A

➤ Version 1.1.1.0

- Defect

■ Correct URI of NvMMMainfunctionTriggerRef

<b>Cause</b>	URI of NvMMMainfunctionTriggerRef in NvM PDF file is wrong so parameter NvMMMainfunctionTriggerRef in NvM can not refer to OsAlarm
<b>ASW Impact</b>	Ea_R44-1.1.0.0, Eep_R44-1.1.0.0, integration_Mem-1.6.0.0
<b>Configuration Impact</b>	/NvMCommon/NvMMMainfunctionTriggerRef
<b>ASW Action</b>	N/A

- Improvement

■ Update include section in NvM.h for supporting TC3xx MCAL ASR440

<b>Cause</b>	The files referenced when creating the "Fee_Cfg.h" file has been changed in TC3xx MCAL ASR440.
<b>ASW Impact</b>	N/A
<b>Configuration Impact</b>	N/A
<b>ASW Action</b>	N/A

- Improvement

■ Change configuration guide

<b>Cause</b>	NvMVersionInfoApi parameter is changeable value but in UM it is fixed value
<b>ASW Impact</b>	N/A
<b>Configuration Impact</b>	/NvMCommon/NvMVersionInfoApi
<b>ASW Action</b>	N/A

➤ Version 1.1.0.0

- Feature

■ Support extended task

<b>Cause</b>	NvM module uses background task to handle mainfunction, so it need to be called many time to complete job.
<b>ASW Impact</b>	Ea_R44-1.1.0.0, Eep_R44-1.1.0.0, integration_Mem-1.6.0.0
<b>Configuration Impact</b>	N/A
<b>ASW Action</b>	N/A

➤ Version 1.0.2.0

- Improvement

■ Solving defect detection in Configuration Guide

<b>Cause</b>	Configuration guide for NvMNvramDeviceId parameter in User Manual difference with Integration Manual guide
--------------	--

<b>ASW Impact</b>	N/A
<b>Configuration Impact</b>	N/A
<b>ASW Action</b>	N/A

- Improvement

■ Solving compiler warning

<b>Cause</b>	Some compilers raise warning message when compile NvM module
<b>ASW Impact</b>	N/A
<b>Configuration Impact</b>	N/A
<b>ASW Action</b>	N/A

➤ Version 1.0.1.0

- Feature

■ Solving generator validation by setting the Fee block Reference

<b>Cause</b>	NvM Generator can not find the Fee reference block
<b>ASW Impact</b>	N/A
<b>Configuration Impact</b>	N/A
<b>ASW Action</b>	N/A

- Improvement

■ Improvement of Static validation result using Polyspace

<b>Cause</b>	There are several polyspace violations in NvM
<b>ASW Impact</b>	N/A
<b>Configuration Impact</b>	N/A
<b>ASW Action</b>	N/A

➤ Version 1.0.0.0

- Feature

■ Initial version

<b>Cause</b>	Initial version
<b>ASW Impact</b>	N/A
<b>Configuration Impact</b>	N/A
<b>ASW Action</b>	N/A

## 4.4 Limitations

- User do not take care about Bswmd\_NvM.arxml.

## 4.5 Deviation

- When creating Swcd, PortName is created according to 8.2.3 chapter in AUTOSAR\_SWS\_NVRAMManager.pdf.

## 5. Configuration Guide

### 5.1 NvM Module

#### 5.1.1 Common Container

Refer to the following settings.

Parameter Name	Value	Category
NvMApiConfigClass	NVM_API_CONFIG_CLASS_3	F
NvMBswMMultiBlockJobStatus Information	True	F
NvMCompiledConfigId <sup>5)</sup>	51	C
NvMCrcNumOfBytes	4	F
NvMDatasetSelectionBits <sup>1)</sup>	User Defined (From SRS)	C
NvMDevErrorDetect	True	F
NvMDrvModeSwitch	true	C
NvMDynamicConfiguration <sup>5)</sup>	User Defined	C
NvMJobPrioritization <sup>2)</sup>	User Defined (From SRS)	F
NvMMultiBlockCallback <sup>6)</sup>	User Defined	C
NvMPollingMode	False	C
NvMRepeatMirrorOperations	0	F
NvMSetRamBlockStatusApi	False	F
NvMSizeImmediateJobQueue <sup>3)</sup>	User Defined	C
NvMSizeStandardJobQueue <sup>3)</sup>	User Defined	C
NvMVersionInfoApi	User Defined	C
NvMMainfunctionTriggerRef <sup>8)</sup>	Integrator Defined	F
NvMUserJobFunction <sup>7)</sup>	Integrator Defined	F
NvMMainFunctionPeriod	Integrator Defined	F
NvMUserIncludeFiles <sup>4)</sup> (AUTOEVER specific)	User Defined	C
NvMReadAllOrderSupport <sup>9)</sup>	User Defined	C
NvMWriteAllOrderSupport <sup>9)</sup>	User Defined	C
NvMUserWdgToggleFunction <sup>10)</sup>	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 2<sup>n</sup> 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<sup>n</sup> \* 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, standard jobs). Otherwise the NvM Module shall use one queue and processes all jobs in FCFS order.

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\_EalnitPerform and Mem\_FeelnitPerform 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.



### 5.1.2 NvMBlockDescriptor Container

Refer to the following settings.

Parameter Name	Value	Category
NvMBlockCrcType <sup>1)</sup>	User Defined	C
NvMBlockHeaderInclude	User Defined	C
NvMBlockJobPriority <sup>2)</sup>	User Defined	C
NvMBlockManagementType <sup>3)</sup>	User Defined	C
NvMBlockUseAutoValidation	User Defined	C
NvMBlockUseCrc <sup>1)</sup>	User Defined	C
NvMBlockUseCRCCompMechanism	User Defined	C
NvMBlockUseSetRamBlockStatus	User Defined	C
NvMBlockUseSyncMechanism	User Defined	C
NvMBlockWriteProt	False	F
NvMBswMBlockStatusInformation	False	F
NvMCalcRamBlockCrc <sup>1)</sup>	User Defined	C
NvMInitBlockCallback <sup>14)</sup>	User Defined	C
NvMMaxNumOfReadRetries	1	C
NvMMaxNumOfWriteRetries	1	C
NvMNvBlockBaseNumber <sup>4)</sup>	User Defined	C
NvMNvBlockLength <sup>5)</sup>	User Defined	C
NvMNvBlockNum <sup>6)</sup>	User Defined	C
NvMNvramBlockIdentifier <sup>7)</sup>	User Defined	C
NvMNvramDeviceId <sup>8)</sup>	User Defined	C
NvMRamBlockDataAddress <sup>9)</sup>	User Defined	C
NvMReadRamBlockFromNvCallback <sup>17)</sup>	User Defined	C
NvMResistantToChangedSw <sup>15)</sup>	User Defined	C
NvMRomBlockDataAddress	-	C
NvMRomBlockNum <sup>16)</sup>	0	C
NvMSelectBlockForFirstInitAll	User Defined	C
NvMSelectBlockForReadAll <sup>10)</sup>	User Defined	C
NvMSelectBlockForWriteAll <sup>10)</sup>	User Defined	C
NvMSingleBlockCallback <sup>11)</sup>	User Defined	C
NvMStaticBlockIDCheck	False	F
NvMWriteBlockOnce	False	F
NvMWriteRamBlockToNvCallback <sup>18)</sup>	User Defined	C
NvMWriteVerification	False	C
NvMWriteVerificationDataSize	1	C
NvMDefaultRomCRCEnabled (AUTOEVER specific)	false	F
NvMTargetBlockReference <sup>12)</sup>	User Defined	C
NvMNameOfFeeBlock/ NvMNameOfEaBlock <sup>13)</sup>	User Defined	C
NvMReadAllOrder <sup>19)</sup>	User Defined	C
NvMWriteAllOrder <sup>19)</sup>	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) NvMNvramDeviceld
  - 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 ShutDown.
  - When using WriteAll, WriteAll Time should be measured and reflected in the time from WdgDelnit 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.

### 5.1.3 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

## 5.2 Ea Module (Internal/External Eeprom)

Ea is a ECU Abstraction module, and refer to the Ea specification for preference.

### 5.2.1 EaGeneral Container

Refer to the following settings.

Parameter Name	Value	Category
EaDevErrorDetect	True	C

EaMainFunctionPeriod	0.005	F
EaNvmJobEndNotification	NvM_JobEndNotification	F
EaNvmJobErrorNotification	NvM_JobErrorNotification	F
EaPollingMode	False	N
EaSetModeSupported	True	N
EaVersionInfoApi	True	C
<sup>1)</sup> EaVirtualPageSize	User Defined	F

1) EaVirtualPageSize:

Mcu dependent item (refer to Ea User/Integration Manual of MCU manufacturer)

### 5.2.2 BlockConfiguration setting

Ea block should be added as many as NvBlock number (NvMNvBlockNum setting value) of NvMBlock. For example, if the NvMNvBlockNum value is 2 (Redundant Block), two Ea Blocks must be created. If it is a Ea Block connected to the same NvMBlock, the rest of the setting value must be the same except for EaBlockNumber, and the EaBlockNumber of the second block is +1 (Index) to the EaBlockNumber of the first Block.

Refer to the following settings.

Parameter Name	Value	Category
EaBlockNumber <sup>1)</sup>	User Defined	C
EaBlockSize <sup>2)</sup>	User Defined	C
EaImmediateData	False	F
EaNumberOfWriteCycles	100000	F
EaDeviceIndex	EepGeneral	F

1) BlockNumber

Basic formula:  $2^n \times \text{NvMNvBlockBaseNumber} + \text{Index}$  (n: NvMDatasetSelectionBits)

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) BlockSize

Same as NvMNvBlockLength connected

## 5.3 Eep Module (Internal/External EEPROM)

Eep is a Mcal module for internal Eeprom or ECU Abstraction module for External Eeprom, and for configuration, the Eep User Manual / Integration Manual of the MCU manufacturer is given priority.

### 5.3.1 EepGeneral Container

Refer to the following settings.

Parameter Name	Value	Category
EepDevErrorDetect	User Defined	C

EepDriverIndex	0	F
EepSetModeApi	True	F
EepUseInterrupts	False	N
EepVersionInfoApi	False	C
EepWriteCycleReduction	False	F
EepEcucPartitionRef	-	C

### 5.3.2 EepInitConfiguration

Refer to the following settings.

Parameter Name	Value	Category
EepBaseAddress	User Defined	F
EepDefaultMode	MEMIF_MODE_SLOW	F
EepFastReadBlockSize	User Defined	F
EepFastWriteBlockSize	User Defined	F
EepJobEndNotification	Ea_JobEndNotification	F
EepJobErrorNotification	Ea_JobErrorNotification	F
EepNormalReadBlockSize	User Defined	F
EepNormalWriteBlockSize	User Defined	F
EepSize	User Defined	F
EepJobCallCycle	User Defined	F

### 5.3.3 EepDemEventParameterRefs

Refer to the following settings.

Parameter Name	Value	Category
EEP_E_COMPARE_FAILED	EEP_E_COMPARE_FAILED	F
EEP_E_ERASE_FAILED	EEP_E_ERASE_FAILED	F
EEP_E_READ_FAILED	EEP_E_READ_FAILED	F
EEP_E_WRITE_FAILED	EEP_E_WRITE_FAILED	F

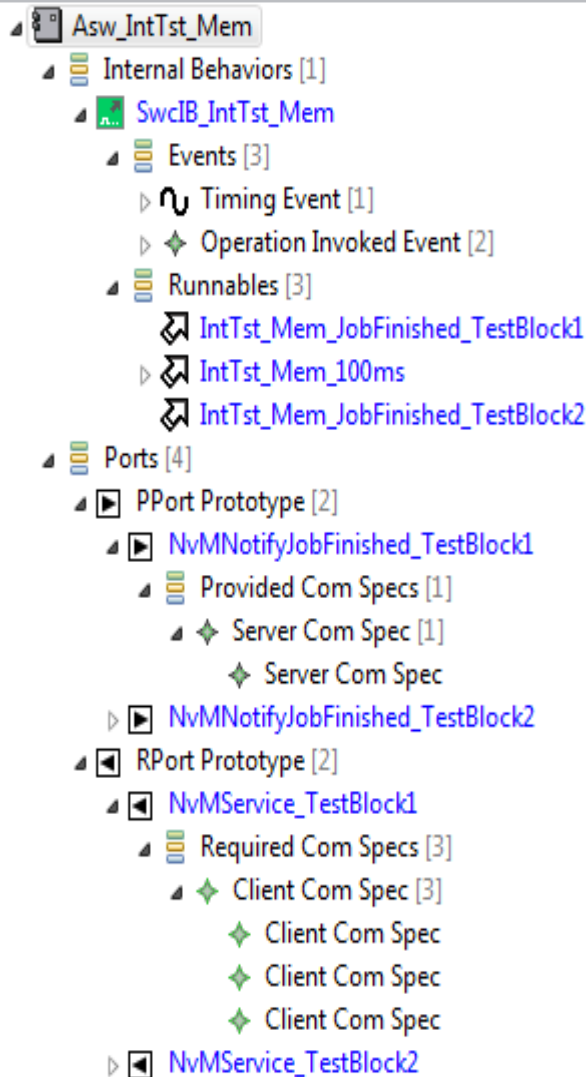
### 5.3.4 EepExternalDriver

Refer to the following settings.

Parameter Name	Value	Category
EepSpiReference	User Defined	F

## 5.4 System Configuration

### 5.4.1 ApplicationSwComponentType setting



☞ 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



### 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

**NvM**

**All Contents**

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

**Navigator**

- HwResource
- IoHwAb
- ComXf
- E2EXf
- Fee
- Fls
- EcuM
- MemIf
- NvM**
- Os
- Os\_Common
- PduR
- WdgIf
- WdgM
- Rte
- Swd\_Bsw\_Dem
- Swd\_Bsw\_Nvm
- Swd\_Bsw\_Csm
- Swd\_Bsw\_EcuM
- Swd\_Bsw\_IoHwAb
- Swd\_Bsw\_Det

**Container Details - Module**

Short Name*	NvM	Edit
Name*	NvM	[icon] unset
Input Files List*	imd, NvM_ECU_Configuration_PDF, Ecud_Os, AUTRON_AUTOSAR_Os_ECU_Configuration_PDF_TC36xA	Add... [icon] unset
Expected Files List*	NvM_Cfg.h, NvM_Cfg.c, NvM_Dem_Cfg.h	Add... [icon] unset
IsMCAL*	<input type="checkbox"/> false	[icon] unset
Used*	<input checked="" type="checkbox"/> true	[icon] unset
AUTOSAR44:	<input checked="" type="checkbox"/> true	[icon] unset

**To Be Configured:**

Bsw Defines:		Add...	[icon] unset
Bsw Include Path:		Add...	[icon] unset
Bsw Flags:		Add...	[icon] unset
Mcal Define Target:			[icon] unset



## 6. Application Programming Interface (API)

### 6.1 Type Definitions

#### 6.1.1 NvM\_ConfigType

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

#### 6.1.2 NvM\_MultiBlockRequestType

<b>Name:</b>	NvM_MultiBlockRequestType		
<b>Type:</b>	Enumeration		
<b>Range:</b>	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
<b>Description:</b>	Identifies the type of request performed on multi block when signaled via the callback function or when reporting to BswM		
<b>Available via:</b>	NvM.h		

### 6.2 Macro Constants

None

### 6.3 Functions

#### 6.3.1 Synchronous Requests

##### 6.3.1.1 NvM\_Init

<b>Service name:</b>	NvM_Init		
<b>Syntax:</b>	void NvM_Init( const NvM_ConfigType* ConfigPtr )		
<b>Service ID[hex]:</b>	0x00		
<b>Sync/Async:</b>	Synchronous		
<b>Reentrancy:</b>	Non Reentrant		
<b>Parameters (in):</b>	ConfigPtr	Pointer to the selected configuration set.	
<b>Parameters (inout):</b>	None		
<b>Parameters (out):</b>	None		
<b>Return value:</b>	None		
<b>Description:</b>	Service for resetting all internal variables.		
<b>Available via:</b>	NvM.h		

##### 6.3.1.2 NvM\_SetDataIndex

<b>Service name:</b>	NvM_SetDataIndex		
----------------------	------------------	--	--

<b>Syntax:</b>	Std_ReturnType NvM_SetDataIndex( NvM_BlockIdType BlockId, uint8 DataIndex )	
<b>Service ID[hex]:</b>	0x01	
<b>Sync/Async:</b>	Synchronous	
<b>Reentrancy:</b>	Reentrant	
<b>Parameters (in):</b>	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.
<b>Parameters (inout):</b>	None	
<b>Parameters (out):</b>	None	
<b>Return value:</b>	Std_ReturnType	E_OK: The index position was set successfully. E_NOT_OK: An error occurred.
<b>Description:</b>	Service for setting the DataIndex of a dataset NVRAM block.	

#### 6.3.1.3 NvM\_GetDataIndex

<b>Service name:</b>	NvM_GetDataIndex	
<b>Syntax:</b>	Std_ReturnType NvM_GetDataIndex( NvM_BlockIdType BlockId, uint8* DataIndexPtr )	
<b>Service ID[hex]:</b>	0x02	
<b>Sync/Async:</b>	Synchronous	
<b>Reentrancy:</b>	Reentrant	
<b>Parameters (in):</b>	BlockId	The block identifier uniquely identifies one NVRAM block descriptor. A NVRAM block descriptor contains all needed information about a single NVRAM block.
<b>Parameters (inout):</b>	None	
<b>Parameters (out):</b>	DataIndexPtr	Pointer to where to store the current dataset index (0..255)
<b>Return value:</b>	Std_ReturnType	E_OK: The index position has been retrieved successfully. E_NOT_OK: An error occurred.
<b>Description:</b>	Service for getting the currently set DataIndex of a dataset NVRAM block	

#### 6.3.1.4 NvM\_SetBlockProtection

<b>Service name:</b>	NvM_SetBlockProtection	
<b>Syntax:</b>	Std_ReturnType NvM_SetBlockProtection( NvM_BlockIdType BlockId, boolean ProtectionEnabled )	
<b>Service ID[hex]:</b>	0x03	
<b>Sync/Async:</b>	Synchronous	
<b>Reentrancy:</b>	Reentrant	
<b>Parameters (in):</b>	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

<b>Parameters (inout):</b>	None	
<b>Parameters (out):</b>	None	
<b>Return value:</b>	Std_ReturnType	E_OK: The block was enabled/disabled as requested E_NOT_OK: An error occurred.
<b>Description:</b>	Service for setting/resetting the write protection for a NV block.	

#### 6.3.1.5 NvM\_GetErrorStatus

<b>Service name:</b>	NvM_GetErrorStatus	
<b>Syntax:</b>	Std_ReturnType NvM_GetErrorStatus( NvM_BlockIdType BlockId, NvM_RequestResultType* RequestResultPtr )	
<b>Service ID[hex]:</b>	0x04	
<b>Sync/Async:</b>	Synchronous	
<b>Reentrancy:</b>	Reentrant	
<b>Parameters (in):</b>	BlockId	The block identifier uniquely identifies one NVRAM block descriptor. A NVRAM block descriptor contains all needed information about a single NVRAM block.
<b>Parameters (inout):</b>	None	
<b>Parameters (out):</b>	RequestResultPtr	Pointer to where to store the request result. See NvM_RequestResultType .
<b>Return value:</b>	Std_ReturnType	E_OK: The block dependent error/status information was read successfully. E_NOT_OK: An error occurred.
<b>Description:</b>	Service to read the block dependent error/status information.	
<b>Available via:</b>	NvM.h	

#### 6.3.1.6 NvM\_GetVersionInfo

<b>Service name:</b>	NvM_GetVersionInfo	
<b>Syntax:</b>	void NvM_GetVersionInfo( Std_VersionInfoType* versioninfo )	
<b>Service ID[hex]:</b>	0x0f	
<b>Sync/Async:</b>	Synchronous	
<b>Reentrancy:</b>	Reentrant	
<b>Parameters (in):</b>	None	
<b>Parameters (inout):</b>	None	
<b>Parameters (out):</b>	versioninfo	Pointer to where to store the version information of this module.
<b>Return value:</b>	None	
<b>Description:</b>	Service to get the version information of the NvM module.	
<b>Available via:</b>	NvM.h	

#### 6.3.1.7 NvM\_SetRamBlockStatus

<b>Service name:</b>	NvM_SetRamBlockStatus	
<b>Syntax:</b>	Std_ReturnType NvM_SetRamBlockStatus( NvM_BlockIdType BlockId, boolean BlockChanged )	
<b>Service ID[hex]:</b>	0x05	
<b>Sync/Async:</b>	Synchronous	

<b>Reentrancy:</b>	Reentrant	
<b>Parameters (in):</b>	BlockId	The block identifier uniquely identifies one NVRAM block descriptor. A NVRAM block descriptor contains all needed information about a single NVRAM block.
	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.
<b>Parameters (inout):</b>	None	
<b>Parameters (out):</b>	None	
<b>Return value:</b>	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.
<b>Description:</b>	Service for setting the RAM block status of a permanent RAM block or the status of the explicit synchronization of a NVRAM block.	
<b>Available via:</b>	NvM.h	

#### 6.3.1.8 NvM\_SetBlockLockStatus

<b>Service name:</b>	NvM_SetBlockLockStatus	
<b>Syntax:</b>	void NvM_SetBlockLockStatus( NvM_BlockIdType BlockId, boolean BlockLocked )	
<b>Service ID[hex]:</b>	0x13	
<b>Sync/Async:</b>	Synchronous	
<b>Reentrancy:</b>	Reentrant	
<b>Parameters (in):</b>	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
<b>Parameters (inout):</b>	None	
<b>Parameters (out):</b>	None	
<b>Return value:</b>	None	
<b>Description:</b>	Service for setting the lock status of a permanent RAM block or of the explicit synchronization of a NVRAM block.	
<b>Available via:</b>	NvM.h	

#### 6.3.1.9 NvM\_CancelJobs

<b>Service name:</b>	NvM_CancelJobs	
<b>Syntax:</b>	Std_ReturnType NvM_CancelJobs( NvM_BlockIdType BlockId )	
<b>Service ID[hex]:</b>	0x10	
<b>Sync/Async:</b>	Synchronous	
<b>Reentrancy:</b>	Reentrant	
<b>Parameters (in):</b>	BlockId	The block identifier uniquely identifies one NVRAM block descriptor. A NVRAM block descriptor contains all needed information about a single NVRAM block.

<b>Parameters (inout):</b>	None
<b>Parameters (out):</b>	None
<b>Return value:</b>	Std_ReturnType E_OK: The job was successfully removed from queue. E_NOT_OK: The job could not be found in the queue.
<b>Description:</b>	Service to cancel all jobs pending for a NV block.
<b>Available via:</b>	NvM.h

### 6.3.2 Asynchronous Single Block Requests

#### 6.3.2.1 NvM\_ReadBlock

<b>Service name:</b>	NvM_ReadBlock	
<b>Syntax:</b>	Std_ReturnType NvM_ReadBlock( NvM_BlockIdType BlockId, void* NvM_DstPtr )	
<b>Service ID[hex]:</b>	0x06	
<b>Sync/Async:</b>	Asynchronous	
<b>Reentrancy:</b>	Reentrant	
<b>Parameters (in):</b>	BlockId	The block identifier uniquely identifies one NVRAM block descriptor. A NVRAM block descriptor contains all needed information about a single NVRAM block.
<b>Parameters (inout):</b>	None	
<b>Parameters (out):</b>	NvM_DstPtr	Pointer to the RAM data block.
<b>Return value:</b>	Std_ReturnType	E_OK: request has been accepted E_NOT_OK: request has not been accepted
<b>Description:</b>	Service to copy the data of the NV block to its corresponding RAM block.	
<b>Available via:</b>	NvM.h	

#### 6.3.2.2 NvM\_WriteBlock

<b>Service name:</b>	NvM_WriteBlock	
<b>Syntax:</b>	Std_ReturnType NvM_WriteBlock( NvM_BlockIdType BlockId, const void* NvM_SrcPtr )	
<b>Service ID[hex]:</b>	0x07	
<b>Sync/Async:</b>	Asynchronous	
<b>Reentrancy:</b>	Reentrant	
<b>Parameters (in):</b>	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.
<b>Parameters (inout):</b>	None	
<b>Parameters (out):</b>	None	
<b>Return value:</b>	Std_ReturnType	E_OK: request has been accepted E_NOT_OK: request has not been accepted
<b>Description:</b>	Service to copy the data of the RAM block to its corresponding NV block.	
<b>Available via:</b>	NvM.h	

#### 6.3.2.3 NvM\_RestoreBlockDefaults

<b>Service name:</b>	NvM_RestoreBlockDefaults	
<b>Syntax:</b>	Std_ReturnType NvM_RestoreBlockDefaults( NvM_BlockIdType BlockId, void* NvM_DestPtr )	
<b>Service ID[hex]:</b>	0x08	
<b>Sync/Async:</b>	Asynchronous	
<b>Reentrancy:</b>	Non Reentrant	

<b>Parameters (in):</b>	BlockId	The block identifier uniquely identifies one NVRAM block descriptor. A NVRAM block descriptor contains all needed information about a single NVRAM block.
<b>Parameters (inout):</b>	None	
<b>Parameters (out):</b>	NvM_DestPtr	Pointer to the RAM data block.
<b>Return value:</b>	Std_ReturnType	E_OK: request has been accepted E_NOT_OK: request has not been accepted
<b>Description:</b>	Service to restore the default data to its corresponding RAM block.	
<b>Available via:</b>	NvM.h	

#### 6.3.2.4 NvM\_EraseNvBlock

<b>Service name:</b>	NvM_EraseNvBlock	
<b>Syntax:</b>	Std_ReturnType NvM_EraseNvBlock( NvM_BlockIdType BlockId )	
<b>Service ID[hex]:</b>	0x09	
<b>Sync/Async:</b>	Asynchronous	
<b>Reentrancy:</b>	Reentrant	
<b>Parameters (in):</b>	BlockId	The block identifier uniquely identifies one NVRAM block descriptor. A NVRAM block descriptor contains all needed information about a single NVRAM block.
<b>Parameters (inout):</b>	None	
<b>Parameters (out):</b>	None	
<b>Return value:</b>	Std_ReturnType	E_OK: request has been accepted E_NOT_OK: request has not been accepted
<b>Description:</b>	Service to erase a NV block.	
<b>Available via:</b>	NvM.h	

#### 6.3.2.5 NvM\_InvalidateNvBlock

<b>Service name:</b>	NvM_InvalidateNvBlock	
<b>Syntax:</b>	Std_ReturnType NvM_InvalidateNvBlock( NvM_BlockIdType BlockId )	
<b>Service ID[hex]:</b>	0x0b	
<b>Sync/Async:</b>	Asynchronous	
<b>Reentrancy:</b>	Reentrant	
<b>Parameters (in):</b>	BlockId	The block identifier uniquely identifies one NVRAM block descriptor. A NVRAM block descriptor contains all needed information about a single NVRAM block.
<b>Parameters (inout):</b>	None	
<b>Parameters (out):</b>	None	
<b>Return value:</b>	Std_ReturnType	E_OK: request has been accepted E_NOT_OK: request has not been accepted
<b>Description:</b>	Service to invalidate a NV block.	
<b>Available via:</b>	NvM.h	

### 6.3.2.6 NvM\_ReadPRAMBlock

<b>Service name:</b>	NvM_ReadPRAMBlock	
<b>Syntax:</b>	Std_ReturnType NvM_ReadPRAMBlock( NvM_BlockIdType BlockId )	
<b>Service ID[hex]:</b>	0x16	
<b>Sync/Async:</b>	Asynchronous	
<b>Reentrancy:</b>	Reentrant	
<b>Parameters (in):</b>	BlockId	The block identifier uniquely identifies one NVRAM block descriptor. A NVRAM block descriptor contains all needed information about a single NVRAM block.
<b>Parameters (inout):</b>	None	
<b>Parameters (out):</b>	None	
<b>Return value:</b>	Std_ReturnType	E_OK: request has been accepted E_NOT_OK: request has not been accepted
<b>Description:</b>	Service to copy the data of the NV block to its corresponding permanent RAM block.	
<b>Available via:</b>	NvM.h	

### 6.3.2.7 NvM\_WritePRAMBlock

<b>Service name:</b>	NvM_WritePRAMBlock	
<b>Syntax:</b>	Std_ReturnType NvM_WritePRAMBlock( NvM_BlockIdType BlockId )	
<b>Service ID[hex]:</b>	0x17	
<b>Sync/Async:</b>	Asynchronous	
<b>Reentrancy:</b>	Reentrant	
<b>Parameters (in):</b>	BlockId	The block identifier uniquely identifies one NVRAM block descriptor. A NVRAM block descriptor contains all needed information about a single NVRAM block.
<b>Parameters (inout):</b>	None	
<b>Parameters (out):</b>	None	
<b>Return value:</b>	Std_ReturnType	E_OK: request has been accepted E_NOT_OK: request has not been accepted
<b>Description:</b>	Service to copy the data of the RAM block to its corresponding permanent RAM block.	
<b>Available via:</b>	NvM.h	

### 6.3.2.8 NvM\_RestorePRAMBlockDefaults

<b>Service name:</b>	NvM_RestorePRAMBlockDefaults	
<b>Syntax:</b>	Std_ReturnType NvM_RestorePRAMBlockDefaults( NvM_BlockIdType BlockId )	
<b>Service ID[hex]:</b>	0x18	
<b>Sync/Async:</b>	Asynchronous	
<b>Reentrancy:</b>	Non Reentrant	
<b>Parameters (in):</b>	BlockId	The block identifier uniquely identifies one NVRAM block descriptor. A NVRAM block descriptor contains all needed information about a single NVRAM block.



<b>Parameters (inout):</b>	None
<b>Parameters (out):</b>	None
<b>Return value:</b>	Std_ReturnType E_OK: request has been accepted E_NOT_OK: request has not been accepted
<b>Description:</b>	Service to restore the default data to its corresponding permanent RAM block.
<b>Available via:</b>	NvM.h

### 6.3.3 Asynchronous Multi Block Requests

#### 6.3.3.1 NvM\_ReadAll

<b>Service name:</b>	NvM_ReadAll
<b>Syntax:</b>	void NvM_ReadAll( void )
<b>Service ID[hex]:</b>	0x0c
<b>Sync/Async:</b>	Asynchronous
<b>Reentrancy:</b>	Non Reentrant
<b>Parameters (in):</b>	None
<b>Parameters (inout):</b>	None
<b>Parameters (out):</b>	None
<b>Return value:</b>	None
<b>Description:</b>	Initiates a multi block read request.
<b>Available via:</b>	NvM.h

#### 6.3.3.2 NvM\_WriteAll

<b>Service name:</b>	NvM_WriteAll
<b>Syntax:</b>	void NvM_WriteAll( void )
<b>Service ID[hex]:</b>	0x0d
<b>Sync/Async:</b>	Asynchronous
<b>Reentrancy:</b>	Non Reentrant
<b>Parameters (in):</b>	None
<b>Parameters (inout):</b>	None
<b>Parameters (out):</b>	None
<b>Return value:</b>	None
<b>Description:</b>	Initiates a multi block write request.
<b>Available via:</b>	NvM.h

#### 6.3.3.3 NvM\_CancelWriteAll

<b>Service name:</b>	NvM_CancelWriteAll
<b>Syntax:</b>	void NvM_CancelWriteAll( void )
<b>Service ID[hex]:</b>	0x0a
<b>Sync/Async:</b>	Asynchronous
<b>Reentrancy:</b>	Non Reentrant
<b>Parameters (in):</b>	None

<b>Parameters (inout):</b>	None
<b>Parameters (out):</b>	None
<b>Return value:</b>	None
<b>Description:</b>	Service to cancel a running NvM_WriteAll request.
<b>Available via:</b>	NvM.h

#### 6.3.3.4 NvM\_ValidateAll

<b>Service name:</b>	NvM_ValidateAll
<b>Syntax:</b>	void NvM_ValidateAll( void )
<b>Service ID[hex]:</b>	0x19
<b>Sync/Async:</b>	Asynchronous
<b>Reentrancy:</b>	Non Reentrant
<b>Parameters (in):</b>	None
<b>Parameters (inout):</b>	None
<b>Parameters (out):</b>	None
<b>Return value:</b>	None
<b>Description:</b>	Initiates a multi block validation request.
<b>Available via:</b>	NvM.h

#### 6.3.3.5 NvM\_FirstInitAll

<b>Service name:</b>	NvM_FirstInitAll
<b>Syntax:</b>	void NvM_FirstInitAll( void )
<b>Service ID[hex]:</b>	0x14
<b>Sync/Async:</b>	Asynchronous
<b>Reentrancy:</b>	Non Reentrant
<b>Parameters (in):</b>	None
<b>Parameters (inout):</b>	None
<b>Parameters (out):</b>	None
<b>Return value:</b>	None
<b>Description:</b>	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.
<b>Available via:</b>	NvM.h

### 6.3.4 Callback Notifications

#### 6.3.4.1 NvM\_JobEndNotification

<b>Service name:</b>	NvM_JobEndNotification
<b>Syntax:</b>	void NvM_JobEndNotification( void )
<b>Service ID[hex]:</b>	0x11
<b>Sync/Async:</b>	Synchronous
<b>Reentrancy:</b>	Non Reentrant

<b>Parameters (in):</b>	None
<b>Parameters (inout):</b>	None
<b>Parameters (out):</b>	None
<b>Return value:</b>	None
<b>Description:</b>	Function to be used by the underlying memory abstraction to signal end of job without error.
<b>Available via:</b>	NvM_MemIf.h

#### 6.3.4.2 NvM\_JobErrorNotification

<b>Service name:</b>	NvM_JobErrorNotification
<b>Syntax:</b>	void NvM_JobErrorNotification( void )
<b>Service ID[hex]:</b>	0x12
<b>Sync/Async:</b>	Synchronous
<b>Reentrancy:</b>	Non Reentrant
<b>Parameters (in):</b>	None
<b>Parameters (inout):</b>	None
<b>Parameters (out):</b>	None
<b>Return value:</b>	None
<b>Description:</b>	Function to be used by the underlying memory abstraction to signal end of job with error.
<b>Available via:</b>	NvM_MemIf.h

#### 6.3.5 Scheduled Functions

##### 6.3.5.1 NvM\_Mainfunction

<b>Service name:</b>	NvM_MainFunction
<b>Syntax:</b>	void NvM_MainFunction( void )
<b>Service ID[hex]:</b>	0x0e
<b>Description:</b>	Service for performing the processing of the NvM jobs.
<b>Available via:</b>	SchM_NvM.h

#### 6.3.6 User Defined Function

##### 6.3.6.1 NvM\_UserJobFunction

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

<b>Reentrancy</b>	Non Reentrant
<b>Parameters (In)</b>	None
<b>Parameters (Inout)</b>	None
<b>Parameters (Out)</b>	None
<b>Return Value</b>	NvM_OpStatusType: NVM_OPSTATUS_IDLE: user job is idle NVM_OPSTATUS_BUSY: user job is busy
<b>Description</b>	<p>When users set NvMUserJobFunction, UserJob function is driven in OsTask_BSW_Mem_Process Task.</p> <p>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.</p> <p>Note: UserJob must be started when the return value of NvM_CddGetStatus is NVM_OPSTATUS_IDLE.</p>
<b>Preconditions</b>	None
<b>Configuration Dependency</b>	This function is available only if configuration parameter NvMUserJobFunction is configured.

### 6.3.6.2 NvM\_UserWdgToggleFunction

<b>Function Name</b>	NvM_UserWdgToggleFunction
<b>Syntax:</b>	void NvM_UserWdgToggleFunction (void)
<b>Service ID</b>	None
<b>Sync/Async</b>	Synchronous
<b>Reentrancy</b>	Non Reentrant
<b>Parameters (In)</b>	None
<b>Parameters (Inout)</b>	None
<b>Parameters (Out)</b>	None
<b>Return Value</b>	None

<b>Description</b>	If the user configures NvMUserWdgToggleFunction, user functions are repeatedly performed in mem_EalnitPerform and Mem_FeelnitPerform 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.
<b>Preconditions</b>	None
<b>Configuration Dependency</b>	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

- 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
- ERR020006: NvMNvramBlockIdentifier must be unique.
  - With NvMBlockDescriptor, /AUTOEVER/NvM/NvMBlockDescriptor/NvMNvramBlockIdentifier must be unique
- 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\_]*	RamBlockDataAdress_1
NvMRomBlockDataAddress			RomBlockDataAddress_1

- 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
- 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/NvMMMainfunctionTriggerRef" parameter is configured. User needs to define OsSecPerTick in 2 ways following:
- Configure Os/OsCounter/OsSecondsPerTick in container reference by "NvMCommon/NvMMMainfunctionTriggerRef" >> 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

<b>Error Name:</b>	NVM_E_HARDWARE	
<b>Short Description:</b>	Reading from or writing to non volatile memory failed	
<b>Long Description:</b>	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.	
<b>Detection Criteria:</b>	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)
<b>Secondary Parameters:</b>	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.	
<b>Time Required:</b>	Not applicable. (there is no timeout monitoring in the NvM)	
<b>Monitor Frequency</b>	continuous	

#### 8.1.2 NVM\_E\_INTEGRITY\_FAILED

<b>Error Name:</b>	NVM_E_INTEGRITY_FAILED	
<b>Short Description:</b>	Processing of the read service detects an inconsistency.	
<b>Long Description:</b>	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.	
<b>Detection Criteria:</b>	Fail	See SWS_NvM_00864
	Pass	See SWS_NvM_00872
<b>Secondary Parameters:</b>	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.	
<b>Time Required:</b>	Not applicable. There is no timeout monitoring or constraint for NvM.	
<b>Monitor Frequency</b>	continuous	

#### 8.1.3 NVM\_E\_REQ\_FAILED

<b>Error Name:</b>	NVM_E_REQ_FAILED	
<b>Short Description:</b>	Processing of the read service failed at a lower layer in the MemStack architecture, including all retries.	
<b>Long Description:</b>	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.	
<b>Detection Criteria:</b>	Fail	See SWS_NvM_00865
	Pass	See: SWS_NvM_00873

<b>Secondary Parameters:</b>	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.
<b>Time Required:</b>	Not applicable. There is no timeout monitoring or constraint for NvM.
<b>Monitor Frequency</b>	continuous

#### 8.1.4 NVM\_E\_WRONG\_BLOCK\_ID

<b>Error Name:</b>	NVM_E_WRONG_BLOCK_ID	
<b>Short Description:</b>	Static block ID check, during read, indicates failure.	
<b>Long Description:</b>	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).	
<b>Detection Criteria:</b>	Fail	See SWS_NvM_00866
	Pass	See SWS_NvM_00874
<b>Secondary Parameters:</b>	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.	
<b>Time Required:</b>	Not applicable. There is no timeout monitoring or constraint for NvM.	
<b>Monitor Frequency</b>	continuous	

#### 8.1.5 NVM\_E\_VERIFY\_FAILED

<b>Error Name:</b>	NVM_E_VERIFY_FAILED	
<b>Short Description:</b>	The write verification failed.	
<b>Long Description:</b>	If, after a successfully finished write, the verification for the written data fails.	
<b>Detection Criteria:</b>	Fail	See SWS_NvM_00867
	Pass	See SWS_NvM_00875
<b>Secondary Parameters:</b>	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.	
<b>Time Required:</b>	Not applicable. There is no timeout monitoring or constraint for NvM.	
<b>Monitor Frequency</b>	continuous	

#### 8.1.6 NVM\_E\_LOSS\_OF\_REDUNDANCY

<b>Error Name:</b>	NVM_E_LOSS_OF_REDUNDANCY	
<b>Short Description:</b>	A redundant block has lost the redundancy.	
<b>Long Description:</b>	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.	
<b>Detection Criteria:</b>	Fail	See SWS_NvM_00868
	Pass	See SWS_NvM_00876
<b>Secondary Parameters:</b>	The condition under which the FAIL or PASS detection is active: checks are performed whenever a reading is requested for a redundant block.	
<b>Time Required:</b>	Not applicable. There is no timeout monitoring or constraint for NvM.	
<b>Monitor Frequency</b>	continuous	

### 8.1.7 NVM\_E\_WRITE\_PROTECTED

<b>Error Name:</b>	NVM_E_WRITE_PROTECTED	
<b>Short Description:</b>	A write is requested for a write protected block.	
<b>Long Description:</b>	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.	
<b>Detection Criteria:</b>	Fail	See: SWS_NvM_00870
	Pass	See: SWS_NvM_00878
<b>Secondary Parameters:</b>	The condition under which the FAIL or PASS detection is active: check is performed for each write request.	
<b>Time Required:</b>	Not applicable. There is no timeout monitoring or constraint for NvM.	
<b>Monitor Frequency</b>	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:  $2n * \text{NvMNvBlockBaseNumber}$
- 6) Second Ea/Fee Block Number:  $2n * \text{NvMNvBlockBaseNumber} + 1$
- 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	Properties
<ul style="list-style-type: none"> <li>NvM           <ul style="list-style-type: none"> <li>Block Descriptor [11]               <ul style="list-style-type: none"> <li>NvMCommon                   <ul style="list-style-type: none"> <li>NvMUserIncludeFiles</li> <li>NvMDemEventParameterRefs_0</li> </ul> </li> </ul> </li> </ul> </li> </ul>	Set Ram Block Status Api*: <input type="checkbox"/> false Size Immediate Job Queue: <input type="text" value="20"/> Size Standard Job Queue*: <input type="text" value="20"/> Main Function Call Cycle: <input type="text" value="0.005"/> Version Info Api*: <input type="checkbox"/> false <b>Read All Order Support: <input checked="" type="checkbox"/> true</b> <b>Write All Order Support: <input checked="" type="checkbox"/> true</b> 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	Properties
<ul style="list-style-type: none"> <li>NvM           <ul style="list-style-type: none"> <li>Block Descriptor [11]               <ul style="list-style-type: none"> <li>NvMBlock_ConfigID</li> <li>NvMBlock_ManagementBlock</li> <li>NvMBlock_EventStatusNvRamBlock</li> <li>NvMBlock_PrimaryEventMemory0</li> <li>NvMBlock_PrimaryEventMemory1</li> </ul> </li> </ul> </li> </ul>	Write Verification Data Size*: <input type="text" value="18"/> DefaultROMCRCEnabled*: <input type="checkbox"/> false <b>Read All Order*: <input type="text" value="7"/></b> <b>Write All Order*: <input type="text" value="1"/></b> 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 Delnit in the Shutdown Sequence phase. DelnitTimeout can be set in WdgM setting, and Wdg Reset occurs when Timeout occurs after WdgM Delnit. So, WriteAll must be completed in DelnitTimeout 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 DelnitTimeout 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

```
052:
053: /*****
054:  * START : Only STD_ON or STD_OFF of the macros can be modified by user
055:  *****/
056:
057: /* STD_ON : When WriteAll is called , mainFunctions of the NvM,Fee,Fls,Eep,Ea
058:  *          are called in the while-loop
059:  * STD_OFF : When WriteAll is called , mainFunctions of the NvM,Fee,Fls,Eep,Ea
060:  *          are called by the GPT or periodic task
061:  */
062:
063: /* CAUTION!!!
064:  *
065:  * WriteAll : Depending on the type of MCU,
066:  *            there may be a timing problems.
067:  * Default : STD_OFF
068:  *
069:  * please contact us.
070:  *****/
071:
072:
073: #define MEM_WRITEALL_FAST_MODE (STD_OFF)
074:
075:
```

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.

◆ BswImplementation\_NvM

◆ All Contents\_V2

Path: ◆ BswImplementation\_NvM [BswImplementation]

Navigator

Container Details - Elements > Bsw Implementation

Short Name\*: ◆ BswImplementation\_NvM Edit

Programming Language: ◆ C

Sw Version: ◆ 1.0.0

Swc Bsw Mapping: ◆ SwcBswMapping\_NvM [/Svc\_NvM/ServiceS ... +

Vendor Id: ◆ 76 Convert...

Ar Release Version: ◆ 4.4.0

## 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

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

\*Rte\_IntTst\_Mem

### Task Mapping

Event Table

☐ Bsw Module Instance ☒ Sw Component Instance /AUTOSAR/Rte\_IntTst\_Mem/SwcInstance\_IntTst\_Mer

Runnable Entity	Period	Operation	Activation	Deactivation
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	-	-	-

Event Table

☐ Bsw Module Instance ☒ 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	-	-

④ Add Remove

### OsTask Mapping Table

☐ Related to Bsw Module ☒ Related to Sw Component ☐ unMapped ② OsTask\_ASW\_FG2\_NvMCallback

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

\*Os

### Task

Path: Os [OS] > OsTask\_ASW\_FG2\_NvMCallback [OsTask]

Navigator

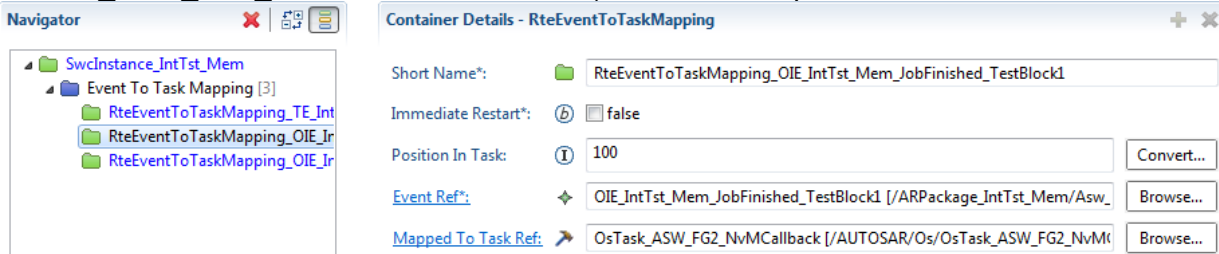
Container Details - OsTask

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				

4. Callback Runnable on Task Mapping

- 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).



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

### 9.6.2 InitBlock Callback

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.