**Non Volatile RAM Manager And**

**Non Volatile RAM Manager Proxy**

**FDD #ES-006A**

**Contents**

[1. High Level Description 4](#_Toc428878928)

[2. Derived Requirements 4](#_Toc428878929)

[3. Sub-Function Data Flow 4](#_Toc428878930)

[4. Design Rationale 5](#_Toc428878931)

[5. Components 6](#_Toc428878932)

[5.1. NvM: Non Volatile Memory (AUTOSAR BSW) 6](#_Toc428878933)

[5.1.1. BSW Configuration 6](#_Toc428878934)

[5.1.1.1. NvMCommon 6](#_Toc428878935)

[5.1.2. Periodic Functions 7](#_Toc428878936)

[5.1.2.1. NvM\_MainFunction 7](#_Toc428878937)

[5.1.2.1.1. Function Definition 7](#_Toc428878938)

[5.1.3. Service Sub-Functions 8](#_Toc428878939)

[5.1.3.1. API Configuration Class 1 9](#_Toc428878940)

[5.1.3.1.1. Sub-Function: NvM\_Init 9](#_Toc428878941)

[5.1.3.1.2. Sub-Function: NvM\_ReadAll 10](#_Toc428878942)

[5.1.3.1.3. Sub-Function: NvM\_WriteAll 11](#_Toc428878943)

[5.1.3.1.4. Sub-Function: NvM\_GetErrorStatus 12](#_Toc428878944)

[5.1.3.1.5. Sub-Function: NvM\_SetRamBlockStatus 13](#_Toc428878945)

[5.1.3.1.6. Sub-Function: NvM\_CancelWriteAll 14](#_Toc428878946)

[5.1.3.2. API Configuration Class 2 15](#_Toc428878947)

[5.1.3.2.1. Sub-Function: NvM\_SetDataIndex 15](#_Toc428878948)

[5.1.3.2.2. Sub-Function: NvM\_GetDataIndex 16](#_Toc428878949)

[5.1.3.2.3. Sub-Function: NvM\_ReadBlock 17](#_Toc428878950)

[5.1.3.2.4. Sub-Function: NvM\_WriteBlock 18](#_Toc428878951)

[5.1.3.2.5. Sub-Function: NvM\_RestoreBlockDefaults 19](#_Toc428878952)

[5.1.3.2.6. Sub-Function: NvM\_CancelJobs 20](#_Toc428878953)

[5.1.3.3. API Configuration Class 3 21](#_Toc428878954)

[5.1.3.3.1. Sub-Function: NvM\_SetBlockProtection 21](#_Toc428878955)

[5.1.3.3.2. Sub-Function: NvM\_EraseNvBlock 22](#_Toc428878956)

[5.1.3.3.3. Sub-Function: NvM\_InvalidateNvBlock 23](#_Toc428878957)

[5.1.4. Type Definitions 24](#_Toc428878958)

[5.1.4.1. Std\_ReturnType 24](#_Toc428878959)

[5.1.4.2. NvM\_RequestResultType 24](#_Toc428878960)

[5.1.4.3. NvM\_BlockIdType 25](#_Toc428878961)

[5.2. NvM\_Proxy: Non Volatile Memory Proxy (Nexteer CDD) 26](#_Toc428878962)

[5.2.1. Design Rationale 26](#_Toc428878963)

[5.2.2. Sub-Functions 26](#_Toc428878964)

[5.2.2.1. Sub-Function: NvMProxy\_Init 26](#_Toc428878965)

[5.2.2.1.1. Hardware Related Design 26](#_Toc428878966)

[5.2.2.1.2. Software Related Design 26](#_Toc428878967)

[6. Timing / Execution Constraints 27](#_Toc428878968)

[6.1. Rationale / Comments 27](#_Toc428878969)

[6.2. Rates and State Execution: NvM 27](#_Toc428878970)

[6.3. Rates and State Execution: NvMProxy 28](#_Toc428878971)

[7. Serial Communications Interfaces 28](#_Toc428878972)

[8. Additional Information 29](#_Toc428878973)

[8.1. NvM block definition Considerations 29](#_Toc428878974)

[8.1.1. Scenario 1 29](#_Toc428878975)

[8.1.2. Scenario 2 29](#_Toc428878976)

[8.2. Software Component Design Considerations 30](#_Toc428878977)

[8.2.1. API Port Selection 30](#_Toc428878978)

[9. Revision Record & Change Approval 31](#_Toc428878979)

# High Level Description

This design document describes the functionality, API, and the configuration of the AUTOSAR basic software (BSW) module NVRAM Manager (NvM) and the NvM Proxy (NvMProxy).

The NvM provides services to ensure the data storage and maintenance of NV (non-volatile) data. The NvM module is able to administrate the NV data for an EEPROM and/or a Flash EEPROM Emulation (FEE) device.

The NvMProxy provides an interface for software components outside of the application of the NvM to communicate with the NvM component.

# Derived Requirements

None

# Sub-Function Data Flow

None

# Design Rationale

The NvM and NvMProxy components are integrated below the application layer in the basic software layer of the AUTOSAR model.

NvMProxy was designed so that all software components can send their NvM requests to the proxy interface, which will communicate the request to the NvM and report the results back to the calling component. This simplifies the design of software components by only requiring one interface for defining NvM needs and providing the needed functionality to switch the OS context in the event the calling application is different than the application NvM is integrated.



# Components

The following sections describe the NvM and NvM proxy components.

## NvM: Non Volatile Memory (AUTOSAR BSW)

### BSW Configuration

#### NvMCommon

|  |  |  |
| --- | --- | --- |
| Configuration Parameter | Value | Rationale |
| API Configuration Class | MVM\_API\_CONFIG\_CLASS\_3 | Class 3 shall be used to provide all API options to software components. This is to prevent rework of existing components if use cases change and require API functions that may not have been available during component development under a different API class. |
| Compiled Configuration Id | 1 | Version of the NV memory layout, always shall start at 1 and be revised if the memory layout changes |
| Crc Number of Bytes | 64 | Dummy value, not used. |
| Dataset Selection Bits | 1 | Shall be set to 1 if the only block types are “native” and “redundant.” If a dataset is required, then the number needs to be set satisfy the following equation:  *2^(Selection Bits) >= max(dataset)*  For example, if the largest dataset for all configured blocks was 30, the selection bits are required to be set to 5.  2^5 = 32 >= 30 |
| Development Error Detection | FALSE | Only should be true for early development. Shall not be used in production level software |
| Drivers Mode Switch | True | Disables processing of background sector switching from startup and shutdown events. |
| Dynamic Configuration Handling | True | Allows for adapting new FEE layouts over existing layouts. See section 5.1.3.2.2 for details on the impact of this setting. |
| Job Prioritization | False | No requirement for prioritization of any blocks |
| Maximum Number of Write Retries | 3 | Default setting |
| Multi Block Callback | NvMProxy\_MultiBlkCallBack |  |
| Multi block Job Status Information | False |  |
| Polling Mode | True | Enabled to provide the application the ability to poll the status of the asynchronous request. |
| Repeat Mirror Operations | 0 | Default setting |
| SetRamBlockStatus API | True | Applications shall use SetRamBlockStatus API to indicate their RAM shadows have updated. |
| Size Of Immediate Status Information | N/A | Not used |
| Size Of Standard Job Queue | 8 | Default setting |
| Version Information API | False | N/A |

### Periodic Functions

#### NvM\_MainFunction

This function has to be called cyclically. It is the entry point for the NvM component. In this function processing of all asynchronous jobs are handled (read/write/erase/invalidate/CRC calculation).

##### Function Definition

|  |
| --- |
| Prototype |
| void NvM\_MainFunction ( void ) |
| Parameter |
| N/A |
| Return Code |
| N/A |

### Service Sub-Functions

The value of the API configuration class determines which API server ports are available to the system. The image below shows the breakdown of the functionality for each API Configuration Class.



In the following sub sections, the APIs are defined for application software components (SWCs) and BSW components. The application function definition shall be used for components that sit above the RTE layer in the AUTOSAR model. The RTE generator will absorb some of the dynamic arguments, such as BlockId, and create a macro with the proper definition for the software component. Complex device drivers and other BSWs that sit below the RTE layer shall use the CDD function definition.

#### API Configuration Class 1

The following sections contain a description of the functions provided by the AUTOSAR NvM basic software component with API Configuration Class 1 configured.

##### Sub-Function: NvM\_Init

###### Hardware Related Design

None

###### Software Related Design

|  |  |
| --- | --- |
| Function Particularities | |
| Request Type | Synchronous |
| Re-entrant | Yes |
| Expected Caller Context | Shall only be called from ECU state manager or equivalent function. |

Before the NvM component can be used, it has to be initialized. Depending on the program the NvM is integrated, the BSWs from lower levels shall be initialized prior to the NvM. The table below is an example of this strategy for Fee and Ea use cases for initialize modules from the low level components up to the NvM.

|  |  |  |
| --- | --- | --- |
|  | Fee | Ea |
| Low level driver | Fls | SPI/EEP |
| Device Abstraction | FEE | EA |
| Non-Volatile Manager | NvM | |

The NvM AUTOSAR component compliant with ASIL-D standards, NvM\_Init shall be called as a trusted function. This will allow the NvM\_Init function access to all the permanent RAM shadows defined in software, regardless of their ASIL rating. This reduces the RAM and throughput during start up to move data from application to another.

Application Function Definition

None

CDD Function Definition

|  |
| --- |
| Prototype |
| void NvM\_Init ( void ) |
| Parameter |
| N/A |
| Return Code |
| N/A |

##### Sub-Function: NvM\_ReadAll

###### Hardware Related Design

None

###### Software Related Design

|  |  |
| --- | --- |
| Function Particularities | |
| Request Type | Asynchronous |
| Re-entrant | No |
| Expected Caller Context | Shall only be called from ECU state manager or equivalent function. |

This function shall only be called after NvM\_Init has been executed. The request loads all the RAM blocks that have the option NVM\_SELECT\_BLOCK\_FOR\_READALL selected.

Note: Non-permanent blocks and data set blocks are skipped during execution of this function and must be loaded manually by calling NvM\_ReadBlock().

During the execution of NvM\_ReadAll(), the value in the *configuration ID* (block 1) is compared with the compiled ID version in the NvM settings. With the *Dynamic Configuration Handling* option set to True, any NvM blocks with the option *Resistant to Changed Software* enabled will be processed and loaded into RAM as if the configuration IDs matched. If the option *Resistant to Changed Software* is not enabled, the blocks will be treated is if they were invalid or blank.

Application Function Definition

None

CDD Function Definition

|  |
| --- |
| Prototype |
| void NvM\_ReadAll ( void ) |
| Parameter |
| N/A |
| Return Code |
| N/A |

##### Sub-Function: NvM\_WriteAll

###### Hardware Related Design

None

###### Software Related Design

|  |  |
| --- | --- |
| Function Particularities | |
| Request Type | Asynchronous |
| Re-entrant | No |
| Expected Caller Context | Shall only be called from ECU state manager or equivalent function. |

Request to write all blocks with RAM data that has been changed and have the option NVM\_SELECT\_BLOCK\_FOR\_WRITEALL selected.

Note: Non-permanent blocks and data set blocks are skipped during execution of this function and must be written manually by calling NvM\_WriteBlock().

Application Function Definition

None

CDD Function Definition

|  |
| --- |
| Prototype |
| void NvM\_WriteAll ( void ) |
| Parameter |
| N/A |
| Return Code |
| N/A |

##### Sub-Function: NvM\_GetErrorStatus

###### Hardware Related Design

None

###### Software Related Design

|  |  |
| --- | --- |
| Function Particularities | |
| Request Type | Synchronous |
| Re-entrant | Yes |
| Expected Caller Context | Application |

The request reads the block dependent status/error information and writes it to the given address. The status/error information was set by a former or current asynchronous request. This API can also be requested with the block id 0 (multi block). The multi block status/error information are only set by NvM\_ReadAll() and NvM\_WriteAll() requests.

Application Function Definition

|  |
| --- |
| Prototype |
| Std\_ReturnType Rte\_Call\_<SWCSvcPortName>\_GetErrorStatus (uint8\* RequestResultPtr ) |
| Parameter |
| RequestResultPtr – Pointer were the result shall be written. See section 5.1.4.2 for details on return values. |
| Return Code |
| Std\_ReturnType – See section 5.1.4.1 definition of this return type |

CDD Function Definition

|  |
| --- |
| Prototype |
| Std\_ReturnType NvM\_GetErrorStatus ( NvMBlockIdType BlockId, uint8\* RequestResultPtr ) |
| Parameter |
| BlockId – The block identifier |
| RequestResultPtr – Pointer were the result shall be written. See section 5.1.4.2 for details on return values. |
| Return Code |
| Std\_ReturnType – See section 5.1.4.1 definition of this return type |

##### Sub-Function: NvM\_SetRamBlockStatus

###### Hardware Related Design

None

###### Software Related Design

|  |  |
| --- | --- |
| Function Particularities | |
| Request Type | Synchronous (If *NVM\_CALC\_RAM\_BLOCK\_CRC* is False)  Asynchronous (If *NVM\_CALC\_RAM\_BLOCK\_CRC* is True) |
| Re-entrant | Yes |
| Expected Caller Context | Application |

The request sets a block’s status to valid/changed and also to unchanged. Setting a block to valid/changed will mark it for writing during NvM\_WriteAll(). If a block has the CRC option NVM\_CALC\_RAM\_BLOCK\_CRC set to True, the CRC calculation of the RAM is initialed as a background task.

Application Function Definition

|  |
| --- |
| Prototype |
| Std\_ReturnType Rte\_Call\_<SWCSvcPortName>\_SetRamBlockStatus (Boolean BlockChanged ) |
| Parameter |
| BlockChanged – TRUE: Validates the RAM block and marks it as changed.  FALSE: Mark the block as unchanged |
| Return Code |
| Std\_ReturnType – See section 5.1.4.1 definition of this return type |

CDD Function Definition

|  |
| --- |
| Prototype |
| Std\_ReturnType NvM\_SetRamBlockStatus ( NvMBlockIdType BlockId, Boolean BlockChanged ) |
| Parameter |
| BlockId – The block identifier |
| BlockChanged – TRUE: Validates the RAM block and marks it as changed.  FALSE: Mark the block as unchanged |
| Return Code |
| Std\_ReturnType – See section 5.1.4.1 definition of this return type |

##### Sub-Function: NvM\_CancelWriteAll

###### Hardware Related Design

None

###### Software Related Design

|  |  |
| --- | --- |
| Function Particularities | |
| Request Type | Asynchronous |
| Re-entrant | No |
| Expected Caller Context | Shall only be called from ECU state manager or equivalent function. |

Request to cancel a running NvM\_WriteAll() request.

Application Function Definition

None

CDD Function Definition

|  |
| --- |
| Prototype |
| void NvM\_CancelWriteAll ( void ) |
| Parameter |
| None |
| Return Code |
| None |

#### API Configuration Class 2

The following sections contain a description of the functions provided by the AUTOSAR NvM basic software component with API Configuration Class 2 configured.

##### Sub-Function: NvM\_SetDataIndex

###### Hardware Related Design

None

###### Software Related Design

|  |  |
| --- | --- |
| Function Particularities | |
| Request Type | Synchronous |
| Re-entrant | Yes |
| Expected Caller Context | Application |

The request sets the specified index to associate a dataset NV block (with or without ROM blocks) with its corresponding RAM block. The DataIndex needs to have a valid value before read, write, erase or invalidate requests are initiated.

If the dataset block has a set of ROM defaults, the function is used to select the appropriate ROM set (prior to NvM\_ReadBlock()).

Application Function Definition

|  |
| --- |
| Prototype |
| Std\_ReturnType Rte\_Call\_<SWCSvcPortName>\_SetDataIndex (uint8 DataIndex ) |
| Parameter |
| DataIndex – Index position of a block in the NV block Dataset type |
| Return Code |
| Std\_ReturnType – See section 5.1.4.1 definition of this return type |

CDD Function Definition

|  |
| --- |
| Prototype |
| Std\_ReturnType NvM\_SetDataIndex ( NvMBlockIdType BlockId, uint8 DataIndex ) |
| Parameter |
| BlockId – The block identifier |
| DataIndex – Index position of a block in the NV block Dataset type |
| Return Code |
| Std\_ReturnType – See section 5.1.4.1 definition of this return type |

##### Sub-Function: NvM\_GetDataIndex

###### Hardware Related Design

None

###### Software Related Design

|  |  |
| --- | --- |
| Function Particularities | |
| Request Type | Synchronous |
| Re-entrant | Yes |
| Expected Caller Context | Application |

The request passes the current DataIndex of the specified dataset block.

Application Function Definition

|  |
| --- |
| Prototype |
| Std\_ReturnType Rte\_Call\_<SWCSvcPortName>\_GetDataIndex (uint8\* DataIndexPtr ) |
| Parameter |
| DataIndexPtr – Address where the current DataIndex shall be written to |
| Return Code |
| Std\_ReturnType – See section 5.1.4.1 definition of this return type |

CDD Function Definition

|  |
| --- |
| Prototype |
| Std\_ReturnType NvM\_GetDataIndex ( NvMBlockIdType BlockId, uint8\* DataIndexPtr ) |
| Parameter |
| BlockId – The block identifier |
| DataIndexPtr – Address where the current DataIndex shall be written to |
| Return Code |
| Std\_ReturnType – See section 5.1.4.1 definition of this return type |

##### Sub-Function: NvM\_ReadBlock

###### Hardware Related Design

None

###### Software Related Design

|  |  |
| --- | --- |
| Function Particularities | |
| Request Type | Asynchronous |
| Re-entrant | Yes |
| Expected Caller Context | Application |

Request to copy the data of the NV block to its corresponding RAM block. This function queues the read request and returns the acceptance result synchronously.

Application Function Definition

|  |
| --- |
| Prototype |
| Std\_ReturnType Rte\_Call\_<SWCSvcPortName>\_ReadBlock (uint8\* NvM\_DstPtr ) |
| Parameter |
| NvM\_DstPtr – Pointer where the data of a non-permanent RAM block shall be written to. If the block is permanent NULL\_PTR shall be passed. |
| Return Code |
| Std\_ReturnType – See section 5.1.4.1 definition of this return type |

CDD Function Definition

|  |
| --- |
| Prototype |
| Std\_ReturnType NvM\_ReadBlock ( NvMBlockIdType BlockId, uint8\* NvM\_DstPtr) |
| Parameter |
| BlockId – The block identifier |
| NvM\_DstPtr – Pointer where the data of a non-permanent RAM block shall be written to. If the block is permanent NULL\_PTR shall be passed. |
| Return Code |
| Std\_ReturnType – See section 5.1.4.1 definition of this return type |

##### Sub-Function: NvM\_WriteBlock

###### Hardware Related Design

None

###### Software Related Design

|  |  |
| --- | --- |
| Function Particularities | |
| Request Type | Asynchronous |
| Re-entrant | Yes |
| Expected Caller Context | Application |

Request for copying data from the RAM block to its corresponding NV block. This function queues the write request and returns the acceptance result synchronously.

If the block has a CRC, the RAM block CRC will be recalculated before the data and the CRC are written to the NV memory, even if the service NvM\_SetRamBlockStatus() was called before and the configuration was set that within this service, the CRC calculation should be done.

If writing the data to NV memory fails, the NVM will retry writing. The number of write retries is a configuration option.

Application Function Definition

|  |
| --- |
| Prototype |
| Std\_ReturnType Rte\_Call\_<SWCSvcPortName>\_WriteBlock (const uint8\* NvM\_SrcPtr ) |
| Parameter |
| NvM\_SrcPtr – Pointer where the data of a non-permanent RAM block shall be read from. If the block is permanent, NULL\_PTR shall be passed. |
| Return Code |
| Std\_ReturnType – See section 5.1.4.1 definition of this return type |

CDD Function Definition

|  |
| --- |
| Prototype |
| Std\_ReturnType NvM\_WriteBlock ( NvMBlockIdType BlockId, const uint8\* NvM\_SrcPtr) |
| Parameter |
| BlockId – The block identifier |
| NvM\_SrcPtr – Pointer where the data of a non-permanent RAM block shall be read from. If the block is permanent, NULL\_PTR shall be passed. |
| Return Code |
| Std\_ReturnType – See section 5.1.4.1 definition of this return type |

##### Sub-Function: NvM\_RestoreBlockDefaults

###### Hardware Related Design

None

###### Software Related Design

|  |  |
| --- | --- |
| Function Particularities | |
| Request Type | Asynchronous |
| Re-entrant | Yes |
| Expected Caller Context | Application |

Request to copy the ROM block default data to its corresponding RAM block. The selected block needs either ROM defaults or an initialization callback. This function queues the restore request and returns the acceptance results synchronously.

Application Function Definition

|  |
| --- |
| Prototype |
| Std\_ReturnType Rte\_Call\_<SWCSvcPortName>\_RestoreBlockDefaults (uint8\* NvM\_DstPtr ) |
| Parameter |
| NvM\_DstPtr – Pointer where the data of a non-permanent RAM block shall be written to. If the block is permanent, NULL\_PTR shall be passed. |
| Return Code |
| Std\_ReturnType – See section 5.1.4.1 definition of this return type |

CDD Function Definition

|  |
| --- |
| Prototype |
| Std\_ReturnType NvM\_RestoreBlockDefaults ( NvMBlockIdType BlockId, uint8\* NvM\_DstPtr) |
| Parameter |
| BlockId – The block identifier |
| NvM\_DstPtr – Pointer where the data of a non-permanent RAM block shall be written to. If the block is permanent, NULL\_PTR shall be passed. |
| Return Code |
| Std\_ReturnType – See section 5.1.4.1 definition of this return type |

##### Sub-Function: NvM\_CancelJobs

###### Hardware Related Design

None

###### Software Related Design

|  |  |
| --- | --- |
| Function Particularities | |
| Request Type | Asynchronous |
| Re-entrant | Yes |
| Expected Caller Context | Application |

Request to cancel pending job for a NV block.

Application Function Definition

None

CDD Function Definition

|  |
| --- |
| Prototype |
| Std\_ReturnType NvM\_CancelJobs ( NvMBlockIdType BlockId ) |
| Parameter |
| BlockId – The block identifier |
| Return Code |
| Std\_ReturnType – See section 5.1.4.1 definition of this return type |

#### API Configuration Class 3

The following sections contain a description of the functions provided by the AUTOSAR NvM basic software component with API Configuration Class 3 configured.

##### Sub-Function: NvM\_SetBlockProtection

###### Hardware Related Design

None

###### Software Related Design

|  |  |
| --- | --- |
| Function Particularities | |
| Request Type | Synchronous |
| Re-entrant | Yes |
| Expected Caller Context | Application |

The request sets the write protection for the NV block. Any further write, erase, and invalidate requests to the NVRAM block is rejected synchronously if the NV block-write protection is set. The data area of the RAM block remains writeable in any case.

Application Function Definition

|  |
| --- |
| Prototype |
| Std\_ReturnType Rte\_Call\_<SWCSvcPortName>\_SetBlockProtection ( Boolean ProtectionEnabled ) |
| Parameter |
| ProtectionEnabled – True: Enable protection. False: Disable protection |
| Return Code |
| Std\_ReturnType – See section 5.1.4.1 definition of this return type |

CDD Function Definition

|  |
| --- |
| Prototype |
| Std\_ReturnType NvM\_ SetBlockProtection ( NvMBlockIdType BlockId, Boolean ProtectionEnabled ) |
| Parameter |
| BlockId – The block identifier |
| ProtectionEnabled – True: Enable protection. False: Disable protection |
| Return Code |
| Std\_ReturnType – See section 5.1.4.1 definition of this return type |

##### Sub-Function: NvM\_EraseNvBlock

###### Hardware Related Design

None

###### Software Related Design

|  |  |
| --- | --- |
| Function Particularities | |
| Request Type | Asynchronous |
| Re-entrant | Yes |
| Expected Caller Context | Application |

Request to erase the specified NV block. This function queues the erase request and returns the acceptance result synchronously. The NvM can notify the application by callback when the service is finished. This function performs the same action as NvM\_InvalidateNvBlock for FEE.

Application Function Definition

|  |
| --- |
| Prototype |
| Std\_ReturnType Rte\_Call\_<SWCSvcPortName>\_EraseNvBlock ( void ) |
| Parameter |
| None |
| Return Code |
| Std\_ReturnType – See section 5.1.4.1 definition of this return type |

CDD Function Definition

|  |
| --- |
| Prototype |
| Std\_ReturnType NvM\_EraseNvBlock ( NvMBlockIdType BlockId ) |
| Parameter |
| BlockId – The block identifier |
| Return Code |
| Std\_ReturnType – See section 5.1.4.1 definition of this return type |

##### Sub-Function: NvM\_InvalidateNvBlock

###### Hardware Related Design

None

###### Software Related Design

|  |  |
| --- | --- |
| Function Particularities | |
| Request Type | Asynchronous |
| Re-entrant | Yes |
| Expected Caller Context | Application |

Request to invalidate the specified NV block. This function queues the erase request and returns the acceptance result synchronously. The NvM can notify the application by callback when the service is finished.

Application Function Definition

|  |
| --- |
| Prototype |
| Std\_ReturnType Rte\_Call\_<SWCSvcPortName>\_InvalidateNvBlock ( void ) |
| Parameter |
| None |
| Return Code |
| Std\_ReturnType – See section 5.1.4.1 definition of this return type |

CDD Function Definition

|  |
| --- |
| Prototype |
| Std\_ReturnType NvM\_InvalidateNvBlock ( NvMBlockIdType BlockId ) |
| Parameter |
| BlockId – The block identifier |
| Return Code |
| Std\_ReturnType – See section 5.1.4.1 definition of this return type |

### Type Definitions

The following section outlines the data types used by the NvM component.

#### Std\_ReturnType

|  |  |  |
| --- | --- | --- |
| C-Type | Description | Value Range |
| uint8 | Standard return type for functions | NVM\_REQ\_OK (0)  *The last request has been accepted* |
|  |  | NVM\_REQ\_NOT\_OK (1)  *The last request has not been accepted* |

#### NvM\_RequestResultType

|  |  |  |
| --- | --- | --- |
| C-Type | Description | Value Range |
| uint8 | An asynchronous API service can have the following results or status that can be polled by NvM\_GetErrorStatus | NVM\_REQ\_OK (0)  *The last request has finished successfully.* |
| NVM\_REQ\_NOT\_OK (1)  *The last request has finished unsuccessfully* |
| NVM\_REQ\_PENDING (2)  *The last request is currently being processed* |
| NVM\_REQ\_INTEGRITY\_FAILED (3)  *A NV block with data corruption caused by a CRC mismatch or FEE or EA components reported an inconsistency* |
| NVM\_REQ\_BLOCK\_SKIPPED (4)  *The block was skipped during a multi-block request (ReadAll or WriteAll requests)* |
| NVM\_REQ\_NV\_INVALIDATED (5)  *The block is marked as invalid* |
| NVM\_REQ\_CANCELLED (6) |

#### NvM\_BlockIdType

|  |  |  |
| --- | --- | --- |
| C-Type | Description | Value Range |
| uint16 | It is the type of a block handle that is used by the application in order to access a NVM block. There are two reserved IDs:   * Block ID 0 for multi block requests (Block ID 0 is only allowed for API NvM\_GetErrorStatus()) * Block ID 1 for the configuration Id block.   The block handles are created as defines in an ascending define list. | 0..(2^(16 - *Dataset Selection Bits*) – 1)  *Dataset Selection Bits* is the maximum number of bits that are needed in order to store the maximum dataset value as is configured by the software integration team.  Following the example from section 5.1.1.1 where *Dataset Selection Bits* was equal to 5, the max number of block handles would be 2^(16-5)-1=2047 |

## NvM\_Proxy: Non Volatile Memory Proxy (Nexteer CDD)

### Design Rationale

The NvM\_Proxy provides and interface to the NvM for all software components. The interface shall consist of the API modules defined in the previous sections for the AUTOSAR NvM component with the prefix of NvMProxy instead of NvM.

If the calling component is in the same ASIL application as the NvM, the proxy shall just forward the request on directly. If the component is not in the same ASIL application as the NvM, the proxy shall call a nontrusted function to switch the OS context to the application of the NvM and then call the desired function.



The NvMProxy component shall also be responsible for settings NTC 0x06, 0x07, 0x08 and 0x0A when a block that is invalid by not being written or when the CRC check fails if one is applied to the block. This check shall be done during the initialization routine of the NvMProxy component.

NvMProxy shall also store all RAM status flags during shutdown to RAM, so they can be restored during a quick ignition cycle. The restoration shall take place if the NvM\_WriteAll is completed, canceled or killed by the BswM.

### Sub-Functions

#### Sub-Function: NvMProxy\_Init

##### Hardware Related Design

None

##### Software Related Design

|  |  |
| --- | --- |
| Function Particularities | |
| Request Type | Synchronous |
| Re-entrant | Yes |
| Expected Caller Context | Shall only be called from ECU state manager or equivalent function. |

Before the NvMProxy component can be used, it has to be initialized. This call also must be performed after the NvM\_ReadAll() has finished executing to properly diagnose all NvM blocks for any data corruption.

# Timing / Execution Constraints

## Rationale / Comments

## Rates and State Execution: NvM

In the following table, functions marked with 0ms run rate are function calls that can be called from the states indicated by “Execute.”

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sub-Function Name** | **Rate (ms)** | **Cold Init** | **Warm Init** | **Operate** | **Disable** |
| NvM\_Init | 0 | Execute | Do Not Execute | Do Not Execute | Do Not Execute |
| NvM\_MainFunction | 10 | Execute\* | Execute | Execute | Execute |
| NvM\_ReadAll | 0 | Execute\* | Do Not Execute | Do Not Execute | Do Not Execute |
| NvM\_WriteAll | 0 | Do Not Execute | Do Not Execute | Do Not Execute | Execute |
| NvM\_GetErrorStatus | 0 | Execute\*\* | Execute | Execute | Execute |
| NvM\_SetRamBlockStatus | 0 | Execute\*\* | Execute | Execute | Execute |
| NvM\_CancelWriteAll | 0 | Do Not Execute | Do Not Execute | Do Not Execute | Execute |
| NvM\_SetDataIndex | 0 | Execute\*\* | Execute | Execute | Execute |
| NvM\_GetDataIndex | 0 | Execute\*\* | Execute | Execute | Execute |
| NvM\_ReadBlock | 0 | Execute\*\* | Execute | Execute | Execute |
| NvM\_WriteBlock | 0 | Execute\*\* | Execute | Execute | Execute |
| NvM\_RestoreBlockDefaults | 0 | Execute\*\* | Execute | Execute | Execute |
| NvM\_CancelJobs | 0 | Execute\*\* | Execute | Execute | Execute |
| NvM\_SetBlockProtection | 0 | Execute\*\* | Execute | Execute | Execute |
| NvM\_EraseNvBlock | 0 | Execute\*\* | Execute | Execute | Execute |
| NvM\_InvalidateNvBlock | 0 | Execute\*\* | Execute | Execute | Execute |
| *\* Execution is allowed, but only after NvM\_Init() has completed.*  *\*\* Execution is allowed, but only after NvM\_ReadAll() has completed.* | | | | | |

## Rates and State Execution: NvMProxy

In the following table, functions marked with 0ms run rate are function calls that can be called from the states indicated by “Execute.”

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sub-Function Name** | **Rate (ms)** | **Cold Init** | **Warm Init** | **Operate** | **Disable** |
| NvMProxy\_Init | 0 | Execute\*\* | Do Not Execute | Do Not Execute | Do Not Execute |
| NvMProxy\_GetErrorStatus | 0 | Execute\* | Execute | Execute | Execute |
| NvMProxy\_SetRamBlockStatus | 0 | Execute\* | Execute | Execute | Execute |
| NvMProxy\_CancelWriteAll | 0 | Do Not Execute | Do Not Execute | Do Not Execute | Execute |
| NvMProxy\_SetDataIndex | 0 | Execute\* | Execute | Execute | Execute |
| NvMProxy\_GetDataIndex | 0 | Execute\* | Execute | Execute | Execute |
| NvMProxy\_ReadBlock | 0 | Execute\* | Execute | Execute | Execute |
| NvMProxy\_WriteBlock | 0 | Execute\* | Execute | Execute | Execute |
| NvMProxy\_RestoreBlockDefaults | 0 | Execute\* | Execute | Execute | Execute |
| NvMProxy\_CancelJobs | 0 | Execute\* | Execute | Execute | Execute |
| NvMProxy\_SetBlockProtection | 0 | Execute\* | Execute | Execute | Execute |
| NvMProxy\_EraseNvBlock | 0 | Execute\* | Execute | Execute | Execute |
| NvMProxy\_InvalidateNvBlock | 0 | Execute\* | Execute | Execute | Execute |
| *\* Execution is allowed, but only after NvMProxy\_Init() has completed.*  *\*\* Execution is allowed, but only after NvM\_ReadAll() has completed.* | | | | | |

# Serial Communications Interfaces

None

# Additional Information

## NvM block definition Considerations

NvM block can be made from a single value to multiple values grouped as one element. It is up to the function designer to determine how their NvM blocks should be structured, but the following considerations should be considered when defining an NvM block.

### Scenario 1

Consider a design with 3 PIMs defined for storage in NvM. If the PIMs were defined in the m-files as separate entries, the software component developer would create three separate NvM blocks for those pieces of data as shown below.



This would mean that each PIM, is independent of the other PIMs, however, it is a waste of FEE memory if the data does not truly need to be independent. Instead the three values should be grouped into a structure so the NvM block created looks like below.



This provides the designer with the same design (three separate variables) but the data is grouped together to make one CRC to cover all three values and only requires one instance of the FEE header saving FEE space.

### Scenario 2

Consider a design with 3 PIMs, 2 of the PIMs are written during manufacturing and the other PIM is learned during function execution and saved at shutdown. In this case, it would make sense to group the PIMs by functionality.



This saves FEE space by grouping like content, but also protects the data that is written during manufacturing from being discarded in case the data that is written during shutdown encountered a power loss or other type of data corruption. Only the learned data would be considered invalid and the other block would be valid.

## Software Component Design Considerations

### API Port Selection

The NvM component will generate different API ports depending on the configuration required a given block. It shall be up to the developer of the software component to configure the service port with the correct API during development to ensure a proper match with the integration project.

Based on the settings in section 5.1.1, the API port will begin with “NvMService\_AC3\_SRBS”. This is because the API class is level 3 (AC3) and the set ram block status API is enable (SRBS).

The following table is for all API access except for NvM\_ SetBlockProtection.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Configuration Matrix | | | | Suffix | Port Name to Use in SWC |
| Block Type | | | ROM Default |
| Native | Redundant | Dataset |
| Y | N | N | N | N/A | NvMService\_AC3\_SRBS |
| Y | N | N | Y | \_Defs | NvMService\_AC3\_SRBS\_Defs |
| N | Y | N | N | N/A | NvMService\_AC3\_SRBS |
| N | Y | N | Y | \_Defs | NvMService\_AC3\_SRBS\_Defs |
| N | N | Y | N | \_DS | NvMService\_AC3\_SRBS\_DS |
| N | N | Y | Y | \_DS\_Defs | NvMService\_AC3\_SRBS\_DS\_Defs |

The following table is only for API to NvM\_ SetBlockProtection.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Configuration Matrix | | | | Suffix | Port Name to Use in SWC |
| Block Type | | | ROM Default |
| Native | Redundant | Dataset |
| Y | N | N | N | N/A | NvMAdministration |
| Y | N | N | Y |
| N | Y | N | N |
| N | Y | N | Y |
| N | N | Y | N |
| N | N | Y | Y |

# Revision Record & Change Approval

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Rev** | **Date** | **Change Control #** | **Drw** | **Change Description** |
| 01.00.00 | 01Sep15 | EA4#471 | KJS | Initial Release |
| 02.00.00 | 01Oct16 | EA4#7778 | KJS | Updates for quick ignition cycle handling |