

<b>Document Title</b>	Specification of Bulk NvData Manager
<b>Document Owner</b>	AUTOSAR
<b>Document Responsibility</b>	AUTOSAR
<b>Document Identification No</b>	949

<b>Document Status</b>	published
<b>Part of AUTOSAR Standard</b>	Classic Platform
<b>Part of Standard Release</b>	R21-11

<b>Document Change History</b>			
<b>Date</b>	<b>Release</b>	<b>Changed by</b>	<b>Description</b>
2021-11-25	R21-11	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>• No content changes</li> </ul>
2020-11-30	R20-11	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>• callback functions renamed</li> <li>• limitation added</li> <li>• specification item prefix adapted</li> <li>• editorial changes</li> </ul>
2019-11-28	R19-11	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>• Initial release</li> </ul>

## Disclaimer

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

## Table of Contents

1	Introduction and functional overview	5
2	Acronyms and Abbreviations	5
3	Related documentation	5
3.1	Input documents & related standards and norms	5
3.2	Related specification	5
4	Constraints and assumptions	6
4.1	Limitations	6
4.2	Applicability to car domains	6
5	Dependencies to other modules	6
6	Requirements Tracing	6
7	Functional specification	6
7.1	Error Classification	9
7.1.1	Development Errors	9
7.1.2	Runtime Errors	9
7.1.3	Transient Faults	9
7.1.4	Production Errors	9
7.1.5	Extended Production Errors	9
8	API specification	10
8.1	Imported types	10
8.2	Type definitions	10
8.2.1	BndM_ConfigType	10
8.2.2	BndM_BlockIdType	10
8.2.3	BndM_Block<BlockId.Shortname>Type	10
8.2.4	BndM_Result	11
8.3	Function definitions	11
8.3.1	BndM_Init	11
8.3.2	BndM_GetVersionInfo	12
8.3.3	BndM_GetBlockPtr	12
8.3.4	BndM_WriteStart	12
8.3.5	BndM_WriteBlock	13
8.3.6	BndM_WriteFinalize	14
8.3.7	BndM_WriteCancel	14
8.4	Callback notifications	14
8.4.1	BndM_JobEndNotification	15
8.4.2	BndM_JobErrorNotification	15
8.5	Scheduled functions	15
8.5.1	BndM_MainFunction	16
8.6	Expected interfaces	16

8.6.1	Mandatory interfaces	16
8.6.2	Optional interfaces	16
8.6.3	Configurable interfaces	17
8.6.3.1	xxx_BndMWriteStartFinish	17
8.6.3.2	xxx_BndMWriteBlockFinish	17
8.6.3.3	xxx_BndMWriteFinalizeFinish	18
8.7	Service Interfaces	18
9	Sequence diagrams	19
10	Configuration specification	19
10.1	How to read this chapter	19
10.2	Containers and configuration parameters	19
10.2.1	BndM	19
10.2.2	BndMGeneral	19
10.2.3	BndMBlockDescriptor	21
10.2.4	BndMCallbackBlock	22
10.2.5	BndMCallbackGeneral	23
10.3	Published Information	24
A	Not applicable requirements	24

# 1 Introduction and functional overview

This specification describes the functionality, API and the configuration for the AUTOSAR Basic Software module BulkNvDataManager.

The demand of non-volatile bulk data is increasing for use-case like variant-coding<sup>1</sup>. Such data is used frequently, but rarely updated. The BulkNvDataManager offers in contrast to the NvM an API to read the data directly from flash memory. In consequence a RAM mirror is avoided, but the writing of the data is more complex.

**Remark:** The whole memory stack in AUTOSAR Classic Platform will have a systematic review within the upcoming release 20/11. This could result in a changed architecture like the integration of the BndM functionality into NvM.

## 2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant to the BulkNvDataManager that are not included in the [1, AUTOSAR glossary].

## 3 Related documentation

### 3.1 Input documents & related standards and norms

- [1] Glossary  
AUTOSAR\_TR\_Glossary
- [2] General Specification of Basic Software Modules  
AUTOSAR\_SWS\_BSWGeneral

### 3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [2, SWS BSW General], which is also valid for BulkNvDataManager.

Thus, the specification SWS BSW General shall be considered as additional and required specification for BulkNvDataManager.

---

<sup>1</sup>Variant coding is a vehicle specific dataset which is calculated in the production for each vehicle (and of course stored in the production).

## 4 Constraints and assumptions

### 4.1 Limitations

The synchronization of a potential parallel access (e.g. FlashEEPROMEmulation) to the underlying flash driver is not part of this AUTOSAR release.

Currently only PFlash writing with A/B Sector switch, present in high end microcontrollers, is supported. This limits the applicability of BndM to architectures supporting this feature.

### 4.2 Applicability to car domains

## 5 Dependencies to other modules

This module depends on the capabilities of the underlying flash driver.

## 6 Requirements Tracing

The following tables reference the requirements specified in <CITATIONS\_OF\_CONTRIBUTED\_DOCUMENTS> and links to the fulfillment of these. Please note that if column “Satisfied by” is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[RS_Diag_04243]	Update of constant parameters through diagnostics	<a href="#">[SWS_BndM_00001]</a> <a href="#">[SWS_BndM_00002]</a> <a href="#">[SWS_BndM_00003]</a> <a href="#">[SWS_BndM_00004]</a> <a href="#">[SWS_BndM_00005]</a> <a href="#">[SWS_BndM_00007]</a> <a href="#">[SWS_BndM_00008]</a> <a href="#">[SWS_BndM_00009]</a> <a href="#">[SWS_BndM_00010]</a> <a href="#">[SWS_BndM_00011]</a> <a href="#">[SWS_BndM_00012]</a> <a href="#">[SWS_BndM_00013]</a> <a href="#">[SWS_BndM_00014]</a>

## 7 Functional specification

In general the concept how the BulkNvDataManager will manage its flash memory is vendor-specific.

The base idea is to have an A/B switching of the data blocks. This means the complete Bulk NvData will be stored in partition A. When the writing is started ([BndM\\_WriteStart](#)) the B partition needs to be erased. The updated blocks ([BndM\\_WriteBlock](#)) will be written to partition B. The finalization ([BndM\\_WriteFinalize](#)) will finally make partition B consistent (e.g. by coping the not updated blocks over to partition B) and switch the active partition to B (further calls to [BndM\\_GetBlockPtr](#) will point to the data in the partition B). Nevertheless the vendor solution could consider alternative solutions like an update through a FlashBootloader.

**[SWS\_BndM\_00001]** [The BndM shall manage its BndM blocks ([BndMBlockDescriptor](#)) in the direct accessible memory (i.e. via pointer).] ([RS\\_Diag\\_04243](#))

**[SWS\_BndM\_00002]** [A call of [BndM\\_GetBlockPtr](#) shall deliver the base pointer to the corresponding BndM block ([BndMBlockDescriptor](#)) in the currently active partition.] ([RS\\_Diag\\_04243](#))

**[SWS\_BndM\_00003]** [A call of [BndM\\_WriteStart](#) shall trigger the preparation of the 2nd (free) partition.] ([RS\\_Diag\\_04243](#))

Note: Depending on the implemented strategy the preparation takes more time. This could be coordinated within the [BndM\\_MainFunction](#). Note: In case of direct writing access to flash the flash-page needs to be erased.

Caveat: Depending on the hardware a parallel read and write access to code flash is not possible. In this case the overall ECU needs to be in a writing mode (e.g. Flash-Bootloader context or all other tasks are interrupted/stopped).

**[SWS\_BndM\_00007]** [After preparation of the 2nd (free) partition [\[SWS\\_BndM\\_00003\]](#) is successfully finished (writing to the 2nd partition is possible) the callback [Xxx\\_BndMWriteStartFinish](#) with the result set to `E_OK` shall be triggered in the context of the [BndM\\_MainFunction](#).] ([RS\\_Diag\\_04243](#))

**[SWS\_BndM\_00014]** [A call of [BndM\\_WriteStart](#) shall be rejected with the error-Code `E_NOT_OK`, if the call is done within an active writing phase (phase between [BndM\\_WriteStart](#) and [BndM\\_WriteFinalize](#)).] ([RS\\_Diag\\_04243](#))

**[SWS\_BndM\_00004]** [A call of [BndM\\_WriteBlock](#) shall trigger the writing of the data to the 2nd (unused) partition. The data (`ImplementationDataType`) shall be not modified to allow a pointer access.] ([RS\\_Diag\\_04243](#))

**[SWS\_BndM\_00008]** [After writing of [\[SWS\\_BndM\\_00004\]](#) the 2nd (free) partition is finished the callback [Xxx\\_BndMWriteBlockFinish](#) with the result set to `E_OK` shall be triggered in the context of the [BndM\\_MainFunction](#).] ([RS\\_Diag\\_04243](#))

**[SWS\_BndM\_00011]** [A call of [BndM\\_WriteBlock](#) shall be rejected with the error-Code `E_NOT_OK`, if the call is done without a previous call of [BndM\\_WriteStart](#). or while another writing of the same or another block is ongoing or the call is done within the finalization mode of the BndM.] ([RS\\_Diag\\_04243](#))

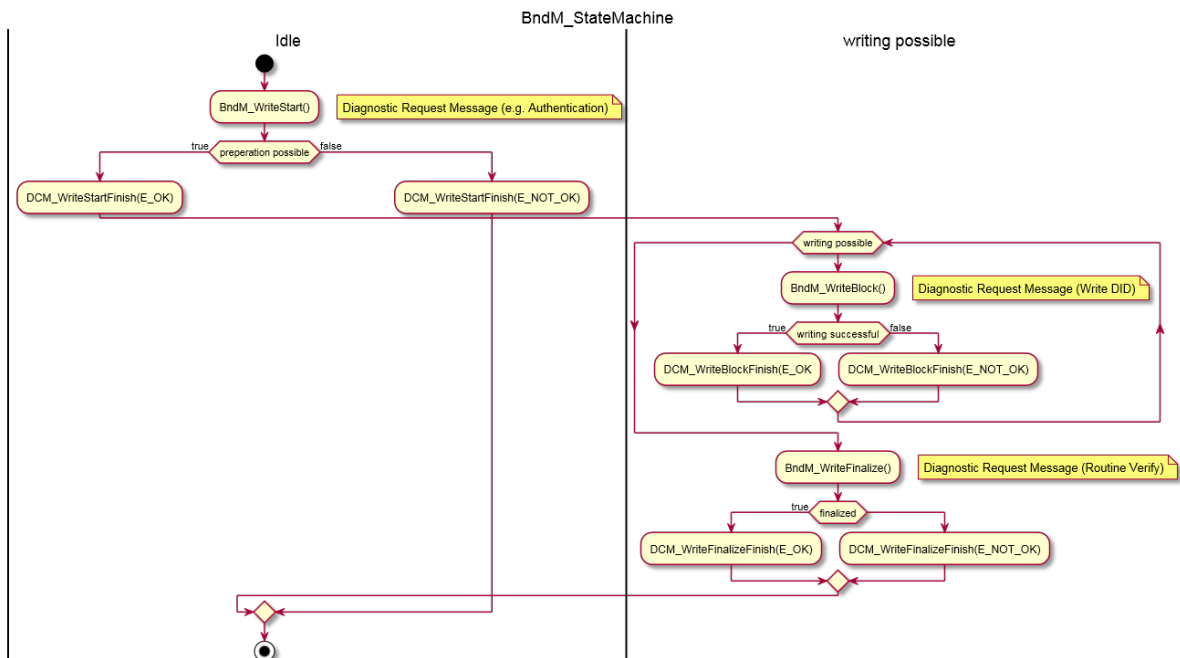
**[SWS\_BndM\_00012]** [A call of `BndM_WriteBlock` shall be rejected with the error-Code `E_NOT_OK`, if the call is done while another writing of the same or another block is ongoing.] (*RS\_Diag\_04243*)

**[SWS\_BndM\_00013]** [A call of `BndM_WriteBlock` shall be rejected with the error-Code `E_NOT_OK`, if the call is done within or after the finalization mode of the BndM.] (*RS\_Diag\_04243*)

**[SWS\_BndM\_00005]** [A call of `BndM_WriteFinalize` shall trigger the finalization of the 2nd (unused) partition. In background the BndM shall make the 2nd (unused) partition consistent by coping all unchanged `BndMBlockDescriptor` to the 2nd (unused) partition. If the finalization is successful the BndM shall make the 2nd (unused) partition to the active partition and trigger the callback `Xxx_BndMWriteFinalizeFinish` with the result set to `E_OK`.] (*RS\_Diag\_04243*) Note: Further calls to `BndM_GetBlockPtr` will point to the data in the 2nd (now active) partition after the finalization is successful.

**[SWS\_BndM\_00009]** [If the finalization is NOT successful (the 2nd partition is not consistent and could therefore not be used) the BndM shall keep the current active partition as the active partition and trigger the callback `Xxx_BndMWriteFinalizeFinish` with the result set to `E_NOT_OK`.] (*RS\_Diag\_04243*)

**[SWS\_BndM\_00010]** [A call of `BndM_WriteFinalize` without a previously called `BndM_WriteStart` or within the finalization mode of the BndM the DET `BndM_E_WRONG_SEQUENCE` error shall be thrown.] (*RS\_Diag\_04243*)



**Figure 7.1: Figure BndMStateMachine**



## 7.1 Error Classification

Section 7.2 "Error Handling" of the document "General Specification of Basic Software Modules" describes the error handling of the Basic Software in detail. Above all, it constitutes a classification scheme consisting of five error types which may occur in BSW modules.

Based on this foundation, the following section specifies particular errors arranged in the respective subsections below.

### 7.1.1 Development Errors

[SWS\_BndM\_00006] [

<i>Type of error</i>	<i>Related error code</i>	<i>Error value</i>
API service called with wrong parameter	BNDM_E_PARAM	0x01
API called in wrong sequence	BNDM_E_WRONG_SEQUENCE	0x02

]()

### 7.1.2 Runtime Errors

There are no runtime errors.

### 7.1.3 Transient Faults

There are no transient faults.

### 7.1.4 Production Errors

There are no production errors.

### 7.1.5 Extended Production Errors

There are no extended production errors.

## 8 API specification

### 8.1 Imported types

In this chapter all types included from the following files are listed.

### 8.2 Type definitions

#### 8.2.1 BndM\_ConfigType

[SWS\_BndM\_01001] [

<b>Name</b>	BndM_ConfigType		
<b>Kind</b>	Structure		
<b>Elements</b>	implementation specific		
	<b>Type</b>	-	
	<b>Comment</b>	-	
<b>Description</b>	This type of the external data structure shall contain the post build initialization data for the BndM.		
<b>Available via</b>	bndm.h		

]()

#### 8.2.2 BndM\_BlockIdType

[SWS\_BndM\_01002] [

<b>Name</b>	BndM_BlockIdType		
<b>Kind</b>	Type		
<b>Derived from</b>	uint16		
<b>Range</b>	0..65535	-	-
<b>Description</b>	Unique identification of an bulk nv block. The BndM_BlockId is assigned by the BndM.		
<b>Available via</b>	bndm.h		

]()

#### 8.2.3 BndM\_Block<BlockId.Shortname>Type

[SWS\_BndM\_01003] [

<b>Name</b>	BndM_Block{BlockId.Shortname}Type
<b>Kind</b>	Structure
<b>Description</b>	The elements of this structure data type is the C-structured representation of the configured ImplementationDataPrototype.
<b>Available via</b>	bndm_externals.h

}]()

## 8.2.4 BndM\_Result

[SWS\_BndM\_01017] [

<b>Name</b>	BndM_ResultType		
<b>Kind</b>	Type		
<b>Derived from</b>	uint8		
<b>Range</b>	E_OK	0x00	Result of the asynchronous job finish notifications
	E_NOT_OK	0x01	-
<b>Description</b>	Result of the asynchronous job finish notifications		
<b>Available via</b>	bndm.h		

}]()

## 8.3 Function definitions

### 8.3.1 BndM\_Init

[SWS\_BndM\_01004] [

<b>Service Name</b>	BndM_Init		
<b>Syntax</b>	<pre>void BndM_Init (     const BndM_ConfigType* ConfigPtr )</pre>		
<b>Service ID [hex]</b>	0x1		
<b>Sync/Async</b>	Synchronous		
<b>Reentrancy</b>	Non Reentrant		
<b>Parameters (in)</b>	ConfigPtr	Pointer to the configuration set in VARIANT-POST-BUILD.	
<b>Parameters (inout)</b>	None		
<b>Parameters (out)</b>	None		
<b>Return value</b>	None		
<b>Description</b>	Initializes or reinitializes this module.		
<b>Available via</b>	BndM.h		

}]()

### 8.3.2 BndM\_GetVersionInfo

[SWS\_BndM\_01005] [

<b>Service Name</b>	BndM_GetVersionInfo	
<b>Syntax</b>	<pre>void BndM_GetVersionInfo (     Std_VersionInfoType* versioninfo )</pre>	
<b>Service ID [hex]</b>	0x2	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non 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>	Returns the version information of this module. API Availability: This API will be available only if (ecuc BndM/BndMGeneral.BndMVersionInfoApi) == true)	
<b>Available via</b>	BndM.h	

]()

### 8.3.3 BndM\_GetBlockPtr

[SWS\_BndM\_01006] [

<b>Service Name</b>	BndM_GetBlockPtr_<BlockId.Shortname>	
<b>Syntax</b>	<pre>Std_ReturnType BndM_GetBlockPtr_&lt;BlockId.Shortname&gt; (     BndM_BlockIdType BlockId,     BndM_Block{BlockId.Shortname}Type** BndM_BlockPtr )</pre>	
<b>Service ID [hex]</b>	0x3	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant Returns an pointer to the structure in flash	
<b>Parameters (in)</b>	BlockId	BlockId
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	BndM_BlockPtr	• BndM_BlockPtr
<b>Return value</b>	Std_ReturnType	-
<b>Description</b>	-	
<b>Available via</b>	BndM_Externals.h	

]()

### 8.3.4 BndM\_WriteStart

[SWS\_BndM\_01007] [

<b>Service Name</b>	BndM_WriteStart	
<b>Syntax</b>	Std_ReturnType BndM_WriteStart ( void )	
<b>Service ID [hex]</b>	0x4	
<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>	Std_ReturnType	E_OK The preparation request is accepted. E_NOT_OK The preparation request is rejected.
<b>Description</b>	Will trigger the start of writing phase. The finish of asynchronous processing will trigger the callback xxx_BndMWriteStartFinish including the result of this operation	
<b>Available via</b>	BndM.h	

]()

Note: It is up to the stack-vendor what can run in parallel while the writing to BndM is possible or not (e.g. FEE might not work anymore).

### 8.3.5 BndM\_WriteBlock

[SWS\_BndM\_01008] [

<b>Service Name</b>	BndM_WriteBlock_<BlockId.Shortname>	
<b>Syntax</b>	Std_ReturnType BndM_WriteBlock_<BlockId.Shortname> ( BndM_BlockIdType BlockId, const BndM_Block{BlockId.Shortname}Type* BndM_SrcPtr )	
<b>Service ID [hex]</b>	0x5	
<b>Sync/Async</b>	Asynchronous	
<b>Reentrancy</b>	Non Reentrant	
<b>Parameters (in)</b>	BlockId	-
	BndM_SrcPtr	-
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	Std_ReturnType	E_OK The write request is accepted. E_NOT_OK The write request is rejected.
<b>Description</b>	Will persist the data in flash, that it can later directly accessed via BndM_GetBlockPtr API. The writing take a while and is finished after the successful callback xxx_BndMWriteBlockFinish	
<b>Available via</b>	BndM_Externals.h	

]()

Note: BndM\_WriteStart needs to be called in advance

### 8.3.6 BndM\_WriteFinalize

[SWS\_BndM\_01009] [

<b>Service Name</b>	BndM_WriteFinalize	
<b>Syntax</b>	Std_ReturnType BndM_WriteFinalize ( void )	
<b>Service ID [hex]</b>	0x6	
<b>Sync/Async</b>	Asynchronous	
<b>Reentrancy</b>	Reentrant Finalize the writing. After the successful callback xxx_BndMWriteFinalizeFinish the finalization is finished (i.e. the new stored data is available).	
<b>Parameters (in)</b>	None	
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	Std_ReturnType	E_OK The finalization request is accepted. E_NOT_OK The finalization request is rejected.
<b>Description</b>	Will trigger the finalization of writing phase. The finish of asynchronous processing will trigger the callback xxx_BndMWriteFinalizeFinish including the result of this operation.	
<b>Available via</b>	BndM.h	

]()

### 8.3.7 BndM\_WriteCancel

[SWS\_BndM\_01010] [

<b>Service Name</b>	BndM_WriteCancel	
<b>Syntax</b>	void BndM_WriteCancel ( void )	
<b>Service ID [hex]</b>	0x7	
<b>Sync/Async</b>	Asynchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	None	
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	Cancels the writing	
<b>Available via</b>	BndM.h	

]()

## 8.4 Callback notifications

This is a list of functions provided for FLS module.

### 8.4.1 BndM\_JobEndNotification

[SWS\_BndM\_01011] [

<b>Service Name</b>	BndM_JobEndNotification
<b>Syntax</b>	<pre>void BndM_JobEndNotification (     void )</pre>
<b>Service ID [hex]</b>	0x8
<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>	This callback function is called when a FLS job has been finished with positive result.
<b>Available via</b>	BndM.h

]()

### 8.4.2 BndM\_JobErrorNotification

[SWS\_BndM\_01012] [

<b>Service Name</b>	BndM_JobErrorNotification
<b>Syntax</b>	<pre>void BndM_JobErrorNotification (     void )</pre>
<b>Service ID [hex]</b>	0x9
<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>	This callback function is called when a FLS job has been canceled or finished with negative result.
<b>Available via</b>	BndM.h

]()

## 8.5 Scheduled functions

These functions are directly called by Basic Software Scheduler. The following functions shall have no return value and no parameter. All functions shall be non re-entrant.

## 8.5.1 BndM\_MainFunction

[SWS\_BndM\_01013] [

<b>Service Name</b>	BndM_MainFunction
<b>Syntax</b>	<pre>void BndM_MainFunction (     void )</pre>
<b>Service ID [hex]</b>	0x55
<b>Description</b>	Schedule function for the background processing.
<b>Available via</b>	SchM_BndM.h

]()

## 8.6 Expected interfaces

In this chapter all interfaces required from other modules are listed.

### 8.6.1 Mandatory interfaces

[SWS\_BndM\_01019] [

<b>API Function</b>	<b>Header File</b>	<b>Description</b>
There are no mandatory interfaces.		

]()

Note: This section defines all interfaces, which are required to fulfill the core functionality of the module.

### 8.6.2 Optional interfaces

This section defines all interfaces, which are required to fulfill an optional functionality of the module.

[SWS\_BndM\_01020] [

<b>API Function</b>	<b>Header File</b>	<b>Description</b>
Det_ReportRuntimeError	Det.h	Service to report runtime errors. If a callout has been configured then this callout shall be called.
Fls_Cancel	Fls.h	Cancels an ongoing job.







API Function	Header File	Description
Fls_Compare	Fls_Com.h	Compares the contents of an area of flash memory with that of an application data buffer.
Fls_Erase	Fls.h	Erases flash sector(s).
Fls_GetJobResult	Fls.h	Returns the result of the last job.
Fls_GetStatus	Fls.h	Returns the driver state.
Fls_Read	Fls.h	Reads from flash memory.
Fls_SetMode	Fls.h	Sets the flash driver's operation mode.
Fls_Write	Fls.h	Writes one or more complete flash pages.

]()

### 8.6.3 Configurable interfaces

In this section, all interfaces are listed where the target function could be configured. The target function is usually a callback function. The names of this kind of interfaces are not fixed because they are configurable.

#### 8.6.3.1 xxx\_BndMWriteStartFinish

[SWS\_BndM\_01016] [

<b>Service Name</b>	Xxx_BndMWriteStartFinish	
<b>Syntax</b>	<pre>void Xxx_BndMWriteStartFinish (     BndM_BlockIdType BlockId,     BndM_ResultType result )</pre>	
<b>Service ID [hex]</b>	0x56	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non Reentrant	
<b>Parameters (in)</b>	BlockId	-
	result	-
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	This callback function is called when BndM_WriteStart is finished.	
<b>Available via</b>	BndM_Externals.h	

]()

#### 8.6.3.2 xxx\_BndMWriteBlockFinish

[SWS\_BndM\_01014] [

<b>Service Name</b>	Xxx_BndMWriteBlockFinish	
<b>Syntax</b>	<pre>void Xxx_BndMWriteBlockFinish (     BndM_BlockIdType BlockId,     BndM_ResultType result )</pre>	
<b>Service ID [hex]</b>	0x57	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non Reentrant	
<b>Parameters (in)</b>	BlockId	-
	result	-
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	This callback function is called when BndM_WriteBlock is finished.	
<b>Available via</b>	BndM_Externals.h	

]()

### 8.6.3.3 xxx\_BndMWriteFinalizeFinish

[SWS\_BndM\_01015] [

<b>Service Name</b>	Xxx_BndMWriteFinalizeFinish	
<b>Syntax</b>	<pre>void Xxx_BndMWriteFinalizeFinish (     BndM_BlockIdType BlockId,     BndM_ResultType result )</pre>	
<b>Service ID [hex]</b>	0x58	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non Reentrant	
<b>Parameters (in)</b>	BlockId	-
	result	-
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	This callback function is called when BndM_WriteFinalize is finished.	
<b>Available via</b>	BndM_Externals.h	

]()

## 8.7 Service Interfaces

The BndM does not have service interfaces.

## 9 Sequence diagrams

## 10 Configuration specification

In general, this chapter defines configuration parameters and their clustering into containers. In order to support the specification Chapter 10.1 describes fundamentals. It also specifies a template (table) you shall use for the parameter specification. We intend to leave Chapter 10.1 in the specification to guarantee comprehension.

Chapter 10.2 specifies the structure (containers) and the parameters of the module BndM.

Chapter 10.3 specifies published information of the module BndM.

### 10.1 How to read this chapter

For details refer to the chapter 10.1 “Introduction to configuration specification” in SWS\_BSWGeneral.

### 10.2 Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters describe Chapter 7 and Chapter 8.

#### 10.2.1 BndM

<b>Module SWS Item</b>	ECUC_BndM_00001	
<b>Module Name</b>	BndM	
<b>Module Description</b>	Configuration of the BulkNvDataManager module.	
<b>Post-Build Variant Support</b>	false	
<b>Supported Config Variants</b>	VARIANT-PRE-COMPILE	
<b>Included Containers</b>		
<b>Container Name</b>	<b>Multiplicity</b>	<b>Scope / Dependency</b>
<a href="#">BndMBlockDescriptor</a>	0..*	Each container defines a Bulk NV Block which can be individually accessed.
<a href="#">BndMCallbackBlock</a>	0..*	This container contains the block-specific callbacks.
<a href="#">BndMCallbackGeneral</a>	0..1	This container contains the general callbacks
<a href="#">BndMGeneral</a>	1	Container for common configuration options.

#### 10.2.2 BndMGeneral

<b>SWS Item</b>	[ECUC_BndM_00002]
<b>Container Name</b>	BndMGeneral
<b>Parent Container</b>	<a href="#">BndM</a>
<b>Description</b>	Container for common configuration options.
<b>Configuration Parameters</b>	

<b>Name</b>	BndMDevErrorDetect [ECUC_BndM_00003]		
<b>Parent Container</b>	<a href="#">BndMGeneral</a>		
<b>Description</b>	Switches the development error detection and notification on or off. <ul style="list-style-type: none"> <li>• true: detection and notification is enabled.</li> <li>• false: detection and notification is disabled.</li> </ul>		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default Value</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: local		

<b>Name</b>	BndMMainFunctionPeriod [ECUC_BndM_00004]		
<b>Parent Container</b>	<a href="#">BndMGeneral</a>		
<b>Description</b>	The period between successive calls to the main function in seconds.		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucFloatParamDef		
<b>Range</b>	]0 .. INF[		
<b>Default Value</b>			
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU		

<b>Name</b>	BndMVersionInfoApi [ECUC_BndM_00005]		
<b>Parent Container</b>	<a href="#">BndMGeneral</a>		
<b>Description</b>	Pre-processor switch to enable / disable the API to read out the modules version information.  true: Version info API enabled. false: Version info API disabled.		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default Value</b>	false		

<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: local		

No Included Containers

### 10.2.3 BndMBlockDescriptor

<b>SWS Item</b>	[ECUC_BndM_00014]
<b>Container Name</b>	BndMBlockDescriptor
<b>Parent Container</b>	<a href="#">BndM</a>
<b>Description</b>	Each container defines a Bulk NV Block which can be individually accessed.
<b>Configuration Parameters</b>	

<b>Name</b>	BndMBlockIdentifier [ECUC_BndM_00007]		
<b>Parent Container</b>	<a href="#">BndMBlockDescriptor</a>		
<b>Description</b>	Unique identification of the block.		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
<b>Range</b>	0 .. 65535		
<b>Default Value</b>			
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: local		

<b>Name</b>	BndMBlockDescriptor [ECUC_BndM_00006]		
<b>Parent Container</b>	<a href="#">BndMBlockDescriptor</a>		
<b>Description</b>	This parameter defines the data structure of the block.		
<b>Multiplicity</b>	1		
<b>Type</b>	Foreign reference to IMPLEMENTATION-DATA-TYPE		
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: local		

<b>Name</b>	BndMCallbackRef [ECUC_BndM_00013]		
<b>Parent Container</b>	<a href="#">BndMBlockDescriptor</a>		
<b>Description</b>	Reference to the block-specific callback function.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	Reference to BndMCallbackBlock		
<b>Post-Build Variant Multiplicity</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: local		

<b>Name</b>	BndMDeviceIndex [ECUC_BndM_00008]		
<b>Parent Container</b>	<a href="#">BndMBlockDescriptor</a>		
<b>Description</b>	Reference to the FLS device this block is stored in.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	Symbolic name reference to FlsGeneral		
<b>Post-Build Variant Multiplicity</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>			

No Included Containers

### 10.2.4 BndMCallbackBlock

<b>SWS Item</b>	[ECUC_BndM_00011]
<b>Container Name</b>	BndMCallbackBlock
<b>Parent Container</b>	<a href="#">BndM</a>
<b>Description</b>	This container contains the block-specific callbacks.
<b>Post-Build Variant Multiplicity</b>	false

<b>Configuration Parameters</b>
---------------------------------

<b>Name</b>	BndMWriteBlockFinishFnc [ECUC_BndM_00012]		
<b>Parent Container</b>	<a href="#">BndMCallbackBlock</a>		
<b>Description</b>	Callback function for the WriteBlockFinish callback.		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucFunctionNameDef		
<b>Default Value</b>			
<b>Regular Expression</b>			
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: local		

<b>No Included Containers</b>
-------------------------------

### 10.2.5 BndMCallbackGeneral

<b>SWS Item</b>	[ECUC_BndM_00015]
<b>Container Name</b>	BndMCallbackGeneral
<b>Parent Container</b>	<a href="#">BndM</a>
<b>Description</b>	This container contains the general callbacks
<b>Configuration Parameters</b>	

<b>Name</b>	BndMWriteFinalizeFinishFnc [ECUC_BndM_00010]		
<b>Parent Container</b>	<a href="#">BndMCallbackGeneral</a>		
<b>Description</b>	Callback function for the WriteFinalizeFinish callback.		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucFunctionNameDef		
<b>Default Value</b>			
<b>Regular Expression</b>			
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: local		

<b>Name</b>	BndMWriteStartFinishFnc [ECUC_BndM_00009]		
<b>Parent Container</b>	<a href="#">BndMCallbackGeneral</a>		
<b>Description</b>	Callback function for the WriteStartFinish callback.		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucFunctionNameDef		
<b>Default Value</b>			
<b>Regular Expression</b>			
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: local		

No Included Containers

### 10.3 Published Information

For details refer to the chapter 10.3 “Published Information” in SWS\_BSWGeneral.

## A Not applicable requirements

[SWS\_BndM\_00999] [These requirements are not applicable to this specification.] ()