VECTOR

MICROSAR Classic BswM

Technical Reference

Version 18.00.01

Authors	visgle, vispet, virpp, visvjn, vispkn, visuca, shagemann
Status	Released

Document Information

History

Author	Date	Version	Remarks
visgle, vispet	2012-08-02	1.00.00	Creation of document.
visgle, vispet	2012-09-27	1.01.00	Addition of feature, support of EthSM . Chapters 2.1, 3.1, 4.2and 5.3.
visgle, vispet	2013-01-31	1.02.00	Addition of feature, support of NvM. Chapter 2.1, 3.1, 4.2 and 4.3.
visgle, vispet	2012-03-26	1.03.00	Support of Post-build variant. Chapters 3.1, 0, 4.1and 4.2. Deviation from AUTOSAR. Header included: Com_Types.h. Chapter 5.1
visgle	2013-10-21	2.00.00	Addition of extension in chapter 5.2. Deletion of limitations in chapter 5.3. DET errors added in chapter 2.7.1. Dynamic files added in chapter 3.1.2. Chapter 0 was changed. Chapter 3.3 was added.
visgle	2013-12-04	2.00.01	Chapter 2.3 was extended. Chapter 2.4.2 was added. Chapter 2.7.1 error code added. Chapter 3.5 was extended Chapter 5.2 was extended.
visgle	2013-02-18	2.01.00	Extended chapters: 2.1, 2.1.2, 2.7.1, 3.1.1, 4.2.18, 4.2.19, 4.2.37, 4.2.38, 4.2.39, 4.2.40 4.3 and 5.2.1. Added chapters: 3.3.3, 4.6, and 5.2.2. Removed deviation about Com_IpduGroupControl usage.
visrpp	2014-06-13	3.00.00	Extended chapters: 2.1.2, 2.5, 4.6.1, 5.2.1, 5.2.8 Added chapters: 4.2.41, 5.2.10, 5.2.11Updated

			Figures: Figure 2-2, Figure 2-3
visrpp	2014-10-22	4.00.00	Extended Chapters: 2.1, 2.7.1, 3.1.1, 3.3.3, 4.2.5 Added chapters: 4.2.42
visrpp	2015-02-02	5.00.00	Extended chapters: 2.7.1, 3.3.3, 5.3.3, 5.3.4 Added chapters: 4.2.22 Removed: Limitation for multiple configurations
visrpp	2015-07-29	6.00.00	Extended chapters: 2.1, 2.1.2, 2.7.1, 3.3.3, 4.3 Added chapters: 3.3.4, 4.2.26, 4.2.31, 4.2.32, 4.2.33, 4.2.34, 4.2.35, 4.2.36
visrpp	2015-12-10	6.00.01	Updated Figure 3-5
visvjn	2016-11-15	7.00.00	Added chapters: 4.2.9 and 4.2.15
visvjn	2017-09-27	8.00.00	Added support for Multi Partition
visvjn	2017-12-20	8.01.00	Extended chapter 2.6
visvjn	2018-06-07	9.00.00	Added support for Kill All Run requests in Ecu State Handling
visvjn	2018-08-14	9.01.00	Updated Figure Figure 3-5 State Machine of the ECU State Handling
vispkn	2019-01-25	10.00.00	Added chapters 2.6, 3.7, 4.2.2 Modified chapters 2.1, 2.2, 2.7.1, 3.1.2
vispkn	2019-07-03	11.00.00	Modified chapter 3.3
vispkn	2019-09-06	12.00.00	Modified chapter 3.3.2
visvjn	2020-01-08	12.01.00	Modified chapter 2.4
visvjn	2020-08-18	13.00.00	Added PduR Pre Transmit Mode Request Port in 4.2.30
visvjn	2020-10-01	14.00.00	Added Service IdsM_BswM_StateChanged in chapter 4.3
visvjn	2022-01-14	15.00.00	Added Service EthIf_StartAllPorts in chapter 4.3

visvjn	2022-03-02	15.01.00	Added BswM_MemMap.h in chapter 3.1.2 Added Restart of EthIf Switch Ports to 3.3.2 Ecu State Handling
visvjn	2022-08-18	15.01.01	Product name updated to MICROSAR Classic
visuca	2022-10-10	16.00.00	Extended chapter 2.6
shagemann	2023-02-06	17.00.00	Extended Figure 1-2 Extended Table 3-1 Added chapter 4.2.12 Added chapter 4.2.13 Extended chapter 5.3.4 Extended Table 6-1 Extended Table 2-4
visuca	2023-08-31	18.00.00	Modified Table 2-3 Modified chapter 3.3 Modified Table 4-4 Modified chapter 5.2.1 Modified Table 4-44 Modified chapter 4.6
visuca	2024-01-25	18.00.01	Extended chapter 3.3.5

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	Specification of Basic Software Manager	R4.1.2
[2]	AUTOSAR	Guide to Mode Management	R4.1.2
[3]	AUTOSAR	Specification of Default Error Tracer	R4.0.3
[4]	AUTOSAR	List of Basic Software Modules	R4.0.3
[5]	AUTOSAR	Specification of Diagnostic Event Manager	R4.0.3
[6]	Vector	TechnicalReference_Rte.pdf	see delivery
[7]	Vector	TechnicalReference_PostBuildLoadable.pdf	see delivery
[8]	Vector	TechnicalReference_Com.pdf	see delivery
[9]	Vector	TechnicalReference_IdentityManager.pdf	see delivery

Scope of the Document

This technical reference describes the general use of the AUTOSAR Basic Software module BSW Mode Manager (BswM).

Caution
We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector’s release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

Contents

Illustrations

Tables

1 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module BswM as specified in [1].

Supported AUTOSAR Release*:	4	
Supported Configuration Variants:	pre-compile, post-build, post-build-selectable	
Vendor ID:	BSWM_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	BSWM_MODULE_ID	42 decimal (according to ref.[4])

* For the precise AUTOSAR Release 4.x please see the release specific documentation.

The BSW Mode Manager is the module that implements the part of the Vehicle Mode Management and Application Mode Management concept that resides in the BSW.

Its responsibility is to arbitrate mode requests from application layer SW-Cs or other

BSW modules based on simple rules, and perform actions based on the arbitration result.

1.1 Architecture Overview

The following figure shows where the BswM is located in the AUTOSAR architecture.

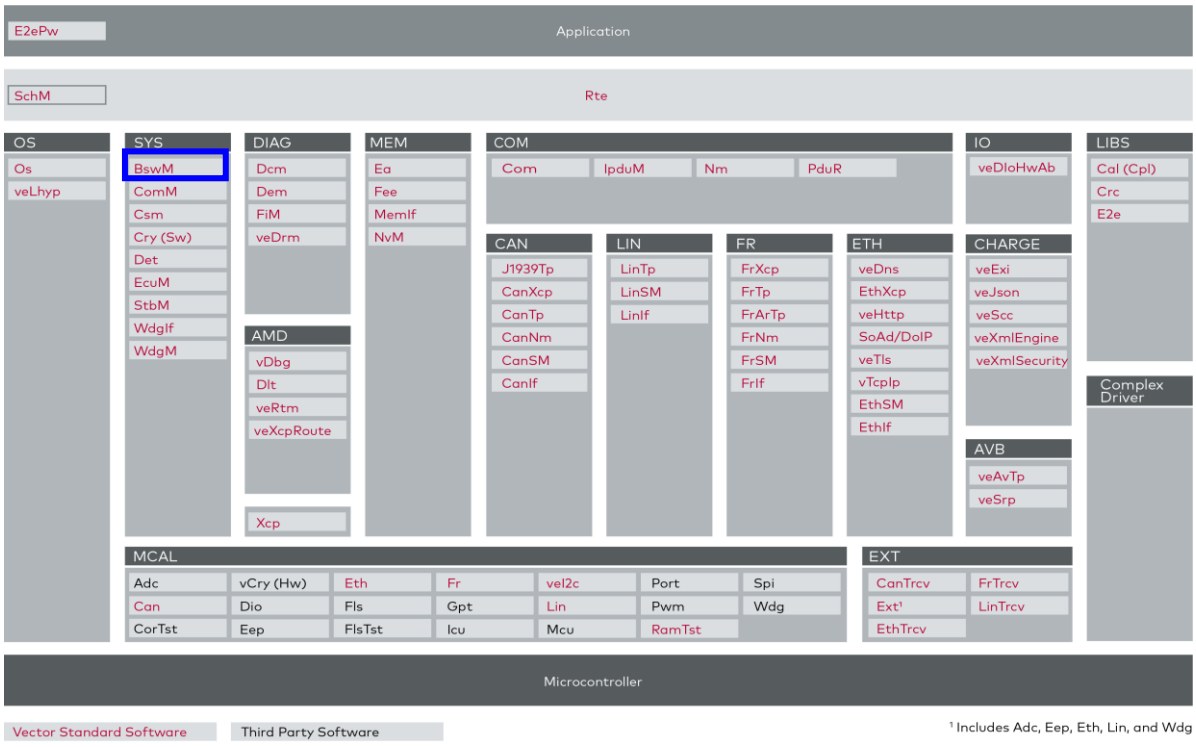

Figure 1-1 AUTOSAR Architecture

The next figure shows the interfaces to adjacent modules of the BswM. These interfaces are described in chapter 4.

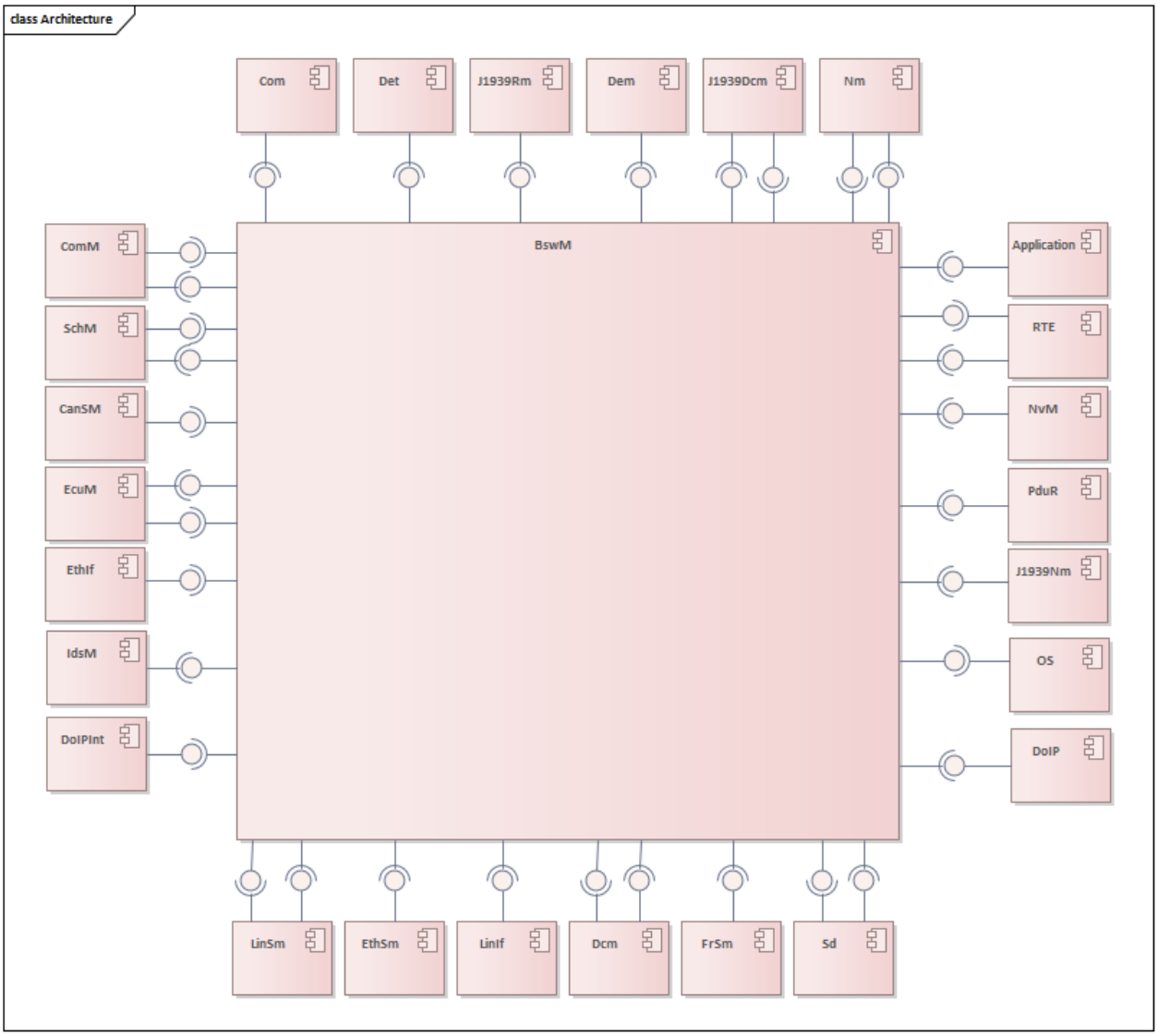

Figure 1-2 Interfaces to adjacent modules of the BswM

2 Functional Description

This chapter describes the general function of the BswM.

2.1 Features

The features listed in the following tables cover the complete functionality specified for the BswM.

The AUTOSAR standard functionality is specified in [1]. The corresponding features are listed in the tables:

- Table 2-1 Supported AUTOSAR Standard Conform Features
- Table 2-2 Not Supported AUTOSAR Standard Conform Features

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
CanSM mode arbitration
ComM mode arbitration
Dcm mode arbitration
EcuM mode arbitration
EthSM mode arbitration
FrSM mode arbitration
J1939Dcm mode arbitration
J1939Nm mode arbitration
LinSM mode arbitration
LinTp mode arbitration
NvM mode arbitration
Sd mode arbitration
Application mode arbitration
I-PDU Group handling (activation/deactivation/deadline monitoring)
Action to handle PduR routing path groups
Nested rule execution
Rte Mode Notification and Switches
Rte Mode Request Interfaces and Ports
Watchdog Manager
Post-Build Loadable
MICROSAR Classic Identity Manager using Post-Build Selectable [9]
Multi Partition Support

Table 2-1 Supported AUTOSAR Standard Conform Features

2.1.1 Deviations

The following features specified in [1] are not supported:

Not Supported AUTOSAR Standard Conform Features
Available Action: BswM_TriggerStartUpPhase2
Available Actions: BswM_TriggerSlaveRTEStop

Table 2-2 Not Supported AUTOSAR Standard Conform Features

See Chapter 5.1 for detailed information about other deviations.

2.1.2 Additions/ Extensions

Features Provided Beyond the AUTOSAR Standard
Timers as mode request ports
Nm as mode request port
User conditions as mode request ports
Generic mode switch as available action
Timer control as available action
Creation of user callouts in BswM_Callout_Stubs.c
Preconfigured State Machines (Communication, Initialization, Service Discovery and ECU State Handling)
Arbitration of rules on initialization values of immediate mode request ports
Rule Control (deactivation of rules during runtime)
Prioritization of Action List Execution Order
PduR mode request ports
EthIf mode arbitration

Table 2-3 Features Provided Beyond the AUTOSAR Standard

2.2 Initialization

The BswM is initialized via the service functions `BswM_PreInit` and `BswM_Init` (refer to chapter 4.2.2). On platforms in which the Random Access Memory (RAM) is not initialized to zero by the start-up code the function `BswM_InitMemory` has to be called first and then a call to `BswM_PreInit` and `BswM_Init` can be realized. All available modes are set to the configured initialization state, which can either be undefined or set to a specific value. If the initialization state is undefined the mode is not arbitrated until the mode request/indication function occurs for the first time.

Note
In case of Multi Partition, each instance of the BswM must be initialized. The BswM_PreInit function should only be called once.

2.3 States

The state machine diagram in Figure 2-1 shows the general handling of the BswM. Each state is described as follows:

> **BSWM_INIT**

The BswM is initialized and ready for immediate mode arbitration requests. Deferred mode arbitration is done within the cyclically called function `BswM_MainFunction`.

> **BSWM_WAIT_IMMEDIATE_REQUEST**

In this state the BswM waits for a mode arbitration request. The state is left if immediate mode arbitration is requested or when `BswM_MainFunction` is called.

> **BSWM_MAIN_FUNCTION**

This state is entered when the `BswM_MainFunction` is called. Within `BswM_MainFunction` the deferred mode arbitration is done. Immediate mode arbitration requests which occur during the execution of `BswM_MainFunction` are queued and will be executed at the end of `BswM_MainFunction`, when all deferred mode arbitration and control is finished. Mode arbitration requests of type “forced immediate” are not queued and interrupt the deferred mode arbitration.

> **BSWM_MODE_ARBITRATION_AND_CONTROL**

In this state the configured mode rule arbitration is done and the true-/false-action lists are executed. New mode arbitration requests of type “immediate” are queued, arbitration requests of type “forced immediate” are arbitrated immediately.

> **BSWM_EMPTY_QUEUE**

In this state the queued mode arbitration requests are executed.

> **BSWM_DEINIT**

This state is entered when the function `BswM_Deinit` is called. No mode arbitration requests are accepted and no mode processing is done. This state can only be left when function `BswM_Init` is called.

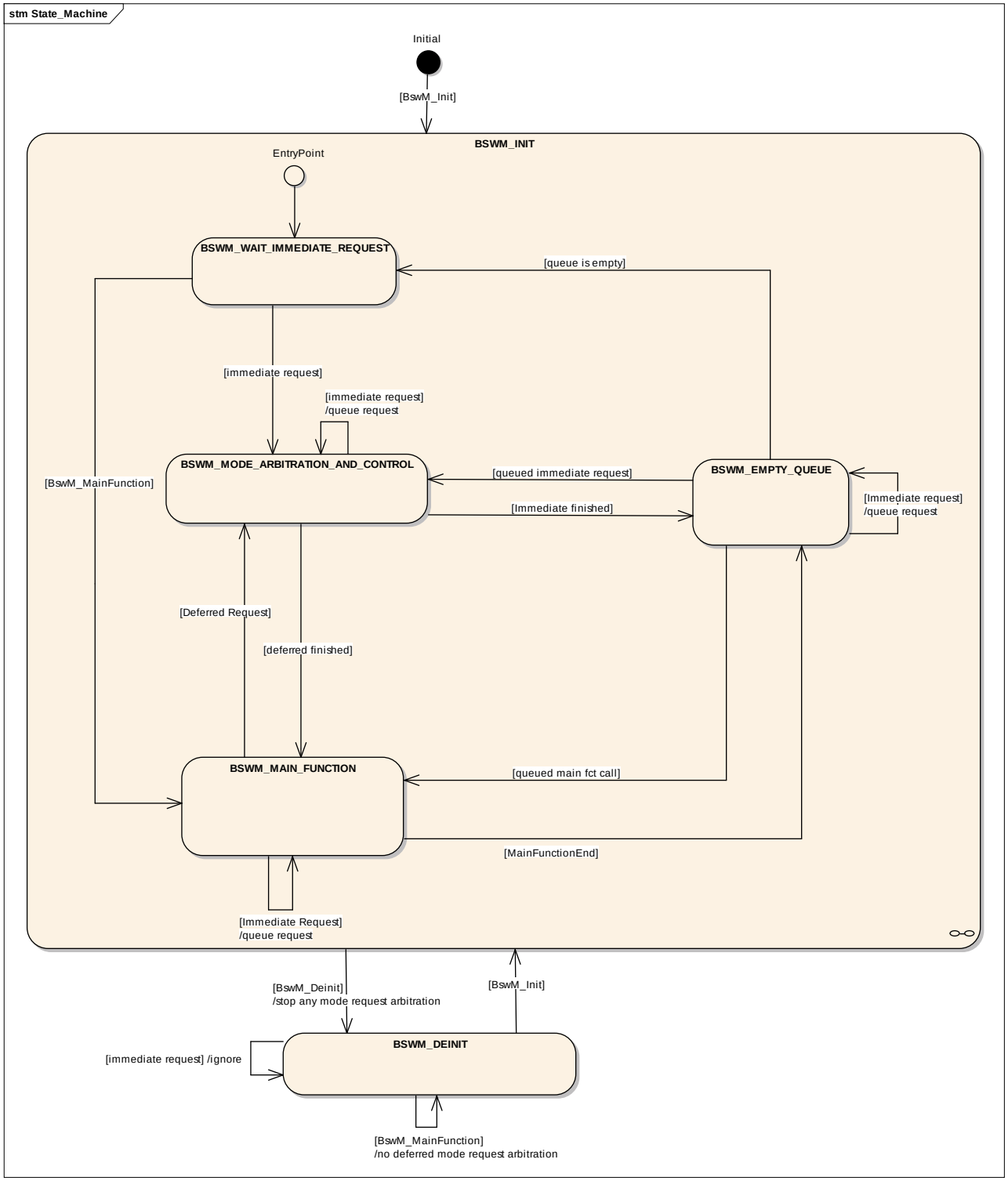

Figure 2-1 States of the BswM

2.4 Mode Management

The BswM manages user defined modes, whose behavior is completely defined by its configuration. A mode consists of the following parts:

- **Mode Source:** this is the trigger for the mode arbitration, a trigger can either be an application indication/request function or a BSW indication/request function or the `BswM_MainFunction()`.
- **Mode Arbitration:** when the mode source trigger occurs the BswM will arbitrate a mode specific rule either immediately or deferred within the `BswM_MainFunction()`. The mode arbitration types are described in detail in chapters 2.4.1 and 2.4.3.
- **Mode Rule:** a rule is a logical Boolean expression which consists of specific conditions which use different operators. The rule is arbitrated by the BswM to be either true or false. Dependent on the evaluation result the BswM executes the configured mode action(s) (true-action(s) or false-action(s)).
- **Mode Actions:** these are either BSW service function calls, user callout function calls or calls to nested rules or action lists which are executed by the BswM after the Mode Arbitration.

2.4.1 Immediate Mode Handling

The immediate mode arbitration is done directly upon the mode request/indication function. If another mode request/indication occurs during mode arbitration the BswM queues this mode arbitration request. The mode request queue is emptied when the current mode arbitration is finished. The sequence diagram in Figure 2-2 shows this procedure.

2.4.2 Forced Immediate Mode Handling

The forced immediate mode arbitration is done directly upon the mode request/indication function. The forced immediate request triggers immediate mode arbitration, interrupting any other immediate mode arbitration or deferred rule processing in the main function. This type of mode handling is not queued.

Caution
In the case that immediate or forced immediate mode arbitration is used, bounded recursions might occur. These recursions are caused by immediate triggered rules with actions which lead to a call of the same BswM mode request API (direct or indirect). The recursion depth depends on the configuration and so this should be used with care. To allow immediate mode processing according to Autosar specification, the recursions are not resolvable by the BswM implementation. The depth of the recursion needs to be taken in account during integration to define a sufficient stack size.

Most likely recursions occur for generic mode request ports but might also happen for other ports.

2.4.3 Deferred Mode Handling

The deferred mode arbitration is done cyclically within the execution of the `BswM_MainFunction()`. If another mode request/indication occurs during mode arbitration the BswM queues this mode arbitration request. The mode request queue is emptied at the end of the `BswM_MainFunction()`. The sequence diagram in Figure 2-3 shows this procedure.

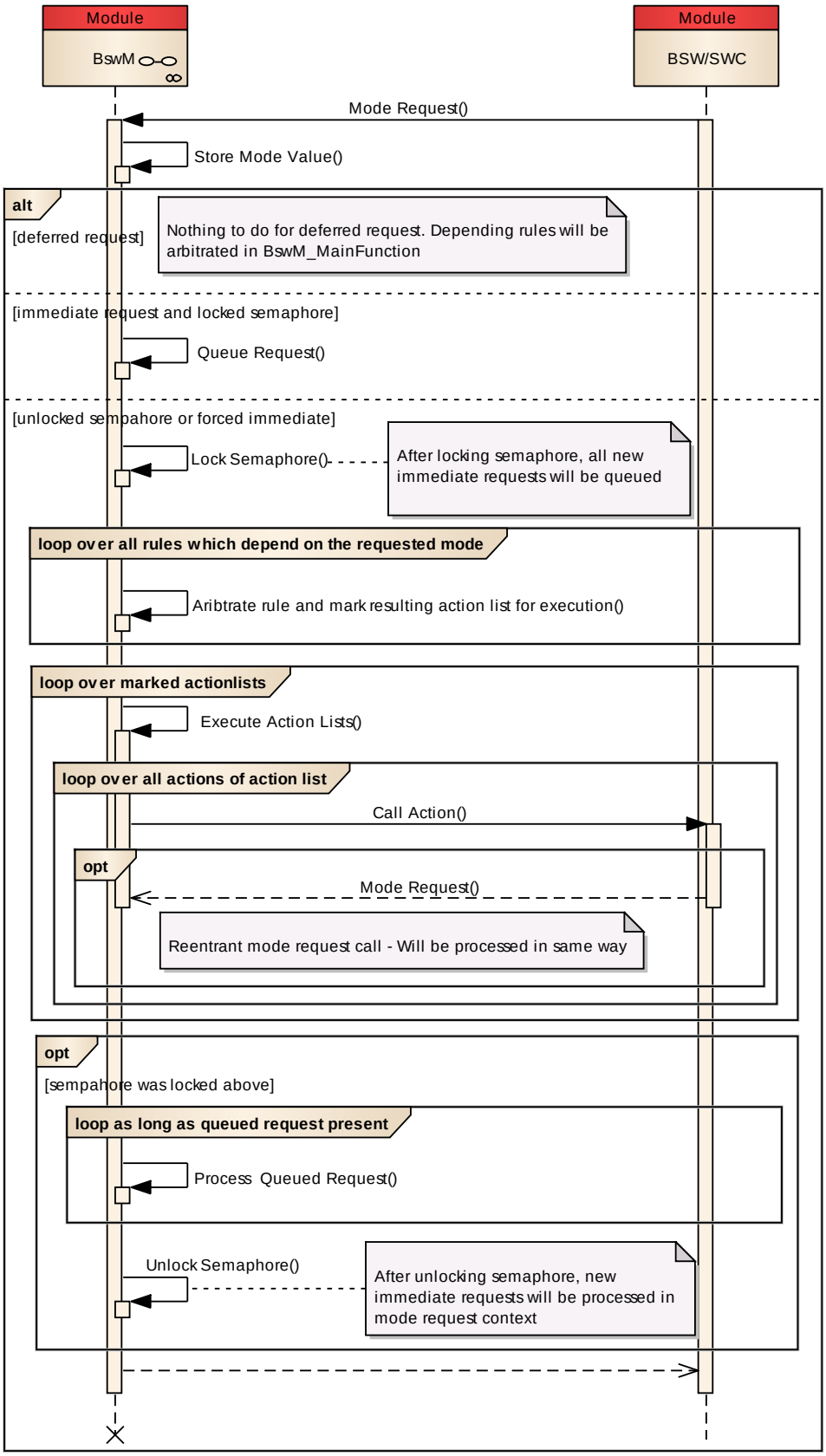

Figure 2-2 Sequence Immediate Processing

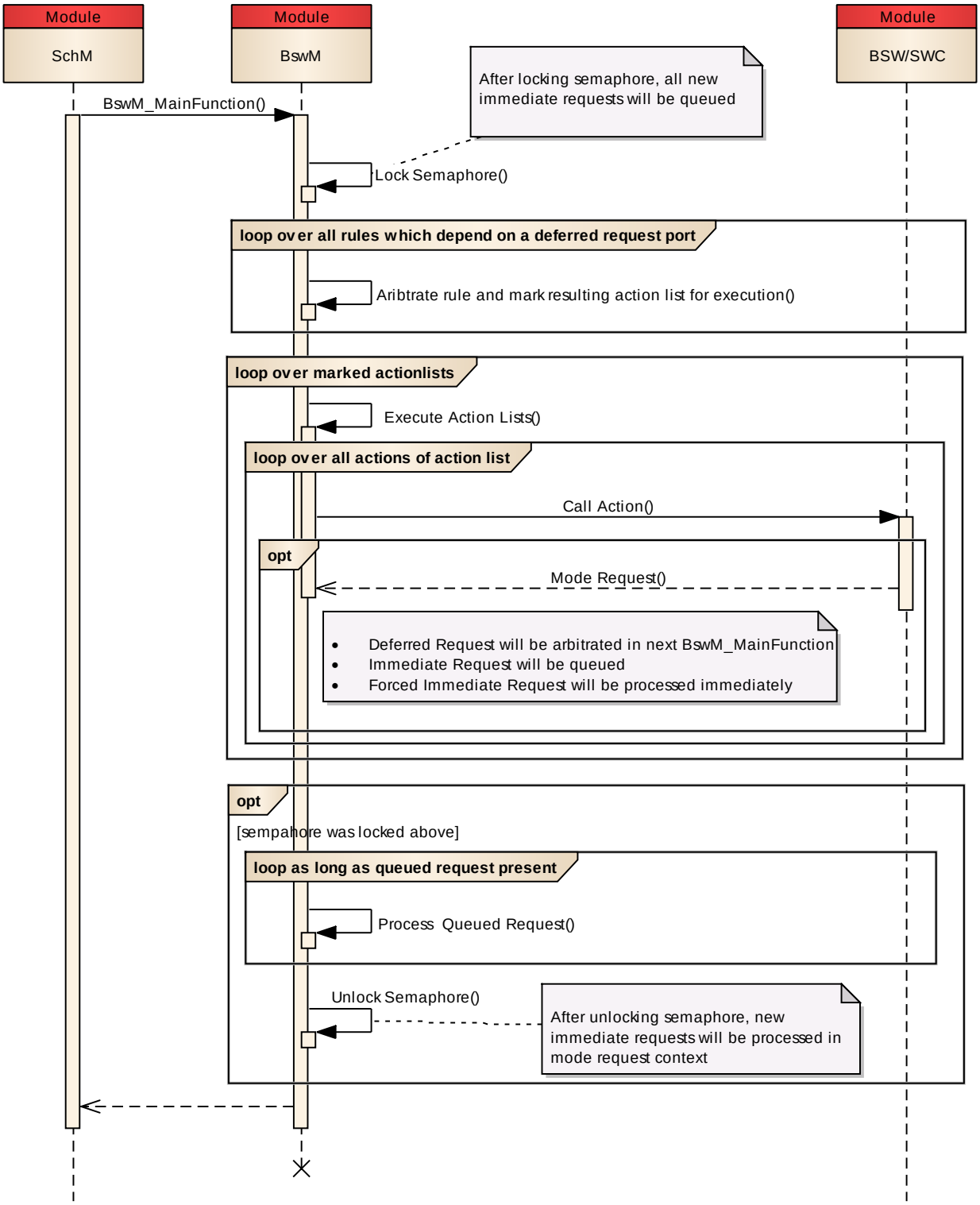

Figure 2-3 Sequence Deferred Mode

2.5 Execution of Action Lists

The execution of actions is done after the rule arbitration phase. Whether an action list shall be executed depends on the arbitration result (true or false). There are two ways to execute an action list based on evaluation of rules: either it is executed every time the rule is evaluated with the corresponding result (so called *conditional execution*), or only when the evaluation result has changed from the previous evaluation (so called *triggered execution*). This execution type is defined via configuration. If arbitration of a rule leads to the execution of an action list, this action list is marked for execution. All marked action lists are executed by their prioritization after the rules have been arbitrated.

2.6 Multi Partition Handling

In case of a Multi Partition configuration, each BswM instance is mapped to its own partition.

All items, such as rules, logical expressions, conditions, action lists and actions, can only be used in the scope of the partition, in which the item is created. The only item that can be used outside of the creation partition is the mode request port.

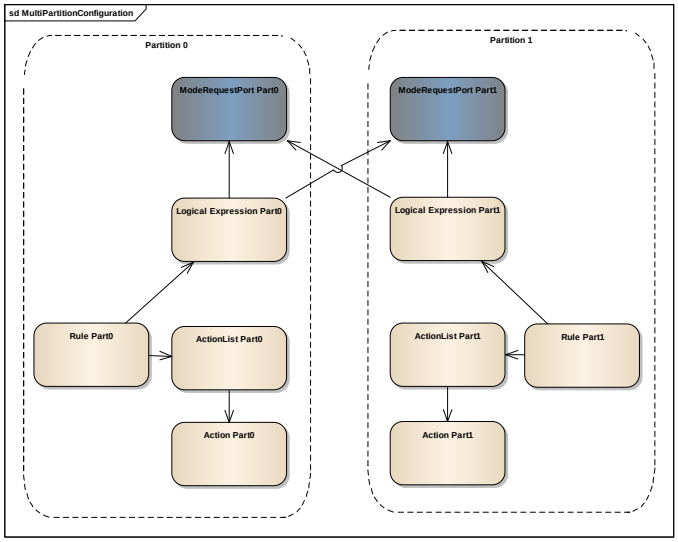

Figure 2-4 Example of rule configuration in Multi Partition use case

Note
Rules, Logical Expressions, Conditions, Action Lists and Actions always belong to exactly one partition and therefore must be configured in exactly one BswM instance. Solely ModeRequestPorts can be used from other partitions.

Caution
If a ModeRequestPort is used across partitions, its mode arbitration type must be configured as deferred. Refer to chapter 2.4.3 for details on Deferred Mode Handling.

2.7 Error Handling

2.7.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [3], if development error reporting is enabled (i.e. pre-compile parameter `BSWM_DEV_ERROR_DETECT==STD_ON`).

If another module is used for development error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Det_ReportError()`.

The reported BswM ID is 42.

The reported service IDs identify the services which are described in chapter 4.2. Table 2-4 presents the service IDs and the related services.

Service ID		Service
0x80	BSWM_INITMEMORY_ID	BswM_InitMemory()
0x00	BSWM_INIT_ID	BswM_Init()
0x01	BSWM_GETVERSIONINFO_ID	BswM_GetVersionInfo()
0x02	BSWM_REQUESTMODE_ID	BswM_RequestMode()
0x03	BSWM_MAINFUNCTION_ID	BswM_MainFunction()
0x04	BSWM_DEINIT_ID	BswM_Deinit()
0x05	BSWM_CANSM_CURRENTSTATE_ID	BswM_CanSM_CurrentState()
0x06	BSWM_DCM_COMMUNICATION_STATE_ID	BswM_Dcm_CommunicationMode_CurrentState()
0x09	BSWM_LINSM_CURRENTSTATE_ID	BswM_LinSM_CurrentState()
0x0A	BSWM_LINSM_CURRENTSCHEDULE_ID	BswM_LinSM_CurrentSchedule()
0x0B	BSWM_LINTP_REQUESTMODE_ID	BswM_LinTp_RequestMode()
0x0C	BSWM_FRSM_CURRENTSTATE_ID	BswM_FrSM_CurrentState()
0x0D	BSWM_ETHSM_CURRENTMODE_ID	BswM_EthSM_CurrentState()
0x0E	BSWM_COMM_CURRENTMODE_ID	BswM_ComM_CurrentMode()
0x0F	BSWM_ECUM_CURRENTSTATE_ID	BswM_EcuM_CurrentState()
0x10	BSWM_ECUM_CURRENTWAKEUP_ID	BswM_EcuM_CurrentWakeup()
0x11	BSWM_WDGM_REQUESTPARTITIONRESET_ID	BswM_WdgM_RequestPartitionReset()
0x14	BSWM_DCM_APPLICATION_UPDATED_ID	BswM_Dcm_ApplicationUpdated()
0x15	BSWM_COMM_PNC_CURRENTMODE_ID	BswM_ComM_CurrentPNCMode()
0x16	BSWM_NVM_CURRENTBLOCKMODE_ID	BswM_NvM_CurrentBlockMode()
0x17	BSWM_NVM_CURRENTJOBMODE_ID	BswM_NvM_CurrentJobMode()
0x18	BSWM_J1939NM_STATE_ID	BswM_J1939Nm_StateChangeNotification()
0x1b	BSWM_J1939DCM_BROADCASTSTATUS_ID	BswM_J1939DcmBroadcastStatus()

Service ID		Service
0x1f	BSWM_SD_CLIENTSERVICE_CURRENT_ID	BswM_Sd_ClientServiceCurrentState()
0x20	BSWM_SD_EVENTHANDLER_CURRENT_ID	BswM_Sd_EventHandlerCurrentState()
0x21	BSWM_SD_CONSUMEDEVENTGROUP_ID	BswM_Sd_ConsumedEventGroupCurrentState()
0x22	BSWM_COMM_INITIATERESET_ID	BswM_ComM_InitiateReset()
0x23	BSWM_ECUM_REQUESTEDSTATE_ID	BswM_EcuM_RequestedState()
0x24	BSWM_DOIP_SET_CHANNEL_READY_ID	BswM_DoIP_SetChannelReady()
0x25	BSWM_DOIPINT_SET_CHANNEL_READY_ID	BswM_DoIPInt_SetChannelReady()
0x81	BSWM_NM_STATE_CHANGE_ID	BswM_Nm_StateChangeNotification()
0x82	BSWM_SWCNOTIFICATION_ID	BswM_Notification_<SWC Notification Name>
0x83	BSWM_SWCREQUESTMODE_ID	BswM_Read_<SWC Mode Request Name>
0x84	BSWM_SETRULESTATE_ID	BswM_RuleControl()
0x85	BSWM_LINSM_SCHEDULEENDNOTIFICATION_ID	BswM_LinSM_ScheduleEndNotification()
0x8E	BSWM_PDUR_PRETRANSMIT_ID	BswM_PduR_PreTransmit()
0x86	BSWM_PDUR_RXINDICATION_ID	BswM_PduR_RxIndication()
0x87	BSWM_PDUR_TPRXINDICATION_ID	BswM_PduR_TpRxIndication()
0x88	BSWM_PDUR_TPSTARTOFRECEPTION_ID	BswM_PduR_TpStartOfReception()
0x89	BSWM_PDUR_TPTXCONFIRMATION_ID	BswM_PduR_TpTxConfirmation()
0x8A	BSWM_PDUR_TRANSMIT_ID	BswM_PduR_Transmit()
0x8B	BSWM_PDUR_TXCONFIRMATION_ID	BswM_PduR_TxConfirmation()
0x8C	BSWM_ETHIF_PORTGROUPLINKSTATECHANGE_ID	BswM_EthIf_PortGroupLinkStateChg()
0x8D	BSWM_PREINIT_ID	BswM_PreInit()
0xE0	BSWM_PUSH_INTO_ACTION_LIST_QUEUE_ID	BswM_PushIntoActionListQueue()

Table 2-4 Service IDs

The errors reported to DET are described in the following table:

Error Code		Description
0x01	BSWM_E_NO_INIT	Service function is called while BswM is not initialized.
0x02	BSWM_E_NULL_POINTER	Service function is called with a null pointer as an argument.
0x03	BSWM_E_PARAM_INVALID	The given parameter is invalid.
0x04	BSWM_E_REQ_USER_OUT_OF_RANGE	A requesting user is out of range.
0x05	BSWM_E_REQ_MODE_OUT_OF_RANGE	A requested mode is out of range.
0x06	BSWM_E_PARAM_CONFIG	The provided configuration is inconsistent.
0x80	BSWM_E_ALREADY_INITIALIZED	The BswM_Init function has been called twice.
0x81	BSWM_E_NO_PREINIT	The BswM_PreInit function was not called before BswM_Init was called.
0xA0	BSWM_E_ALREADY_QUEUED	An immediate request was made before the last request of the same port was processed. In most cases this error occurs due to an incorrect configuration, i.e. port shall be arbitrated on its initialization value of port but initialization value of rule is incorrect. If configuration is correct and loss of the earlier mode request is acceptable, this error can be ignored for this port. Otherwise, processing of port can be changed to BSWM_FORCED_IMMEDIATE.

Error Code		Description
0xA1	BSWM_E_REQ_USER_INV_PARTITION	A mode request was performed on a partition different than the configured partition.
0xB0	BSWM_E_ACTION_LIST_QUEUE_ERROR	Internal processing error, please check configuration of BswM exclusive areas.

Table 2-5 Errors reported to DET

2.7.2 Production Code Error Reporting

By default, production code related errors are reported to the DEM using the service `Dem_ReportErrorStatus()` as specified in [5].

The module BswM does not report any DEM error itself. However, it can be configured that an action member of an action list reports a DEM error when it fails.

3 Integration

This chapter gives necessary information for the integration of the MICROSAR Classic BswM into an application environment of an ECU.

3.1 Scope of Delivery

The delivery of the BswM contains the files which are described in chapters 3.1.1 and 3.1.2:

3.1.1 Static Files

File Name	Source Code Delivery	Object Code Delivery	Description
BswM.c	■		This is the source file of the BswM. It contains the initialization function, the deinitialization function, the cyclic main function and all the BSW mode indication functions.
BswM.h	■	■	This is the header file of the BswM. It contains the interfaces to the BswM API functions.
BswM_CanSM.h	■	■	This header file contains the prototypes of the callback functions of the CAN State Manager.
BswM_ComM.h	■	■	This header file contains the prototypes of the callback functions of the Communication Manager.
BswM_Dcm.h	■	■	This header file contains the prototypes of the callback functions of the Diagnostic Communication Manager.
BswM_DoIP.h	■	■	This header file contains the prototypes of the callback functions of the Diagnostic over IP.
BswM_DoIPInt.h	■	■	This header file contains the prototypes of the callback functions of the Diagnostic over IP for vehicle internal communication.
BswM_EcuM.h	■	■	This header file contains the prototypes of the callback functions of the Electronic Control Unit State Manager.
BswM_EthSm.h	■	■	This header file contains the prototypes of the callback functions of the Ethernet State Manager.
BswM_FrSM.h	■	■	This header file contains the prototypes of the callback functions of the FlexRay State Manager.
BswM_J1939Dcm.h	■	■	This header file contains the prototypes of the callback functions of the J1939Dcm module.
BswM_J1939Nm.h	■	■	This header file contains the prototypes of the callback functions of the J1939Nm module.
BswM_LinSM.h	■	■	This header file contains the prototypes of the callback functions of the LIN State Manager.
BswM_LinTp.h	■	■	This header file contains the prototypes of the callback functions of the LIN Transport Protocol.

File Name	Source Code Delivery	Object Code Delivery	Description
BswM_Nm.h	■	■	This header file contains the prototypes of the callback functions of the Network Manager.
BswM_NvM.h	■	■	This header file contains the prototypes of the callback functions of the Non Volatile Random-access memory Manager.
BswM_PduR.h	■	■	This header file contains the prototypes of the callback functions of the Pdu Router module.
BswM_Sd.h	■	■	This header file contains the prototypes of the callback functions of the Service Discovery module.
BswM_WdgM.h	■	■	This header file contains the prototypes of the callback functions of the Watchdog Manager module.

Table 3-1 Static Files

3.1.2 Dynamic Files

The dynamic files are generated by the configuration tool.

File Name	Description
BswM_Lcfg.c	This file contains the configuration parameters for pre-compile and for post-build variant.
BswM_Cfg.h	This header file contains general and configuration definitions for pre-compile and post-build variant.
BswM_Private_Cfg.h	This file contains the necessary includes and the declarations of libraries and variables used by the BswM.
BswM_PBcfg.c	This file contains the variables used for mode arbitration in post-build variant. This file is only generated in case of post-build loadable.
BswM_Callout_Stubs.c	This file contains the definitions of the call back functions which were configured to be created.
BswM_MemMap.h	This file is generated by the MICROSAR Classic MemMap module and provides the module specific memory sections.

Table 3-2 Dynamic Files

In case of multiple partitions additional files are generated.

File Name	Description
BswM_Lcfg_<OsApplication>.c	This file contains the configuration parameters for pre-compile and for post-build variant for the respective OsApplication.
BswM_Lcfg_<OsApplication>.h	This header file contains the partition specific declarations for pre-compile and post-build variant configurations.
BswM_PBcfg_<OsApplication>.c	This file contains the configuration parameters for post-build loadable for the respective OsApplication.

File Name	Description
BswM_PBcfg_<OsApplication>.h	This header file contains the partition specific declarations for post-build loadable configurations.

Table 3-3 Dynamic Multi Partition Files

3.2 Initialization of Other Software Modules

The BswM is able to initialize software components through User Callout functions. The BswM can realize the initialization after the EcuM has finished its “post OS sequence”, in which it initializes the operating system, the Schedule Manager and the BswM.

3.2.1 Using the Basic Editor

In order to configure the BswM to initialize other modules:

- Create “Actions” of type “User Callout” which contain the initialization functions.
- Create an “Action List” which contains the before mentioned “User Callout” actions.
- Click on the container “BswMModeControl”, to make the “Init Action List Reference” visible.
- Add a reference to the action list that contains the initialization callouts of the other modules.
- These actions will be called at the end of BswM_Init.

Caution
It is important that the execution of the initialization is not interrupted by any other main function. The initialization of all the configured modules should be concluded before any other function is called.

Illustratively, a list of initialization functions is listed below, as exemplified in the Guide to Mode Management [2] (This guide can be also consulted for further Mode management information). Note that this list is not complete and depends on the BSW modules you have in your delivery.

Initialization of basic drivers to access the NVRAM:

```
Spi_Init(NULL_PTR);
Eep_Init(NULL_PTR);
Fls_Init(NULL_PTR);
NvM_Init(NULL_PTR);
NvM_ReadAll();
```

After the `NvM_ReadAll()` job is finished the initialization of the remaining modules can continue:

```
Can_Init(NULL_PTR);
CanIf_Init(NULL_PTR);
CanSM_Init(NULL_PTR);
CanTp_Init(NULL_PTR);
Lin_Init(NULL_PTR);
LinIf_Init(NULL_PTR);
LinSM_Init(NULL_PTR);
LinTp_Init(NULL_PTR);
Fr_Init(NULL_PTR);
FrIf_Init(NULL_PTR);
FrSm_Init(NULL_PTR);
FrTp_Init(NULL_PTR);
StbM_Init();
PduR_Init(NULL_PTR);
CanNm_Init(NULL_PTR);
LinNM_Init(NULL_PTR);
FrNm_Init(NULL_PTR);
Nm_Init(NULL_PTR);
IpduM_Init(NULL_PTR);
Com_Init(NULL_PTR);
ComM_Init(NULL_PTR);
Dcm_Init(NULL_PTR);
Dem_Init(NULL_PTR);
RteStart();
```

Note that when in Post-Build variant, the previous initialization functions could contain post-build specific parameters. For detailed information see document [7], chapter: BSW Module Initialization, which summarizes the steps required to initialize post-build loadable BSW modules.

Caution
Note that the parameters of the initialization functions used in the example may differ from the actual expected parameters of the corresponding modules depending on the configuration. Please refer to the Technical Reference of each module for the proper initialization call.

3.2.2 Using the Comfort View

In order to facilitate the configuration of the initialization of other modules, the “Auto Configuration: Module Initialization” can be used. For further information see chapters 3.3 and 0.

3.3 Support of Preconfigured State Machines (Auto-Configuration)

The BswM supports preconfigured state machines. The content of these state machines is based on the currrent configuration. The state machines can be activated and modified by the user. They can be found in the “Mode Management” view of the DaVinci Configurator 5 Pro. To make use of the auto configured state machines:

1. In the configuration editor click on “Mode Management”.
2. Open “BSW Management” window.
3. Click on “Auto Configuration: <Name of the State Machine>”.
4. Click on the link “Configure <Name of the State Machine>” to start configuring.

Caution
Created Rules, Actions, Conditions, etc. are only an advice and may be edited by the integrator. The user has to monitor the validity of all created elements.

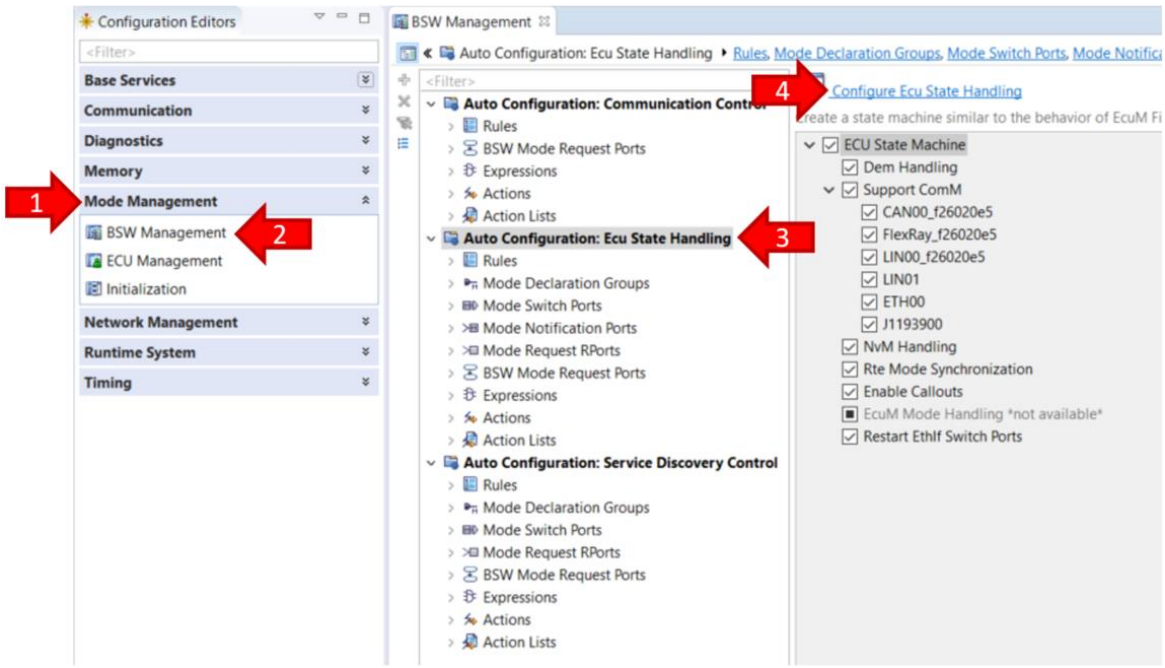

Figure 3-1 Auto-configured state machines

3.3.1 Module Initialization

The BswM has knowledge of how to initialize several modules: which function to call, with which parameters and which header to include. These modules are listed in the “known modules” list. However, the preconfigured initialization functions and include headers can be changed/adapted by the integrator.

The “foreign modules” list contains modules unknown to the BswM. An initialization function and an include header are suggested, but it is necessary to assure the correctness of the preconfigured parameters and adapt them in case it is necessary. The foreign modules will be initialized after the known modules by default.

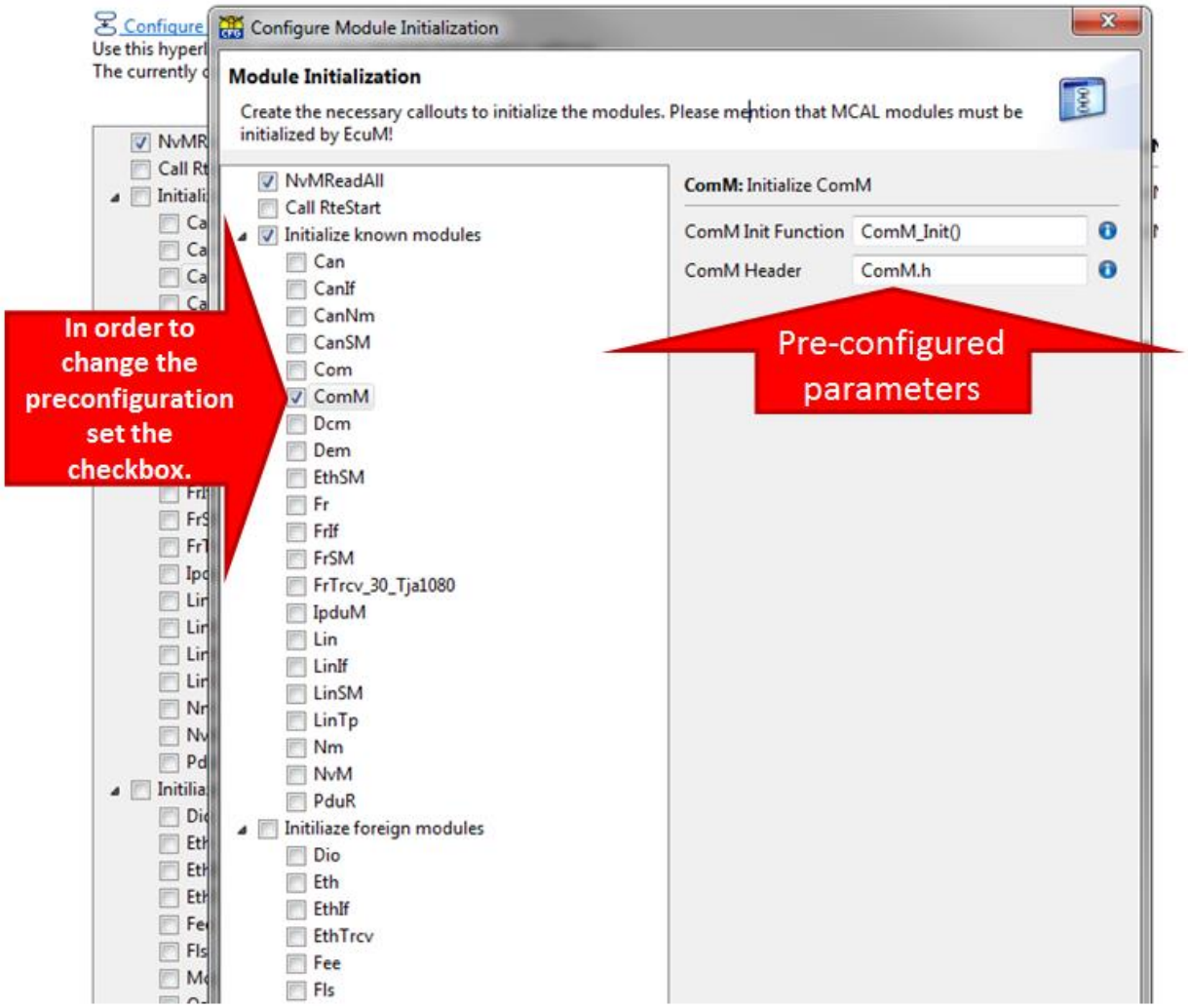

Figure 3-2 Configure module initialization

The list of modules shows them in alphabetical order. But the initialization function calls are generated according to the internal logic of the generator. In order to see the actual order in which the functions will be generated, click on Auto Configuration: Module Initialization -> Action Lists -> INIT_AL_Initialize.

A list of items is shown in the order in which they are generated. The order of the items can be changed manually.

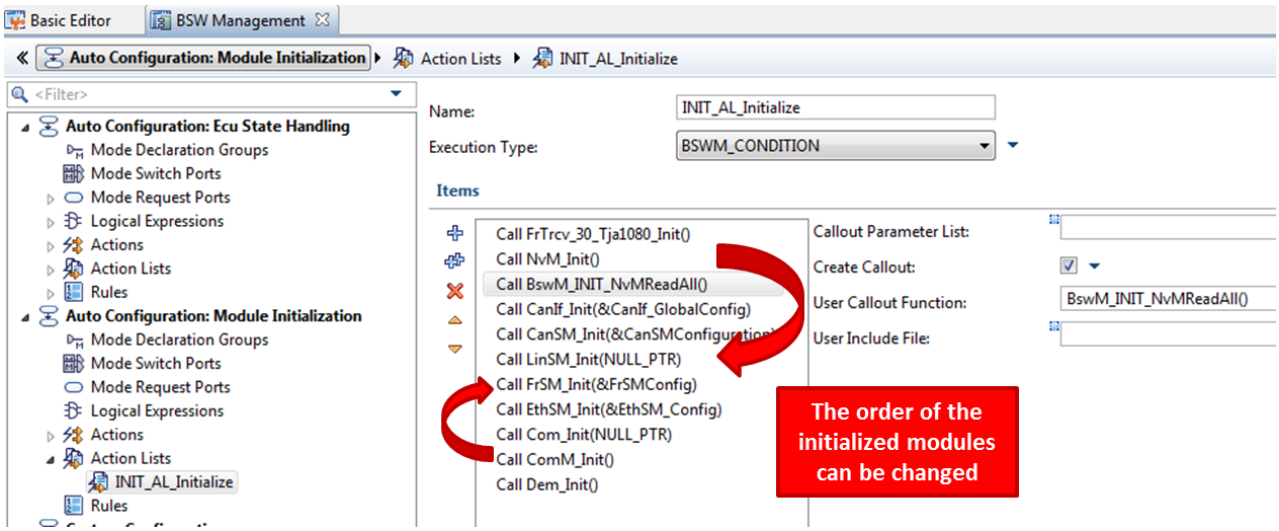

Figure 3-3 Edit initialization order

If the module initialization is edited with the configuration window again, the default order of the items will be restored and the changes previously made in the action list items order will be lost.

To avoid changing the already edited action list items order, it is necessary to clear the “Restore Default Sequence” checkbox when configuring again.

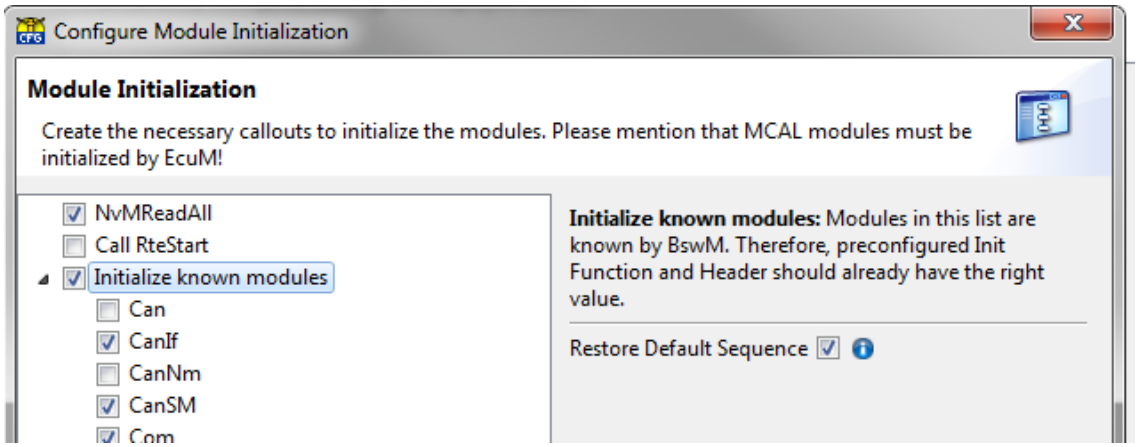

Figure 3-4 Restore default sequence

3.3.2 Ecu State Handling

The BswM is able to create rules and actions which take care of starting and shutting down the ECU. This behavior is similar to EcuM in ASR 3.

The following features can be activated:

- DEM initialization and shut-down generation
- Enabling and disabling of ComM communication

- Activation of NvM handling
- Notifications of the RTE about mode changes
- Callouts within state transitions
- Restart of Ethernet Switches after an unexpected ECU Reset

Furthermore, the number of users that request run request and post-run request and the period of time that the state machine spends in the run mode state can be configured.

In general there is the option to configure the “Kill All Run Request Port”, with this SWC Mode Request port it is possible to enforce a shutdown without respecting any run / post-run user, self run request timer or channel state.

The state machine of the ECU state Handling is illustrated in Figure 3-5 State Machine of the ECU State Handling. The rectangular states are notified to the RTE if synchronisation is enabled.

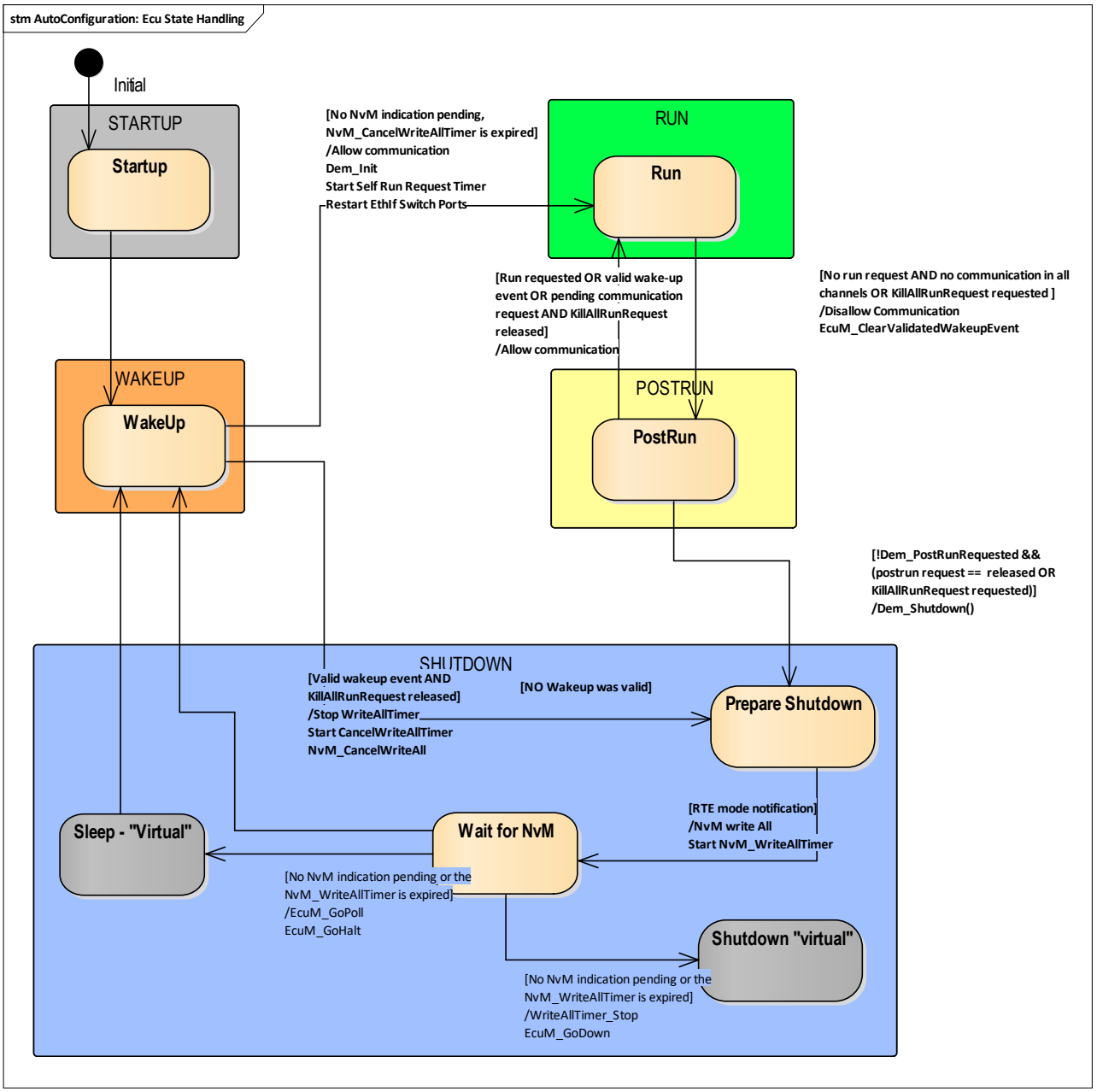

Figure 3-5 State Machine of the ECU State Handling

3.3.3 Communication Control

The BswM is able to create rules and actions which take care of starting and stopping the communication of an ECU.

The features supported by the auto configuration of the Communication Control are:

> Configuration I-PDU groups switching of CAN, ETH, LIN, FR and J1939 as long as the I-PDUs belong to only one channel.

- In case the I-PDU Group has I-PDUs from different channels, it will be listed as “Not available” and the configuration has to be realized manually.

> Reinitialization of transmission (TX) and reception (RX) I-PDUs is possible.
  - In case of CAN, the reinitialization will only be performed in the Bus State transition from NO_COM to FULL _COM, in case of BUS_OFF or SILENT no reinitialization will be performed.

> Enabling and disabling of the NM for CAN, ETH and FlexRay bus, if NM is present in that channel.

> Consideration of the DCM Modes when switching I-PDU Groups that belong to CAN, ETH or to FlexRay bus.

> Consideration of selected Nm States when switching TX I-PDU Groups that belong to a CAN bus.

> Configuration of Partial Networking (PNC) is supported for CAN, ETH and FlexRay bus.
  - If a I-PDU Group can be assigned to a PNC, the I-PDU Group is listed as a sub feature of the corresponding PNC and it is switched on or off depending on the PNC Status.
  - Consider that the PNC can only be determined if there are PNC-Mapping entries in the System-Description.

> Configuration of the J1939 module.
  - Standard rules will be configured which consider the state of the J1939Nm for the rule condition. As action lists the states of the modules J1939Dcm and J1939Rm are set.
  - The I-PDU Groups which contains only I-PDUs of the same Node will be switched on or off depending on the Node status.
  - The I-PDU Groups which are determined as broadcast groups will be switched on or off depending on the Dcm broadcast status.
  - Enabling and disabling of Routing-Paths in PduR depending on the channel and node state.

> Switching of LIN I-PDU groups.
  - The I-PDU Groups which contains only I-PDUs of the same Schedule will be switched on or off depending on the schedule status.
  - Setting a start schedule table.

3.3.4 Service Discovery Control

The BswM is able to create the necessary ports to control the Service Discovery by application.

The auto configuration, which is only available if the Sd module is in the current configuration, supports the following features:

> Creation of a BswMSwitchPort (P-Port) for each selected SdClientService, SdEventHandler or SdConsumedEventGroup to provide its state to the application.

> Creation of a BswMSwcModeRequest (R-Port) for each selected SdClientService, SdServerService or SdConsumedEventGroup to catch the request from application and forward it to the Sd.

3.3.5 Multi Partition Support

The auto configured state machines “Communication Control”, “Ecu State Handling” and “Service Discovery Control” consider partition specific information to distribute their elements to corresponding partitions.

The partition where an element is assigned to depends on several factors, e.g. where a ComMChannel or ComIPdu is located or where a BSW module has its master partition.

The partition which is used as default/master module partition by the Auto-Configuration can be configured in the module specific EcuC partition references of BswM as shown in Figure 3-6.

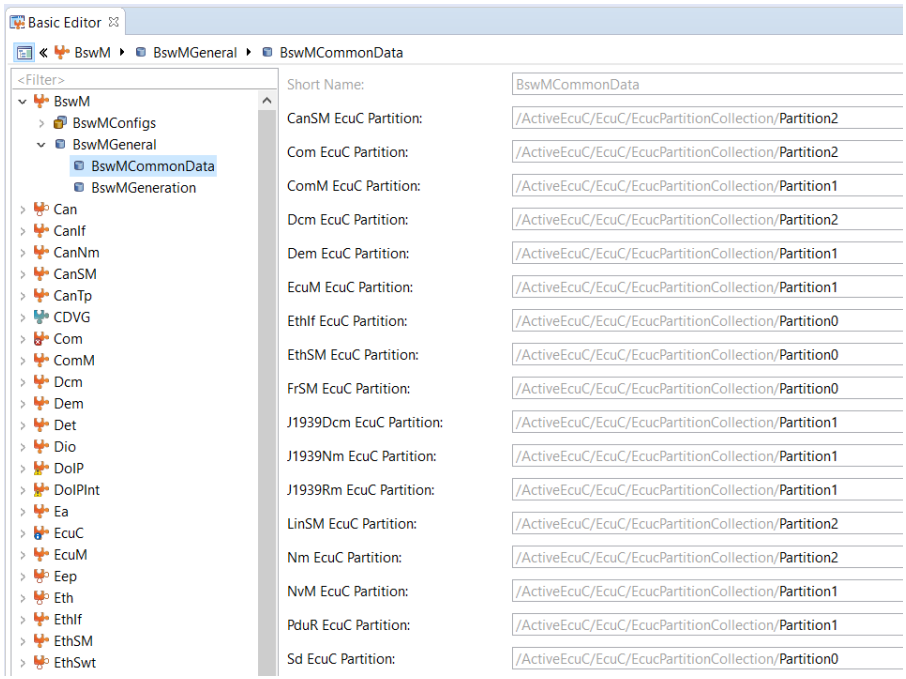

Figure 3-6 Module specific EcuC partition references for Auto-Configuration

If a required partition reference is not configured, a corresponding warning is displayed in the “Mode Management” view declaring the missing reference.

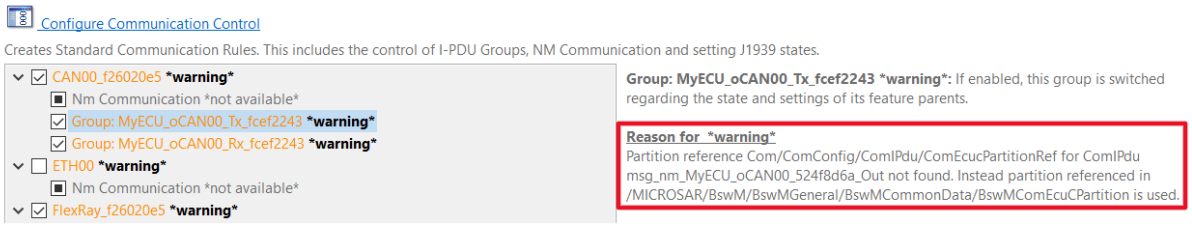

Figure 3-7 Warning about missing partition reference in Auto-Configuration

Example
In the example shown in Figure 3-7 the partition reference of a ComIPdu is needed to determine on which partition the calls to `Com_IpduGroupStart()` shall be executed. To fix the warning and ensure that the API is executed on the correct partition, the reference /Com/ComConfig/ComIPdu/ComEcucPartitionRef needs to be filled for mentioned ComIPdu.

As the partition reference of the ComIPdu is missing, the module specific partition reference of Com (as seen in Figure 3-6) is used instead to ensure that the Auto-Configuration rules can still be created regardless of the missing partition reference.

Note
The data that is collected by the “Module Initialization” state machine does not contain any partition specific information. Therefore, the “Module Initialization” state machine can be configured on all partitions. The user has to distribute the initialization to correct partitions.

3.4 Critical Sections

The BswM has code sections which must not be interrupted by incoming mode requests. Therefore the BswM uses one exclusive area which requires a global interrupt lock:

BSWM_EXCLUSIVE_AREA_0

The main functions of the BSW modules that use BswM to provide mode indications should not interrupt each other.

Note
Refer to [6] for details about exclusive areas.

3.5 Cyclic Task

The BswM has one cyclic main function `BswM_MainFunction()` which must be called cyclically if either a deferred mode request port exists, a timer is used or a RTE mode switch action is configured. The cyclic time is up to the user but must be considered for deferred mode handling.

Note
In case of a Multi Partition configuration the `BswM_MainFunction()` must be triggered on each partition if a deferred mode request port exists, a timer is used or a RTE mode switch action is configured on that partition.

3.6 NvM – BswM configuration

When configuring NvM request ports in BswM it is necessary that the general configuration of the NvM has the necessary boxes checked.

In NvMCommon check the box: “Multiblock Job status Information”

In NVMConfigBlock check the box “Block status information”

3.7 MultiPartition Initialization

The BswM has multiple synchronization points in order to assure that all BswM instances (partitions) have executed the same mandatory code sections.

The following diagram displays an overview of the initialization routine with the necessary synchronization points. At first all partitions are executed one after the other. At the end the core master partition (= partition with lowest priority of the InitTask) is waiting for all other core masters to reach the exact same point in order to synchronize. After each core, and therefore all partitions, have reached the same point, the tasks switch back to the highest priority.

Note
The `BswM_Init()` is called in the context of EcuM_StartupTwo().

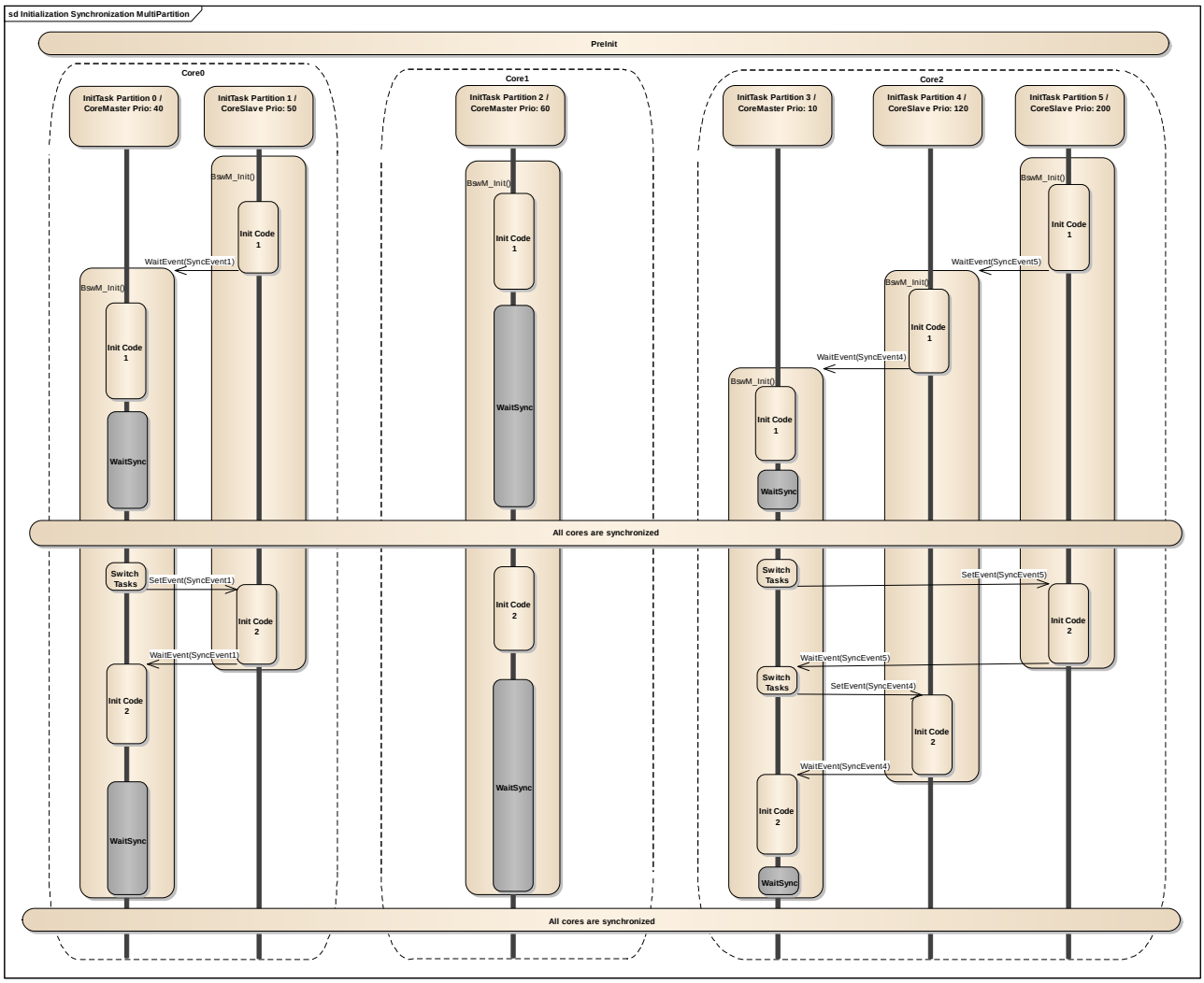

Figure 3-8 Sequence of Multi Partition initialization

4 API Description

For an interfaces overview please see Figure 1-2.

4.1 Type Definitions

The types defined by the BswM are described in this chapter.

Type Name	C-Type	Description	Value Range
BswM_ConfigType	struct	Used for the pointers of post-build configurations during the initialization of the BswM. In pre-compile, it is not used.	
BswM_ModeType	uint16	Data type that identifies the modes that can be requested by BswM Users	0 … 65535 Used if the total number of modes is greater than 255.
BswM_UserType	uint16	Data type that identifies a BswM User that makes mode requests to the BswM.	0 … 65535 Used if the total number of users is greater than 255.
BswM_HandleType	uint8 / uint16	Data type which is used for action list and rule IDs.	0 … 65535 Depends on number of action lists and rules.

Table 4-1 Type definitions

4.2 Services Provided by BswM

4.2.1 BswM_InitMemory

Prototype	
`void` **`BswM_InitMemory`** `(void)`	
Parameter	
None	-
Return code	
`void`	-
Functional Description	
Initializes the BSW Mode Manager module variables in case an initializing startup code is not used. This function sets the BswM into an uninitialized state.	
Particularities and Limitations	
> If this function is used it shall be called before any other BSWM function after startup. > This function is synchronous. > This function is non-reentrant.	
Call Context	
> This function can be called from task context.	

Table 4-2 BswM_InitMemory

4.2.2 BswM_PreInit

Prototype	
`void` **`BswM_PreInit`** `(const BswM_ConfigType *ConfigPtr)`	
Parameter	
`ConfigPtr`	Pointer to post-build configuration data. For the pre-compile case a NULL pointer shall be used.
Return code	
`void`	-
Functional Description	
PreInitializes the BSW Mode Manager.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant. > This function has to be called only once.	
Call Context	
> This function can be called from task context.	

Table 4-3 BswM_PreInit

4.2.3 BswM_Init

Prototype	
`void BswM_Init (const BswM_ConfigType *ConfigPtr)`	
Parameter	
`ConfigPtr`	Pointer to post-build configuration data. For the pre-compile case a NULL pointer shall be used.
Return code	
`void`	-
Functional Description	
Initializes the BSW Mode Manager.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant. > This function has to be called on each partition with a BswM instance.	
Call Context	
> This function can be called from task context.	

Table 4-4 BswM_Init

4.2.4 BswM_Deinit

Prototype	
`void BswM_Deinit (void)`	
Parameter	
None	-
Return code	
`void`	-
Functional Description	
Deinitializes the BSW Mode Manager. All pending requests are cleared and no further mode requests are accepted by the BswM. This state can only be left by calling the function `BswM_Init()`.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant.	
Call Context	
> This function can be called from task context.	

Table 4-5 BswM_Deinit

4.2.5 BswM_GetVersionInfo

Prototype	
`void BswM_GetVersionInfo (Std_VersionInfoType *VersionInfo)`	
Parameter	
`VersionInfo`	Pointer to address where the version information shall be copied to.
Return code	
`void`	None
Functional Description	
Returns the version information of this module. The versions are BCD-coded.	
Particularities and Limitations	
> The caller must ensure to allocate a variable of the type Std_VersionInfoType before the function call. > This function is synchronous. > This function is reentrant.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-6 BswM_GetVersionInfo

4.2.6 BswM_RequestMode

Prototype	
`void BswM_RequestMode (BswM_UserType requesting_user, BswM_ModeType requested_mode)`	
Parameter	
`requesting_user`	The user that requests the mode.
`requested_mode`	The requested mode.
Return code	
`void`	-
Functional Description	
Generic function call to request modes. This function shall only be used by other BSW modules that do not have a specific mode request interface and/or for generic mode requests.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for different users. > This function is only allowed to be used by applications for generic mode requests. Otherwise, applications must not use this function.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-7 BswM_RequestMode

4.2.7 BswM_ComM_CurrentMode

Prototype	
`void BswM_ComM_CurrentMode (NetworkHandleType Network, ComM_ModeType RequestedMode)`	
Parameter	
`Network`	The ComM communication channel that the indicated state corresponds to.
`RequestedMode`	The current state of the ComM communication channel
Return code	
`void`	-
Functional Description	
Function called by ComM to indicate the current communication mode of a ComM channel.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for different networks. > Must only be called by the ComM.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-8 BswM_ComM_CurrentMode

4.2.8 BswM_ComM_CurrentPNCMode

Prototype	
`void BswM_ComM_CurrentPNCMode (PNCHandleType PNC, ComM_PncModeType RequestedMode)`	
Parameter	
`PNC`	The handle of the PNC for which the current state is reported.
`RequestedMode`	The current mode of the PNC.
Return code	
`void`	-
Functional Description	
Function called by ComM to indicate the current mode of the PNC.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for different PNCs. > Must only be called by the ComM.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-9 BswM_ComM_CurrentPNCMode

4.2.9 BswM_ComM_InitiateReset

Prototype	
`void` **`BswM_ComM_InitiateReset`** `(void)`	
Parameter	
`void`	-
Return code	
`void`	-
Functional Description	
Function called by ComM to request a ECU reset.	
Particularities and Limitations	
> This function is synchronous. > Must only be called by the ComM.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-10 BswM_ComM_InitiateReset

4.2.10 BswM_Dcm_ApplicationUpdated

Prototype	
`void` **`BswM_Dcm_ApplicationUpdated`** `(void)`	
Parameter	
None	-
Return code	
`void`	-
Functional Description	
Function called by DCM to inform the BswM that the application has being updated.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant. > Must only be called by the Dcm.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-11 BswM_Dcm_ApplicationUpdated

4.2.11 BswM_Dcm_CommunicationMode_CurrentState

Prototype	
`void` **`BswM_Dcm_CommunicationMode_CurrentState`** `(NetworkHandleType Network, Dcm_CommunicationModeType RequestedMode)`	
Parameter	
`Network`	The communication channel that the diagnostic mode corresponds to.
`RequestedMode`	The requested diagnostic communication mode.
Return code	
`void`	-
Functional Description	
Function called by DCM to inform the BswM about the current state of the communication mode.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for different networks. > Must only be called by the Dcm.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-12 BswM_Dcm_CommunicationMode_CurrentState

4.2.12 BswM_DoIP_SetChannelReady

Prototype	
`void` **`BswM_DoIP_SetChannelReady`** `(NetworkHandleType Network)`	
Parameter	
`Network`	The communication channel for which a routing activation succeeded.
Return code	
`void`	-
Functional Description	
Function called by DoIP to report channel ready to BswM in case a routing activation has been successfully performed.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for different networks. > Must only be called by the DoIP.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-13 BswM_DoIP_SetChannelReady

4.2.13 BswM_DoIPInt_SetChannelReady

Prototype	
`void` **`BswM_DoIPInt_SetChannelReady`**`(NetworkHandleType Network)`	
Parameter	
`Network`	The communication channel for which a connection has been established successfully.
Return code	
`void`	-
Functional Description	
Function called by DoIPInt to report channel ready to BswM in case a connection has been successfully established.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for different networks. > Must only be called by the DoIPInt.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-14 BswM_DoIPInt_SetChannelReady

4.2.14 BswM_CanSM_CurrentState

Prototype	
`void` **`BswM_CanSM_CurrentState`** `( NetworkHandleType Network, CanSM_BswMCurrentStateType CurrentState)`	
Parameter	
`Network`	The CAN channel that the indicated state corresponds to.
`CurrentState`	The current state of the CAN channel.
Return code	
`void`	-
Functional Description	
Function called by CanSM to indicate its current state.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for different networks. > Must only be called by the CanSM.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-15 BswM_CanSM_CurrentState

4.2.15 BswM_EthIf_PortGroupLinkStateChg

Prototype	
`void BswM_EthIf_PortGroupLinkStateChg ( EthIf_SwitchPortGroupIdxType PortGroupIdx, EthTrcv_LinkStateType PortGroupState)`	
Parameter	
`PortGroupIdx`	The port group index in the context of the Ethernet Interface.
`PortGroupState`	The current state of the port group.
Return code	
`void`	-
Functional Description	
Function called by EthIf to indicate the link state change of a certain ethernet switch port group.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for different ethernet port groups. > Must only be called by the EthIf.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-16 BswM_EthIf_PortGroupLinkStateChg

4.2.16 BswM_EthSM_CurrentState

Prototype	
`void BswM_EthSM_CurrentState ( NetworkHandleType Network, EthSM_NetworkModeStateType CurrentState)`	
Parameter	
`Network`	The Ethernet channel that the indicated state corresponds to.
`CurrentState`	The current state of the Ethernet channel.
Return code	
`void`	-
Functional Description	
Function called by EthSM to indicate its current state.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for different networks. > Must only be called by the EthSM.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-17 BswM_EthSM_CurrentState

4.2.17 BswM_FrSM_CurrentState

Prototype	
`void BswM_FrSM_CurrentState ( NetworkHandleType Network, FrSM_BswM_StateType CurrentState)`	
Parameter	
`Network`	The FlexRay cluster that the indicated state corresponds to.
`CurrentState`	The current state of the FlexRay cluster.
Return code	
`void`	-
Functional Description	
Function called by FrSM to indicate its current state.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for different networks. > This function must only be called by the FrSM.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-18 BswM_FrSM_CurrentState

4.2.18 BswM_J1939DcmBroadcastStatus

Prototype	
`void BswM_J1939DcmBroadcastStatus ( uint16 NetworkMask)`	
Parameter	
`NetworkMask`	Mask containing one bit for each available network. 1:Network enabled 0: Network disabled.
Return code	
`void`	-
Functional Description	
This API tells the BswM the desired communication status of the available networks. The status will typically be activated via COM I-PDU group switches.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant. > This function must only be called by the J1939Dcm.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-19 BswM_J1939DcmBroadcastStatus

4.2.19 BswM_J1939Nm_StateChangeNotification

Prototype	
`void BswM_J1939Nm_StateChangeNotification ( NetworkHandleType Network, uint8 Node, Nm_StateType NmState)`	
Parameter	
`Network`	Identification of the J1939 channel.
`Node`	Identification of the J1939 node
`NmState`	Current (new) state of the J1939 node
Return code	
`void`	-
Functional Description	
Notification of current J1939Nm state after state changes.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for different combinations of network and node. > This function must only be called by the J1939Nm.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-20 BswM_J1939Nm_StateChangeNotification

4.2.20 BswM_LinSM_CurrentState

Prototype	
`void BswM_LinSM_CurrentState ( NetworkHandleType Network, LinSM_ModeType CurrentState)`	
Parameter	
`Network`	The LIN channel that the indicated state corresponds to.
`CurrentState`	The current state of the LIN channel.
Return code	
`void`	-
Functional Description	
Function called by LinSM to indicate its current state.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for different networks. > This function must only be called by the LinSM.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-21 BswM_LinSM_CurrentState

4.2.21 BswM_LinSM_CurrentSchedule

Prototype	
`void BswM_LinSM_CurrentSchedule ( NetworkHandleType Network, LinIf_SchHandleType CurrentSchedule)`	
Parameter	
`Network`	The LIN channel that the indicated schedule corresponds to.
`CurrentSchedule`	The currently active schedule table of the LIN channel.
Return code	
`void`	-
Functional Description	
Function called by LinSM to indicate its current schedule.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for different networks. > This function must only be called by the LinSM.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-22 BswM_LinSM_CurrentSchedule

4.2.22 BswM_LinSM_ScheduleEndNotification

Prototype	
`void BswM_LinSM_ScheduleEndNotification (NetworkHandleType Network, LinIf_SchHandleType Schedule)`	
Parameter	
`Network`	The LIN channel that the indicated schedule corresponds to.
`Schedule`	The schedule table of the LIN channel wich has ended.
Return code	
`void`	-
Functional Description	
Function called by LinSM to notify the end of a schedule.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for different networks. > This function must only be called by the LinSM.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-23 BswM_LinSM_ScheduleEndNotification

4.2.23 BswM_LinTp_RequestMode

Prototype	
`void BswM_LinTp_RequestMode ( NetworkHandleType Network, LinTp_Mode LinTpRequestedMode)`	
Parameter	
`Network`	The LIN channel that the LIN TP mode request corresponds to.
`LinTpRequestedMode`	The requested LIN TP mode.
Return code	
`void`	-
Functional Description	
Function called by LinTp to request a mode for the corresponding LIN channel. The LinTp_Mode mainly correlates to the LIN schedule table that should be used.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for different networks. > This function must only be called by the LinTp.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-24 BswM_LinTp_RequestMode

4.2.24 BswM_EcuM_CurrentState

Prototype	
`void BswM_EcuM_CurrentState (EcuM_StateType CurrentState)`	
Parameter	
`CurrentState`	The requested ECU Operation Mode
Return code	
`void`	-
Functional Description	
Function called by EcuM to indicate the current ECU Operation Mode.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant. > Must only be called by the EcuM.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-25 BswM_EcuM_CurrentState

4.2.25 BswM_EcuM_CurrentWakeup

Prototype	
`void BswM_EcuM_CurrentWakeup ( EcuM_WakeupSourceType source, EcuM_WakeupStateType state)`	
Parameter	
`source`	Wakeup source(s) that changed state.
`state`	The new state of the wakeup source(s).
Return code	
`void`	-
Functional Description	
Function called by EcuM to indicate the current state of a wakeup source.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for different sources. > Must only be called by the EcuM.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-26 BswM_EcuM_CurrentWakeup

4.2.26 BswM_EcuM_RequestedState

Prototype	
`void BswM_ EcuM_RequestedState (EcuM_StateType State, EcuM_RunStatusType CurrentStatus)`	
Parameter	
`State`	The requested state by EcuMFlex.
`CurrentStatus`	The new result of the Run Request Protocol.
Return code	
`void`	-
Functional Description	
Function called by EcuM to indicate the request of a run request protocol state.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for different states. > Must only be called by the EcuM.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-27 BswM_EcuM_RequestedState

4.2.27 BswM_MainFunction

Prototype	
`void BswM_MainFunction (void)`	
Parameter	
`None`	-
Return code	
`void`	-
Functional Description	
Main function of the BswM.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant. > This function must be called with the configured cycle time by the SchM [6].	
Call Context	
> This function can be called from task context.	

Table 4-28 BswM_MainFunction

4.2.28 BswM_NvM_CurrentBlockMode

Prototype	
`void BswM_NvM_CurrentBlockMode(NvM_BlockIdType Block, NvM_RequestResultType CurrentBlockMode)`	
Parameter	
`Block`	The Block that the new NvM Mode corresponds to.
`CurrentBlockMode`	The current block mode of the NvM block.
Return code	
`void`	-
Functional Description	
Function called by NvM to indicate the current block mode of an NvM block.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for different blocks. > This function must only be called by NvM.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-29 BswM_NvM_CurrentBlockMode

4.2.29 BswM_NvM_CurrentJobMode

Prototype	
`void` **`BswM_NvM_CurrentJobMode`**`(uint8 ServiceId, NvM_RequestResultType CurrentJobMode)`	
Parameter	
`ServiceId`	Indicates whether the callback refers to multi block services NvM_ReadAll or NvM_WriteAll.
`CurrentJobMode`	Current state of the multi block job indicated by parameter ServiceId.
Return code	
`void`	-
Functional Description	
Function called by NvM to inform the BswM about the current state of a multi block job.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for different services. > This function must only be called by NvM.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-30 BswM_NvM_CurrentJobMode

4.2.30 BswM_PduR_PreTransmit

Prototype	
`void BswM_PduR_PreTransmit(PduIdType TxPduId, const PduInfoType *PduInfoPtr)`	
Parameter	
`TxPduId`	The PduR ID of the PDU to transmit.
`PduInfoPtr`	Pointer which stores all information about the PDU. Not used by current implementation.
Return code	
`void`	-
Functional Description	
Function called by PduR to inform the BswM about an upcoming PDU Transmit Event.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for an id which does not belong to the same configured port. > This function must only be called by the PduR.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-31 BswM_PduR_PreTransmit

4.2.31 BswM_PduR_RxIndication

Prototype	
`void BswM_PduR_RxIndication(PduIdType RxPduId, const PduInfoType *PduInfoPtr)`	
Parameter	
`RxPduId`	The PduR ID of the received PDU.
`PduInfoPtr`	Pointer which stores all information about the PDU. Not used by current implementation.
Return code	
`void`	-
Functional Description	
Function called by PduR to inform the BswM about a received PDU.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for a RxPduId which does not belong to the same configured port. > This function must only be called by the PduR.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-32 BswM_PduR_RxIndication

4.2.32 BswM_PduR_TpRxIndication

Prototype	
`void BswM_PduR_TpRxIndication(PduIdType id, Std_ReturnType result)`	
Parameter	
`id`	The PduR ID of the received PDU.
`result`	Result of the reception. Not used by current implementation.
Return code	
`void`	-
Functional Description	
Function called by PduR to inform the BswM about a received TP PDU.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for an id which does not belong to the same configured port. > This function must only be called by the PduR.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-33 BswM_PduR_TpRxIndication

4.2.33 BswM_PduR_TpStartOfReception

Prototype	
`void BswM_PduR_TpStartOfReception ( PduIdType id, PduInfoType *info, PduLengthType TpSduLength,PduLengthType *bufferSizePtr)`	
Parameter	
`id`	The PduR ID of the received PDU.
`info`	Pointer which stores all information about the PDU. Not used by current implementation.
`TpSduLength`	Total length of the I-PDU to be received. Not used by current implementation.
`bufferSizePtr`	Pointer to the receive buffer. Not used by current implementation.
Return code	
`void`	-
Functional Description	
Function called by PduR to inform the BswM about the start of TP PDU Reception.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for an id which does not belong to the same configured port. > This function must only be called by the PduR.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-34 BswM_PduR_TpStartOfReception

4.2.34 BswM_PduR_TpTxConfirmation

Prototype	
`void BswM_PduR_TpTxConfirmation(PduIdType id, Std_ReturnType result)`	
Parameter	
`id`	The PduR ID of the sent TP PDU.
`result`	Result of the transmission. Not used by current implementation.
Return code	
`void`	-
Functional Description	
Function called by PduR to inform the BswM about a sent TP PDU.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for an id which does not belong to the same configured port. > This function must only be called by the PduR.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-35 BswM_PduR_TpTxConfirmation

4.2.35 BswM_PduR_Transmit

Prototype	
`void BswM_PduR_Transmit( PduIdType id, const PduInfoType *PduInfoPtr)`	
Parameter	
`id`	The PduR ID of the PDU to transmit.
`PduInfoPtr`	Pointer which stores all information about the PDU. Not used by current implementation.
Return code	
`void`	-
Functional Description	
Function called by PduR to inform the BswM about a PDU Transmit Event	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for an id which does not belong to the same configured port. > This function must only be called by the PduR.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-36 BswM_PduR_Transmit

4.2.36 BswM_PduR_TxConfirmation

Prototype	
`void BswM_PduR_TxConfirmation( PduIdType TxPduId )`	
Parameter	
`TxPduId`	The PduR ID of the sent PDU.
Return code	
`void`	-
Functional Description	
Function called by PduR to inform the BswM about a sent PDU.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for a TxPduId which does not belong to the same configured port. > This function must only be called by the PduR.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-37 BswM_PduR_TxConfirmation

4.2.37 BswM_Sd_EventHandlerCurrentState

Prototype	
`void BswM_Sd_EventHandlerCurrentState(uint16 SdEventHandlerHandleId, Sd_EventHandlerCurrentStateType EventHandlerStatus )`	
Parameter	
`SdEventHandlerHandleId`	HandleId to identify the EventHandler
`EventHandlerStatus`	Status of the EventHandler
Return code	
`void`	-
Functional Description	
Function called by Service Discovery to indicate current status of the EventHandler (requested/released).	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for different handles. > This function must only be called by Sd.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-38 BswM_Sd_EventHandlerCurrentState

4.2.38 BswM_Sd_ClientServiceCurrentState

Prototype	
`void BswM_Sd_ClientServiceCurrentState(uint16 SdClientServiceHandleId, Sd_ClientServiceCurrentStateType CurrentClientState)`	
Parameter	
`SdClientServiceHandleId`	HandleId to identify the ClientService.
`CurrentClientState`	Current state of the ClientService.
Return code	
`void`	-
Functional Description	
Function called by Service Discovery to indicate current state of the Client Service (available/down).	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for different handles. > This function must only be called by Sd.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-39 BswM_Sd_ClientServiceCurrentState

4.2.39 BswM_Sd_ConsumedEventGroupCurrentState

Prototype	
`void BswM_Sd_ConsumedEventGroupCurrentState( uint16 SdConsumedEventGroupHandleId, Sd_ConsumedEventGroupCurrentStateType ConsumedEventGroupState)`	
Parameter	
`SdConsumedEventGroupHandleId`	HandleId to identify the Consumed Eventgroup.
`ConsumedEventGroupState`	Status of the Consumed Eventgroup.
Return code	
`void`	-
Functional Description	
Function called by Service Discovery to indicate current status of the Consumed Eventgroup (available/down).	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for different handles. > This function must only be called by Sd.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-40 BswM_Sd_ConsumedEventGroupCurrentState

4.2.40 BswM_Nm_StateChangeNotification

Prototype	
`void BswM_Nm_StateChangeNotification(NetworkHandleType nmNetworkHandle, Nm_StateType nmPreviousState, Nm_StateType nmCurrentState)`	
Parameter	
`nmNetworkHandle`	Identification of the NM-channel
`nmPreviousState`	Previous state of the NM-channel (Parameter not used)
`nmCurrentState`	Current (new) state of the NM-channel
Return code	
`void`	-
Functional Description	
Function called by Nm to inform the BswM about its current state.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for different networks. > This function must only be called by Nm.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-41 BswM_Nm_StateChangeNotification

4.2.41 BswM_RuleControl

Prototype	
`void BswM_RuleControl (BswM_HandleType ruleId, uint8 state)`	
Parameter	
`ruleId`	The external ID of the rule which shall be changed. Symbolic Name Define shall be used.
`state`	The new rule state. Following values are valid: Disable Rule: `BSWM_DEACTIVATED` Enable Rule: `BSWM_UNDEFINED`, `BSWM_TRUE` or `BSWM_FALSE`
Return code	
`void`	-
Functional Description	
Sets a new state to a given rule whereby rule can be enabled or disabled.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant for different rules. > This function should be called by an action of BswM.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-42 BswM_RuleControl

4.2.42 BswM_WdgM_RequestPartitionReset

Prototype	
`void BswM_WdgM_RequestPartitionReset (ApplicationType Application)`	
Parameter	
Application	The Block that the new NvM Mode corresponds to.
Return code	
`void`	-
Functional Description	
Function called by WdgM to request a reset of the corresponding partition of given application.	
Particularities and Limitations	
> This function is synchronous. > This function is reentrant. > This function must only be called by WdgM.	
Call Context	
> This function can be called from task and interrupt context.	

Table 4-43 BswM_WdgM_RequestPartitionReset

4.3 Services Used by BswM

In the following table services provided by other components, which are used by the BswM are listed. For details about prototype and functionality refer to the documentation of the providing component.

Component	API
ComM	ComM_CommunicationAllowed
ComM	ComM_LimitChannelToNoComMode
ComM	ComM_RequestComMode
Com	Com_IpduGroupStart
Com	Com_IpduGroupStop
Com	Com_EnableReceptionDM
Com	Com_DisableReceptionDM
Com	Com_SwitchIpduTxMode
Com	Com_TriggerIPDUSend
Dcm	Dcm_SetChannelReady
DEM	Dem_Init
DEM	Dem_Shutdown
DET	Det_ReportError
EcuM	EcuM_GoDown
EcuM	EcuM_GoHalt
EcuM	EcuM_GoPoll
EcuM	EcuM_SelectShutdownTarget
EcuM	EcuM_SetState
EcuM	EcuM_ClearValidatedWakeupEvent
EthIf	EthIf_StartAllPorts
IdsM	IdsM_BswM_StateChanged
J1939Dcm	J1939Dcm_SetState
J1939Rm	J1939Rm_SetState
LinSM	LinSM_ScheduleRequest
Nm	Nm_DisableCommunication
Nm	Nm_EnableCommunication
NvM	NvM_WriteAll
NvM	NvM_CancelWriteAll
PduR	PduR_EnableRouting
PduR	PduR_DisableRouting
RTE	Rte mode switch. The API name is configurable.
SchM	SchM_Enter_BswM_BSWM_EXCLUSIVE_AREA_0
SchM	SchM_Exit_BswM_BSWM_EXCLUSIVE_AREA_0
Sd	Sd_ConsumedEventGroupSetState

Component	API
Sd	Sd_ClientServiceSetState
Sd	Sd_ServerServiceSetState

Table 4-44 Services used by the BswM

4.4 Callback Functions

There are no callback functions in the BswM.

4.5 Configurable Interfaces

4.5.1 Callout Functions

A User Callout Function can be used as an item of an Action List. If the declaration of the callout function already exists, the integrator must provide an extern declaration of the function via a user include file.

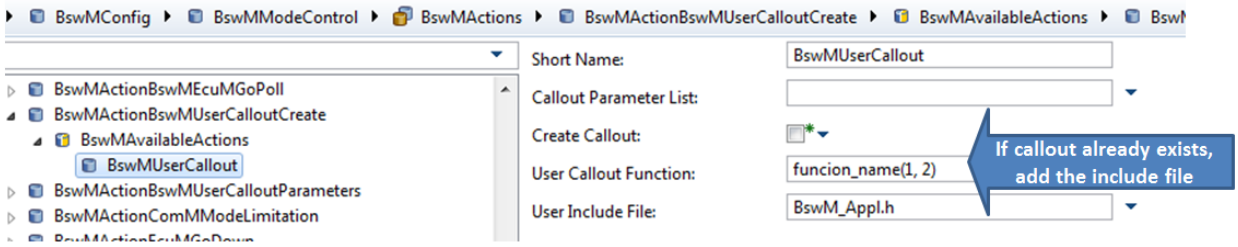

Figure 4-1 Existing callout functions

If the BswM is to generate the user callout prototype: the checkbox “Create Callout” should be set and the parameter prototypes should be defined in the given field as list separated with semicolons. The function prototype is generated in “BswM_Callout_Stubs.c”

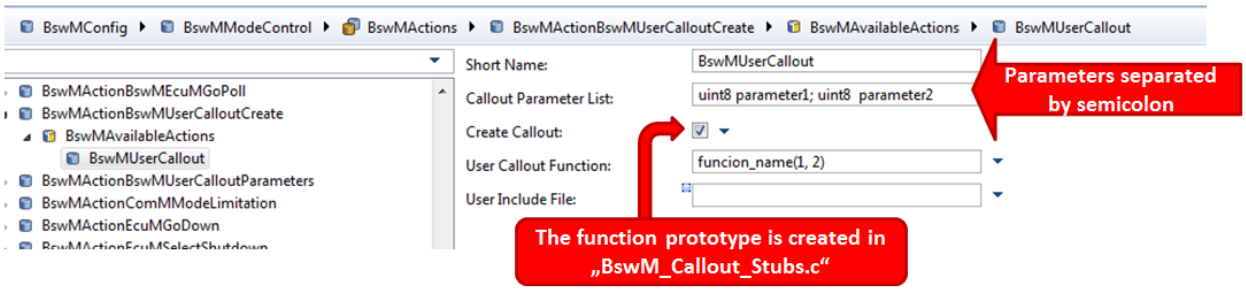

Figure 4-2 Generate prototype of callout functions

The BswM callout function declaration is described in the following table:

Prototype	
`void [Callout Function Name] ( <parameters> )`	
Parameter	
-	-
Return code	
-	-
Functional Description	
If a User Callout is configured as an item of an Action List the BswM calls this function in the context of the appropriate rule.	
Particularities and Limitations	
-	
Call context	
> Interrupt or task context, depends on the mode/rule configuration in which the callout is used.	

Table 4-45 User Callout

4.6 Service Ports

The BswM has a service component which depends on the following containers:

- BswMSwcModeRequest
- BswMSwcModeNotification
- BswMSwitchPort
- BswMRteModeRequestPort
- BswMModeDeclaration

These containers are described in the following chapters.

4.6.1 BswMSwcModeRequest (R-Port)

BswM is able to receive modes by Sender-Receiver mode ports (Require Port). This can be done by using BswMSwcModeRequest.

The BswMSwcModeRequest has a reference to a Mode-Declaration-Group-Prototype and an Instance-Reference to a VARIABLE-DATA-PROTOTYPE. If the reference to the Mode-Declaration-Group-Prototype is configured, it is not possible to determine a relationship to a Sender-Receiver-Interface. Therefore, it is necessary to create a new Sender-Receiver-Interface. The given BswMModeRequestDataElementPrototypeName will be used as DataElement name.

Sender-Receiver-Interfaces are named

- BswM_SRI_{Mode-Switch-Interface Name}_{Mode-Declaration-Group-Prototype Name}

If the Instance-Reference to a VARIABLE-DATA-PROTOTYPE is used, BswM reuses the existing Sender-Receiver interface.

In both cases, created Ports are named:

> Receive_{BswMSwcModeRequest Name}

4.6.2 BswMSwcModeNotification (R-Port)

BswM is able to receive modes by Mode-Switch mode ports (Require Port). This can be done by using BswMSwcModeNotification.

The BswM has a reference to a Mode-Declaration-Group-Prototype. From this prototype it is possible to determine a Mode-Switch-Interface which will be reused for the created port.

Created ports are named:

> Notification_{BswMSwcModeNotification Name}

4.6.3 BswMSwitchPort (P-Port)

BswM is able to switch modes by Mode-Switch mode ports (Provide Port). This can be done by using a BswMSwitchPort. The BswM has a reference to a Mode Declaration Group Prototype. From this prototype it is possible to determine a Mode-Switch-Interface which will be reused for the created port.

Created ports are named:

> Switch_{BswMSwitchPort Name}

4.6.4 BswMRteModeRequestPort (P-Port)

BswM is able to send modes by Sender-Receiver mode ports (Require Port). This can be done by using a port of type BswMRteModeRequestPort in a BswMRteModeRequest action. The BswM uses an Instance-Reference to a VARIABLE-DATA-PROTOTYPE, which represents the DataElement of an already existing Sender-Receiver-Interface. This interface is used by the created P-Port.

Created ports are named:

> Provide_{BswMRteModeRequestPort Name}

4.6.5 BswMModeDeclaration

To facilitate SWC ModeRequest Handling, the BswM is able to provide Mode-Declarations by itself. To use this, a BswMModeDeclaration container with corresponding modes can be created. The BswM SWC Validation creates automatically a Mode-Declaration, the corresponding Implementation-Type and a Mode-Switch-Interface with a Mode-Declaration-Group-Prototype.

The Mode-Switch-Interface is named:

> BswM_MSI_{BswMModeDeclaration Name}

The corresponding Mode-Declaration-Group-Prototype is named:

> BswM_MDGP_{BswmModeDeclaration Name}

The Implementation-Type is named:

> BswM_{BswMModeDeclaration Name}

5 AUTOSAR Standard Compliance

5.1 Deviations

5.1.1 Inclusion of the header Com_Types.h

A non-AUTOSAR header is used within the code. The source file BswM_Cfg.h includes Com_Types.h. This header has been included because it defines the type `Com_IpduGroupIdType`.

In case the project in use does not contain a MICROSAR Classic Com module, it is necessary to add a header file with the name “Com_Types.h”, which defines the type “Com_IpduGroupIdType”.

5.1.2 Port Names

Notice that in the BswM AUTOSAR SWS the name of the ports is specified as:

modeNotificationPort_{Name}

modeRequestPort_{Name}

modeSwitchPort_{Name}

However, the structure of the name port is as follows:

Notification_{Name}

Request_{Name}

Switch_{Name}

Furthermore, BswMRteModeRequestPort are named:

Provide_{Name}

5.2 Additions/ Extensions

5.2.1 Optional Interfaces

The BswM supports the following “Optional Interfaces” defined in [1] [BswM0008]:

- ComM_LimitChannelToNoComMode
- ComM_RequestComMode
- Com_IpduGroupStart
- Com_IpduGroupStop
- Com_EnableReceptionDM
- Com_DisableReceptionDM
- Com_SwitchIpduTxMode

- Det_ReportError
- EcuM_GoDown
- EcuM_GoHall
- EcuM_GoPoll
- EcuM_SelectShutdownTarget
- J1939Dcm_SetState
- J1939Rm_SetState
- LinSM_ScheduleRequest
- Nm_DisableCommunication
- Nm_EnableCommunication
- Sd_ClientServiceSetState
- Sd_ConsumedEventGroupSetState
- Sd_ServerServiceSetState

5.2.2 Nm Indication

BswM supports the NM indication by using the API “BswM_Nm_StateChangeNotification”. The mode request source is of type “BswMNMIndication”. In order to use this feature the Nm module must be configured as follows:

- NmStateChangeIndEnabled must be set to true
- NmStateChangeIndCallback must be set to “BswM_Nm_StateChangeNotification”
- NmCallbacksPrototypeHeader must be set to “BswM_Nm.h” (or any other header which includes BswM_Nm.h)

5.2.3 User Condition Functions

A User Condition Function can be used in a Rule Condition. The integrator must provide an extern declaration of the function via an application header file.

In the same manner, with the request port of type “User Condition”, it is possible to compare any variable.

The integrator must make sure that the return value of the function is compatible with the value to compare with.

5.2.4 Creation of Mode Declarations

The BswM is able to provide Mode Declarations by itself in order to facilitate the SWC Mode Request Handling. For further information see 4.6.5.

5.2.5 Timers

A Timer offers the possibility to execute action time dependently. Therefore, a Mode Request Port of type BswMTimer must be created. This port represents a timer which can be started and stopped by a BswMTimerControl Action. The value for the timer start can be set in the TimerControlAction.

The timer should be a multiple of the BswMMainFunctionPeriod (timer is decreased in the MainFunction). In case the timer is not multiple of the main function period, it will be rounded up. The timer must be used in a condition to trigger the corresponding actions. The state of a timer can be STARTED, STOPPED or EXPIRED.

5.2.6 Generic Symbolic Values

Generic ports offer the possibility to define Symbolic Values. In order to realize this, create a BswMGenericRequestMode inside the BswMGenericRequest container. These Symbolic Values are necessary for the Generic Actions (see chapter 5.2.7).

5.2.7 Generic Actions

BswM supports setting a generic mode by an action. In order to configure it, a BswMGenericModeSwitch action must be created. Here, the generic mode and the corresponding value can be chosen.

5.2.8 Immediate request in BswM_Init()

All configured immediate request are processed once within the function BswM_Init, in order to arbitrate the initial states. This behavior can be changed for each port in the configuration.

5.2.9 Mode Handling Forced Immediate

The additional mode handling type “Forced Immediate” allows mode requests to be executed immediately interrupting other requests. For more information see chapter 2.4.2.

5.2.10 Rule Control

If Rule Control is used, rules can be activated or deactivated during runtime. Furthermore, rules can be deactivated in configuration by using `BSWM_DEACTIVATED` as initialization value. For further information see 4.2.41.

5.2.11 Support of Com ASR3 IPduGroup APIs

If Microsar Classic Com is used and Com is configured to use ASR3 IPduGroup APIs, BswM will use the following APIs in its IPduGroup actions instead of the ASR4 APIs:

- Com_IpduGroupStart
- Com_IpduGroupStop
- Com_EnableReceptionDM
- Com_DisableReceptionDM

For further information see [8].

5.3 Limitations

5.3.1 Configurable interfaces that are not supported

5.3.1.1 EcuM Indication for EcuM Flex

The ModeRequestPort of type EcuMIndication is not supported for MICROSAR Classic EcuM Flex without enabled ModeHandling. This is due to the fact, that BswM calls most of EcuM Function itself. So, the notifications from EcuM to BswM will be done in the context of the BswM and this leads to a queued processing of mode changes.

If EcuM notifies more than one mode change, previously notified mode changes get lost and Rules which should be triggered to this mode will be skipped. As this is not the desired behavior, the EcuM Indication is no longer supported during configuration of the module.

5.3.2 Optional Interfaces

Within the predefined actions, the BswM does not support the following “Optional Interfaces” defined by [1] [BswM0008]:

- ComM_GetCurrentComMode
- ComM_GetInhibitionStatus
- ComM_GetMaxComMode
- ComM_GetRequestedComMode
- ComM_GetStatus
- ComM_GetVersionInfo
- ComM_LimitECUToNoComMode
- ComM_MainFunction_<Channel_Id>
- ComM_PreventWakeUp
- ComM_ReadInhibitCounter
- ComM_ResetInhibitCounter
- ComM_SetECUGroupClassification
- ControlIdle

5.3.3 Configuration Variants

Configuration variant Link-Time is not supported.

5.3.4 BSW Modules

Only these BSW Modules are supported for mode indications and arbitrations: CanSM, ComM, Dcm, DoIP, DoIPInt, EcuM, EthIf, EthSm, FrSM, J1939Dcm, J1939Nm, LinSM, LinTp, Nm, NvM , Sd, WdgM and RTE.

6 Glossary and Abbreviations

6.1 Glossary

Term	Description
DaVinci Configurator	Generation tool for MICROSAR Classic components

6.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
CAN	Controller Area Network
Com	Communication (AUTOSAR BSW)
ComM	Communication Manager
CanSM	CAN State Manager
DCM	Diagnostic Communication Manager
DoIP	Diagnostic over IP
DoIPInt	Diagnostic over IP for vehicle internal communication
DEM	Diagnostic Event Manager
DET	Development Error Tracer
ECU	Electronic Control Unit
ECUM	ECU Manager
EthIf	Ethernet Interface
EthSM	Ethernet State Management
Fr	FlexRay
FrSM	FlexRay State Manager
HIS	Hersteller Initiative Software
I-PDU	Interaction Layer Protocol Data Unit
ISR	Interrupt Service Routine
J1939Dcm	J1939 Diagnostic Communication Manager
J1939Nm	J1939 Network Manager
J1939Rm	J1939 Request Manager
LIN	Local Interconnect Network
LinIf	LIN Interface
LinSM	LIN State Manager
LinTp	LIN Transport Protocol
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)

Nm	Network Manager
NvM	Non-Volatile RAM Manager
PduR	Protocol Data Unit Router
PNC	Partial Networking Cluster
RAM	Random Access Memory
RTE	Runtime Environment
Sd	Service Discovery
SchM	Schedule Manager
SWC	Software Component
SWS	Software Specification

Table 6-1 Abbreviations

7 Contact

Visit our website for more information on

- News
- Products
- Demo Software
- Support
- Training data
- Addresses

www.vector.com