

Technical Reference

Version 6.00.01

Authors	Leticia Garcia Herrera, Thomas Kuhl, Philipp Ritter
Status	Released

Leticia Garcia, Thomas Kuhl	2012-08-02	1.00.00	Creation of document.
Leticia Garcia, Thomas Kuhl	2012-09-27	1.01.00	Addition of feature, support of EthSM . Chapters 3.1, 4.1, 5.2and 6.3.
Leticia Garcia, Thomas Kuhl	2013-01-31	1.02.00	Addition of feature, support of NvM. Chapter 3.1, 4.1, 5.2 and 5.3.
Leticia Garcia, Thomas Kuhl	2012-03-26	1.03.00	Support of Post-build variant. Chapters 4.1, 4.2, 5.1and 5.2. Deviation from AUTOSAR. Header included: Com_Types.h. Chapter 6.1
Leticia Garcia	2013-10-21	2.00.00	Addition of extension in chapter 6.2. Deletion of limitations in chapter 6.3. DET errors added in chapter 3.6.1. Dynamic files added in chapter 4.1.2. Chapter 4.2 was changed. Chapter 4.3 was added.
Leticia Garcia	2013-12-04	2.00.01	Chapter 3.3 was extended. Chapter 3.4.2 was added. Chapter 3.6.1 error code added. Chapter 4.5 was extended Chapter 6.2 was extended.
Leticia Garcia	2013-02-18	2.01.00	Extended chapters: 3.1, 3.1.2, 3.6.1, 4.1.1, 5.2.13, 5.2.14, 5.2.31, 5.2.32, 5.2.33, 5.2.34 5.3 and 6.2.1. Added chapters: 4.3.3, 5.6, and 6.2.2. Removed deviation about Com_IpduGroupControl usage.
Philipp Ritter	2014-06-13	3.00.00	Extended chapters: 3.1.2, 3.5, 5.6.1, 6.2.1, 6.2.8 Added chapters: 5.2.35, 6.2.10, 6.2.11 Updated Figures: Figure 3-2,

			Figure 3-3
Philipp Ritter	2014-10-22	4.00.00	Extended Chapters: 3.1, 3.6.1, 4.1.1, 4.3.3, 5.2.4 Added chapters: 5.2.36
Philipp Ritter	2015-02-02	5.00.00	Extended chapters: 3.6.1, 4.3.3, 6.3.3, 6.3.4 Added chapters: 5.2.17 Removed: Limitation for multiple configurations
Philipp Ritter	2015-07-29	6.00.00	Extended chapters: 3.1, 3.1.2, 3.6.1, 4.3.3, 5.3 Added chapters: 4.3.4, 5.2.21, 5.2.25, 5.2.26, 5.2.27, 5.2.28, 5.2.29, 5.2.30
Philipp Ritter	2015-12-10	6.00.01	Updated Figure 4-6

[1]	AUTOSAR	AUTOSAR_SWS_BSWModeManager.pdf	1.4.0
[2]	AUTOSAR	AUTOSAR_EXP_ModemanagementGuide	2.1.0
[3]	AUTOSAR	AUTOSAR_SWS_DevelopmentErrorTracer.pdf	3.2.0
[4]	AUTOSAR	AUTOSAR_TR_BSWModuleList.pdf	1.6.0
[5]	AUTOSAR	AUTOSAR_SWS_DiagnosticEventManager.pdf	4.2.0
[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).



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.

.....	
2.1	Architecture Overview 11
.....	
3.1	Features 13
3.1.1	Deviations 14
3.1.2	Additions/ Extensions 14
3.2	Initialization 14
3.3	States 14
3.4	Mode Management 16
3.4.1	Immediate Mode Handling 17
3.4.2	Forced Immediate Mode Handling 17
3.4.3	Deferred Mode Handling 17
3.5	Execution of Action Lists 20
3.6	Error Handling 20
3.6.1	Development Error Reporting 20
3.6.2	Production Code Error Reporting 22
.....	
4.1	Scope of Delivery 23
4.1.1	Static Files 23
4.1.2	Dynamic Files 24
4.2	Initialization of Other Software Modules 24
4.2.1	Using the Basic Editor 24
4.2.2	Using the Comfort View 26
4.3	Support of Preconfigured State Machines (Auto-Configuration) 26
4.3.1	Initialization 27
4.3.2	ECU State Handling 29
4.3.3	Communication Control 31
4.3.4	Service Discovery Control 33
4.4	Critical Sections 33
4.5	Cyclic Task 33
4.6	NvM – BswM configuration 33
.....	
5.1	Type Definitions 34
5.2	Services Provided by BswM 35

5.2.1	BswM_InitMemory	35
5.2.2	BswM_Init	35
5.2.3	BswM_Deinit	36
5.2.4	BswM_GetVersionInfo	36
5.2.5	BswM_RequestMode	37
5.2.6	BswM_ComM_CurrentMode	37
5.2.7	BswM_ComM_CurrentPNCMode	38
5.2.8	BswM_Dcm_ApplicationUpdated	38
5.2.9	BswM_Dcm_CommunicationMode_CurrentState	39
5.2.10	BswM_CanSM_CurrentState	39
5.2.11	BswM_EthSM_CurrentState	40
5.2.12	BswM_FrSM_CurrentState	40
5.2.13	BswM_J1939DcmBroadcastStatus	41
5.2.14	BswM_J1939Nm_StateChangeNotification	41
5.2.15	BswM_LinSM_CurrentState	42
5.2.16	BswM_LinSM_CurrentSchedule	42
5.2.17	BswM_LinSM_ScheduleEndNotification	43
5.2.18	BswM_LinTp_RequestMode	43
5.2.19	BswM_EcuM_CurrentState	44
5.2.20	BswM_EcuM_CurrentWakeup	44
5.2.21	BswM_EcuM_RequestedState	45
5.2.22	BswM_MainFunction	45
5.2.23	BswM_NvM_CurrentBlockMode	46
5.2.24	BswM_NvM_CurrentJobMode	46
5.2.25	BswM_PduR_RxIndication	47
5.2.26	BswM_PduR_TpRxIndication	47
5.2.27	BswM_PduR_TpStartOfReception	48
5.2.28	BswM_PduR_TpTxConfirmation	48
5.2.29	BswM_PduR_Transmit	49
5.2.30	BswM_PduR_TxConfirmation	49
5.2.31	BswM_Sd_EventHandlerCurrentState	50
5.2.32	BswM_Sd_ClientServiceCurrentState	50
5.2.33	BswM_Sd_ConsumedEventGroupCurrentState	51
5.2.34	BswM_Nm_StateChangeNotification	52
5.2.35	BswM_RuleControl	53
5.2.36	BswM_WdgM_RequestPartitionReset	53
5.3	Services Used by BswM	54
5.4	Callback Functions	55
5.5	Configurable Interfaces	55
5.5.1	Callout Functions	55
5.6	Service Ports	56

5.6.1	BswMSwcModeRequest (R-Port).....	56
5.6.2	BswMSwcModeNotification (R- Port)	56
5.6.3	BswMSwitchPort (P- Port).....	57
5.6.4	BswMRteModeRequestPort (P-Ports).....	57
5.6.5	BswMModeDeclaration	57
.....		
6.1	Deviations	58
6.1.1	Inclusion of the header Com_Types.h	58
6.1.2	Port Names	58
6.2	Additions/ Extensions.....	58
6.2.1	Optional Interfaces	58
6.2.2	Nm Indication	59
6.2.3	User Condition Functions	59
6.2.4	Creation of Mode Declarations	60
6.2.5	Timers.....	60
6.2.6	Generic Symbolic Values	60
6.2.7	Generic Actions.....	60
6.2.8	Immediate request in BswM_Init()	60
6.2.9	Mode Handling Forced Immediate	60
6.2.10	Rule Control.....	60
6.2.11	Support of Com ASR3 IPduGroup APIs.....	61
6.3	Limitations.....	61
6.3.1	Configurable interfaces that are not supported.....	61
6.3.1.1	EcuM Indication for EcuM Flex	61
6.3.2	Optional Interfaces	61
6.3.3	Configuration Variants.....	62
6.3.4	BSW Modules	62
.....		
7.1	Glossary	63
7.2	Abbreviations	63
.....		

Figure 2-1	AUTOSAR Architecture.....	11
Figure 2-2	Interfaces to adjacent modules of the BswM.....	12
Figure 3-1	States of the BswM.....	16
Figure 3-2	Sequence Immediate Processing	18
Figure 3-3	Sequence Deferred Mode.....	19
Figure 4-1	Auto-configured state machines.....	27
Figure 4-2	Configure module initialization	28
Figure 4-3	Edit initialization order.....	29
Figure 4-4	Restore default sequence	29
Figure 4-5	Configuration of the features for ECU State Handling	30
Figure 4-6	State Machine of the ECU State Handling	31
Figure 5-1	Existing callout functions	55
Figure 5-2	Generate prototype of callout functions.....	55

Table 1-1	Component history.....	10
Table 3-1	Supported AUTOSAR Standard Conform Features	13
Table 3-2	Not Supported AUTOSAR Standard Conform Features.....	14
Table 3-3	Features Provided Beyond the AUTOSAR Standard	14
Table 3-4	Service IDs	21
Table 3-5	Errors reported to DET	21
Table 4-1	Static Files.....	24
Table 4-2	Dynamic Files	24
Table 5-1	Type definitions.....	34
Table 5-2	BswM_InitMemory	35
Table 5-3	BswM_Init.....	35
Table 5-4	BswM_Deinit.....	36
Table 5-5	BswM_GetVersionInfo	36
Table 5-6	BswM_RequestMode.....	37
Table 5-7	BswM_ComM_CurrentMode.....	37
Table 5-8	BswM_ComM_CurrentPNCMode	38
Table 5-9	BswM_Dcm_ApplicationUpdated.....	38
Table 5-10	BswM_Dcm_CommunicationMode_CurrentState	39
Table 5-11	BswM_CanSM_CurrentState.....	39
Table 5-12	BswM_EthSM_CurrentState	40
Table 5-13	BswM_FrSM_CurrentState	40
Table 5-14	BswM_J1939DcmBroadcastStatus.....	41
Table 5-15	BswM_J1939Nm_StateChangeNotification	41
Table 5-16	BswM_LinSM_CurrentState.....	42
Table 5-17	BswM_LinSM_CurrentSchedule	42
Table 5-18	BswM_LinSM_ScheduleEndNotification	43
Table 5-19	BswM_LinTp_RequestMode.....	43
Table 5-20	BswM_EcuM_CurrentState.....	44
Table 5-21	BswM_EcuM_CurrentWakeup	44
Table 5-22	BswM_EcuM_RequestedState	45
Table 5-23	BswM_MainFunction	45
Table 5-24	BswM_NvM_CurrentBlockMode	46
Table 5-25	BswM_NvM_CurrentJobMode	47
Table 5-26	BswM_PduR_RxIndication	47
Table 5-27	BswM_PduR_TpRxIndication	47

Table 5-28	BswM_PduR_TpStartOfReception.....	48
Table 5-29	BswM_PduR_TpTxConfirmation.....	49
Table 5-30	BswM_PduR_Transmit	49
Table 5-31	BswM_PduR_TxConfirmation	49
Table 5-32	BswM_Sd_EventHandlerCurrentState	50
Table 5-33	BswM_Sd_ClientServiceCurrentState	51
Table 5-34	BswM_Sd_ConsumedEventGroupCurrentState	51
Table 5-35	BswM_Nm_StateChangeNotification	52
Table 5-36	BswM_RuleControl	53
Table 5-37	BswM_WdgM_RequestPartitionReset	53
Table 5-38	Services used by the BswM.....	54
Table 5-39	User Callout.....	56
Table 7-1	Abbreviations.....	64

1

The component history gives an overview over the important milestones that are supported in the different versions of the component.

1.00.00	Creation
1.01.00	Support of Ethernet components was added
1.02.00	Support to NvM.
1.03.00	Post-Build loadable support. SWC mode requests support
2.00.00	Support of timers and user conditions as request ports Generic modes handling extended. Initialization automation and preconfigured state machine to emulate the behavior of EcuM in ASR 3.
2.00.01	Forced Immediate mode handling was added.
2.01.00	Support for NM, J1939Nm, J1939Dcm and Service Discovery (Sd), R Request Port of type SwcModeRequest, SwcModeNotification support immediate request processing and Support of P-Ports (BswMRteModeRequestPort).
3.00.00	Mode Arbitration algorithm changed (first arbitrate all rules, execute action lists afterwards), disabling of rules (Rule Control), support of Com ASR3 IPduGroup APIs, prioritization of action list execution order.
4.00.00	Support for Post-Build selectable and WdgMPartitionReset.
5.00.00	Support of LinScheduleEndNotification
6.00.00	Support of BswM_EcuM_RequestedState, BswM_PduR_RxIndication, BswM_PduR_TpRxIndication, BswM_PduR_TpStartOfReception, BswM_PduR_TpTxConfirmation, BswM_PduR_Transmit, BswM_PduR_TxConfirmation

Table 1-1 Component history

2

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

	4	
	pre-compile, post-build, post-build-selectable	
	BSWM_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
	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.

2.1 Architecture Overview

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

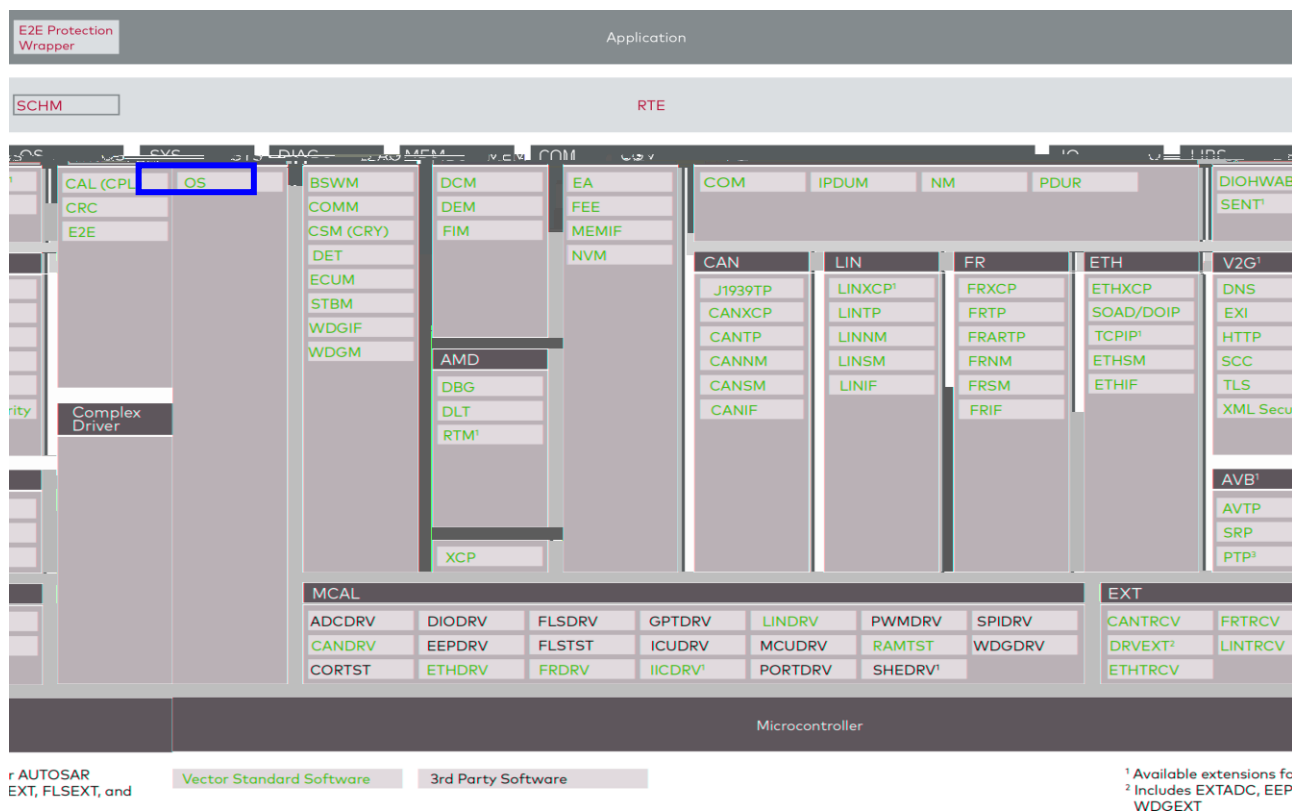


Figure 2-1 AUTOSAR Architecture

3

This chapter describes the general function of the BswM.

3.1 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 3-1 Supported AUTOSAR Standard Conform Features
- > Table 3-2 Not Supported AUTOSAR Standard Conform Features

The following features specified in [1] are supported:

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
Cascade rule conditions
Rte Mode Notification and Switches
Rte Mode Request Interfaces and Ports
Watchdog Manager
Post-Build Loadable
MICROSAR Identity Manager using Post-Build Selectable [9]

Table 3-1 Supported AUTOSAR Standard Conform Features

3.1.1 Deviations

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

Available Action: BswM_TriggerStartUpPhase2
Available Actions: BswM_TriggerSlaveRTESStop

Table 3-2 Not Supported AUTOSAR Standard Conform Features

See Chapter 6.1 for detailed information about other deviations.

3.1.2 Additions/ Extensions

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
Support of ASR3 IPduGroup APIS
PduR mode request ports

Table 3-3 Features Provided Beyond the AUTOSAR Standard

3.2 Initialization

The BswM is initialized via the service function `BswM_Init` (refer to chapter 5.2.2). 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_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.

3.3 States

The state machine diagram in Figure 3-1 shows the general handling of the BswM. Each state is described as follows:

>

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

>

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.

>

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.

>

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.

>

In this state the queued mode arbitration requests are executed.

>

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.



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

- > 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()`.

- >

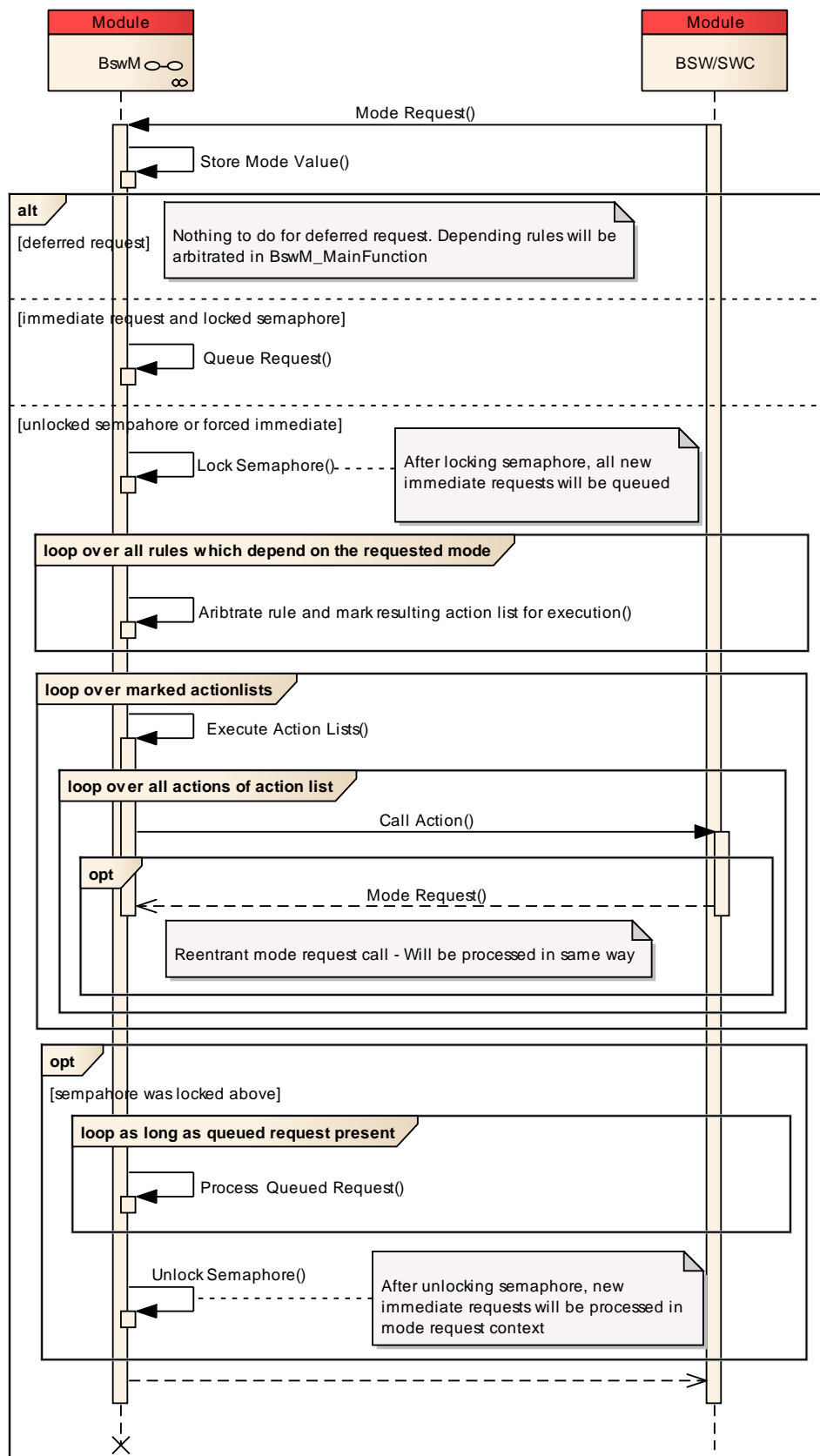


Figure 3-2 Sequence Immediate Processing

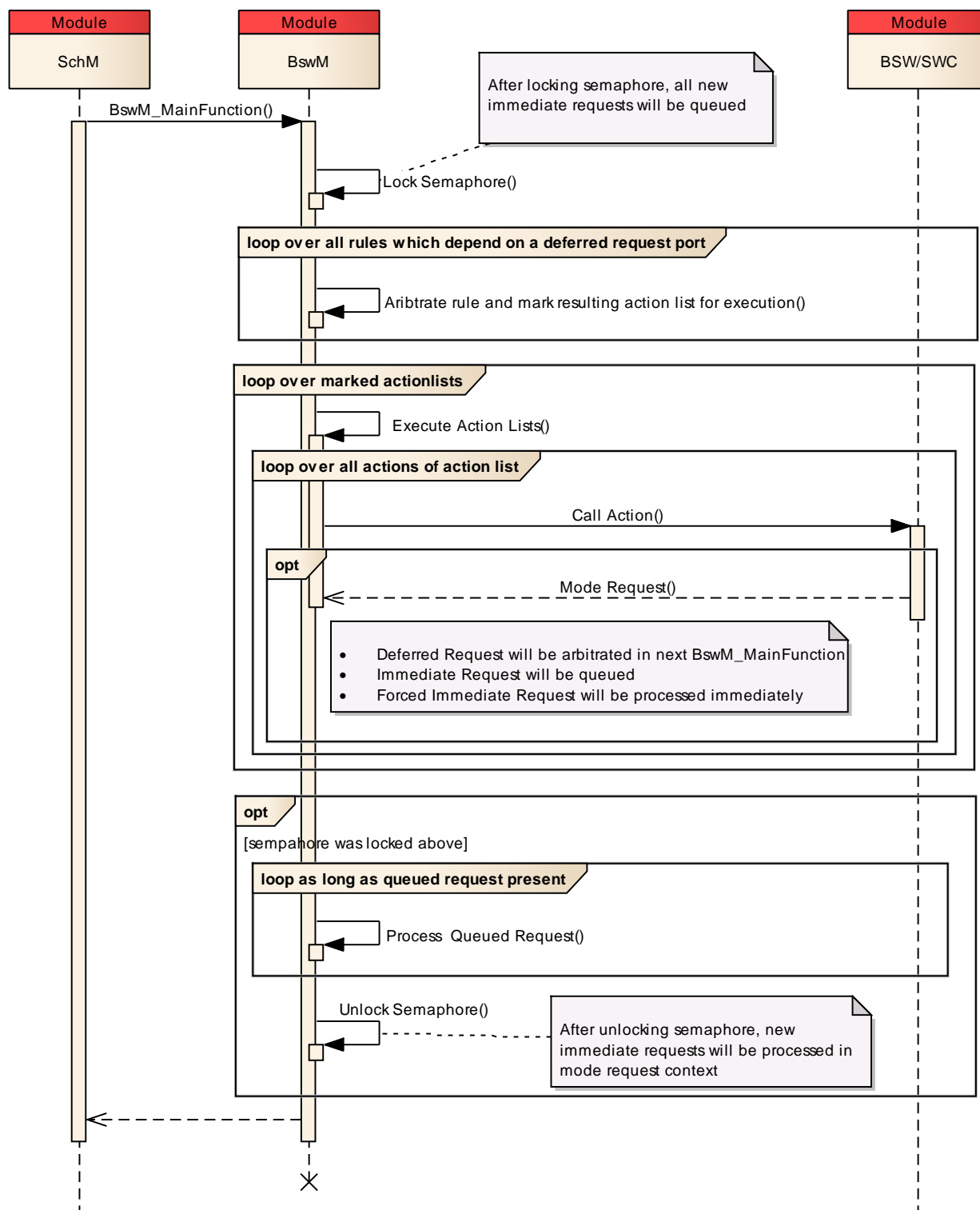


Figure 3-3 Sequence Deferred Mode

3.5 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.

3.6 Error Handling

3.6.1 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 5.2. Table 3-4 presents the service IDs and the related services.

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

BSWM_NVM_CURRENTBLOCKMODE_ID	(0x16)	BswM_NvM_CurrentBlockMode()
BSWM_NVM_CURRENTJOBMODE_ID	(0x17)	BswM_NvM_CurrentJobMode()
BSWM_J1939NM_STATE_ID	(0x18)	BswM_J1939Nm_StateChangeNotification()
BSWM_J1939DCM_BROADCASTSTATUS_ID	(0x1b)	BswM_J1939DcmBroadcastStatus()
BSWM_SD_CLIENTSERVICE_CURRENT_ID	(0x1f)	BswM_Sd_ClientServiceCurrentState()
BSWM_SD_EVENTHANDLER_CURRENT_ID	(0x20)	BswM_Sd_EventHandlerCurrentState()
BSWM_SD_CONSUMEDEVENTGROUP_ID	(0x21)	BswM_Sd_ConsumedEventGroupCurrentState()
BSWM_ECUM_REQUESTEDSTATE_ID	(0x23)	BswM_EcuM_RequestedState()
BSWM_NM_STATE_CHANGE_ID	(0x81)	BswM_Nm_StateChangeNotification()
BSWM_SWCNOTIFICATION_ID	(0x82)	BswM_Notification_<SWC Notification Name>
BSWM_SWCREQUESTMODE_ID	(0x83)	BswM_Read_<SWC Mode Request Name>
BSWM_SETRULESTATE_ID	(0x84)	BswM_RuleControl()
BSWM_LINSM_SCHEDULEENDNOTIFICATION_ID	(0x85)	BswM_LinSM_ScheduleEndNotification()
BSWM_PDUR_RXINDICATION_ID	(0x86)	BswM_PduR_RxIndication()
BSWM_PDUR_TPRXINDICATION_ID	(0x87)	BswM_PduR_TpRxIndication()
BSWM_PDUR_TPSTARTOFRECEPTION_ID	(0x88)	BswM_PduR_TpStartOfReception()
BSWM_PDUR_TPTXCONFIRMATION_ID	(0x89)	BswM_PduR_TpTxConfirmation()
BSWM_PDUR_TRANSMIT_ID	(0x8A)	BswM_PduR_Transmit()
BSWM_PDUR_TXCONFIRMATION_ID	(0x8B)	BswM_PduR_TxConfirmation()

Table 3-4 Service IDs

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

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.
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.

Table 3-5 Errors reported to DET

3.6.2 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.

4

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

4.1 Scope of Delivery

The delivery of the BswM contains the files which are described in chapters 4.1.1 and 4.1.2:

4.1.1 Static Files

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_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.
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 4-1 Static Files

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool.

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.
BswM_Callout_Stubs.c	This file contains the definitions of the call back functions which were configured to be created.

Table 4-2 Dynamic Files

4.2 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.

4.2.1 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.



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.



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.

4.2.2 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 4.3 and 4.3.1.

4.3 Support of Preconfigured State Machines (Auto-Configuration)

The BswM supports preconfigured state machines. The content of these state machines is based on the current 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 Module Initialization” to start configuring.



Created Rules, Actions, Conditions, etc. are only an advice and may be edited by the integrator.

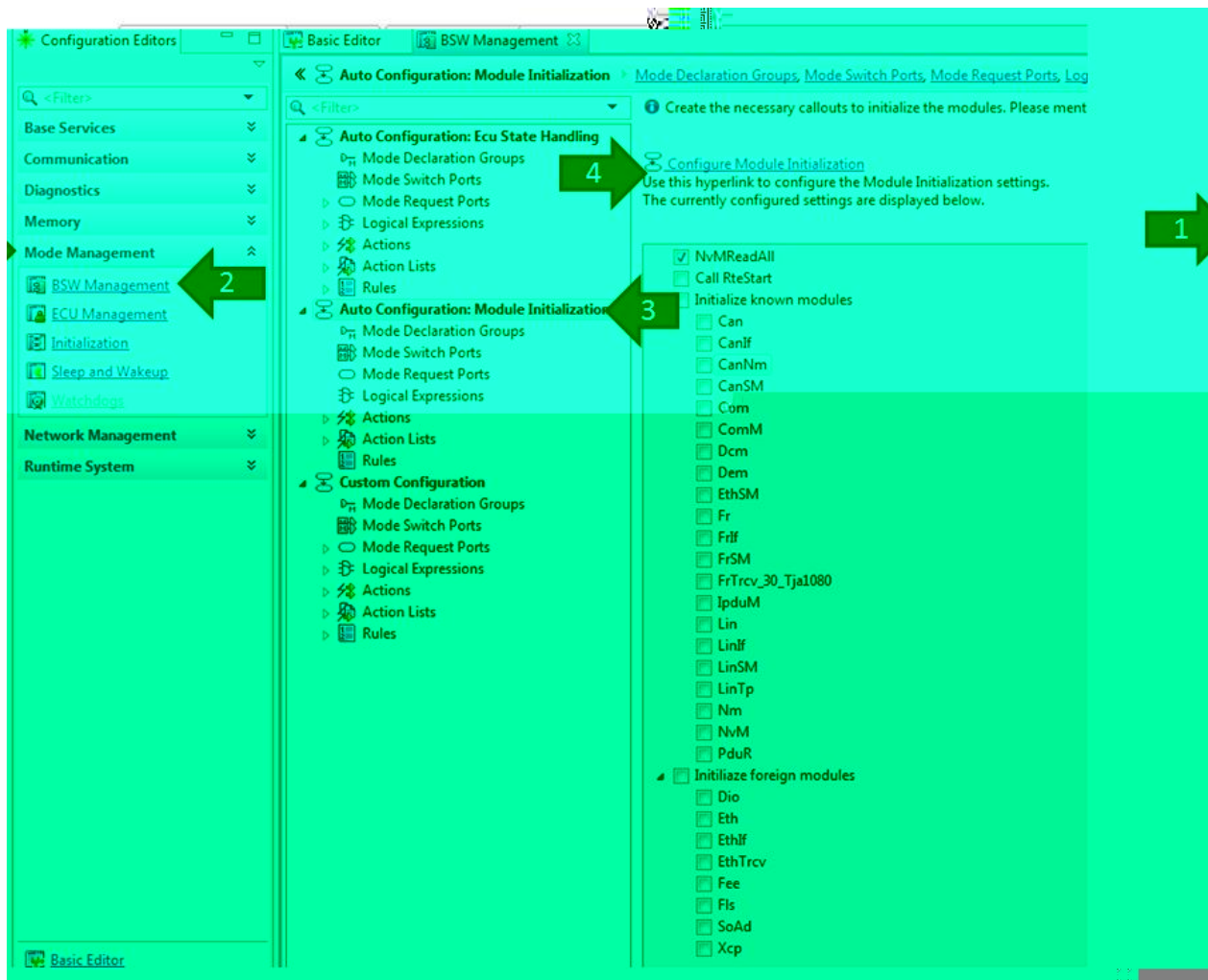


Figure 4-1 Auto-configured state machines

4.3.1 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.

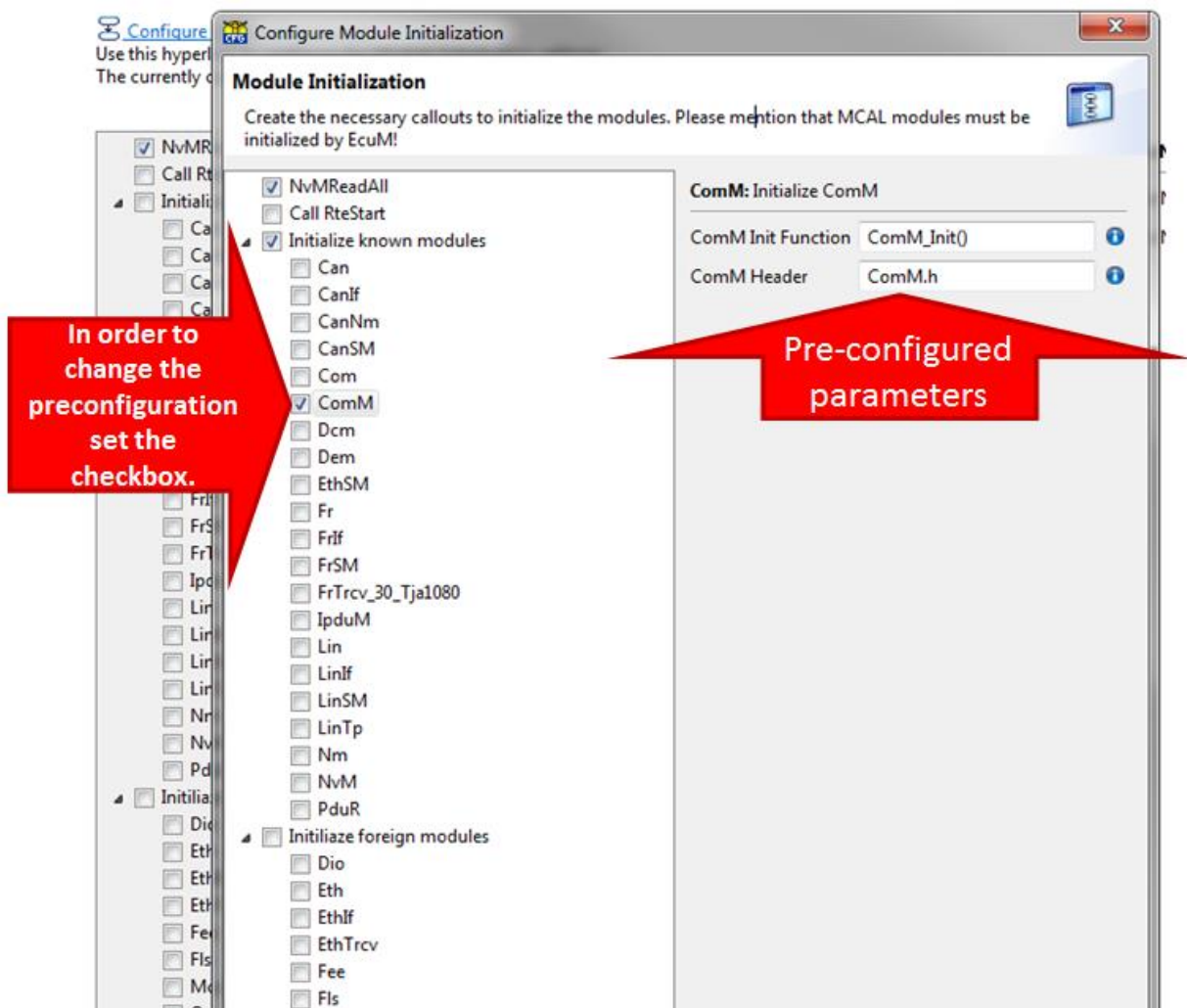


Figure 4-2 Configure module initialization

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.

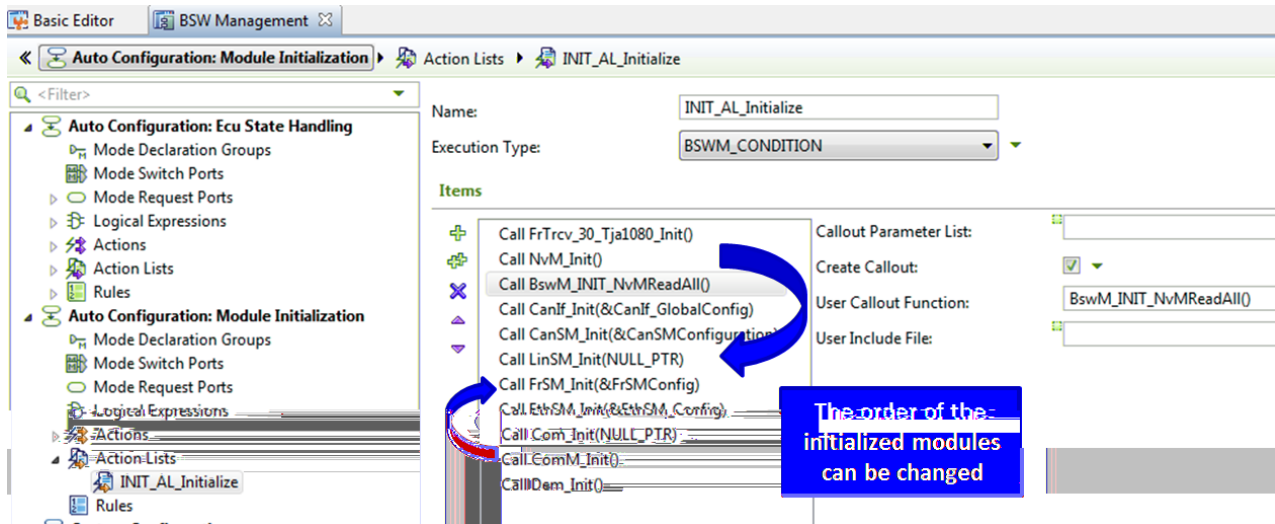


Figure 4-3 Edit initialization order

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.

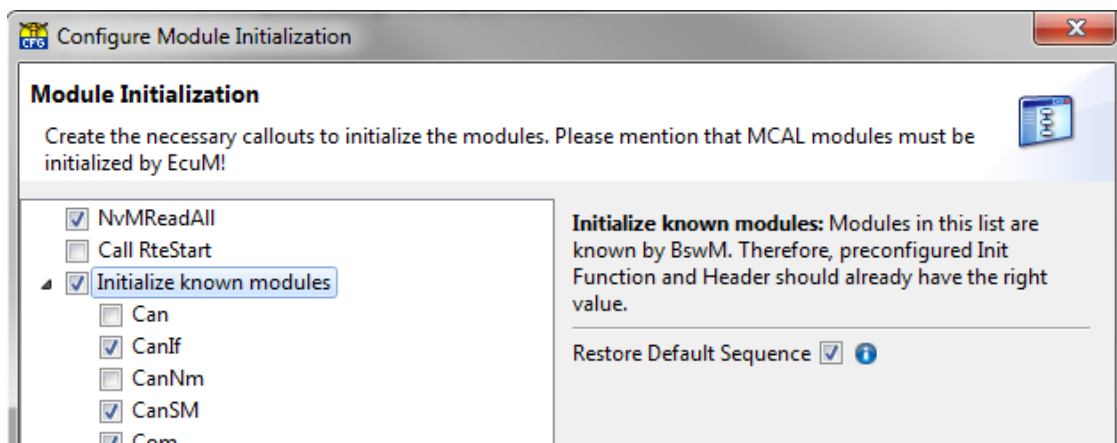


Figure 4-4 Restore default sequence

4.3.2 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.

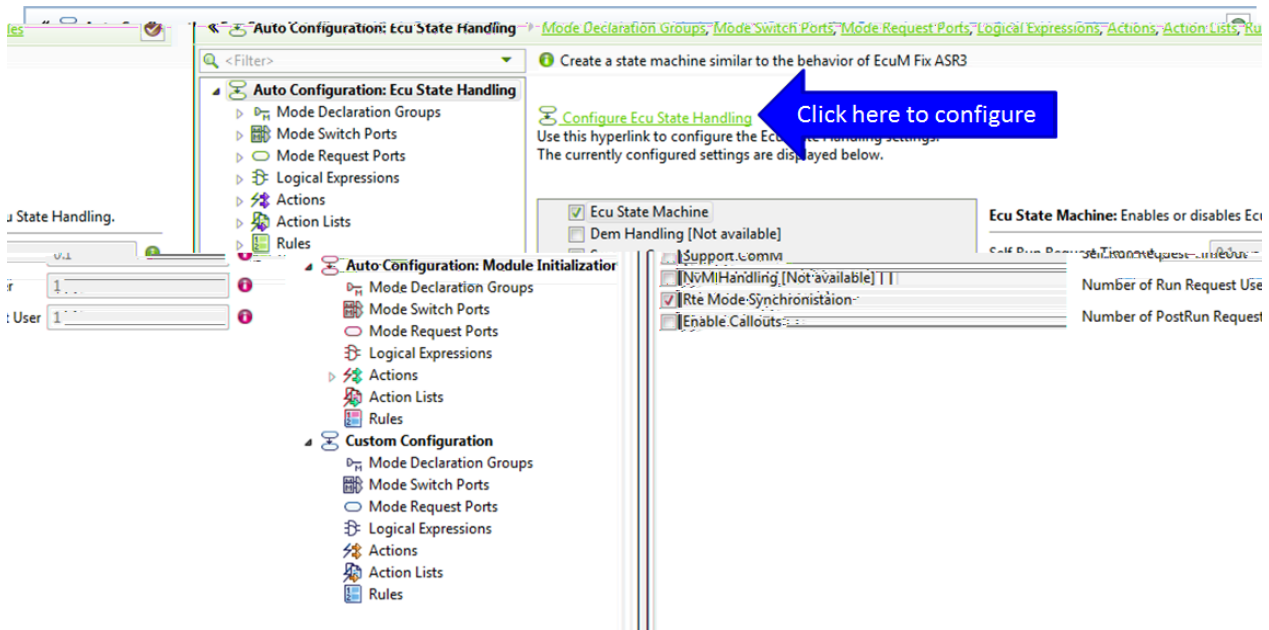


Figure 4-5 Configuration of the features for ECU State Handling

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 and transition call outs are enabled.

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.

The state machine of the ECU state Handling is illustrated in Figure 4-6 State Machine of the ECU State Handling. The rectangular states are notified to the RTE if synchronisation is enabled.

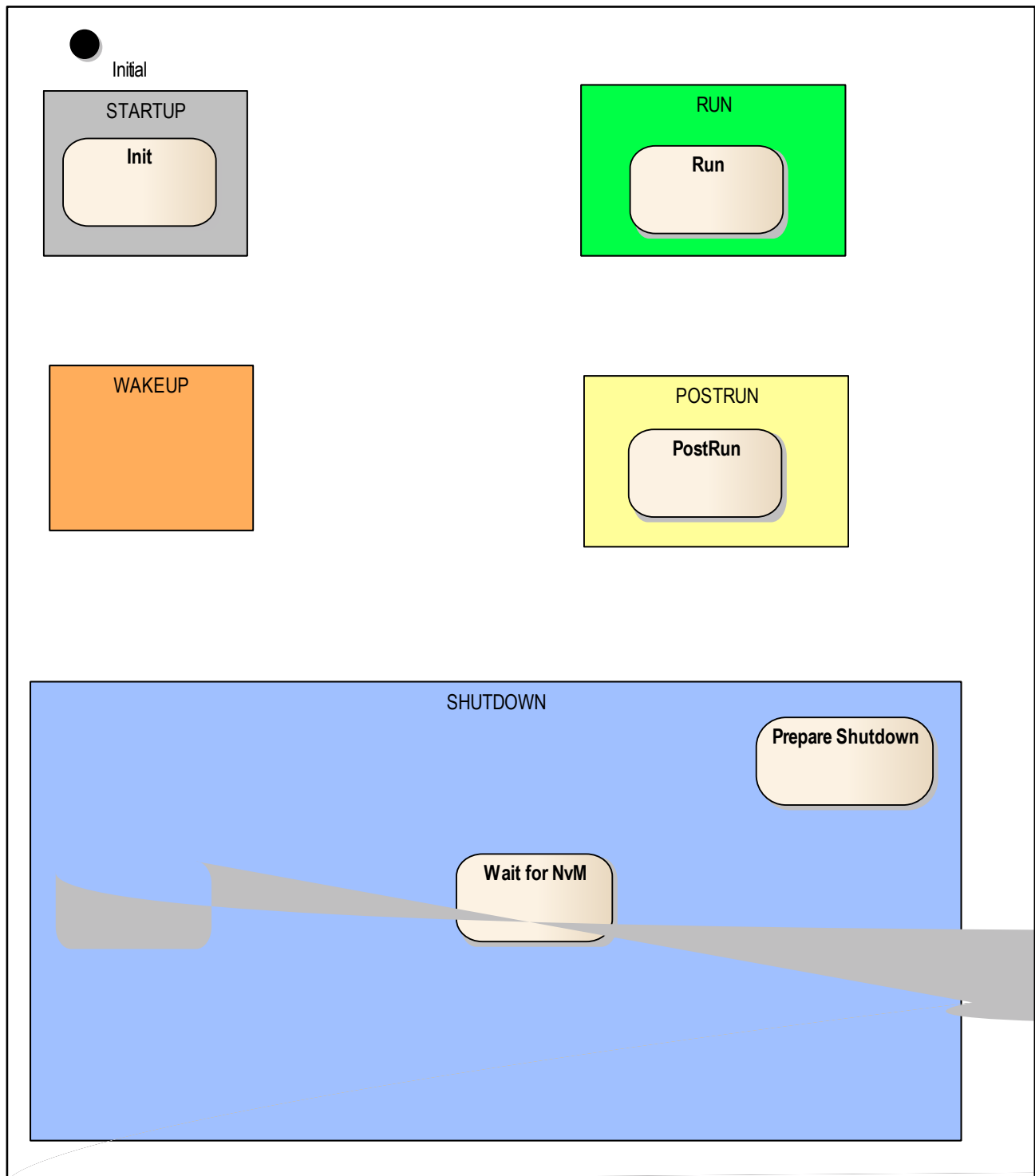


Figure 4-6 State Machine of the ECU State Handling

4.3.3 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 PCN 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-Pathes 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.

4.3.4 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.

4.4 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.



Refer to [6] for details about exclusive areas.

4.5 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.

4.6 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”

5

For an interfaces overview please see Figure 2-2.

5.1 Type Definitions

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

	-		
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 5-1 Type definitions

5.2 Services Provided by BswM

5.2.1 BswM_InitMemory

void (void)	
None	-
void	-
<p>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.</p>	
<ul style="list-style-type: none"> > 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	
<ul style="list-style-type: none"> > This function can be called from task context. 	

Table 5-2 BswM_InitMemory

5.2.2 BswM_Init

void (const BswM_ConfigType *ConfigPtr)	
ConfigPtr	Pointer to post-build configuration data. For the pre-compile case a NULL pointer shall be used.
void	-
<p>Initializes the BSW Mode Manager.</p>	
<ul style="list-style-type: none"> > This function is synchronous. > This function is non-reentrant. 	
Call Context	
<ul style="list-style-type: none"> > This function can be called from task context. 	

Table 5-3 BswM_Init

5.2.3 BswM_Deinit

void (void)	
None	-
void	-
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 <code>BswM_Init()</code> .	
<ul style="list-style-type: none"> > This function is synchronous. > This function is non-reentrant. 	
Call Context	
<ul style="list-style-type: none"> > This function can be called from task context. 	

Table 5-4 BswM_Deinit

5.2.4 BswM_GetVersionInfo

void (Std_VersionInfoType *VersionInfo)	
VersionInfo	Pointer to address where the version information shall be copied to.
void	None
Returns the version information of this module. The versions are BCD-coded.	
<ul style="list-style-type: none"> > The caller must ensure to allocate a variable of the type <code>Std_VersionInfoType</code> before the function call. > This function is synchronous. > This function is reentrant. 	
Call Context	
<ul style="list-style-type: none"> > This function can be called from task and interrupt context. 	

Table 5-5 BswM_GetVersionInfo

5.2.5 BswM_RequestMode

void (BswM_UserType requesting_user, BswM_ModeType requested_mode)	
requesting_user	The user that requests the mode.
requested_mode	The requested mode.
void	-
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.	
<ul style="list-style-type: none"> > 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	
<ul style="list-style-type: none"> > This function can be called from task and interrupt context. 	

Table 5-6 BswM_RequestMode

5.2.6 BswM_ComM_CurrentMode

void (NetworkHandleType Network, ComM_ModeType RequestedMode)	
Network	The ComM communication channel that the indicated state corresponds to.
RequestedMode	The current state of the ComM communication channel
void	-
Function called by ComM to indicate the current communication mode of a ComM channel.	
<ul style="list-style-type: none"> > This function is synchronous. > This function is reentrant for different networks. > Must only be called by the ComM. 	
Call Context	
<ul style="list-style-type: none"> > This function can be called from task and interrupt context. 	

Table 5-7 BswM_ComM_CurrentMode

5.2.7 BswM_ComM_CurrentPNCMode

void (PNCHandleType PNC, ComM_PncModeType RequestedMode)	
PNC	The handle of the PNC for which the current state is reported.
RequestedMode	The current mode of the PNC.
void	-
Function called by ComM to indicate the current mode of the PNC.	
<ul style="list-style-type: none"> > This function is synchronous. > This function is reentrant for different PNCs. > Must only be called by the ComM. 	
Call Context	
<ul style="list-style-type: none"> > This function can be called from task and interrupt context. 	

Table 5-8 BswM_ComM_CurrentPNCMode

5.2.8 BswM_Dcm_ApplicationUpdated

void (void)	
None	-
void	-
Function called by DCM to inform the BswM that the application has being updated.	
<ul style="list-style-type: none"> > This function is synchronous. > This function is reentrant. > Must only be called by the Dcm. 	
Call Context	
<ul style="list-style-type: none"> > This function can be called from task and interrupt context. 	

Table 5-9 BswM_Dcm_ApplicationUpdated

5.2.9 BswM_Dcm_CommunicationMode_CurrentState

<pre>void (NetworkHandleType Network, Dcm_CommunicationModeType RequestedMode)</pre>	
Network	The communication channel that the diagnostic mode corresponds to.
RequestedMode	The requested diagnostic communication mode.
<pre>void</pre>	
-	
Function called by DCM to inform the BswM about the current state of the communication mode.	
<ul style="list-style-type: none"> > This function is synchronous. > This function is reentrant for different networks. > Must only be called by the Dcm. 	
Call Context	
<ul style="list-style-type: none"> > This function can be called from task and interrupt context. 	

Table 5-10 BswM_Dcm_CommunicationMode_CurrentState

5.2.10 BswM_CanSM_CurrentState

<pre>void (NetworkHandleType Network, CanSM_BswMCurrentStateType CurrentState)</pre>	
Network	The CAN channel that the indicated state corresponds to.
CurrentState	The current state of the CAN channel.
<pre>void</pre>	
-	
Function called by CanSM to indicate its current state.	
<ul style="list-style-type: none"> > This function is synchronous. > This function is reentrant for different networks. > Must only be called by the CanSM. 	
Call Context	
<ul style="list-style-type: none"> > This function can be called from task and interrupt context. 	

Table 5-11 BswM_CanSM_CurrentState

5.2.11 BswM_EthSM_CurrentState

void (NetworkHandleType Network, EthSM_NetworkModeStateType CurrentState)	
Network	The Ethernet channel that the indicated state corresponds to.
CurrentState	The current state of the Ethernet channel.
void -	
Function called by EthSM to indicate its current state.	
<ul style="list-style-type: none"> > This function is synchronous. > This function is reentrant for different networks. > Must only be called by the EthSM. 	
Call Context	
<ul style="list-style-type: none"> > This function can be called from task and interrupt context. 	

Table 5-12 BswM_EthSM_CurrentState

5.2.12 BswM_FrSM_CurrentState

void (NetworkHandleType Network, FrSM_BswM_StateType CurrentState)	
Network	The FlexRay cluster that the indicated state corresponds to.
CurrentState	The current state of the FlexRay cluster.
void -	
Function called by FrSM to indicate its current state.	
<ul style="list-style-type: none"> > This function is synchronous. > This function is reentrant for different networks. > This function must only be called by the FrSM. 	
Call Context	
<ul style="list-style-type: none"> > This function can be called from task and interrupt context. 	

Table 5-13 BswM_FrSM_CurrentState

5.2.13 BswM_J1939DcmBroadcastStatus

void (uint16 NetworkMask)	
NetworkMask	Mask containing one bit for each available network. 1:Network enabled 0: Network disabled.
void	-
<p>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.</p> <ul style="list-style-type: none"> > 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 5-14 BswM_J1939DcmBroadcastStatus

5.2.14 BswM_J1939Nm_StateChangeNotification

void (NetworkHandleType Network, uint8 Node, Nm_StateType NmState)	
Network	Identification of the J1939 channel.
Node	Identification of the J1939 node
NmState	Current (new) state of the J1939 node
void	-
<p>Notification of current J1939Nm state after state changes.</p> <ul style="list-style-type: none"> > 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 5-15 BswM_J1939Nm_StateChangeNotification

5.2.15 BswM_LinSM_CurrentState

void (NetworkHandleType Network, LinSM_ModeType CurrentState)	
Network	The LIN channel that the indicated state corresponds to.
CurrentState	The current state of the LIN channel.
void -	
Function called by LinSM to indicate its current state.	
<ul style="list-style-type: none"> > This function is synchronous. > This function is reentrant for different networks. > This function must only be called by the LinSM. 	
Call Context	
<ul style="list-style-type: none"> > This function can be called from task and interrupt context. 	

Table 5-16 BswM_LinSM_CurrentState

5.2.16 BswM_LinSM_CurrentSchedule

void (NetworkHandleType Network, LinIf_SchHandleType CurrentSchedule)	
Network	The LIN channel that the indicated schedule corresponds to.
CurrentSchedule	The currently active schedule table of the LIN channel.
void -	
Function called by LinSM to indicate its current schedule.	
<ul style="list-style-type: none"> > This function is synchronous. > This function is reentrant for different networks. > This function must only be called by the LinSM. 	
Call Context	
<ul style="list-style-type: none"> > This function can be called from task and interrupt context. 	

Table 5-17 BswM_LinSM_CurrentSchedule

5.2.17 BswM_LinSM_ScheduleEndNotification

void (NetworkHandleType Network, LinIf_SchHandleType Schedule)	
Network	The LIN channel that the indicated schedule corresponds to.
Schedule	The schedule table of the LIN channel which has ended.
void	-
Function called by LinSM to notify the end of a schedule.	
<ul style="list-style-type: none"> > This function is synchronous. > This function is reentrant for different networks. > This function must only be called by the LinSM. 	
Call Context	
<ul style="list-style-type: none"> > This function can be called from task and interrupt context. 	

Table 5-18 BswM_LinSM_ScheduleEndNotification

5.2.18 BswM_LinTp_RequestMode

void (NetworkHandleType Network, LinTp_Mode LinTpRequestedMode)	
Network	The LIN channel that the LIN TP mode request corresponds to.
LinTpRequestedMode	The requested LIN TP mode.
void	-
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.	
<ul style="list-style-type: none"> > This function is synchronous. > This function is reentrant for different networks. > This function must only be called by the LinTp. 	
Call Context	
<ul style="list-style-type: none"> > This function can be called from task and interrupt context. 	

Table 5-19 BswM_LinTp_RequestMode

5.2.19 BswM_EcuM_CurrentState

void (EcuM_StateType CurrentState)	
CurrentState	The requested ECU Operation Mode
void	-
Function called by EcuM to indicate the current ECU Operation Mode.	
<ul style="list-style-type: none"> > This function is synchronous. > This function is non-reentrant. > Must only be called by the EcuM. 	
Call Context	
<ul style="list-style-type: none"> > This function can be called from task and interrupt context. 	

Table 5-20 BswM_EcuM_CurrentState

5.2.20 BswM_EcuM_CurrentWakeup

void (EcuM_WakeupSourceType source, EcuM_WakeupStateType state)	
source	Wakeup source(s) that changed state.
state	The new state of the wakeup source(s).
void	-
Function called by EcuM to indicate the current state of a wakeup source.	
<ul style="list-style-type: none"> > This function is synchronous. > This function is reentrant for different sources. > Must only be called by the EcuM. 	
Call Context	
<ul style="list-style-type: none"> > This function can be called from task and interrupt context. 	

Table 5-21 BswM_EcuM_CurrentWakeup

5.2.21 BswM_EcuM_RequestedState

void (EcuM_StateType State, EcuM_RunStatusType CurrentStatus)	
State	The requested state by EcuMFlex.
CurrentStatus	The new result of the Run Request Protocol.
void	-
Function called by EcuM to indicate the request of a run request protocol state.	
<ul style="list-style-type: none"> > This function is synchronous. > This function is reentrant for different states. > Must only be called by the EcuM. 	
Call Context	
<ul style="list-style-type: none"> > This function can be called from task and interrupt context. 	

Table 5-22 BswM_EcuM_RequestedState

5.2.22 BswM_MainFunction

void (void)	
None	-
void	-
Main function of the BswM.	
<ul style="list-style-type: none"> > 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	
<ul style="list-style-type: none"> > This function can be called from task context. 	

Table 5-23 BswM_MainFunction

5.2.23 BswM_NvM_CurrentBlockMode

<pre>void (NvM_BlockIdType Block, NvM_RequestResultType CurrentBlockMode)</pre>	
Block	The Block that the new NvM Mode corresponds to.
CurrentBlockMode	The current block mode of the NvM block.
void	-
Function called by NvM to indicate the current block mode of an NvM block.	
<ul style="list-style-type: none"> > This function is synchronous. > This function is reentrant for different blocks. > This function must only be called by NvM. 	
Call Context	
<ul style="list-style-type: none"> > This function can be called from task and interrupt context. 	

Table 5-24 BswM_NvM_CurrentBlockMode

5.2.24 BswM_NvM_CurrentJobMode

<pre>void (uint8 ServiceId, NvM_RequestResultType CurrentJobMode)</pre>	
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.
void	-
Function called by NvM to inform the BswM about the current state of a multi block job.	
<ul style="list-style-type: none"> > This function is synchronous. > This function is reentrant for different services. > This function must only be called by NvM. 	
Call Context	
<ul style="list-style-type: none"> > This function can be called from task and interrupt context. 	

Table 5-25 BswM_NvM_CurrentJobMode

5.2.25 BswM_PduR_RxIndication

<pre>void (PduIdType RxPduId, const PduInfoType *PduInfoPtr)</pre>	
RxPduId	The PduR ID of received PDU.
PduInfoPtr	Pointer which stores all informations about the PDU. Not used by current implementation.
void	-
Function called by PduR to inform the BswM about a received PDU.	
<ul style="list-style-type: none"> > 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	
<ul style="list-style-type: none"> > This function can be called from task and interrupt context. 	

Table 5-26 BswM_PduR_RxIndication

5.2.26 BswM_PduR_TpRxIndication

<pre>void (PduIdType id, Std_ReturnType result)</pre>	
id	The PduR ID of received PDU.
result	Result of the reception. Not used by current implementation.
void	-
Function called by PduR to inform the BswM about a received TP PDU.	
<ul style="list-style-type: none"> > 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	
<ul style="list-style-type: none"> > This function can be called from task and interrupt context. 	

Table 5-27 BswM_PduR_TpRxIndication

5.2.27 BswM_PduR_TpStartOfReception

void (PduIdType id, PduInfoType *info, PduLengthType TpSduLength, PduLengthType *bufferSizePtr)	
id	The PduR ID of received PDU.
info	Pointer which stores all informations 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.
void	-
Function called by PduR to inform the BswM about the start of TP PDU Reception.	
<ul style="list-style-type: none"> > 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	
<ul style="list-style-type: none"> > This function can be called from task and interrupt context. 	

Table 5-28 BswM_PduR_TpStartOfReception

5.2.28 BswM_PduR_TpTxConfirmation

void (PduIdType id, Std_ReturnType result)	
id	The PduR ID of sent TP PDU.
result	Result of the transmission. Not used by current implementation.
void	-
Function called by PduR to inform the BswM about a sent TP PDU.	
<ul style="list-style-type: none"> > 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	
<ul style="list-style-type: none"> > This function can be called from task and interrupt context. 	

Table 5-29 BswM_PduR_TpTxConfirmation

5.2.29 BswM_PduR_Transmit

void (PduIdType id, const PduInfoType *PduInfoPtr)	
id	The PduR ID of PDU to transmit.
PduInfoPtr	Pointer which stores all informations about the PDU. Not used by current implementation.
void	-
Function called by PduR to inform the BswM about a PDU Transmit Event	
<ul style="list-style-type: none"> > 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	
<ul style="list-style-type: none"> > This function can be called from task and interrupt context. 	

Table 5-30 BswM_PduR_Transmit

5.2.30 BswM_PduR_TxConfirmation

void (PduIdType TxPduId)	
TxPduId	The PduR ID of sent PDU.
void	-
Function called by PduR to inform the BswM about a sent PDU.	
<ul style="list-style-type: none"> > 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	
<ul style="list-style-type: none"> > This function can be called from task and interrupt context. 	

Table 5-31 BswM_PduR_TxConfirmation

5.2.31 BswM_Sd_EventHandlerCurrentState

<pre>void BswM_Sd_EventHandlerCurrentState(uint16 SdEventHandlerHandleId, Sd_EventHandlerCurrentStateType EventHandlerStatus)</pre>	
SdEventHandlerHandleId	HandleId to identify the EventHandler
EventHandlerStatus	Status of the EventHandler
void	-
Function called by Service Discovery to indicate current status of the EventHandler (requested/released).	
<ul style="list-style-type: none"> > This function is synchronous. > This function is reentrant for different handles. > This function must only be called by Sd. 	
Call Context	
<ul style="list-style-type: none"> > This function can be called from task and interrupt context. 	

Table 5-32 BswM_Sd_EventHandlerCurrentState

5.2.32 BswM_Sd_ClientServiceCurrentState

<pre>void (uint16 SdClientServiceHandleId, Sd_ClientServiceCurrentStateType CurrentClientState)</pre>	
SdClientServiceHandleId	HandleId to identify the ClientService.
CurrentClientState	Current state of the ClientService.
void	-
Function called by Service Discovery to indicate current state of the Client Service (available/down).	
<ul style="list-style-type: none"> > This function is synchronous. > This function is reentrant for different handles. > This function must only be called by Sd. 	
Call Context	
<ul style="list-style-type: none"> > This function can be called from task and interrupt context. 	

Table 5-33 BswM_Sd_ClientServiceCurrentState

5.2.33 BswM_Sd_ConsumedEventGroupCurrentState

<pre>void (uint16 SdConsumedEventGroupHandleId, Sd_ConsumedEventGroupCurrentStateType ConsumedEventGroupState)</pre>	
SdConsumedEventGroupHandleId	HandleId to identify the Consumed Eventgroup.
ConsumedEventGroupState	Status of the Consumed Eventgroup.
void	-
Function called by Service Discovery to indicate current status of the Consumed Eventgroup (available/down).	
<ul style="list-style-type: none">> This function is synchronous.> This function is reentrant for different handles.> This function must only be called by Sd.	
Call Context	
<ul style="list-style-type: none">> This function can be called from task and interrupt context.	

Table 5-34 BswM_Sd_ConsumedEventGroupCurrentState

5.2.34 BswM_Nm_StateChangeNotification

void (NetworkHandleType nmNetworkHandle, Nm_StateType nmPreviousState, Nm_StateType nmCurrentState)	
nmNetworkHandle	Identification of the NM-channel
nmPreviousState	Previous state of the NM-channel (Parameter not used)
nmCurrentState	Current (new) state of the NM-channel
void	-
Function called by Nm to inform the BswM about its current state.	
<ul style="list-style-type: none"> > This function is synchronous. > This function is reentrant for different networks. > This function must only be called by Nm. 	
Call Context	
<ul style="list-style-type: none"> > This function can be called from task and interrupt context. 	

Table 5-35 BswM_Nm_StateChangeNotification

5.2.35 BswM_RuleControl

void (BswM_HandleType ruleId, uint8 state)	
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
void	-
Sets a new state to a given rule whereby rule can be enabled or disabled.	
<ul style="list-style-type: none"> > This function is synchronous. > This function is reentrant for different rules. > This function should be called by an action of BswM. 	
Call Context	
<ul style="list-style-type: none"> > This function can be called from task and interrupt context. 	

Table 5-36 BswM_RuleControl

5.2.36 BswM_WdgM_RequestPartitionReset

void (ApplicationType Application)	
Application	The Block that the new NvM Mode corresponds to.
void	-
Function called by WdgM to request a reset of the corresponding partition of given application.	
<ul style="list-style-type: none"> > This function is synchronous. > This function is reentrant. > This function must only be called by WdgM. 	
Call Context	
<ul style="list-style-type: none"> > This function can be called from task and interrupt context. 	

Table 5-37 BswM_WdgM_RequestPartitionReset

5.3 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.

ComM	ComM_CommunicationAllowed
ComM	ComM_LimitChannelToNoComMode
ComM	ComM_RequestComMode
Com	Com_IpduGroupControl
Com	Com_ReceptionDMControl
Com	Com_SetIpduGroup
Com	Com_SwitchIpduTxMode
Com	Com_TriggerIPDUSend
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
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
Sd	Sd_ClientServiceSetState
Sd	Sd_ServerServiceSetState

Table 5-38 Services used by the BswM

5.4 Callback Functions

There are no callback functions in the BswM.

5.5 Configurable Interfaces

5.5.1 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.

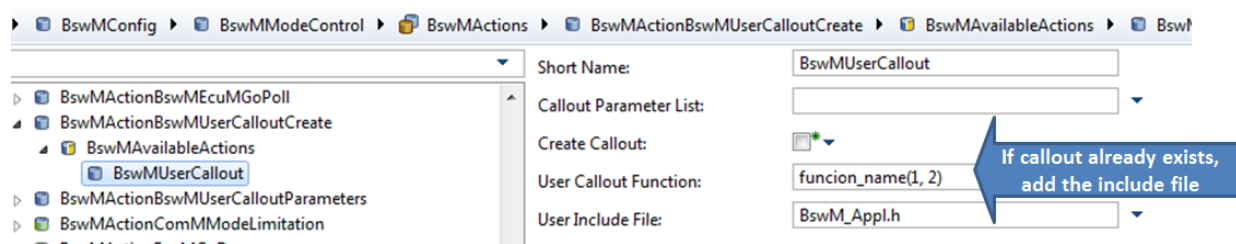


Figure 5-1 Existing callout functions

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”

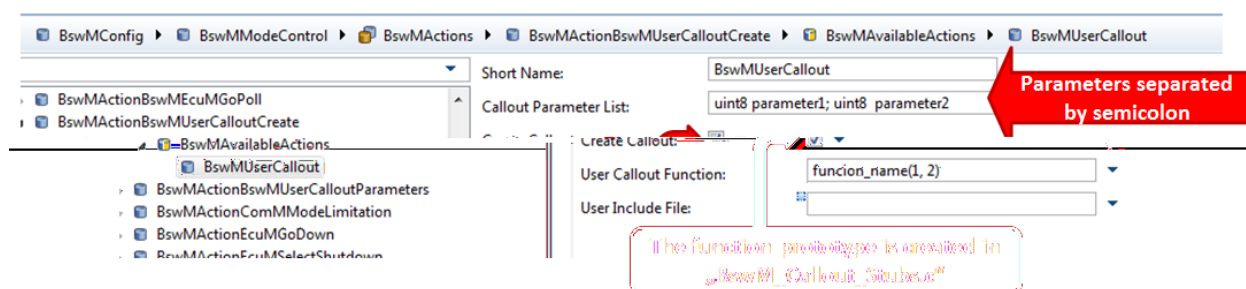


Figure 5-2 Generate prototype of callout functions

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

void	
(<parameters>)	
-	-
-	-

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.	
-	
Call context	
>	Interrupt or task context, depends on the mode/rule configuration in which the callout is used.

Table 5-39 User Callout

5.6 Service Ports

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

- > BswMSwcModeRequest
- > BswMSwcModeNotification
- > BswMSwitchPort
- > BswMModeDeclaration

These containers are described in the following chapters.

5.6.1 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_{Name of BswMSwcModeRequest}

5.6.2 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 }

5.6.3 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 }

5.6.4 BswMRteModeRequestPort (P-Ports)

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 }

5.6.5 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_{Name of BswMModeDeclaration}

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

- > BswM_MDGP_{Name of BswmModeDeclaration}

The Implementation-Type is named:

- > BswM_{Name of BswMModeDeclaration}

6

6.1 Deviations

6.1.1 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 Com module, it is necessary to add a header file with the name "Com_Types.h", which defines the type "Com_IpduGroupIdType".

6.1.2 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}

6.2 Additions/ Extensions

6.2.1 Optional Interfaces

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

- > ComM_LimitChannelToNoComMode
- > ComM_RequestComMode
- > Com_ClearIpduGroupVector
- > Com_IpduGroupControl
- > Com_ReceptionDMControl
- > Com_SetIpduGroup
- > 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

6.2.2 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)

6.2.3 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.

6.2.4 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 5.6.5.

6.2.5 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.

6.2.6 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 6.2.7).

6.2.7 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.

6.2.8 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.

6.2.9 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 3.4.2.

6.2.10 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 5.2.35.

6.2.11 Support of Com ASR3 IPduGroup APIs

If Microsar 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].

6.3 Limitations

6.3.1 Configurable interfaces that are not supported

6.3.1.1 EcuM Indication for EcuM Flex

The ModeRequestPort of type EcuMIndication is not supported for MICROSAR 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.

6.3.2 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
- > ControllIdle

6.3.3 Configuration Variants

Configuration variant Link-Time is not supported.

6.3.4 BSW Modules

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

7

7.1 Glossary



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 7-1 Abbreviations

8

Visit our website for more information on

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