MICROSAR Classic EcuM Flex

Technical Reference

SysService_Asr4EcuM

Version 11.00.00

Authors	visvjn, vispkn, visols, mbach, vismfi, vischh
Status	Released

Document Information

History

Author	Date	Version	Remarks
visvjn	2012-06-06	1.00.00	> Initial Setup
visvjn	2013-01-30	1.00.01	> ESCAN00064669 Updated compiler abstraction and memory mapping
visvjn	2013-05-03	1.01.00	> Added support of post-build-loadable > Added support of asynchronous transceiver handling in 3.9.2 > Added API EcuM_ClearValidatedWakeupEvent() > in 5.3.10 > Extended description of EcuM_StartupTwo() in 5.3.3
visvjn	2013-10-31	2.00.00	> ESCAN00069010 Added support for Alarm Clock in 3.14 > ESCAN00071546 Added Multi Core support in 3.15 > New API EcuM_GoToSelectedShutdownTarget > ESCAN00071553 Changed handling of wakeup source states in 5.1 > Changes in chapter 4.2 Critical Sections > ESCAN00071552 Removed BswM_EcuM_CurentState notification
visvjn	2014-06-03	3.00.00	> Added Support for EcuM fixed > ESCAN00073631 Fixed missing description of EcuM_BswErrorHook()
visvjn	2014-11-04	4.00.00	> Added Support for Post-Build Selectable > Added chapter 3.15.1.2.1 Driver initialization on the Slave Core. > Added chapter 3.15.5 Reconfiguration of the BSW Core ID > Added MICROSAR Classic specific CanSM handling in 3.18.2.3.3 > ESCAN00079382 Fixed missing description of the StateRequest Port in 5.9.1.1 > ESCAN00077124 Fixed description of Critical Sections in 4.2

			> ESCAN00079407, ESCAN00068331 Fixed description in Type Definitions of EcuM_WakeupStateType in 5.1
visvjn	2014-11-25	4.00.01	> Adapted description of EcuM_DeterminePbConfiguration
visvjn	2015-01-26	4.01.00	> Updated the Include structure and added two files in 4.1.2 > Updated access on PB and Variant data in DriverInitLists in Ch. 5.8.2
visvjn	2015-07-14	5.00.00	> Added new EcuM error ID for invalid CoreID in Ch. 3.11.3 > Added support for Mode Handling, see Ch. 3.16, 5.4.13 and 5.6 > Removed subchapters “Parameter Checking” from Ch. 3.11 > Added missing API ID in Table 3-8 Service IDs
visvjn	2016-11-15	6.00.00	> Added support for PNC notifications to > ComM about Wakeup Events
visvjn	2017-11-30	6.00.01	> ESCAN00096797 Added hint to EcuM_Shutdown API description
visvjn	2018-07-06	7.00.00	> STORYC-2829 Added support for initialization and shutdown of multiple partitions
visvjn	2018-08-14	7.01.00	> Added missing error ID ECUM_E_HOOK_INVALID_APPLICATIONID in 3.11.3
vispkn	2018-12-17	7.01.01	ESCAN00098617 Added ErrorId ECUM_E_INVALID_GEN_DATA in ch. 3.11.1 Added ch. 4.3 Memory Mapping with multiple partitions > STORYC-7328 Removed restriction in ch. 6.3.3 > STORYC-7335 Added ch. 5.2 and 5.2.1 Modified ch. 5.3.17
visols	2020-05-04	7.01.02	> ESCAN00106260 Added critical section ECUM_EXCLUSIVE_AREA_3 in ch. 4.2 and ch. 5.6
visols	2020-09-16	7.01.03	> Added further explanation on safety levels in ch. 4.3
visols	2020-12-07	8.00.00	> NMM-1060 Added support for SchM initialization according to AUTOSAR 4.3.1 in ch. 3.6.2, 3.12.1, 3.15.1

visols	2021-01-21	8.00.01	> ESCAN00108355 Changed order of execution of functions in EcuM_StartupTwo in ch. 3.6.2.3 and 3.12.1
visols	2022-01-17	9.00.00	> NMM-1210 Adding dependency to MemMap module in ch. 4.1.2 and 4.3
mbach	2022-07-26	10.00.00	> Product name updated to MICROSAR Classic > NMM-1486 Added support for multiple channel notifications to ComM about Wakeup Events
vischh	2023-04-19	11.00.00	> 3.18.2.3.3 EcuM_StartWakeupSources and EcuM_StopWakeupSources in the case of a MICROSAR Classic CanSM

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_ECUStateManager.pdf	V3.0.0
[2]	AUTOSAR	AUTOSAR_SWS_DevelopmentErrorTracer.pdf	V3.2.0
[3]	AUTOSAR	AUTOSAR_SWS_DiagnosticEventManager.pdf.pdf	V4.2.0
[4]	AUTOSAR	AUTOSAR_TR_BSWModuleList.pdf	V1.6.0
[5]	AUTOSAR	AUTOSAR_EXP_ModemanagementGuide.pdf	V1.0.0
[6]	VECTOR	TechnicalReference_PostBuildLoadable.pdf	see delivery
[7]	AUTOSAR	AUTOSAR_SWS_ECUStateManagerFixed.pdf	V1.4.0
[8]	VECTOR	TechnicalReference_IdentityManager.pdf	see delivery

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 Component History

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

Component Version	New Features
1.00.00	Adaption to AUTOSAR Release 4
1.01.00	Added support of configuration variant Post-Build Loadable Added support of asynchronous transceiver handling
2.00.00	Added support for handling of MultiCore ECUs Added support of Alarm Clock to provide the absolute time and handling of time triggered wake-ups.
3.00.00	Added support for EcuM with fixed state machine
4.00.00	Added support for Post-Build Selectable
5.00.00	Added support for Mode Handling in EcuM Flex
7.00.00	Added support for initialization and shutdown of multiple partitions
8.00.00	Added support for SchM initialization according to AUTOSAR 4.3.1
9.00.00	Added support for automated Memory Section creation

Table 1-1 Component history

2 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module EcuM as specified in [1] and [7].

Supported AUTOSAR Release*:	4.0.3	
Supported Configuration Variants:	Pre-Compile, Post-Build Loadable	
Vendor ID:	ECUM_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	ECUM_MODULE_ID	10 decimal (according to ref. [4])

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

This document describes the functionality and API of the ECU State Manager (EcuM) as a hardware independent module.

The main tasks of the EcuM are:

- Initialization of BSW (Basis Software) modules that are needed to start the operating system
- Preparation of the microcontroller for a sleep phase and the following wake up
- Performing an ordered shut down or reset of the ECU
- Validation of occurred wake ups via the wake-up validation protocol

2.1 Architecture Overview

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

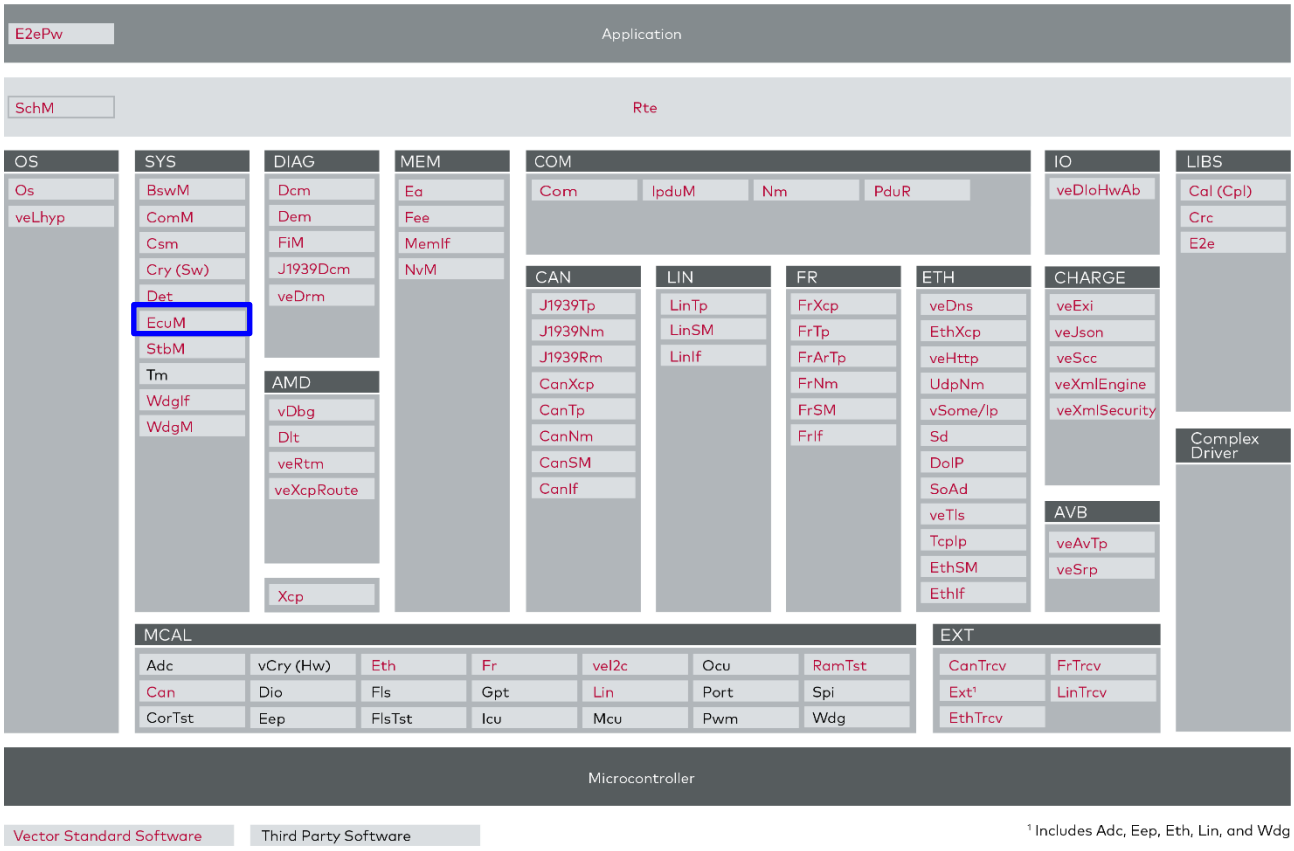

Figure 2-1 AUTOSAR 4.1 Architecture Overview

The next figure shows the interfaces to adjacent modules of the EcuM. These interfaces are described in chapter 5.2 and 5.6 Services Used by EcuM.

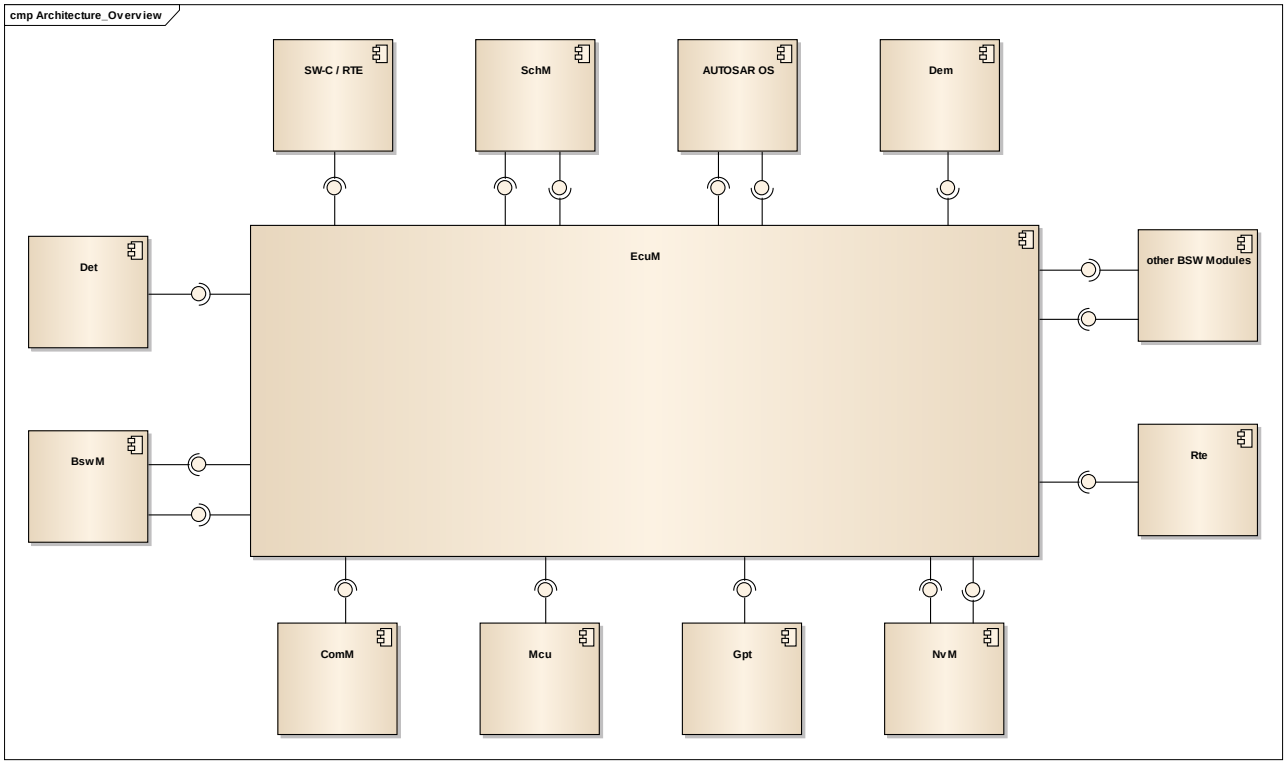

Figure 2-2 Interfaces to adjacent modules of the EcuM

3 Functional Description

3.1 Features

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

The AUTOSAR standard functionality is specified in [1] and [7], the corresponding features are listed in the tables:

> Table 3-1 Supported AUTOSAR EcuM common features

> Table 3-2 Supported AUTOSAR EcuM flex features

> Table 3-3 Supported AUTOSAR EcuM fixed features

For further information of not supported features see also chapter 6.

Vector Informatik provides further EcuM functionality beyond the AUTOSAR standard. The corresponding features are listed in the table:

> Table 3-4 Features provided beyond the AUTOSAR standard

The following features specified in [1] and [7] are supported:

Supported AUTOSAR Standard Conform Features
Configuration of different wake-up sources.
Configuration of EcuM users.
Configurable startup sequence of the BSW stack that is needed before starting the OS.
Possibility to add additional initialization code into the initialization lists.
Notification of the BswM if a wake-up event occurs on a wake-up source.
Notification of the ComM if a wake-up event occurs on communication channels.
Assignment of multiple communication channels to wake-up sources.
Configuration of different sleep modes.
Selection of different shutdown targets.
Selection of different shutdown causes.
Generation of the SW-C description file needed for the generation of the RTE.
Service Port: EcuM_ShutdownTarget
Service Port: EcuM_BootTarget
Consistency hash checking according to AUTOSAR specification
Post-build configuration of the EcuM
Support of MultiCore ECUs
Run / Post_Run Request Protocol
Mode Port: EcuM_CurrentMode

Table 3-1 Supported AUTOSAR EcuM common features

The following EcuM flex features specified in [1] are supported:

Supported AUTOSAR EcuM flex Features
Configuration of different reset modes.
Service Port: EcuM_AlarmClock
Defensive Behavior to check the valid call of EcuM_GoDown
Alarm clock to provide an absolute time and handling of time triggered wake-ups.
Added support for initialization and shutdown of multiple partitions

Table 3-2 Supported AUTOSAR EcuM flex features

The following EcuM fixed features specified in [7] are supported:

Supported AUTOSAR EcuM fixed Features
Full initialization of the Stack via configurable DriverInitLists
Fixed state machine to control the ECU states
Allow communication via ComM_CommunicationAllowed when entering the ECUM_STATE_RUN
Handle NvM_WriteAll() and NvM_CancelWriteAll()
Start and stop of the RTE

Table 3-3 Supported AUTOSAR EcuM fixed features

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
Adding of additional initialization code by the configuration tool
Wake-up Events are buffered until the BswM and the ComM are initialized
Support of asynchronous transceiver handling (Introduced API EcuM_StartCheckWakeup + EcuM_EndCheckWakeup)
Providing an additional API EcuM_ClearValidatedWakeupEvent() to clear only validated, but not pending wake-up events
Providing an additional API EcuM_GoToSelectedShutdownTarget() to decide EcuM internal if EcuM_GoPoll(), EcuM_GoHalt() or EcuM_GoDown() has to be called, depending on the selected shutdown target [EcuM flex only]
Configuration of the Core ID on which the BSW is initialized
Notification of the ComM if a wake-up event occurs on a PNC

Table 3-4 Features provided beyond the AUTOSAR standard

3.2 States of EcuM flex

These states indicate the current internal EcuM Operation State.

Module State	Activities	Point in Time
ECUM_STATE_STARTUP	Initializes the drivers out of the EcuM_DriverInitZero list.	Entered during EcuM_Init().
ECUM_STATE_STARTUP_ONE	Initializes the drivers out of the EcuM_DriverInitOne list. Reset reason translation, setting of the default shutdown target and at the end start the operating system.	Entered during EcuM_Init().
ECUM_STATE_STARTUP_TWO	Initializes the BswM and the SchM. Former buffered Wake-up Events are notified to the BswM.	Entered during EcuM_StartupTwo().
ECUM_STATE_APP_RUN	After initializing the necessary BSW, the EcuM is in the Run state.	Entered during EcuM_StartupTwo(), EcuM_GoSleep(), EcuM_GoPoll() or during the MainFunction.
ECUM_STATE_GO_SLEEP	Prepares the ECU for the upcoming sleep phase.	Entered during EcuM_GoSleep().
ECUM_STATE_SLEEP	Handles the sleep.	Entered during EcuM_GoHalt() or EcuM_GoPoll().
ECUM_STATE_GO_OFF_ONE	Prepares the ECU for the upcoming Off phase. The SchM and the BswM are deinitialized in this phase and the EcuM_OnGoOffOne() Callout is invoked. Finally the operating system will be shut down.	Entered during EcuM_GoDown().
ECUM_STATE_GO_OFF_TWO	The configured shutdown target is called by the EcuM.	Entered during EcuM_Shutdown().
ECUM_STATE_WAKEUP_ONE	The hardware is reinitialized after a former sleep mode.	
ECUM_STATE_WAKEUP_VALIDATION	Waits for the validation of an occurred wake up.	After a wake-up event has occurred that needs validation.

Table 3-5 States of the EcuM

3.3 States of EcuM fixed

These states indicate the current internal EcuM Operation State which can be retrieved via the API 5.5.1 EcuM_GetState.

All the states, except ECUM_STATE_STARTUP and ECUM_STATE_ERROR are notified to the BswM. In some state transitions an RTE mode switch will be performed.

Module State	Activities	RTE Mode
ECUM_STATE_STARTUP	Initializes the drivers via the DriverInitLists. Reset reason translation, setting of the default shutdown target and at the end start the operating system. Initializes the BswM, the SchM and the RTE. Former buffered Wake-up Events are notified to the BswM.	ECUM_RTE_STARTUP (initial mode)
ECUM_STATE_APP_RUN	EcuM stays in this state while there are active Run Requests, the EcuM Self Run Request timeout has not expired or ComM Channels are in communication.	ECUM_RTE_RUN
ECUM_STATE_APP_POST_RUN	Post Run Requests keep the EcuM in this state.	ECUM_RTE_POST_RUN
ECUM_STATE_PREP_SHUTDOWN	Shutdown the DEM and transit directly to ECUM_STATE_GO_SLEEP or ECUM_STATE_GO_OFF_ONE	ECUM_RTE_POST_RUN
ECUM_STATE_GO_SLEEP	EcuM triggers the NvM_WriteAll() job. EcuM remains in this state until the NvM calls EcuM_CB_NfyNvMJobEnd() or the occurrence of a wake up event cancels the sleep process. In case of a wake up event, NvM_CancelWriteAll() is called.	ECUM_RTE_SLEEP
ECUM_STATE_SLEEP	Handles the sleep and a wake up from sleep.	ECUM_RTE_SLEEP
ECUM_STATE_GO_OFF_ONE	Stops the RTE and triggers NvM_WriteAll(). EcuM remains in this state until the NvM call EcuM_CB_NfyNvMJobEnd().	ECUM_RTE_SHUTDOWN
ECUM_STATE_WAKEUP_VALIDATION	Waits for the validation of an occurred wake up.	ECUM_RTE_SLEEP
ECUM_STATE_WAKEUP_REACTION	Wait for completion of a potential NvM_CancelWriteAll().	ECUM_RTE_SLEEP

Module State	Activities	RTE Mode
ECUM_STATE_WAKEUP_WAKESLEEP	-	ECUM_RTE_WAKE_SLEEP
ECUM_STATE_ERROR	The EcuM_ErrorHook is called in this state. This state is only reached if the ShutdownOS() or EcuM_AL_SwitchOff returns to the EcuM.	-

Table 3-6 States of the EcuM

3.4 The State Diagram of the EcuM flex

The following figure shows the EcuM state diagram with all state transitions, the corresponding conditions and actions:

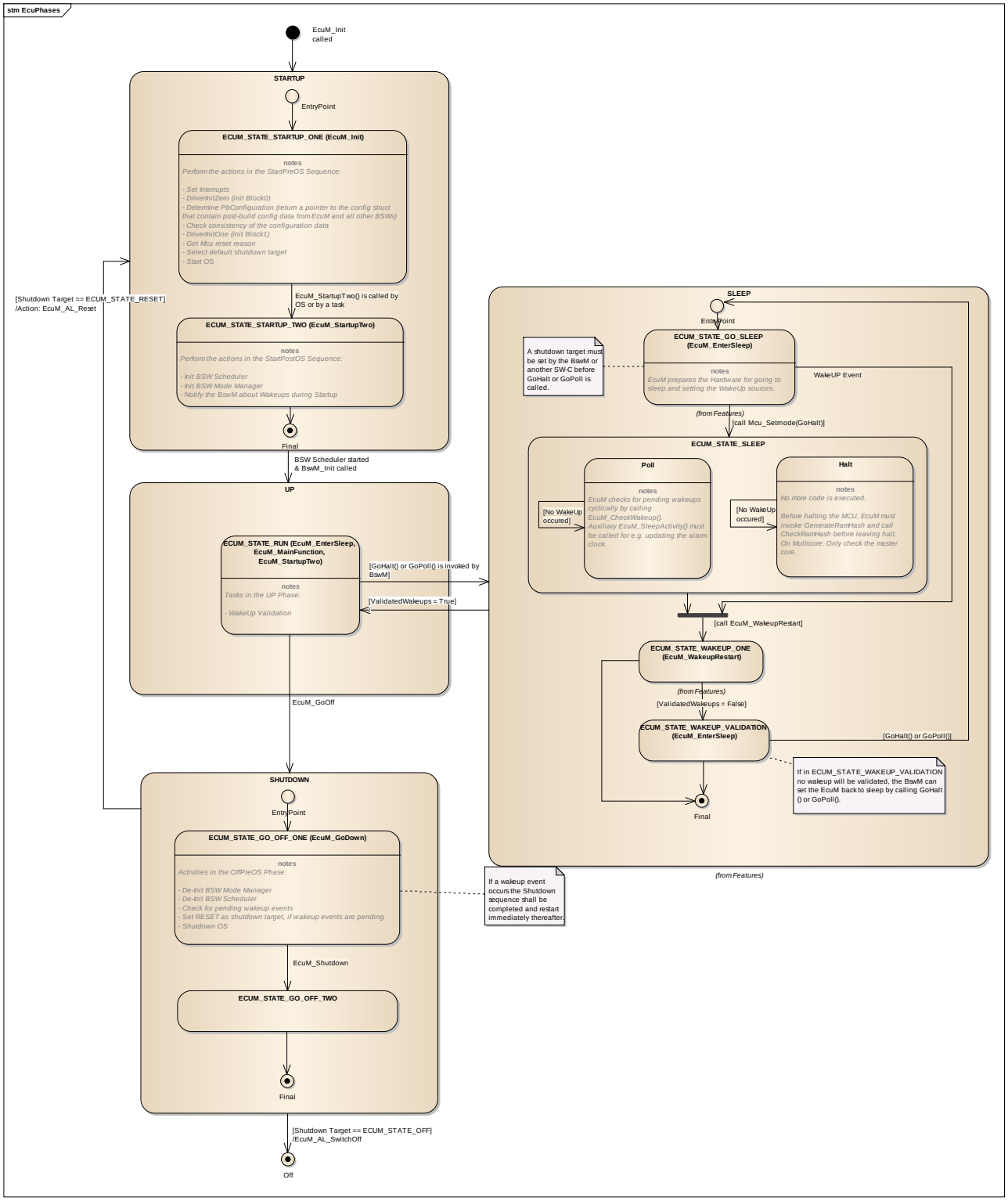

Figure 3-1 The state diagram of the EcuM flex

3.5 The State Diagram of the EcuM with fixed state machine

The following figure shows the EcuM state diagram with all state transitions and the corresponding RTE modes:

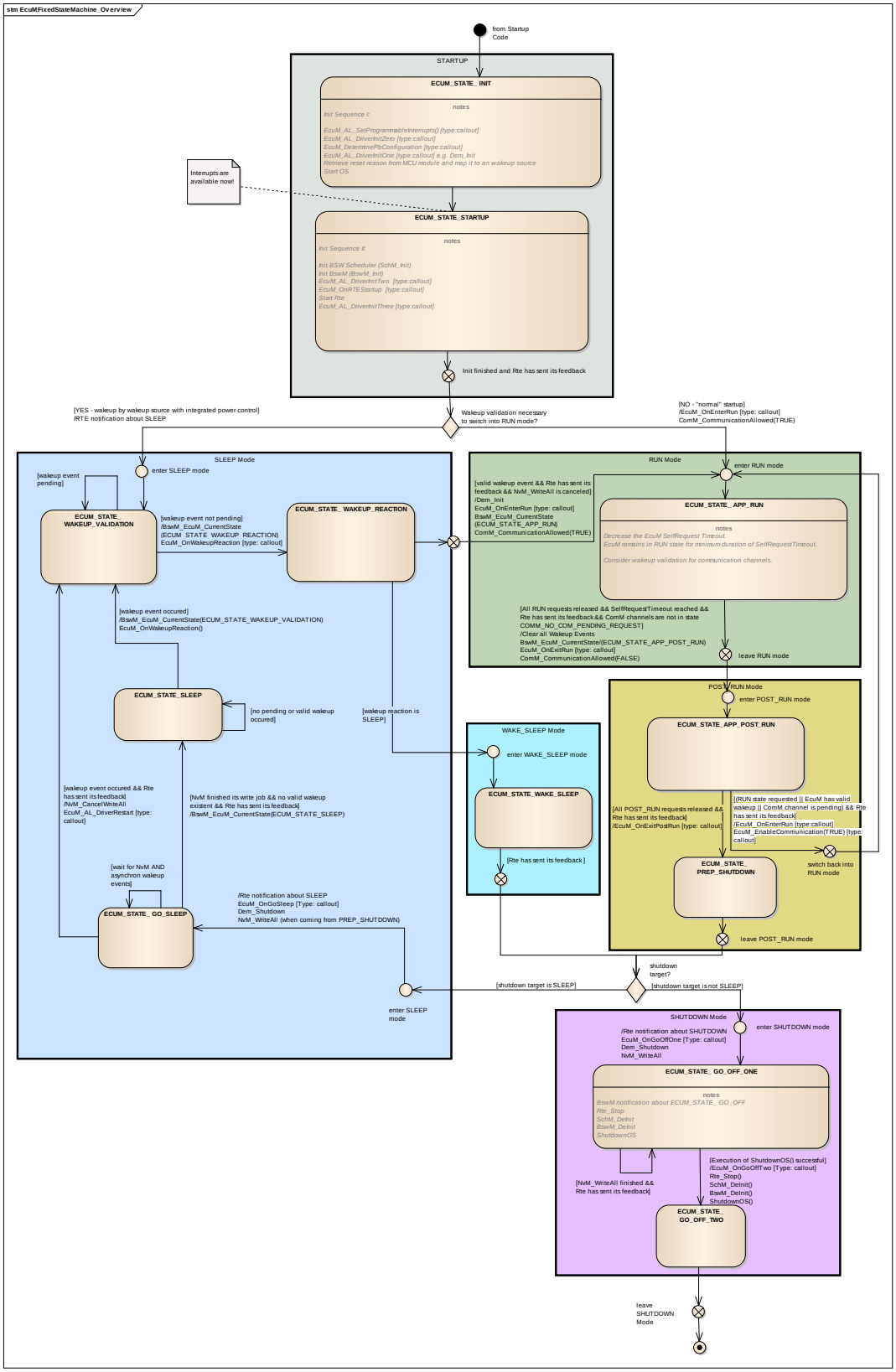

Figure 3-2 State Diagram of the EcuM with fixed state machine

3.6 Initialization

The initialization of the EcuM is split into two parts: one part is the initialization before the OS is up and running and the second part must be executed when the OS is started.

3.6.1 EcuM_Init

The first part will be performed by the function EcuM_Init() (refer to chapter 5.3.2). This function executes the DriverInitLists “EcuMDriverInitListZero” and “EcuMDriverInitListOne” where the basic driver initialization should be performed. EcuM_Init() starts the AUTOSAR OS by calling the function StartOS() (refer to chapter 5.3.3).

3.6.2 EcuM_StartupTwo

The second part of the initialization sequence will be executed by the EcuM API EcuM_StartupTwo(). The integrator must ensure that this function is called once right after the start of the OS.

3.6.2.1 EcuM_StartupTwo in case of EcuM flex

When EcuM_StartupTwo() is left, the EcuM flex is in Run state and passes the control of the ECU to the BswM.

Caution
At the end of the EcuM_StartupTwo the EcuM is fully initialized. That does not mean that the whole stack is initialized, it means only that the EcuM has passed the control over to the BswM. Further initialization is done by the BswM.

3.6.2.2 EcuM_StartupTwo in case of EcuM fixed

In case of EcuM fixed, in EcuM_StartupTwo() the DriverInitLists “EcuMDriverInitListTwo” and “EcuMDriverInitListThree” can be used to initialize the whole stack.

Caution
At the end of the EcuM_StartupTwo the EcuM fixed transits to ECUM_STATE_APP_RUN in case of a validated wake up, e.g. set by the MCU Reset Reason (refer to chapter 3.9.5).

If this wake up was cleared (in EcuMDriverInitListTwo), the EcuM transits to ECUM_STATE_WAKEUP_VALIDATION and performs a wake up validation if any wake up source is pending.

3.6.2.3 EcuM_StartupTwo in Case of AUTOSAR 4.3.1

Im case of AUTOSAR >= 4.3.1, in EcuM_StartupTwo the SchM and BswM are initialized with the following sequence of function calls:

SchM_Start(), SchM_Init(), BswM_Init(),SchM_StartTiming()

Caution
AUTOSAR 4.3.1 defines the sequence of function calls as SchM_Start(), BswM_Init(), SchM_Init(), SchM_StartTiming(). Note that our implementation exchanges the order of BswM_Init and SchM_Init to restore the old default.

3.6.2.4 EcuM_StartupTwo in Case of AUTOSAR older than 4.3.1

Im case of AUTOSAR < 4.3.1, in EcuM_StartupTwo the SchM and BswM are initialized with the following sequence of function calls:

SchM_Init(), BswM_Init()

3.6.3 Initialization Order

Depending on which modules are needed for starting the operating system the initialization lists can look different.
In the following an example initialization order is given. Init Block 0 corresponds to the EcuM_AL_DriverInitZero() (refer to chapter 5.8.2.6) and Init Block 1 corresponds to EcuM_AL_DriverInitOne() (refer to chapter 5.8.2.7).

Initialization Group
Init Block0
Det_Init()
Dem_PreInit(ConfigPointer)
Init Block1
Mcu_Init(ConfigPointer)
Gpt_Init(ConfigPointer)
Wdg_Init()
WdgM_Init()
Adc_Init(ConfigPointer)
Icu_Init(ConfigPointer)
Pwm_Init(ConfigPointer)

Table 3-7 Initialization Order

3.6.4 Additional Code in the Initialization Callouts

If the user needs more than the initialization routines offered by the AUTOSAR modules, the configuration tool offers the facility to add own Code to the DriverInitLists. To use this feature the user has to choose “Code” instead of a MSN, then the code can be added to a special field.

The user code is added to the Init Block 0 or Init Block 1 as configured by the user.

Example
In this example the routine Mcu_InitClock() is added to the DriverInitListOne:

> Open the Initialization dialogue
> Go to the configuration of DriverInitListOne in the Pre-OS Init Sequence
> Add an InitItem to the list and choose a name like “McuInitClock”
> Choose “Code” in the field Type
> In the field “Code” you can insert: “Mcu_InitClock();”
> Reorder the position of the InitItem

3.6.5 Inclusion of Additional Header Files

If the user needs additional headers for using in the EcuM_Callout_Stubs.c file, the EcuM offers the possibility of adding them by the configuration tool.

Note
All header files of the modules that are initialized in the DriverInitLists must be included into the additional header files because they are not included automatically.

3.6.6 Configuration Set Selection

The AUTOSAR compatible mechanism to select the configuration set which should be used for module initialization considers the following aspects:

- Most of the AUTOSAR modules provide a configuration reference to the provided configuration sets
- Some modules are initialized without a configuration pointer (Init-function signature <MSN>_Init(void))
- Some modules have an Init-function signature with configuration pointer but make no use of it, therefore, they need to be initialized with a NULL_PTR.

The user must decide which routines use a configuration pointer. For these routines the configuration reference must be configured.

- Module uses a configuration pointer for its initialization:
  - Select in the DriverInitList a MSN via the field Type (e.g. Dem)
  - Select the corresponding Service (e.g. Dem_PreInit)
  - Configure the corresponding Configuration Pointer for that MSN (e.g. DemConfigSet)
  - Result: The EcuM generates “Dem_PreInit(&DemConfigSet)”
- Module has a void Init-function signature
  - Select in the DriverInitList a MSN via the field Type (e.g. Det)
  - Select the corresponding Service (e.g. Det_Init)
  - Do not configure the corresponding Configuration Pointer for this MSN
  - Result: The EcuM generates: “Det_Init()”

Caution
If a module initialization routine requires a configuration set as parameter, the corresponding reference to the module must be configured.

This is also necessary if the initialization routine does not use the parameter. The reference must be configured, otherwise the parameter list will be generated empty.

3.7 Initialization of a MultiCore ECU

The initialization of a MultiCore Ecu is described in chapter 3.15.1 Initialization of a MultiCore ECU.

3.8 Shutdown Targets

The EcuM provides the possibility to select a shutdown target that is used for the next shutdown, initiated by calling EcuM_GoDown() (refer to chapter 5.4.5), EcuM_GoPoll()(refer to chapter 5.4.4) or EcuM_GoHalt()(refer to chapter 5.4.3).

The following three different targets can be selected by a SWC or a BSW module:

- ECUM_STATE_SLEEP
- ECUM_STATE_RESET
- ECUM_STATE_OFF

Note
The two targets ECUM_STATE_SLEEP and ECUM_STATE_RESET have an additional mode parameter, which is used to identify the configuration for the Sleep mode or to identify the reason for an upcoming reset of the ECU.

3.8.1 Using the API EcuM_SelectShutdownTarget()

The API EcuM_SelectShutdownTarget()(refer to chapter 5.3.5) can only be used when the EcuM is in the state ECUM_STATE_RUN. In the startup phase or during the sleep phase it is not allowed to change the shutdown target.

3.8.2 Default Shutdown Target

A Default shutdown target must be set during the configuration. This is the first target that is selected as shutdown target after a startup. During runtime the shutdown target can be changed by another BSW or SWC via the API EcuM_SelectShutdownTarget().

3.8.3 Reset Modes

The reset modes can be used to identify the reason for an upcoming ECU reset. A set of reset modes is defined by the AUTOSAR standard. Additional modes can be added by the configuration.
The reset mode is passed over to the Callout EcuM_AL_Reset(EcuM_ResetType) and the user can implement different ways to reset the ECU, depending on the reason for this reset.

The Vector extension ECUM_RESET_WAKEUP is used as the reset mode in the case of a late wake-up event in the shutdown phase. If a wake-up occurs during the shutdown procedure, the shutdown target is changed by the EcuM to ECUM_STATE_RESET and the described mode is used.

Note
The following reset mode is defined by Vector as an extension to the standard AUTOSAR modes:

- ECUM_RESET_WAKEUP

Caution
Reset Modes are only available if EcuM flex is used.

3.8.4 Sleep Modes

A sleep mode holds the information about the configured sleep modes and the corresponding relevant settings. The following items can be set for a sleep mode:

> Reference to a configured MCU mode that is executed for that sleep mode.

> Active Wake-up Sources during this sleep mode.

3.9 Wake-up Sources

The EcuM flex offers the possibility to configure wake-up sources for all modules that have the functionality to wake up the ECU. The EcuM handles the Wake-up Validation Protocol for these sources as described in 3.10.1 Wake-up Validation Protocol.

The Wake-up Sources have several configurable attributes as described in the following section.

3.9.1 Validation Timeout

For every source, except for the standard sources 0 – 4, a validation timeout timer can be configured. This timer specifies the time (in seconds) until the wake-up source must be validated by calling EcuM_ValidateWakeupEvent().

If the wake-up event is not validated during that time the EcuM sets this event to “expired” and reports it to the BswM.

Note
The following standard wake-up sources are pre-configured and do not need the wake-up validation protocol:

- ECUM_WKSOURCE_POWER
- ECUM_WKSOURCE_RESET
- ECUM_WKSOURCE_INTERNAL_RESET
- ECUM_WKSOURCE_INTERNAL_WDG
- ECUM_WKSOURCE_EXTERNAL_WDG

3.9.2 Check-Wakeup Validation Timeout

For every source, except for the standard sources 0 – 4, a check wake-up validation timeout timer can be configured. This timer specifies the time (in seconds) until the wake-up source must be set by calling EcuM_SetWakeupEvent().

This timer can be used for e.g. asynchronous transceiver drivers, which cannot check the wake-up source in the context of EcuM_CheckWakeup.

3.9.3 ComM Channel Reference

If the configured Wake-up Source comprises one or more ComM Channels, the references to the corresponding channels can be configured by the parameter EcuMComMChannelRef.

If these references are configured and a validated wake-up event occurred, the EcuM calls the function ComM_EcuM_WakeupIndication() and reports them to the ComM.

Note
By default only one Wake-up Source which represent a ComM Channel can lead to a wake up in the state ECUM_STATE_RUN. Other Wake-up Sources are ignored during this state.
If all Wake-up Sources should be considered during the state ECUM_STATE_RUN, the according configuration option has to be set.

3.9.4 Polling of Wake-up Sources

If a Wake-up Source needs to be polled to detect wake-up events this parameter must be set. In that case, the sleep can be entered by calling EcuM_GoPoll() and the EcuM polls all Wake-up Sources that are active during that Sleep mode and the polling parameter is set.

3.9.5 MCU Reset Reason

The EcuM calls the routine Mcu_GetResetReason() to acquire the reason for the recent reset. The EcuM iterates over all configured Wake-up Sources and checks if the configured Reset Reason of one Wake-up Source matches to the return value of the MCU.

If a reset reason is found, the EcuM maps this MCU reset reason to an EcuM Wake-up Source and reports the event to the BswM. The regular wake-up validation is done by the EcuM in case it is required by the source.

Note
If the reset reason translation is not successful and no reset reason can be determined, the EcuM reports to the BswM the default reset reason ECUM_WKSOURCE_RESET.

3.10 Main Functions

3.10.1 Wake-up Validation Protocol

The wake-up validation protocol provides a standardized way to recognize valid controller wake ups after a sleep phase.

For all user configured wake-up sources the parameter “Validation Timeout” is configurable. If the parameter is set to a value which is not 0, the wake-up validation protocol is active for that source.

Example

In the following example the whole wake-up validation procedure can be seen. A wake-up event occurs for the ComMChannel CanIf and needs validation. The validation is processed and the wake-up event is notified to the BswM and to the ComM.

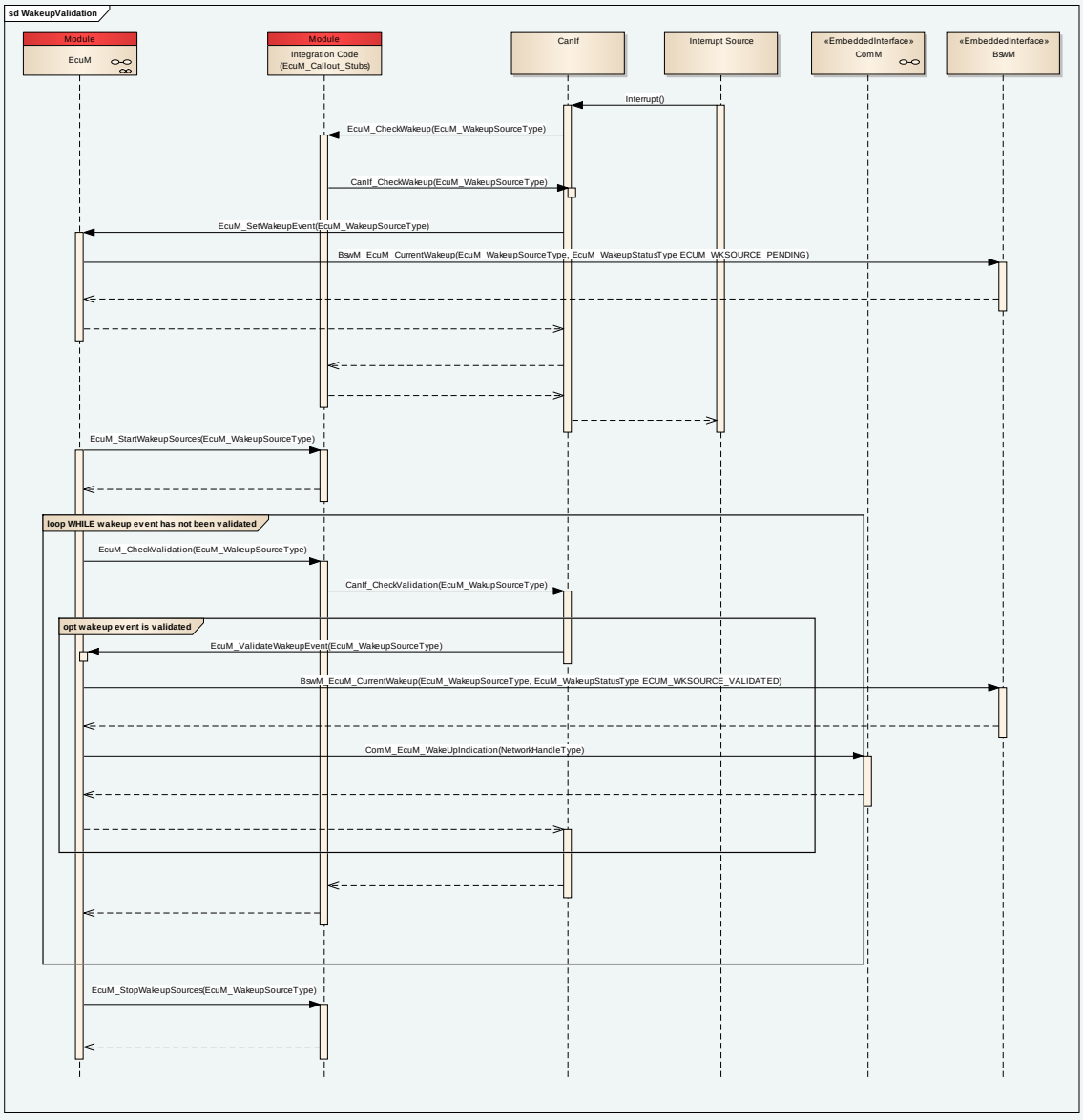

Figure 3-3 Example Wake-up Validation

3.10.2 Wake-up Validation Protocol for asynchronous Can transceiver

For all user configured wake-up sources the parameter “Check Validation Timeout” is configurable. If the parameter is set to a value which is not 0, the check wake-up validation protocol is active for that source.

For these sources the call of EcuM_SetWakeupEvent must not occur in the context of EcuM_CheckWakeup.

Example

In the following example parts of the wake-up validation procedure can be seen for an asynchronous Can transceiver.

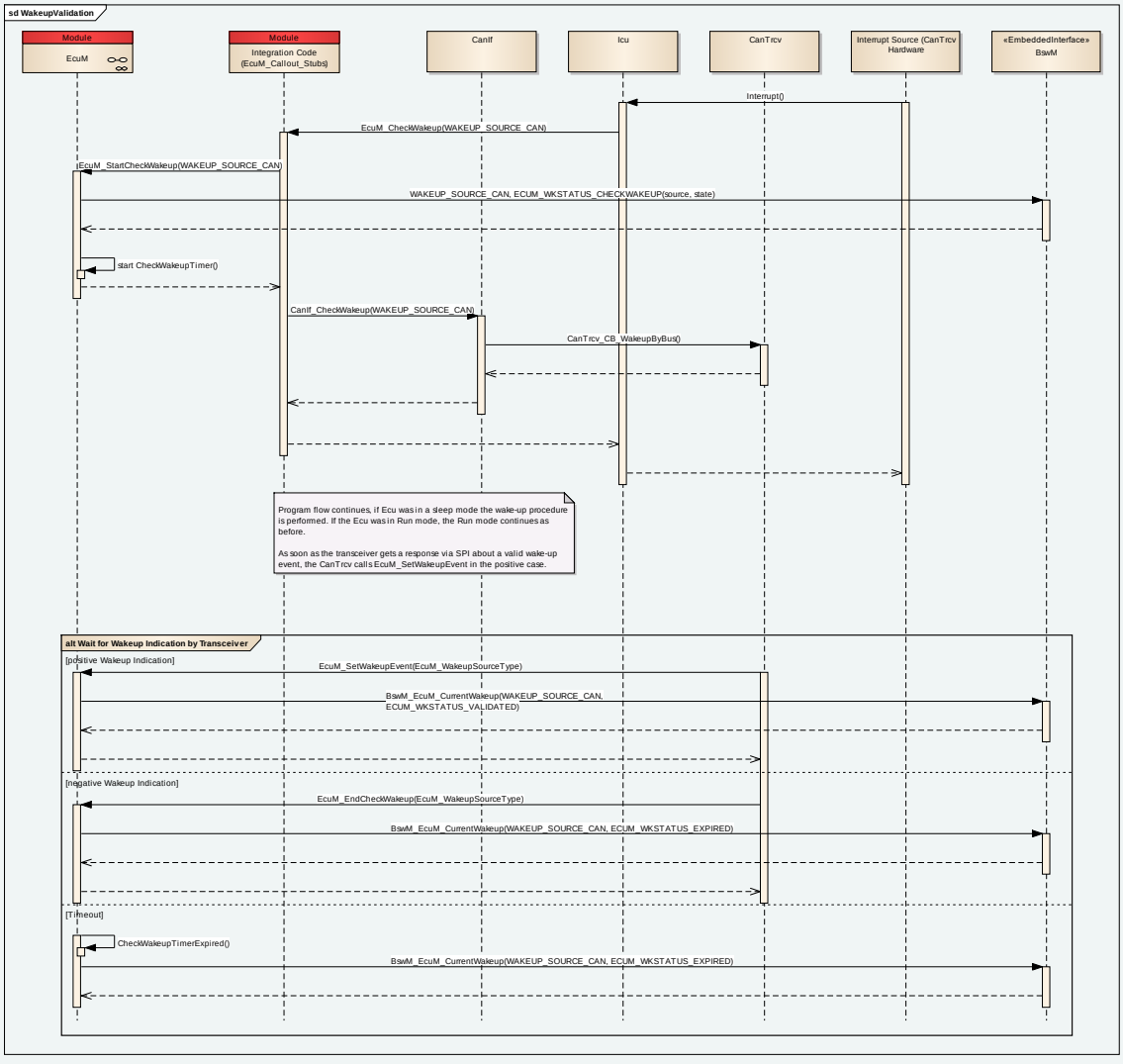

Figure 3-4 Example Wake-up Validation for asynchronous Can Transceivers

3.11 Error Handling

3.11.1 Development Error Reporting

Development errors are reported to the DET using the service Det_ReportError() as specified in [2], if development error reporting is enabled (ECUM_DEV_ERROR_DETECT==STD_ON).

The reported EcuM ID is 10.

The reported service IDs identify the services which are described in 5.2. The following table presents the service IDs and the related services:

Service ID	Service
0x00	EcuM_GetVersionInfo()
0x01	EcuM_Init()
0x02	EcuM_Shutdown()
0x03	EcuM_RequestRun()
0x04	EcuM_ReleaseRun()
0x05	EcuM_KillAllRUNRequests()
0x06	EcuM_SelectShutdownTarget()
0x07	EcuM_GetState()
0x08	EcuM_GetLastShutdownTarget()
0x09	EcuM_GetShutdownTarget()
0x0A	EcuM_RequestPOST_RUN()
0x0B	EcuM_ReleasePOST_RUN()
0x0C	EcuM_SetWakeupEvent()
0x0D	EcuM_GetPendingWakeupEvents()
0x12	EcuM_SelectBootTarget()
0x13	EcuM_GetBootTarget()
0x14	EcuM_ValidateWakeupEvent()
0x15	EcuM_GetValidatedWakeupEvents()
0x16	EcuM_ClearWakeupEvent()
0x18	EcuM_MainFunction()
0x19	EcuM_GetExpiredWakeupEvents()
0x1A	EcuM_StartupTwo()
0x1B	EcuM_SelectShutdownCause()
0x1C	EcuM_GetShutdownCause()
0x1D	EcuM_GetMostRecentShutdown()[not supported in this release]
0x1E	EcuM_GetNextRecentShutdown()[not supported in this release]
0x1F	EcuM_GoDown()
0x20	EcuM_GoHalt()
0x21	EcuM_GoPoll()
0x22	EcuM_SetRelWakeupAlarm()

Service ID	Service
0x23	EcuM_SetAbsWakeupAlarm()
0x24	EcuM_AbortWakeupAlarm()
0x25	EcuM_GetCurrentTime()
0x26	EcuM_GetWakeupTime()
0x27	EcuM_SetClock()
0x28	EcuM_StartCheckWakeup()
0x29	EcuM_EndCheckWakeup()
0x30	EcuM_ClearValidatedWakeupEvent()
0x2A	EcuM_KillAllPostRUNRequests()
0x2B	EcuM_SetState()
0x65	EcuM_CB_NfyNvMJobEnd()

Table 3-8 Service IDs

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

Error Code		Description
0x10	ECUM_E_UNINIT	A service was called prior to initialization.
0x11	ECUM_E_SERVICE_DISABLED	Error code defined by AUTOSAR SWS (not used in this implementation).
0x12	ECUM_E_NULL_POINTER	A null pointer was passed as an argument.
0x13	ECUM_E_INVALID_PAR	A parameter was invalid (not specified)
0x14	ECUM_E_MULTIPLE_RUN_REQUESTS	EcuM_RequestRUN or EcuM_ RequestPOST_RUN was called two times by the same user without release.
0x15	ECUM_E_MISMATCHED_RUN_RELEASE	EcuM_ReleaseRUN or EcuM_ ReleasePOST_RUN was called by a user without a previous request.
0x16	ECUM_E_STATE_PAR_OUT_OF_RANGE	API service EcuM_SelectShutdownTarget() called with parameter not in expected range
0x17	ECUM_E_UNKNOWN_WAKEUP_SOURCE	Wake-up source ID is not known by ECU State Manager
0x18	ECUM_E_INVALID_GEN_DATA	Generated data is not valid.
0x20	ECUM_E_MODULE_NOT_IN_STARTUP	EcuM_StartupTwo() is called and the EcuM is not in state EcuM_Startup_One which is entered in EcuM_Init().
0x21	ECUM_E_MODULE_NOT_IN_PREPSHUTDOWN	EcuM_Shutdown() was invoked without calling EcuM_GoDown().
0x22	ECUM_E_MODULE_NOT_IN_RUN_STATE	This error will be reported if the callout EcuM_AL_SwitchOff() does not switch off the ECU.
0x23	ECUM_E_NO_SLEEPMODE_CONFIGURED	This error will be reported if EcuM_GoPoll() or EcuM_GoHalt() is called and no SleepMode is configured.
0x24	ECUM_E_INVALID_STATEREQUEST	A state which was requested is invalid, perhaps because a former request is not finished yet.

Table 3-9 Errors reported to DET

3.11.2 Production Code Error Reporting

By default, production code related errors are reported to the DEM using the service `Dem_ReportErrorStatus()` as specified in [3], if production error reporting is enabled (In the case that a reference to a Dem event parameter is configured in EcuMDemEventParameterRefs).

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

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

Error Code	Description
ECUM_E_RAM_CHECK_FAILED	The RAM check during wake-up failed.
ECUM_E_CONFIGURATION_DATA_INCONSISTENT	Post build configuration data is inconsistent.
ECUM_E_IMPROPER_CALLER	Defensive behavior checks have detected improper use of the module.
ECUM_E_ALL_RUN_REQUESTS_KILLED	The API EcuM_KillAllRUNRequests() was called.

Table 3-10 Errors reported to DEM

Caution
Only ECUM_E_IMPROPER_CALLER and ECUM_E_ALL_RUN_REQUESTS_KILLED are passed to the Dem directly out of the static code. In the other cases EcuM_ErrorHook (see 3.11.3) is called and the integrator has to decide what happens in the case of these errors.

3.11.3 EcuM_ErrorHook

The EcuM has an own ErrorHook which offers the integrator the possibility to react on occurring errors during runtime.

Error Code	Description
`ECUM_E_HOOK_RAM_CHECK_FAILED`	If the Ram check has failed after a sleep phase, the ErrorHook is called with this parameter.
`ECUM_E_HOOK_CONFIGURATION_DATA_INCONSISTENT`	If the consistency check of pre-compile and link-time parameters in variant post-build has failed, the ErrorHook is called with this parameter.
`ECUM_E_HOOK_WRONG_ECUM_USAGE`	If the call of ShutdownOS returns to the EcuM. ShutdownOS has to call

Error Code	Description
	EcuM_Shutdown() to perform a shutdown.
ECUM_E_HOOK_INVALID_COREID	The OS returned an invalid CoreID via the API GetCoreID().
ECUM_E_HOOK_INVALID_APPLICATIONID	The OS returned an invalid ApplicationID via the API GetCurrentApplicationID().

Table 3-11 Description of EcuM internal Error Codes

The integrator has to implement the behavior of the EcuM in this situation. The EcuM reports the error not by default to the Dem. If this is desired, the integrator has to call the Dem.

3.12 Callout Execution Sequences

This chapter describes the execution order of callouts and important functions. This may be useful while integrating the software stack.

Caution
The execution sequences are not relevant for EcuM fixed.

3.12.1 Callouts from Startup to Run

STARTUP – RUN
Execution in EcuM_Init()
▪ EcuM_AL_SetProgrammableInterrupts() ▪ EcuM_AL_DriverInitZero() ▪ EcuM_AL_DriverInitOne() ▪ Mcu_GetResetReason() ▪ EcuM_SetWakeupEvent(ResetReason) ▪ StartOS(ECUM_DEFAULTAPPMODE)
Execution in EcuM_StartupTwo() according to AUTOSAR < 4.3.1

▪ SchM_Init()
▪ BswM_Init(NULL_PTR / CfgPtr_BswM)
  ▪ If Wake-up Events have occurred before BswM_Init:
  ▪ BswM_EcuM_CurrentWakeup(WakeupSource, ECUM_WKSTATUS_VALIDATED)

Execution in EcuM_StartupTwo() according to AUTOSAR >= 4.3.1, with the deviation described in ch. 3.6.2.3

▪ SchM_Start()
▪ SchM_Init()
▪ BswM_Init(NULL_PTR / CfgPtr_BswM)
  ▪ If Wake-up Events have occurred before BswM_Init:
  ▪ BswM_EcuM_CurrentWakeup(WakeupSource, ECUM_WKSTATUS_VALIDATED)
▪ SchM_StartTiming()

Table 3-12 Callouts from Startup to Run

3.12.2 Callouts from Run to Sleep (Halt) and back to Run

Run – Sleep (Halt) – Run
Selection of the ShutdownTarget must be done before the transition to sleep e.g. by the BswM
▪ EcuM_SelectShutdownTarget(ECUM_STATE_SLEEP, resetSleepMode)
All validated wake-up events must be cleared, e.g. by the BswM
▪ EcuM_ClearValidatedWakeupEvent(ECUM_WKSOURCE_ALL_SOURCES)
GoHalt must be called e.g. by the BswM
▪ EcuM_GoHalt()
Execution in EcuM_GoHalt()
▪ BswM_EcuM_CurrentWakeup(wakeupSource, ECUM_WKSTATUS_NONE) ▪ EcuM_EnableWakeupSources(wakeupSource) ▪ GetResource(ECUM_OS_RESOURCE) ▪ DisableAllInterrupts() ▪ EcuM_GenerateRamHash() ▪ Mcu_SetMode(ECUM_SLEEPMODELIST[ECUM_CURRENTSLEEPMODE].mcuMode) ▪ EnableAllInterrupts() ▪ EcuM_CheckRamHash() ▪ If CheckRamHash has failed ▪ EcuM_ErrorHook(ECUM_E_HOOK_RAM_CHECK_FAILED) ▪ DisableAllInterrupts() ▪ Mcu_SetMode(ECUM_NORMALMCUMODEREF) ▪ EnableAllInterrupts() ▪ EcuM_DisableWakeupSources(EcuM_PendingWakeups | EcuM_ValidatedWakeups)) ▪ BswM_EcuM_CurrentWakeup(EcuM_PendingWakeups | EcuM_ValidatedWakeups), ECUM_WKSTATUS_DISABLED) ▪ EcuM_AI_DriverRestart() ▪ ReleaseResource(ECUM_OS_RESOURCE)

Table 3-13 Callouts from Run to Sleep (Halt) and back to Run

3.12.3 Callouts from Run to Reset

Run – Reset
Selection of the ShutdownTarget must be done before the transition to Off e.g. by the BswM
▪ EcuM_SelectShutdownTarget(ECUM_STATE_RESET, resetMode)
GoDown must be called e.g. by the BswM
▪ EcuM_GoDown()
Execution in EcuM_GoDown()
▪ EcuM_OnGoOffOne() ▪ BswM_Deinit() ▪ SchM_Deinit() ▪ ShutdownOS(E_OK)
Shutdown must be called from the ShutdownHook
▪ EcuM_Shutdown()
Execution in EcuM_Shutdown()
▪ EcuM_OnGoOffTwo() ▪ EcuM_AL_Reset(EcuM_CurrentShutdownMode)

Table 3-14 Callouts from Run to Reset

3.12.4 Callouts from Run to Off

Run – Reset
Selection of the ShutdownTarget must be done before the transition to Off e.g. by the BswM
▪ EcuM_SelectShutdownTarget(ECUM_STATE_Off, 0)
All validated wake-up events must be cleared, e.g. by the BswM
▪ EcuM_ClearValidatedWakeupEvent(ECUM_WKSOURCE_ALL_SOURCES)
GoDown must be called e.g. by the BswM
▪ EcuM_GoDown()
Execution in EcuM_GoDown()
▪ EcuM_OnGoOffOne() ▪ BswM_Deinit() ▪ SchM_Deinit() > If a wake-up event has occurred, the Shutdown Target will be changed to ECUM_STATE_RESET and the reset mode will be ECUM_RESET_WAKEUP ▪ ShutdownOS(E_OK)
Shutdown must be called from the ShutdownHook
▪ EcuM_Shutdown()
Execution in EcuM_Shutdown()
▪ EcuM_OnGoOffTwo() ▪ EcuM_AL_SwitchOff()

Table 3-15 Callouts from Run to Off

3.13 EcuM Flex Users and Defensive Behavior

The EcuM offers the facility to configure flex Users to identify the caller of the routine EcuM_GoDown. The calling module has to use its Module ID as specified by AUTOSAR in [4].

Note
To use this feature, the switch EcuMEnableDefBehaviour must be active.

3.14 Alarm Clock

The EcuM flex offers the possibility to configure a clock which provides the absolute time since the last power-on reset of the ECU. This clock can be used to retrieve the current system time via the API EcuM_GetCurrentTime and to wake up the ECU from sleep phases.

In sleep phases the ECU will be woken up by the Gpt every second, depending if the Gpt supports this. If the wake up by the Gpt is the only wakeup event, the EcuM will increment the system clock and falls back to sleep again. If a wake up alarm has expired, the EcuM will call EcuM_SetWakeupEvent() to indicate a valid wake up of the ECU.

Note
To use this feature, the switch EcuMAlarmClockPresent must be active.

3.14.1 Configuring the Gpt to provide the Time base

To support the Alarm Clock, a Gpt channel must be configured in a way which leads to an interrupt every second. For a correct behavior of the Alarm Clock, even in sleep phases, the channel must be configured as followed:

Gpt Channel Parameter	Value
GptChannelMode	GPT_CH_MODE_CONTINUOUS
GptEnableWakeup	True
GptNotification	EcuM_AlarmCheckWakeup
GptWakeupSourceRef	Choose here the same Wakeup Source as configured for EcuM parameter EcuMAlarmWakeupSource

Table 3-16 Gpt Channel Configuration

Caution
The implementation of the EcuM alarm clock requires that the Gpt provides a time base of exactly one second. If this is not supported by Gpt, the EcuM does not perform a correction of the time base.

3.14.2 Configuring the EcuM for using the Alarm Clock

For setting a wake up alarm during the runtime of the ECU, an EcuMAlarmClock with a reference to an EcuMFlexUserConfig must be configured.

The Gpt channel configured in 3.14.1 must be referenced by the EcuM parameter EcuMGptChannelRef.

3.14.3 Setting of the EcuM Clock

The API EcuM_SetClock is offered to allow configuring an EcuMFlexUser to modify the system time during runtime. This user must be set as reference in the configuration parameter EcuMSetClockAllowedUserRef.

Only if this reference is configured, the usage of the API EcuM_SetClock is allowed for this user.

3.14.4 Setting of a Time Triggered Wake Up Alarm

Via the APIs EcuM_SetRelWakeupAlarm and EcuM_SetAbsWakeupAlarm the configured EcuMFlexUsers can set wake up alarms during the runtime of the ECU. This wake up alarm will be active during the next sleep phase.

The wake up alarm can be cancelled by the user during runtime of the ECU via the API EcuM_AbortWakeupAlarm.

Note
One single EcuMFlexUser can only set one single wake up alarm.

Caution
All wake up alarms are cleared if the ECU wakes up from a sleep phase, even if the reason for this wake up was not time triggered. The wake up alarms must be rearmed before the next sleep phase is entered.

3.15 MultiCore Ecu

The EcuM offers the possibility to handle multi core ECUs. The handling of the initialization, sleep and shutdown differs to a single core ECU and is described in the following.

3.15.1 Initialization of a MultiCore ECU

3.15.1.1 Initialization on the Master Core

After power-on of the ECU, the master core starts running and EcuM_Init() should be called in the startup code. At the end of EcuM_Init() the callout EcuM_StartOS() is called.

In the callout EcuM_StartOS() all other slave cores are started via the OS API StartCore().

Example

In the following example the startup sequence of the master core for a ECU with 4 cores can be seen:

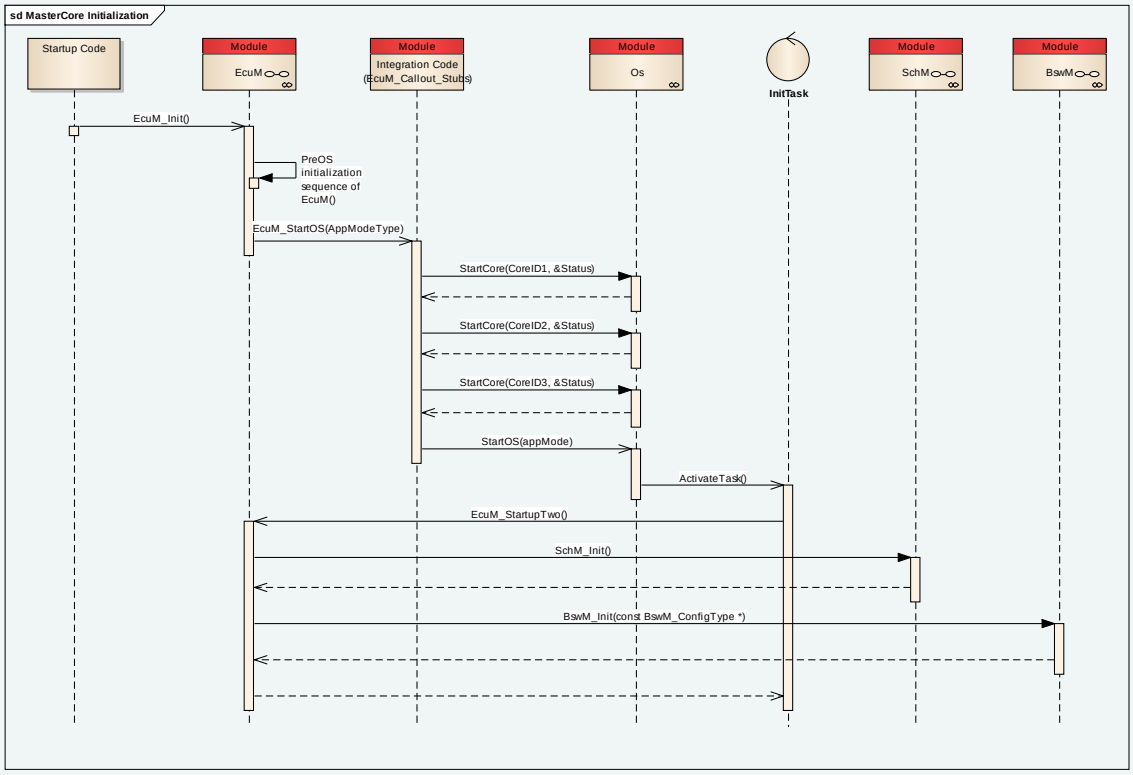

Figure 3-5 Startup Sequence on a Master Core (StartupTwo sequence displayed for AUTOSAR < 4.3.1)

Note

The callout EcuM_StartOS() is filled by the configuration tool per default. In some cases it might be necessary to adapt this callout.

3.15.1.2 Initialization on the Slave Core

After the slave core has been started by the master core, it also starts running with the startup code. EcuM_Init() is called from the startup code, but on the slave core only driver initialization and a call to StartOS() is performed via the callout EcuM_StartOS().

Example

In the following example the startup sequence of a slave core for a ECU with multiple cores can be seen:

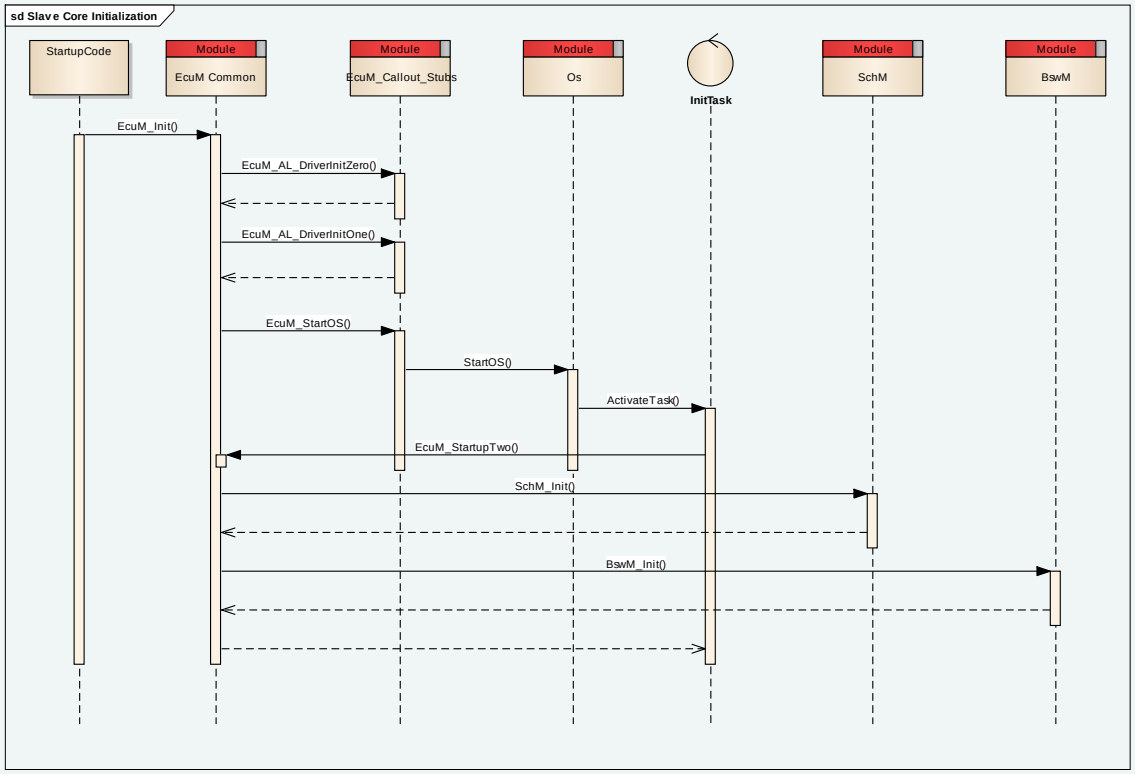

Figure 3-6 Startup Sequence on a Slave Core (StartupTwo sequence displayed for AUTOSAR < 4.3.1)

Caution

On the slave core a call to EcuM_StartupTwo() is only necessary if the initialization of the SchM / BswM should be done by the EcuM.

BswM is only initialized if it is configured in context of this partition.

3.15.1.2.1 Driver initialization on the Slave Core

The callouts EcuM_AL_DriverInitZero() and EcuM_AL_DriverInitOne() are also called on slave cores, but the generated code is only executed on the master core.

On which core the driver initialization is called, is determined via the OS API GetCoreID(), as it can be seen in the code example below.

A slave core specific handling has to be implemented by the user.

Example

```
/*********************************************************************
* EcuM_AL_DriverInitZero
*********************************************************************/
FUNC(void, ECUM_CODE) EcuM_AL_DriverInitZero(void)
{
  if(GetCoreID() == ECUM_CORE_ID_MASTER)
  {
    MasterCore_Init();
  }

/*********************************************************************
 * DO NOT CHANGE THIS COMMENT!   <USERBLOCK EcuM_AL_DriverInitZero>
DO NOT CHANGE THIS COMMENT!
*********************************************************************/
  /* Add implementation of EcuM_AL_DriverInitZero() */

  return;
/*********************************************************************
 * DO NOT CHANGE THIS COMMENT!   </USERBLOCK>   DO NOT CHANGE THIS
COMMENT!
*********************************************************************/
}
```

3.15.2 Sleep handling of slave cores

The EcuM flex supports two different ways to set the ECU to sleep, with and without synchronization of all cores. Which handling is used depends on the boolean parameter EcuMSlaveCoreHandling

EcuMSlaveCoreHandling	Behavior
False	The Master Core does not care about slave cores during the sleep mode. Depending on the used hardware, it might happen that the Master Core has switched already to sleep and the slave cores are still running.
True	The Master Core waits on the way to sleep (initiated via EcuM_GoHalt() / EcuM_GoPoll()) till all slave cores has already switched to sleep. During wait for the slave cores, the callout EcuM_WaitForSlaveCores is called cyclically till all cores have switched their state to sleep. The callout can be used to set the slaves to sleep.

Table 3-17 Sleep handling on Slave Cores

3.15.3 Blocking of the BSW Scheduler during Sleep

If only one BSW scheduler is used on the master core, it is sufficient to configure only one OsResource which is blocked during the sleep mode.

If there is more than one BSW scheduler running on several cores, it is necessary to configure an OsResource for every core. The configuration tool assigns automatically the configured OsResource to the corresponding core.

3.15.4 Shutdown of the MultiCore ECU

It is necessary to call EcuM_GoDown() on all cores which have a running SchM to assure a regular de-initialization of the SchM.
Finally after EcuM_GoDown() was called for all these slave cores, the API can be called on the master core. This leads via the callout EcuM_ShutdownOS to a call of the OS API ShutdownAllCores(). This API synchronizes all cores and stops the slaves.

Note
If the SchM is only running on the master core it is sufficient to call EcuM_GoDown() on the master core only.

3.15.5 Reconfiguration of the BSW Core ID

The EcuM supports the configuration of the BSW Core Id. Per default the master Core Id is mapped to the OS define OS_CORE_ID_MASTER (Id 0).

If the BSW shall run on another Core, the Id has to be configured via the configuration tool.

3.16 Mode Handling for EcuM Flex

3.16.1 Mode Handling

The BswM can set a specific EcuM state (via EcuM_SetState) which is mapped to the corresponding mode and an Rte mode switch will be initiated by the EcuM. The mapping of states to modes can be seen in Table 3-18.

After the mode switch is initiated, the EcuM polls the Rte in each MainFunction cycle if the mode switch is executed successfully. After the Rte has acknowledged the successful mode switch execution, the EcuM will notify the BswM about the finished mode switch.

EcuM State	EcuM Mode
ECUM_STATE_STARTUP	RTE_MODE_EcuM_Mode_STARTUP
ECUM_STATE_SLEEP	RTE_MODE_EcuM_Mode_SHUTDOWN or RTE_MODE_EcuM_Mode_SLEEP
ECUM_STATE_APP_RUN	RTE_MODE_EcuM_Mode_RUN
ECUM_STATE_APP_POST_RUN	RTE_MODE_EcuM_Mode_POST_RUN
ECUM_STATE_SHUTDOWN	RTE_MODE_EcuM_Mode_SHUTDOWN or RTE_MODE_EcuM_Mode_SLEEP

Table 3-18 Mapping of States to Modes

Note
In case of a requested state ECUM_STATE_SHUTDOWN or ECUM_STATE_SLEEP, the corresponding mode depends on the currently configured shutdown target.

3.16.2 Run Request Protocol

The run request protocol is a mechanism for applications or Software Components (SW-C) to request RUN state explicitly via EcuM_RequestRUN. The EcuM notifies the BswM about an active application request. If the application has nothing to do anymore it must release the previous requested RUN state. If no other SW-C has requested RUN state the ECU State Manger will notify the BswM that no application request is active anymore.

If SW-C needs special preparation for one of the shutdown states (SLEEP, OFF, RESET) the SW-C must request POST RUN state. This is the same mechanism like requesting RUN state. So, the POST RUN state has to be released after the job of the application is finished. It is very important for SW-C's which needs POST RUN state activities to request the POST RUN state before releasing the RUN request. Otherwise it is possible that the application doesn't get the chance to execute its POST RUN activities, depending on the BswM configuration.

To request RUN or POST RUN state each SW-C must be a configured user of the ECU State Manager. Therefore it is necessary to define one user for each SW-C to place requests.

3.17 Generated Template Files

A generated template file in this document is a file which:

- is generated by the generation tool at every generation process
- the user can modify this template for his needs
- the changes made by the user will not be overwritten at the next generation process

In order not to overwrite the changes made by the user, the template file contains special comments. The user must insert his code between the two comments which delimit the user block. The comments have the following format:

```
/**********************************************************************************************************************
 * DO NOT CHANGE THIS COMMENT!  <USERBLOCK FUNCTIONNAME>  DO NOT CHANGE THIS COMMENT!
**********************************************************************************************************************/

/**********************************************************************************************************************
 * DO NOT CHANGE THIS COMMENT!  </USERBLOCK>              DO NOT CHANGE THIS COMMENT!
**********************************************************************************************************************/
```

Caution
Do not modify or delete these comments.

3.18 Wake-up Event Handling and Wake-up Validation

The handling of Wake-up Sources and Wake-up Validation has to be configured and implemented specifically for every ECU. The following list provides a short overview which callouts are affected:

- EcuM_EnableWakeupSources(), (refer to Ch. 5.8.2.17)
- EcuM_DisableWakeupSources(), (refer to Ch. 5.8.2.18)
- EcuM_CheckWakeup(), (refer to Ch. 5.8.2.21)
- EcuM_StartWakeupSources(), (refer to Ch. 5.8.2.19)
- EcuM_StopWakeupSources(), (refer to Ch. 5.8.2.20)

The integration task is to fill these callouts with code which fulfill the ECU specific requirements. The following paragraphs illustrate two example use cases:

- Wake-up after a physical sleep mode
- Wake-up validation of communication channels (EcuM in Run state)

3.18.1 Wake-up after a Physical Sleep Mode

3.18.1.1 Use Case Description

A raising edge on an ICU channel shall bring the ECUM into RUN state. A wake-up source “ECUM_WKSOURCE_ICU_CH0” is configured for that. The name of the configured ICU channel is Icu_Channel0.

No wake-up validation shall be performed on that wake-up event. This wake-up event is the only active wake-up event for the desired sleep mode.

3.18.1.2 Execution Flow

- EcuM is in ECUM_STATE_RUN
- BswM calls EcuM_GoHalt()
  - Callout EcuM_EnableWakeupSources() is executed.
  - EcuM transits to sleep, Mcu_SetMode() is called
- External event triggers ICU hardware to raise an interrupt
- Callout EcuM_CheckWakeup() is executed by ISR
- API function EcuM_SetWakeupEvent() is executed
- EcuM executes implicitly EcuM_ValidateWakeupEvent() because wake-up event is instantly valid
- EcuM transits from ECUM_STATE_SLEEP to ECUM_STATE_WAKEUP_ONE
- EcuM transits from ECUM_STATE_WAKEUP_TWO to ECUM_STATE_ RUN
  - Callout EcuM_DisableWakeupSources() is executed

3.18.1.3 Callout Implementation Examples

```
FUNC(void, ECUM_CODE) EcuM_EnableWakeupSources(EcuM_WakeupSourceType
wakeupSource)
{
  /* Check for each configured wake-up source the corresponding bit
   * is set. Here the bit for the ICU wake-up source must be set
   */
  if ((wakeupSource & ECUM_WKSOURCE_ICU_CH0) != 0)
  {
    Icu_EnableNotification(Icu_Channel0);
    Icu_EnableWakeup(Icu_Channel0);
    Icu_SetMode(ICU_MODE_SLEEP);
  }
  /* … */
}

FUNC(void, ECUM_CODE) EcuM_CheckWakeup(EcuM_WakeupSourceType wakeupSource)
{
  if ((wakeupSource & ECUM_WKSOURCE_ICU_CH0) != 0)
  {
    /* no validation necessary, so call EcuM_SetWakeupEvent() */

    EcuM_SetWakeupEvent(ECUM_WKSOURCE_ICU_CH0);
  }
  /* … */
}



FUNC(void, ECUM_CODE) EcuM_DisableWakeupSources(EcuM_WakeupSourceType
wakeupSource)
{
  if ((wakeupSource & ECUM_WKSOURCE_ICU_CH0) != 0)
  {
    Icu_DisableNotification(Icu_Channel0);
    Icu_DisableWakeup(Icu_Channel0);
    Icu_SetMode(ICU_MODE_NORMAL);
  }
}
```

3.18.2 Wake-up Validation of Communication Channels (ECUM in RUN State)

3.18.2.1 Use Case Description

A wake-up capable CAN hardware is assumed. A message on a CAN channel shall be recognized and set the CAN channel into normal operation mode (which will be triggered by ComM). A wake-up source ECUM_WKSOURCE_CAN0 is configured for that. Wake-up Validation shall be performed for that channel.

3.18.2.2 Execution Flow

> ECUM is in RUN state, the CAN channel is in sleep state and is able to detect wake-up events

> Callout EcuM_CheckWakeup() is executed by ISR

> API EcuM_SetWakeupEvent() is executed, EcuM starts wake-up validation timeout

> EcuM_MainFunction() triggered by SCHM

- (a) ECUM detects a pending wake-up event and executes callout EcuM_StartWakeupSources()
- (b) ECUM executes callout EcuM_CheckValidation()
- Note: step (b) may be executed several times, with each EcuM_MainFunction() call until the wake-up event is validated or expired, but EcuM_StartWakeupSources() is executed only once.

> Case Validation successful:

- API EcuM_ValidateWakeupEvent() is executed, within this routine ComM_WakeUpIndication() is called
- EcuM_MainFunction() triggered by SCHM
  - ECUM stops validation timeout

> Case Validation failed:

- ECUM executes callout EcuM_StopWakeupSources()
- The pending wake-up changes to an expired wake-up source

3.18.2.3 Callout Implementation Examples

3.18.2.3.1 EcuM_CheckWakeup

```
FUNC(void, ECUM_CODE) EcuM_CheckWakeup(EcuM_WakeupSourceType wakeupSource)
{
  if((wakeupSource & ECUM_WKSOURCE_CAN0) != 0)
  {
    CanIf_CheckWakeup(ECUM_WKSOURCE_CAN0);
  }
}
```

3.18.2.3.2 EcuM_CheckValidation

```
FUNC(void, ECUM_CODE) EcuM_CheckValidation(EcuM_WakeupSourceType wakeupSource)
{
  if ((wakeupSource & ECUM_WKSOURCE_CAN0) != 0)
  {
    /* Query the driver if the wake-up event was valid */
    CanIf_CheckValidation(ECUM_WKSOURCE_CAN0);
  }
}
```

3.18.2.3.3 EcuM_StartWakeupSources and EcuM_StopWakeupSources in the case of a MICROSAR Classic CanSM

If the used CanSM module is a MICROSAR Classic module, the following implementation can be used.

```
void EcuM_StartWakeupSources(EcuM_WakeupSourceType wakeupSource)
{
  if ((wakeupSource & <WKSOURCE_CAN>) != 0)
  { /* CanSM_StartWakeupSources must not be called cyclic */
    if (CanSM_StartWakeupSources(<CorrespondingNetworkHandle>) == E_NOT_OK)
    {
      /* Request denied, Execution not possible */
    }
  }
}
```

```
void EcuM_StopWakeupSources(EcuM_WakeupSourceType wakeupSource)
{
  if ((wakeupSource & <WKSOURCE_CAN>) != 0)
  {/* CanSM needs the corresponding Network Handle e.g. 0x00 */
    if (CanSM_StopWakeupSources(<CorrespondingNetworkHandle>, wakeupSource) ==
CANSM_E_STOP_WAKEUPSOURCES_RUNNING)
    {
      /*  Avoid Ecu shutdown, set a "wakeupValidation" EcuM request Run */
      /*  Poll the "state" of the StopWakeupValidation
       *  in a cyclic application task */
    }
  }
}
```

```
void <Appl_CyclicTask>(void)
{
  if ( (CanSM_StopWakeupSources(<CorrespondingNetworkHandle>, 0 /*dummyValue*/ )
==  E_OK)||
(CanSM_StopWakeupSources(<CorrespondingNetworkHandle>, 0 /*dummyValue*/ ) ==
CANSM_E_REQUESTED_COMM_MODE) )
  {
      /* Enable Ecu shutdown, remove “wakeupValidation” EcuM request Run */
  }
}
```

3.18.2.3.4 EcuM_StartWakeupSources and EcuM_StopWakeupSources in the case of a non MICROSAR Classic CanSM

If the used CanSM module is a non MICROSAR Classic module, the following implementation can be used.

```
FUNC(void, ECUM_CODE) EcuM_StartWakeupSources(EcuM_WakeupSourceType
wakeupSource)
{
  CanIf_ControllerModeType CanIfCtrlMode;
  if ((wakeupSource & ECUM_WKSOURCE_CAN0) != 0)
  {
    /* determine in which is the current Can Controller state */
    (void)CanIf_GetControllerMode(0, &CanIfCtrlMode);
    /* in case the Can Controller is not CANIF_CS_STARTED */

    if (CANIF_CS_STARTED != CanIfCtrlMode)
    {
    /* Set the controller and transceiver mode into normal operation mode*/
    CanIf_SetTrcvMode(0, CANIF_TRCV_MODE_NORMAL);
    CanIf_SetControllerMode(0, CANIF_CS_STOPPED);
    CanIf_SetControllerMode(0, CANIF_CS_STARTED);
}
else
{
  /* Stack already up and running */
}
 }
}


FUNC(void, ECUM_CODE) EcuM_StopWakeupSources(EcuM_WakeupSourceType wakeupSource)
{
  if ((wakeupSource & ECUM_WKSOURCE_CAN0) != 0)
  {
    /* Validation was not successful, set the CAN controller and
    * Transceiver back to sleep mode. */
    CanIf_SetControllerMode(0, CANIF_CS_STOPPED);
    CanIf_SetControllerMode(0, CANIF_CS_SLEEP);
    CanIf_SetTrcvMode(0, CANIF_TRCV_MODE_STANDBY);
  }
}
```

4 Integration

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

4.1 Scope of Delivery

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

4.1.1 Static Files

File Name	Source Code Delivery	Object Code Delivery	Description
EcuM.c	■		This is the source file of the EcuM. It contains the implementation of the EcuM interfaces.
EcuM.h	■	■	This is the header file of the EcuM. It declares the interfaces of the MIRCROSAR module EcuM.
EcuM_Cbk.h	■	■	Contains the prototypes of the provided callbacks and callout functions.

Table 4-1 Static files

Do not edit manually

The static files listed above must not be edited by the user!

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool.

File Name	Description
EcuM_Cfg.h	Contains the configuration of the EcuM.
EcuM_Cfg.c	Contains the generated configuration data of the EcuM
EcuM_PrivateCfg.h	Contains configuration data which is only relevant for the EcuM implementation. This file must be only included by the EcuM implementation files.
EcuM_Generated_Types.h	Contains all provided types of the EcuM.
EcuM_PBcfg.c	Contains the post-build configuration of the EcuM.
EcuM_Callout_Stubs.c	Template for the callout code which has to be filled by the integrator.
EcuM_Init_PBcfg.c	This file contains configuration pointers to post-build modules.
EcuM_Init_PBcfg.h	This file contains the definition of the global post-build struct.
EcuM_Init_Cfg.c	This file contains configuration pointers to variant modules.
EcuM_Init_Cfg.h	This file contains the definition of the variant modules struct.
EcuM_Error.h	This file provides an BSW Error function for post-build-loadable

File Name	Description
EcuM_MemMap.h	This file is generated by the MICROSAR MemMap module and provides the module specific memory sections.

Table 4-2 Generated files

4.2 Critical Sections

The EcuM calls the following function when entering a critical section:

>void SchM_Enter_EcuM_ECUM_EXCLUSIVE_AREA_0(void)

> When the critical section is left the following function is called by the EcuM:

>void SchM_Exit_EcuM_ECUM_EXCLUSIVE_AREA_0(void)

Critical Section Define	Interrupt Lock
ECUM_EXCLUSIVE_AREA_0	No interrupt by any wake-up interrupt is allowed. These interrupts must be locked in this exclusive area.
ECUM_EXCLUSIVE_AREA_1	If it cannot be assured that a 32bit variable is written atomically, this exclusive area must be configured as a spin lock to protect access on global state variables. **Note** The configuration of this exclusive area is only necessary in the case of a multi core ECU
ECUM_EXCLUSIVE_AREA_2	No task switch and no interrupt from the Gpt is allowed in this exclusive area to protect the global system time. **Note** The configuration of this exclusive area is only necessary if the feature Alarm Clock is enabled
ECUM_EXCLUSIVE_AREA_3	This exclusive area must be configured as a spin lock to protect access on global shutdown target variables.

Table 4-3 Critical Sections

4.3 Memory Mapping with multiple partitions

The partition specific memory sections are created with the following rules:

Flex / FixFlex Configuration

```
ECUM_START_SEC_VAR_PARTITION_<NameOfOsApplication>_NOCACHE_NOINIT_32BIT
ECUM_STOP_SEC_VAR_PARTITION_<NameOfOsApplication>_NOCACHE_NOINIT_32BIT
```

Fixed Configuration

```
ECUM_START_SEC_VAR_PARTITION_CORE_<NumberOfCore>_NOCACHE_NOINIT_32BIT
ECUM_STOP_SEC_VAR_PARTITION_CORE_<NumberOfCore>_NOCACHE_NOINIT_32BIT
```

Caution
The respective section must be mapped to a section where no caching is allowed!

In a partitioning solution, the API EcuM_Init can be considered as safety related (see safety manual), the rest of the module can be considered QM.

After the operating system is started, the EcuM should always be assigned to the same OsApplication as the BSW stack is mapped to.

The EcuM memory sections should also always be assigned to this OsApplication.

Please note that before the operating system is started, no OsApplications exist. Care must be taken that memory of the EcuM cannot be overwritten by other QM modules if EcuM is considered safety related.

The Vector MICROSAR operating system provides a PreStartTask as a special context to restrict memory access using the MPU even prior to StartOS.

4.4 Include Structure

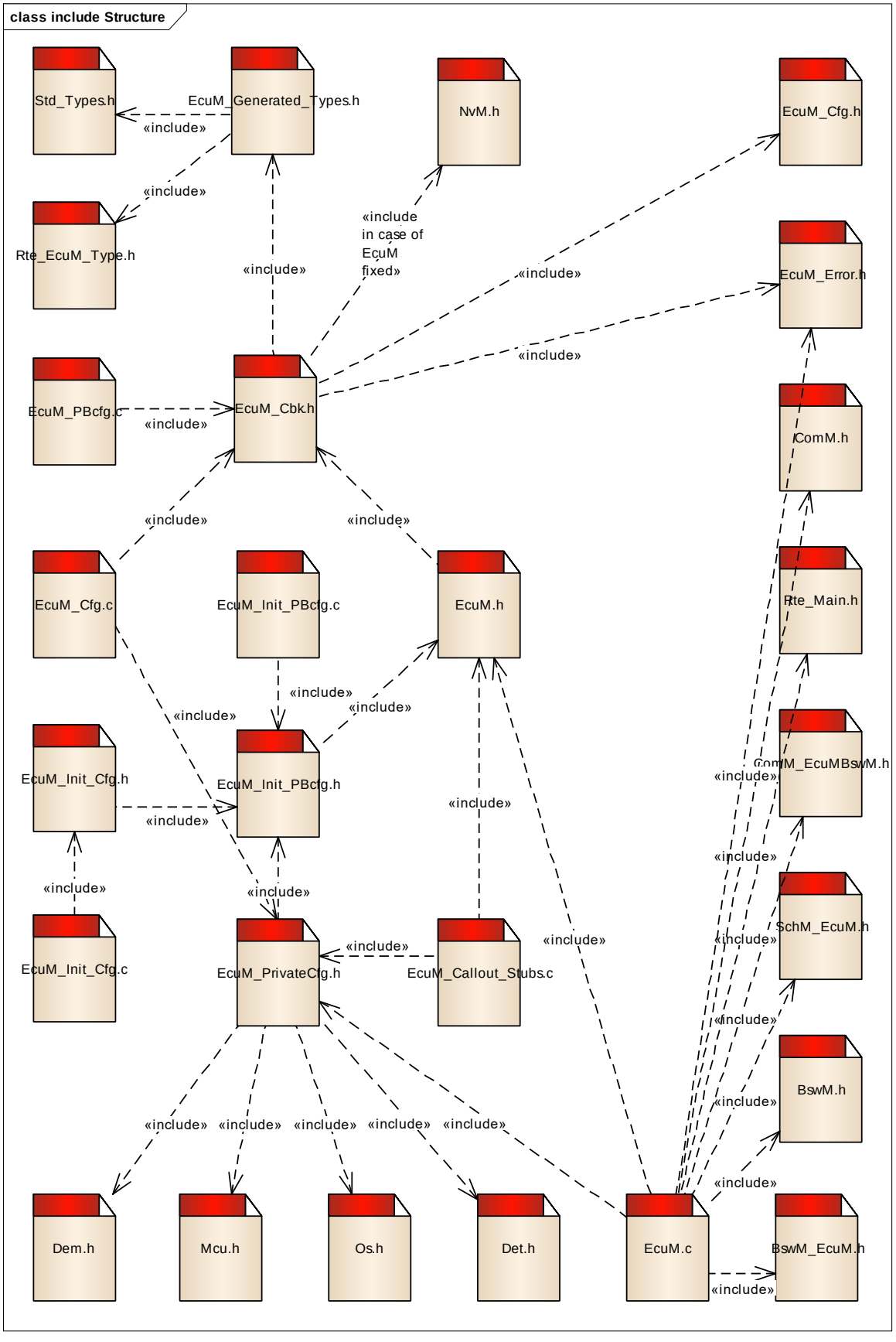

Figure 4-1 Include structure

4.5 Dependencies on other BSW Modules

4.5.1 BswM

The EcuM module depends on the BswM. The EcuM performs the initialization of the BswM during EcuM_StartupTwo().

The states of all wake-up sources are reported to the BswM in the case of a changing wake-up source.

The usage of the BswM cannot be switched off.

4.5.1.1 BswM and EcuM fixed

The EcuM reports all state changes described in 3.6.2.1 to the BswM.

4.5.2 AUTOSAR OS

The EcuM module depends on the AUTOSAR OS. It starts and performs the shutdown of the OS.
The EcuM needs a valid reference within the EcuC file to a configured OS application mode. Additionally an OsResource must be configured to block other tasks during a sleep mode.
The usage of the OS cannot be switched off.

4.5.3 MCU

The EcuM module depends on a MCU. The MCU mode settings are used to enter power saving modes in the phases ECUM_STATE_SLEEP and ECUM_STATE_OFF, it is also used to restore the normal MCU mode. Every sleep mode must have configured a MCU mode which will be entered in that sleep mode.

After a reset, the MCU is called to get the reason for the current reset.

The usage of the MCU cannot be switched off.

4.5.4 DEM

The EcuM depends on the DEM. The EcuM supports the pre-initialization of the DEM and if the production errors for the EcuM are configured as active, the EcuM reports some Errors to the DEM. Refer to chapter 3.11.2 for more information.
The usage of the DEM can be switched off.

4.5.5 DET

The EcuM depends on the DET. The EcuM performs the initialization and reports development errors for diagnostic purposes. Refer to chapter 3.11.1 for more information.
The usage of the DET can be switched off.

4.5.6 ComM

This module depends on the ComM. The EcuM manages the validation of communication channels. In the case of a validated wake-up event from a communication channel, the EcuM reports this event to the ComM.

4.5.6.1 ComM and EcuM fixed

In the transition to ECUM_STATE_APP_RUN, the EcuM calls ComM_CommunicationAllowed() for all configured communication channels.

In ECUM_STATE_APP_RUN, the ComM API ComM_GetState() is called for every communication channel in EcuM_MainFunction.

If ComM_GetState() returns COMM_NO_COM_NO_PENDING_REQUEST for all channels, the EcuM can leave the ECUM_STATE_APP_RUN.

4.5.7 SchM

The EcuM module depends on the SchM. The EcuM performs the initialization of the SchM during EcuM_StartupTwo().

The usage of the SchM cannot be switched off.

4.5.8 Gpt

In the case that the Alarm Clock is enabled, the EcuM depends on the Gpt. The EcuM initialize the Gpt (has to be done in EcuM_AL_DriverInitOne) and starts the corresponding timer during EcuM_StartupTwo(). On the way to sleep, the mode of the Gpt is switched to sleep and the normal mode is recovered after a wake-up from sleep.

4.5.9 NvM

The EcuM handles the call of NvM_WriteAll() and NvM_CancelWriteAll(). Both calls are protected with a configurable timeout to guarantee a shutdown of the ECU even in case of a defect NvM.

Caution
Dependency to the NvM exists only in case of EcuM fixed.

5 API Description

5.1 Type Definitions

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

Type Name	C-Type	Description	Value Range
EcuM_StateType	uint8	Encodes all states and sub states provided by the ECU State Manager.	ECUM_SUBSTATE_MASK Get the current state by AND gating the state with this mask. All states are delivered including substates.
			ECUM_STATE_STARTUP STARTUP super state
			ECUM_STATE_STARTUP_ONE Initialization of drivers which don't need OS support.
			ECUM_STATE_STARTUP_TWO Initialization of drivers which need OS support.
			ECUM_STATE_WAKEUP WAKE-UP super state Not used in this EcuM flex Implementation!
			ECUM_STATE_WAKEUP_ONE Reinitializing of drivers for normal operation.
			ECUM_STATE_WAKEUP_VALIDATION Waits for validation of a wake-up event
			ECUM_STATE_WAKEUP_REACTION Computes the appropriate wake-up reaction Not used in this EcuM flex Implementation!
			ECUM_STATE_WAKEUP_TWO Prepares the ECU for RUN state Not used in this EcuM flex Implementation!
			ECUM_STATE_WAKEUP_WAKESLEEP A short system phase where the ECU transit from a wake-up directly to sleep again. Not used in this EcuM flex Implementation!
			ECUM_STATE_WAKEUP_TTII

Type Name	C-Type	Description	Value Range
			Performs the TTII protocol Not used in this EcuM flex Implementation!
			ECUM_STATE_RUN Normal ECU operation super state
			ECUM_STATE_APP_RUN ECU is in normal operation state Not used in this EcuM flex Implementation!
			ECUM_STATE_APP_POST_RUN ECU performs POST RUN activities Not used in this EcuM flex Implementation!
			ECUM_STATE_SHUTDOWN Shutdown super state
			ECUM_STATE_PREP_SHUTDOWN Prepares the ECU for the following shutdown sequence. Not used in this EcuM flex Implementation!
			ECUM_STATE_GO_SLEEP Activation of the wake-up sources
			ECUM_STATE_GO_OFF_ONE Shutdown of system services
			ECUM_STATE_GO_OFF_TWO Performs a RESET or switches off the ECU
			ECUM_STATE_SLEEP ECU is in sleep state (this information cannot be retrieved)
			ECUM_STATE_OFF ECU is without power supply (this information cannot be retrieved)
EcuM_WakeupSourceType	uint32	Each bit in this type identifies a wake-up source.	ECUM_WKSOURCE_POWER Identifies a power on reset (bit 0)
			ECUM_WKSOURCE_RESET Identifies a hardware reset (bit 1)
			ECUM_WKSOURCE_INTERNAL_RESET Identifies resets which only reset the core of the microcontroller but not the peripherals. This source also indicates unhandled exceptions (bit 2)

Type Name	C-Type	Description	Value Range
			ECUM_WKSOURCE_INTERNAL_WDG Identifies a reset by internal watchdog (bit 3)
			ECUM_WKSOURCE_EXTERNAL_WDG Identifies a reset by external watchdog (bit 4). (This is only possible if the hardware supports this feature)
			ECUM_WKSOURCE_ALL_SOURCES Identifies each wake-up source
			ECUM_WKSOURCE_NONE Value 0. This is a MICROSAR ECUM extension and identifies an invalid wake-up source.
			ECUM_WKSOURCE_<NAME> Can be extended by configuration
EcuM_UserType	uint8	ID of the Users which are able to request RUN state. Each user must have a unique ID.	0..255 The Range depends on the number of configured users
EcuM_WakeupStateType	uint8	The type describes possible results of the WAKE-UP VALIDATION state.	ECUM_WKSTATUS_NONE The wake-up source is Disabled
			ECUM_WKSTATUS_PENDING The wake-up event was detected but not yet validated
			ECUM_WKSTATUS_VALIDATED The wake-up event is valid
			ECUM_WKSTATUS_EXPIRED The wake-up event has not been validated and has already expired.
			ECUM_WKSTATUS_ENABLED The wake-up source is enabled (armed) and is ready for detecting wake-up events.
			ECUM_WKSTATUS_CHECKWAKEUP Asynchronous wake-up event is detected but SetWakeupEvent has not been called yet.
EcuM_BootTargetType	uint8	Defines the boot target which should be chosen in the next start up.	ECUM_BOOT_TARGET_APP Boot into application mode
			ECUM_BOOT_TARGET_BOOTLOADER Boot into boot loader mode
EcuM_ResetType	uint8		ECUM_RESET_MCU

Type Name	C-Type	Description	Value Range
		This type describes the reset mechanisms supported by the ECU State Manager. It can be extended by configuration.	Microcontroller reset via Mcu_PerformReset
			ECUM_RESET_WDG Watchdog reset via WdgM_PerformReset
			ECUM_RESET_IO Reset by toggling an I/O line
			ECUM_RESET_WAKEUP Reset in the case of a wake-up event during shutdown
			ECUM_RESET_<NAME> Can be extended by configuration.
EcuM_ShutdownCauseType	uint8	This type describes the cause for a shutdown by the ECU State Manager. It can be extended by configuration.	ECUM_CAUSE_UNKNOWN No cause was set.
			ECUM_CAUSE_ECU_STATE ECU state machine entered a state for shutdown
			ECUM_CAUSE_WDGM Watchdog Manager detected a failure
			ECUM_CAUSE_DCM Diagnostic Communication Manager requests a shutdown due to a service request
			ECUM_CAUSE_<NAME> Can be extended by configuration.

Table 5-1 Type definitions

5.2 Global Constants

5.2.1 Component Versions

The source code versions of ECUM are provided by three BCD coded macros.

Name	Type	Description
ECUM_SW_MAJOR_VERSION (EcuM_MainVersion)	BCD	Contains the major component version number.
ECUM_SW_MINOR_VERSION (EcuM_SubVersion)	BCD	Contains the minor component version number.
ECUM_SW_PATCH_VERSION (EcuM_ReleaseVersion)	BCD	Contains the patch level component version number.

5.3 Services Provided by EcuM

5.3.1 EcuM_MainFunction

Prototype	
void **EcuM_MainFunction** (void)	
Parameter	
void	none
Return code	
void	none
Functional Description	
The service which implements all activities of the ECU state Manager while OS is up and running. In the MainFunction the wake-up validation is handled. This service must be called on a periodic basis from an adequate OS task. ▪ The service also carries out the wake-up validation protocol. The smallest validation timeout typically should limit the period. ▪ As a rule of thumb, the period of this service should be in the order of half as long as the shortest time constant mentioned in the topics above **Note** In case that the EcuM handles the slave cores, the EcuM_MainFunction has to be called on each relevant partition to achieve that the SchM and the BswM are de-initialized in case of a shutdown.	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is non-reentrant. ▪ Called by SchM	
Call Context	
▪ Function could be called from task level	

Table 5-2 EcuM_MainFunction

5.3.2 EcuM_Init

Prototype	
void **EcuM_Init** (void)	
Parameter	
void	none
Return code	
void	none
Functional Description	
The Init function is called to initiate the startup procedure that takes place before the OS is started. Additionally in this API all EcuM variables that need initialization are initialized. **Caution** After EcuM_Init() the EcuM is not in the running state, to achieve this state EcuM_StartupTwo() has to be called.	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is non-reentrant. ▪ called by start-up code	
Call Context	
▪ Function could be called from interrupt level or from task level	

Table 5-3 EcuM_Init

5.3.3 EcuM_StartupTwo

Prototype	
void **EcuM_StartupTwo** (void)	
Parameter	
void	none
Return code	
void	none
Functional Description	
The function implements the startup phase where the OS is already running. EcuM_AL_DriverInitTwo() is called within this function. This function should be scheduled by a task directly after StartOS() and only be called once. **Caution** The integrator has to ensure that the EcuM_StartupTwo is not interrupted by any other function or task.	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is non-reentrant.	
Call Context	
▪ Function could be called from interrupt level or from task level	

Table 5-4 EcuM_StartupTwo

5.3.4 EcuM_Shutdown

Prototype	
void **EcuM_Shutdown** (void)	
Parameter	
void	none
Return code	
void	none
Functional Description	
This function performs a reset or switches off the ECU (depending on which shutdown target is currently chosen). **Note** This function shall be called inside the OS ShutdownHook() routine. The integrator is responsible to perform this task. **Caution** The API EcuM_Shutdown must be called only for the core which is responsible for the shutdown of the ECU. If the OS ShutdownHook() is called on each core, the implementation of the hook has to take care that EcuM_Shutdown is not called on every core.	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is non-reentrant. ▪ Must only be called on the core which is responsible for the shutdown.	
Call Context	
▪ Function should be called from the ShutdownHook of the Os.	

Table 5-5 EcuM_Shutdown

5.3.5 EcuM_SelectShutdownTarget

Prototype	
Std_ReturnType **EcuM_SelectShutdownTarget** (EcuM_StateType targetState, uint8 resetSleepMode)	
Parameter	
targetState	One of these values: ▪ ECUM_STATE_OFF ▪ ECUM_STATE_SLEEP ▪ ECUM_STATE_RESET
resetSleepMode	Depending on the parameter targetState this represents a sleep mode or a reset mode.
Return code	
E_OK	success
E_NOT_OK	error
Functional Description	
This service selects a shutdown target in which the shutdown sequence should change	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is reentrant. The EcuM must be in RUN state. ▪ The ECU State Manager does not define any mechanism to resolve issues arising from parallel requests. It is rather assumed that there will be one application which is ECU specific and handles these kinds of issues.	
Call Context	
▪ Function could be called from interrupt level or from task level	

Table 5-6 EcuM_SelectShutdownTarget

5.3.6 EcuM_GetShutdownTarget

Prototype	
Std_ReturnType **EcuM_GetShutdownTarget** (EcuM_StateType *target, uint8 *resetSleepMode)	
Parameter	
target	One of these values: ▪ ECUM_STATE_OFF ▪ ECUM_STATE_SLEEP ▪ ECUM_STATE_RESET
resetSleepMode	Depending on the parameter target this represents a sleep mode or a reset mode. If the target is ECUM_STATE_OFF this parameter is 0.
Return code	
E_OK	success
E_NOT_OK	error
Functional Description	
Returns the actual chosen shutdown target.	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is reentrant.	
Call Context	
▪ Function could be called from interrupt level or from task level	

Table 5-7 EcuM_GetShutdownTarget

5.3.7 EcuM_GetLastShutdownTarget

Prototype	
Std_ReturnType **EcuM_GetLastShutdownTarget**	(EcuM_StateType *target, uint8 *resetSleepMode)
Parameter	
target	One of these values: ▪ ECUM_STATE_OFF ▪ ECUM_STATE_SLEEP ▪ ECUM_STATE_RESET
resetSleepMode	Depending on the parameter target this represents a sleep mode or a reset mode. If the target is ECUM_STATE_OFF this parameter is 0.
Return code	
E_OK	success
E_NOT_OK	error
Functional Description	
This function returns not the current shutdown target but the shutdown target set before the last reset. This function always shall return the same value until the next shutdown.	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is reentrant.	
Call Context	
▪ Function could be called from interrupt level or from task level	

Table 5-8 EcuM_GetLastShutdownTarget

5.3.8 EcuM_GetPendingWakeupEvents

Prototype	
EcuM_WakeupSourceType **EcuM_GetPendingWakeupEvents** (void)	
Parameter	
void	none
Return code	
EcuM_WakeupSourceType	Every bit set in the return value indicates a wake-up source where the validation is in progress.
Functional Description	
Returns wake-up events which have been set but not yet validated.	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is reentrant.	
Call Context	
▪ Function could be called from interrupt level or from task level	

Table 5-9 EcuM_GetPendingWakeupEvents

5.3.9 EcuM_ClearWakeupEvent

Prototype	
void **EcuM_ClearWakeupEvent** (EcuM_WakeupSourceType WakeupSource)	
Parameter	
WakeupSource	Wake-up event(s) which should be cleared
Return code	
void	none
Functional Description	
Clears the pending, validated and expired wake-up events which are passed by the parameter.	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is reentrant.	
Call Context	
▪ Function could be called from interrupt level or from task level	

Table 5-10 EcuM_ClearWakeupEvent

5.3.10 EcuM_ClearValidatedWakeupEvent

Prototype	
void **EcuM_ClearValidatedWakeupEvent** (EcuM_WakeupSourceType WakeupSource)	
Parameter	
WakeupSource	Wake-up event(s) which should be cleared
Return code	
void	none
Functional Description	
Clears only the validated wake-up events which are passed by the parameter.	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is reentrant.	
Call Context	
▪ Function could be called from interrupt level or from task level	

Table 5-11 EcuM_ClearValidatedWakeupEvent

5.3.11 EcuM_GetValidatedWakeupEvents

Prototype	
EcuM_WakeupSourceType **EcuM_GetValidatedWakeupEvents** (void)	
Parameter	
void	none
Return code	
EcuM_WakeupSourceType	ID of the wake-up source which was responsible for the wake-up
Functional Description	
This function returns wake-up event which causes the wake-up of the ECU from the previous sleep mode. **Caution** The validated Wake-up Events must be cleared before the EcuM is set to sleep. The EcuM does not clear those events by itself.	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is reentrant.	
Call Context	
▪ Function could be called from interrupt level or from task level	

Table 5-12 EcuM_GetValidatedWakeupEvents

5.3.12 EcuM_GetExpiredWakeupEvents

Prototype	
EcuM_WakeupSourceType **EcuM_GetExpiredWakeupEvents** (void)	
Parameter	
void	none
Return code	
EcuM_WakeupSourceType	ID's of wake-up sources which are expired in the validation process.
Functional Description	
Returns all events that have been set and for which validation has failed. Events which do not need validation must never be reported by this service.	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is reentrant.	
Call Context	
▪ Function could be called from interrupt level or from task level	

Table 5-13 EcuM_GetExpiredWakeupEvents

5.3.13 EcuM_GetBootTarget

Prototype	
Std_ReturnType **EcuM_GetBootTarget** (EcuM_BootTargetType *BootTarget)	
Parameter	
BootTarget	The current selected BootTarget
Return code	
E_OK	success
E_NOT_OK	error
Functional Description	
Returns the current selected boot target of the ECU.	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is reentrant.	
Call Context	
▪ Function could be called from interrupt level or from task level	

Table 5-14 EcuM_GetBootTarget

5.3.14 EcuM_SelectBootTarget

Prototype	
Std_ReturnType **EcuM_SelectBootTarget** (EcuM_BootTargetType BootTarget)	
Parameter	
BootTarget	The selected BootTarget
Return code	
E_OK	success
E_NOT_OK	error
Functional Description	
Sets the boot target for the next boot.	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is reentrant.	
Call Context	
▪ Function could be called from interrupt level or from task level	

Table 5-15 EcuM_SelectBootTarget

5.3.15 EcuM_StartCheckWakeup

Prototype	
void **EcuM_StartCheckWakeup** (EcuM_WakeupSourceType WakeupSource)	
Parameter	
WakeupSource	ID of the asynchronous wake-up source
Return code	
void	none
Functional Description	
This function starts the check wakeup timeout mechanism and marks that the wakeup source has an unapproved CheckWakeup call if applicable on given wakeup source (check wakeup timeout > 0). **Caution** This service shall only be called by EcuM_CheckWakeup(). The call is generated automatically if at least one wake-up source has a configured check wakeup timeout.	
Particularities and Limitations	
▪ This service is synchronous. ▪ This service is reentrant for the same WakeupSource. ▪ This service is always available.	
Call Context	
▪ Expected to be called in interrupt context.	

Table 5-16 EcuM_StartCheckWakeup

5.3.16 EcuM_EndCheckWakeup

Prototype	
void **EcuM_EndCheckWakeup** (EcuM_WakeupSourceType WakeupSource)	
Parameter	
WakeupSource	ID of the asynchronous wake-up source
Return code	
void	none
Functional Description	
This function stops the check wakeup timeout mechanism and removes the wakeup source from the list of unapproved CheckWakeup calls.	
Particularities and Limitations	
▪ This service is synchronous. ▪ This service is reentrant for the same WakeupSource. ▪ This service is always available.	
Call Context	
▪ Expected to be called in interrupt context.	

Table 5-17 EcuM_EndCheckWakeup

5.3.17 EcuM_GetVersionInfo

Prototype	
void **EcuM_GetVersionInfo** (Std_VersionInfoType *versioninfo)	
Parameter	
versioninfo	pointer to store the version information
Return code	
void	none
Functional Description	
Returns the version information of the ECU State Manager.	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is non-reentrant. ▪ The availability of this service depends on ECUM_VERSION_INFO_API. ▪ The versions are BCD-coded.	
Call Context	
▪ Function could be called from task level	

Table 5-18 EcuM_GetVersionInfo

5.3.18 EcuM_RequestRUN

Prototype	
Std_ReturnType **EcuM_RequestRUN (EcuM_UserType user)**	
Parameter	
user	User ID which requests the RUN state
Return code	
E_OK	Request accepted
E_NOT_OK	Request not accepted
Functional Description	
Places a RUN request for this user, Users represents normally an application. The tracking of the requests are specific for each user. **Note** RUN request will be ignored after an API call to EcuM_KillAllRUNRequest().	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is non-reentrant.	
Call Context	
▪ Function could be called from application context.	

Table 5-19 EcuM_RequestRUN

5.3.19 EcuM_ReleaseRUN

Prototype	
Std_ReturnType **EcuM_ReleaseRUN (EcuM_UserType user)**	
Parameter	
user	User ID which requests the RUN state
Return code	
E_OK	Request accepted
E_NOT_OK	Request not accepted
Functional Description	
Releases the RUN request previously done with a call to EcuM_RequestRUN().	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is non-reentrant.	
Call Context	
▪ Function could be called from application context.	

Table 5-20 EcuM_ReleaseRUN

5.3.20 EcuM_RequestPOST_RUN

Prototype	
Std_ReturnType **EcuM_RequestPOST_RUN (EcuM_UserType user)**	
Parameter	
user	User ID which requests the RUN state
Return code	
E_OK	Request accepted
E_NOT_OK	Request not accepted
Functional Description	
Places a POST_RUN request for this user, Users represents normally an application. The tracking of the requests are specific for each user. **Note** POST_RUN request will be ignored after an API call to EcuM_KillAllPostRUNRequest().	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is non-reentrant.	
Call Context	
▪ Function could be called from application context.	

Table 5-21 EcuM_RequestPOST_RUN

5.3.21 EcuM_ReleasePOST_RUN

Prototype	
Std_ReturnType **EcuM_ReleasePOST_RUN (EcuM_UserType user)**	
Parameter	
user	User ID which requests the RUN state
Return code	
E_OK	Request accepted
E_NOT_OK	Request not accepted
Functional Description	
Releases the POST_RUN request previously done with a call to EcuM_RequestPOST_RUN().	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is non-reentrant.	
Call Context	
▪ Function could be called from application context.	

Table 5-22 EcuM_ReleasePOST_RUN

5.4 Services Provided by EcuM flex

In the following the services are described which are only relevant for EcuM flex:

5.4.1 EcuM_SelectShutdownCause

Prototype		
Std_ReturnType **EcuM_SelectShutdownCause**		(EcuM_ShutdownCauseType shutdownCause)
Parameter		
shutdownCause	current shutdown cause	
Return code		
E_OK	success	
E_NOT_OK	error	
Functional Description		
Selects a new Shutdown cause for an intended shutdown.		
Particularities and Limitations		
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is reentrant.		
Call Context		
▪ Function could be called from interrupt level or from task level		

Table 5-23 EcuM_SelectShutdownCause

5.4.2 EcuM_GetShutdownCause

Prototype	
Std_ReturnType **EcuM_GetShutdownCause**	(EcuM_ShutdownCauseType *shutdownCause)
Parameter	
shutdownCause	current shutdown cause
Return code	
E_OK	success
E_NOT_OK	error
Functional Description	
Get the currently set shutdown cause.	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is reentrant.	
Call Context	
▪ Function could be called from interrupt level or from task level	

Table 5-24 EcuM_GetShutdownCause

5.4.3 EcuM_GoHalt

Prototype	
Std_ReturnType **EcuM_GoHalt** (void)	
Parameter	
void	none
Return code	
E_OK	success
E_NOT_OK	error
Functional Description	
This API is called in some modes for saving power. In this mode no more code is executed after entering that state.	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is non-reentrant. ▪ The selected shutdown target must set to ECUM_STATE_SLEEP	
Call Context	
▪ Function could be called from interrupt level or from task level	

Table 5-25 EcuM_GoHalt

5.4.4 EcuM_GoPoll

Prototype	
Std_ReturnType **EcuM_GoPoll** (void)	
Parameter	
void	none
Return code	
E_OK	success
E_NOT_OK	error
Functional Description	
This API is called in some modes for saving power. In this mode code is executed, so the EcuM poll for wake-up events. Only those Wake-up Sources with configured parameter EcuMWakeupSourcePolling are polled during that sleep mode. Other active sources can set wake-up events via interrupts.	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is non-reentrant. ▪ The selected shutdown target must set to ECUM_STATE_SLEEP	
Call Context	
▪ Function could be called from interrupt level or from task level	

Table 5-26 EcuM_GoPoll

5.4.5 EcuM_GoDown

Prototype	
Std_ReturnType **EcuM_GoDown** (uint16 caller)	
Parameter	
void	none
Return code	
Std_ReturnType	none
Functional Description	
This routine is called to initiate a shutdown or a reset. The routine checks if the caller is one of the allowed callers (if defensive behavior is configured) and then the EcuM calls ShutdownOS() and thereafter the API EcuM_Shutdown() is called by the shutdown hook.	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is non-reentrant. ▪ The selected shutdown target must set to ECUM_STATE_OFF or ECUM_STATE_RESET	
Call Context	
▪ Function could be called from interrupt level or from task level	

Table 5-27 EcuM_GoDown

5.4.6 EcuM_GoToSelectedShutdownTarget

Prototype	
Std_ReturnType **EcuM_GoToSelectedShutdownTarget**(void)	
Parameter	
void	none
Return code	
E_OK	
E_NOT_OK	
Functional Description	
This API can be called e.g. from the BswM without knowledge about the currently configured shutdown target. The EcuM decides if EcuM_GoHalt(), EcuM_GoPoll() or EcuM_GoDown() has to be called.	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is non-reentrant.	
Call Context	
▪ Function could be called from interrupt level or from task level	

Table 5-28 EcuM_GoToSelectedShutdownTarget

5.4.7 EcuM_SetRelWakeupAlarm

Prototype	
Std_ReturnType **EcuM_SetRelWakeupAlarm** (EcuM_UserType user, uint32 time)	
Parameter	
user	The user that wants to set up the wake up alarm
time	Relative time for the wake-up alarm in seconds
Return code	
E_OK	Alarm was successfully started
E_NOT_OK	No Alarm was started because of an invalid user parameter
E_EARLIER_ACTIVE	An earlier alarm was already started

Functional Description

This API can be used to set a relative wake up alarm during runtime of the ECU. For further information about this see chapter 3.14.

Caution
All wake up alarms are cleared if the ECU wakes up from a sleep phase, even if the reason for this wake up was not time triggered. The wake up alarms must be rearmed before the next sleep phase is entered.

Note
Each user can only set one wake-up alarm.

Particularities and Limitations

- Service ID: see table 'Service IDs'
- This function is synchronous.
- This function is non-reentrant.

Call Context

- Function could be called from task level

Table 5-29 EcuM_SetRelWakeupAlarm

5.4.8 EcuM_SetAbsWakeupAlarm

Prototype	
Std_ReturnType **EcuM_SetAbsWakeupAlarm** (EcuM_UserType user, uint32 time)	
Parameter	
user	The user that wants to set up the wake-up alarm
time	Absolute time for the wake-up alarm in seconds
Return code	
E_OK	Alarm was successfully started
E_NOT_OK	No Alarm was started because of an invalid user parameter
E_EARLIER_ACTIVE	An earlier alarm was already started
E_PAST	The time has already passed
Functional Description	
This API can be used to set an absolute wake up alarm during runtime of the ECU. For further information about this see chapter 3.14.	
Caution All wake up alarms are cleared if the ECU wakes up from a sleep phase, even if the reason for this wake up was not time triggered. The wake up alarms must be rearmed before the next sleep phase is entered.	
Note Each user can only set one wake-up alarm.	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is non-reentrant.	
Call Context	
▪ Function could be called from task level	

Table 5-30 EcuM_SetAbsWakeupAlarm

5.4.9 EcuM_AbortWakeupAlarm

Prototype	
Std_ReturnType **EcuM_AbortWakeupAlarm** (EcuM_UserType user)	
Parameter	
user	The user that wants to abort the wake-up alarm
Return code	
E_OK E_NOT_OK E_NOT_ACTIVE	Alarm was successfully aborted The parameter ‘user’ was not valid No alarm was active for this user
Functional Description	
This API can be used to abort a wake-up alarm which was set via the APIs EcuM_SetRelWakeupAlarm or EcuM_SetAbsWakeupAlarm.	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is non-reentrant.	
Call Context	
▪ Function could be called from task level	

Table 5-31 EcuM_AbortWakeupAlarm

5.4.10 EcuM_GetWakeupTime

Prototype	
Std_ReturnType **EcuM_GetWakeupTime** (uint32 *time)	
Parameter	
time	Absolute time of the next configured wake-up alarm in seconds
Return code	
E_OK E_NOT_OK	Time was successfully returned A null pointer was passed as parameter ‘time’
Functional Description	
Returns the time of the next active wake-up alarm which was set via the APIs EcuM_SetAbsWakeupAlarm or EcuM_SetRelWakeupAlarm. **Note** If the returned value equals ‘0xFFFFFFFF’, no wake-up alarm is currently active	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is non-reentrant.	
Call Context	
▪ Function could be called from task level	

Table 5-32 EcuM_GetWakeupTime

5.4.11 EcuM_SetClock

Prototype	
Std_ReturnType **EcuM_SetClock** (EcuM_UserType user, uint32 time)	
Parameter	
user time	The user that wants to set up the clock The absolute time value designated for the new time in seconds
Return code	
E_OK E_NOT_ALLOWED	Time was successfully modified The user was not allowed to modify the time
Functional Description	
This API can be used to modify the current time. Only special users are allowed to modify this time, e.g. for test purposes.	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is non-reentrant.	
Call Context	
▪ Function could be called from task level	

Table 5-33 EcuM_SetClock

5.4.12 EcuM_GetCurrentTime

Prototype	
Std_ReturnType **EcuM_GetCurrentTime** (uint32 *time)	
Parameter	
time	Current system time in seconds
Return code	
E_OK E_NOT_OK	Time was successfully returned A null pointer was passed as parameter ‘time’
Functional Description	
This API can be used to get the current system time.	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is non-reentrant.	
Call Context	
▪ Function could be called from task level	

Table 5-34 EcuM_GetCurrentTime

5.4.13 EcuM_SetState

Prototype	
void **EcuM_SetState**(EcuM_StateType state);	
Parameter	
state	State indicated by BswM
Return code	
void	none
Functional Description	
Requests a specific state which will be mapped to the corresponding RTE mode. This mode will be used to trigger a RTE mode switch.	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is non-reentrant.	
Call Context	
▪ Function could be called from task level	

Table 5-35 EcuM_SetState

5.5 Services Provided by EcuM fixed

In the following the services are described which are only relevant for EcuM fixed:

5.5.1 EcuM_GetState

Prototype	
Std_ReturnType **EcuM_GetState** (EcuM_StateType* state)	
Parameter	
state	Current EcuM State
Return code	
E_OK	The parameter state was a not NULL_PTR
E_NOT_OK	The parameter state was a NULL_PTR
Functional Description	
This API returns the current EcuM State. The possible EcuM States for the fixed EcuM can be seen in chapter **3.3** States of EcuM fixed.	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is reentrant.	
Call Context	
▪ Function could be called from task level	

Table 5-36 EcuM_GetState

5.5.2 EcuM_KillAllRUNRequests

Prototype	
void **EcuM_ KillAllRUNRequests (void)**	
Parameter	
void	none
Return code	
void	none
Functional Description	
Deletes all RUN requests and ensures that no new RUN request is accepted. Additionally the EcuM self-run request period will be aborted. **Note** The benefit of this function over an ECU reset is that the shutdown sequence is executed, which e.g. takes care of writing back NV memory contents.	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is non-reentrant.	
Call Context	
▪ Function could be called from application context.	

Table 5-37 EcuM_ KillAllRUNRequests

5.5.3 EcuM_KillAllPostRUNRequests

Prototype	
void **EcuM_ KillAllPostRUNRequests (void)**	
Parameter	
void	none
Return code	
void	none
Functional Description	
Deletes all POST_RUN requests and ensures that no new POST_RUN request is accepted.	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is non-reentrant.	
Call Context	
▪ Function could be called from application context.	

Table 5-38 EcuM_ KillAllPostRUNRequests

5.6 Services Used by EcuM

In the following table services provided by other components, which are used by the EcuM are listed. Also refer to chapter 2.1.

For details about prototype and functionality refer to the documentation of the providing component.

Component	API	EcuM flex	EcuM fixed
BswM	BswM_EcuM_CurrentWakeup()	■	■
	BswM_Init()	■	■
	BswM_Deinit()	■	■
	BswM_EcuM_RequestedState()	■	
	BswM_EcuM_CurrentState()	■	■
ComM	ComM_EcuM_WakeUpIndication()	■	■
	ComM_EcuM_PNCWakeUpIndication()	■	■
	ComM_GetStatus()	■	■
	ComM_GetState()		■
	ComM_CommunicationAllowed()		■
	ComM_DeInit()		■
Det	Det_ReportError()	■	■
Dem	Dem_ReportErrorStatus()	■	■
	Dem_Init()		■
	Dem_Shutdown()		■
Gpt	Gpt_EnableNotification()	■	
	Gpt_EnableWakeup()	■	
	Gpt_SetMode()	■	
	Gpt_StartTimer()	■	
Mcu	Mcu_SetMode()	■	■
	Mcu_GetResetReason()	■	■
NvM	NvM_WriteAll()		■
	NvM_CancelWriteAll()		■
	NvM_KillWriteAll()		■
OS	ShutdownOS()	■	■
	StartOS()	■	■
	GetResource()	■	■
	ReleaseResource()	■	■
	EnableAllInterrupts()	■	■
	DisableAllInterrupts()	■	■
RTE	Rte_Start()		■
	Rte_Stop()		■
	Rte_Switch_currentMode_currentMode()		■
	Rte_Feedback_currentMode_currentMode()		■
SchM	SchM_Init()	■	■
	SchM_Deinit()	■	■
	SchM_Enter_EcuM_ECUM_EXCLUSIVE_AREA_0()	■	■
	SchM_Exit_EcuM_ECUM_EXCLUSIVE_AREA_0()	■	■
	SchM_Enter_EcuM_ECUM_EXCLUSIVE_AREA_1()	■	■
	SchM_Exit_EcuM_ECUM_EXCLUSIVE_AREA_1()	■	■

Component	API	EcuM flex	EcuM fixed
	SchM_Enter_EcuM_ECUM_EXCLUSIVE_AREA_2()	■	■
	SchM_Exit_EcuM_ECUM_EXCLUSIVE_AREA_2()	■	■
	SchM_Enter_EcuM_ECUM_EXCLUSIVE_AREA_3()	■	■
	SchM_Exit_EcuM_ECUM_EXCLUSIVE_AREA_3()	■	■

Table 5-39 Services used by the EcuM

5.7 Callback Functions

This chapter describes the callback functions that are implemented by the EcuM and can be invoked by other modules. The prototypes of the callback functions are provided in the header file EcuM_Cbk.h by the EcuM.

5.7.1 EcuM_SetWakeupEvent

Prototype	
void **EcuM_SetWakeupEvent** (EcuM_WakeupSourceType WakeupSource)	
Parameter	
WakeupSource	the source of the wake-up event.
Return code	
void	none
Functional Description	
Marks a wake-up event as pending if validation is required. If no validation is required then EcuM_ValidateSetWakeupEvent will be called within this function.	
Particularities and Limitations	
▪ None	
Call Context	
▪ Function could be called from interrupt level or from task level	

Table 5-40 EcuM_SetWakeupEvent

5.7.2 EcuM_ValidateWakeupEvent

Prototype	
void **EcuM_ValidateWakeupEvent** (EcuM_WakeupSourceType WakeupSource)	
Parameter	
WakeupSource	the wake-up source which should be validated
Return code	
void	none
Functional Description	
After wake-up, the ECU State Manager will stop the process during the WAKE-UP VALIDATION state to wait for validation of the wake-up event. The validation is carried out with a call of this API service.	
Particularities and Limitations	
▪ Only ComM channels can validate Wake-up Events during ECUM_STATE_RUN.	
Call Context	
▪ Function could be called from interrupt level or from task level	

Table 5-41 EcuM_ValidateWakeupEvent

5.7.3 EcuM_AlarmCheckWakeup

Prototype	
void **EcuM_AlarmCheckWakeup**(void)	
Parameter	
void	none
Return code	
void	none
Functional Description	
This API is used to update the system clock. The API is called by the EcuM callout EcuM_CheckWakeup or directly by the GPT after one second has elapsed. If during sleep the wake-up alarm which was set via the APIs EcuM_SetAbsWakeupAlarm or EcuM_SetRelWakeupAlarm has expired, this API call will lead to a wake-up event.	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is non-reentrant.	
Call Context	
▪ Function could be called from interrupt level	

Table 5-42 EcuM_AlarmCheckWakeup

5.7.4 Callback Functions by EcuM fixed

5.7.4.1 EcuM_CB_NfyNvMJobEnd

Prototype	
void **EcuM_CB_NfyNvMJobEnd** (uint8 ServiceID, NvM_RequestResultType JobResult)	
Parameter	
ServiceID	Unique service ID of NVRAM manger service.
JobResult	[parameter is ignored by EcuM fixed]
Return code	
void	none
Functional Description	
Used to notify about the end of NVRAM jobs initiated by ECUM.	
Particularities and Limitations	
▪ Service ID: see table 'Service IDs' ▪ This function is synchronous. ▪ This function is non-reentrant.	
Call Context	
▪ Function could be called from interrupt level	

Table 5-43 EcuM_AlarmCheckWakeup

5.8 Configurable Interfaces

5.8.1 Notifications

The EcuM does not provide notifications.

5.8.2 Callout Functions

At its configurable interfaces the EcuM defines callout functions. The declarations of the callout functions are provided by the BSW module, i.e. the EcuM. It is the integrator's task to provide the corresponding function definitions. The definitions of the callouts can be adjusted to the system's needs. The EcuM callout function declarations are described in the following tables:

5.8.2.1 EcuM_ErrorHook

Prototype	
void **EcuM_ErrorHook** (Std_ReturnType reason)	
Parameter	
reason	The reason for the current call of the ErrorHook.
Return code	
void	none
Functional Description	
The ECU State Manager calls the Errorhook if the following error code occur: ▪ ECUM_E_HOOK_RAM_CHECK_FAILED In that case it is not possible to continue processing. The integrator has to take the decision how the ECU shall react in that situation.	
Particularities and Limitations	
▪ This function has to be filled with code by the integrator.	
Call Context	
▪ Expected to be called in application context.	

Table 5-44 EcuM_ErrorHook

5.8.2.2 EcuM_OnGoOffOne

Prototype	
void **EcuM_OnGoOffOne** (void)	
Parameter	
void	none
Return code	
void	none
Functional Description	
Allows the execution of additional activities in GO OFF I state.	
Particularities and Limitations	
▪ This function has to be filled with code by the integrator.	
Call Context	
▪ Invoked in task context ▪ Called right after entering ECUM_STATE_GO_OFF_ONE.	

Table 5-45 EcuM_OnGoOffOne

5.8.2.3 EcuM_OnGoOffTwo

Prototype	
void **EcuM_OnGoOffTwo** (void)	
Parameter	
void	none
Return code	
void	none
Functional Description	
Allows the execution of additional activities in GO OFF II state.	
Particularities and Limitations	
▪ This function has to be filled with code by the integrator.	
Call Context	
▪ Invoked in task context ▪ Called right after entering ECUM_STATE_GO_OFF_TWO.	

Table 5-46 EcuM_OnGoOffTwo

5.8.2.4 EcuM_AL_SwitchOff

Prototype	
void **EcuM_AL_SwitchOff** (void)	
Parameter	
void	none
Return code	
void	none
Functional Description	
This callout shall take the code for shutting off the power supply of the ECU.	
Particularities and Limitations	
▪ This function has to be filled with code by the integrator.	
Call Context	
▪ Invoked in EcuM_Shutdown(), task context	

Table 5-47 EcuM_AL_SwitchOff

5.8.2.5 EcuM_AL_Reset

Prototype	
void **EcuM_AL_Reset** (EcuM_ResetType Reset)	
Parameter	
Reset	The parameter Reset describes the ResetType (refer to 5.1) that is currently configured via EcuM_SelectShutdownTarget () (refer to 5.3.5).
Return code	
void	none
Functional Description	
This callout shall take the decision what kind of reset to be performed depending on the given Reset mode.	
Particularities and Limitations	
▪ This function has to be filled with code by the integrator.	
Call Context	
▪ Expected to be called in application context.	

Table 5-48 EcuM_AL_Reset

5.8.2.6 EcuM_AL_DriverInitZero

Prototype	
void **EcuM_AL_DriverInitZero** (void)	
Parameter	
void	none
Return code	
void	none
Functional Description	
This callout is invoked early in the PreOS Sequence during EcuM_Init().	
Particularities and Limitations	
▪ This function is filled by the configuration tool. It can be extended by the integrator by using the Userblocks.	
Call Context	
▪ Invoked in EcuM_Init(), task context	

Table 5-49 EcuM_AL_DriverInitZero

5.8.2.7 EcuM_AL_DriverInitOne

Prototype	
void **EcuM_AL_DriverInitOne** (void)	
Parameter	
void	none
Return code	
void	none
Functional Description	
This callout is invoked late in the PreOS Sequence during EcuM_Init().	
Particularities and Limitations	
▪ This function is filled by the configuration tool. It can be extended by the integrator by using the Userblocks. **Note** PostBuild data can be accessed via the global pointer EcuM_GlobalPBConfig_Ptr, example: EcuM_GlobalPBConfig_Ptr->CfgPtr_Com_Init. **Note** Variant data can be accessed via the global pointer EcuM_GlobalPCConfig_Ptr, example: EcuM_GlobalPCConfig_Ptr->CfgPtr_ComM_Init.	
Call Context	
▪ Invoked in EcuM_Init(), task context	

Table 5-50 EcuM_AL_DriverInitOne

5.8.2.8 EcuM_AL_DriverRestart

Prototype	
void **EcuM_AL_DriverRestart** (void)	
Parameter	
void	none
Return code	
void	none
Functional Description	
This callout shall provide driver initialization and other hardware related startup activities after a wake-up event from SLEEP state. This callout should be a combination of EcuM_DriverInitZero and EcuM_DriverInitOne.	
Particularities and Limitations	
▪ This function is filled by the configuration tool. It can be extended by the integrator by using the Userblocks. **Note** PostBuild data can be accessed via the global pointer EcuM_GlobalPBConfig_Ptr, example: EcuM_GlobalPBConfig_Ptr->CfgPtr_Com_Init. **Note** Variant data can be accessed via the global pointer EcuM_GlobalPCConfig_Ptr, example: EcuM_GlobalPCConfig_Ptr->CfgPtr_ComM_Init.	
Call Context	
▪ Invoked directly after the wake-up phase	

Table 5-51 EcuM_AL_DriverRestart

5.8.2.9 EcuM_AL_SetProgrammableInterrupts

Prototype	
void **EcuM_AL_SetProgrammableInterrupts** (void)	
Parameter	
void	none
Return code	
void	none
Functional Description	
On ECUs with programmable interrupt priorities, these priorities must be set before the OS is started.	
Particularities and Limitations	
▪ This function has to be filled with code by the integrator.	
Call Context	
▪ Invoked in EcuM_Init(), task context	

Table 5-52 EcuM_AL_SetProgrammableInterrupts

5.8.2.10 EcuM_McuSetMode

Prototype	
void **EcuM_McuSetMode** (Mcu_ModeType McuMode)	
Parameter	
McuMode	Mode for the upcoming sleep mode
Return code	
void	none
Functional Description	
Switches the Mcu to a power saving mode during a sleep phase.	
Particularities and Limitations	
▪ This function is filled by the configuration tool. It can be adapted by the integrator.	
Call Context	
▪ Expected to be called by EcuM_GoHalt() or EcuM_GoPoll()	

Table 5-53 EcuM_McuSetMode

5.8.2.11 EcuM_WaitForSlaveCores

Prototype	
void **EcuM_WaitForSlaveCores** (void)	
Parameter	
void	none
Return code	
void	none
Functional Description	
Is only called if EcuMSlaveCoreHandling is active. During the master core is waiting for the slave cores to be ready for the upcoming shutdown, this callout is called cyclically. In context of this callout the slave cores can be initiated to enter sleep.	
Particularities and Limitations	
▪ This function has to be filled with code by the integrator.	
Call Context	
▪ Expected to be called by EcuM_GoHalt() or EcuM_GoPoll()	

Table 5-54 EcuM_WaitForSlaveCores

5.8.2.12 EcuM_StartOS

Prototype	
void **EcuM_StartOS** (AppModeType appMode)	
Parameter	
appMode	Default OS application Mode
Return code	
void	none
Functional Description	
This callout is called at the end of EcuM_Init() to start the OS. **Note** In case of a MultiCore ECU all slave cores are started from the Master Core via the OS API StartCore() before the OS is started with a call to StartOS().	
Particularities and Limitations	
▪ This function is filled by the configuration tool. It can be adapted by the integrator.	
Call Context	
▪ Expected to be called by EcuM_Init()	

Table 5-55 EcuM_StartOS

5.8.2.13 EcuM_ShutdownOS

Prototype	
void **EcuM_ShutdownOS** (Std_ReturnType ErrCode)	
Parameter	
ErrCode	E_OK
Return code	
void	none
Functional Description	
This callout is called at the end of EcuM_GoDown() to shut down the OS via ShutdownOS(E_OK). **Note** In case of a MultiCore ECU this callout should lead to a call of ShutdownAllCores(E_OK), inside this OS API all cores are synchronized and stopped.	
Particularities and Limitations	
▪ This function is filled by the configuration tool. It can be adapted by the integrator.	
Call Context	
▪ Expected to be called by EcuM_GoDown()	

Table 5-56 EcuM_ShutdownOS

5.8.2.14 EcuM_GenerateRamHash

Prototype	
void **EcuM_GenerateRamHash** (void)	
Parameter	
void	none
Return code	
void	none
Functional Description	
This callout is intended to provide a RAM integrity test. The goal of this test is to ensure that after a long SLEEP duration, RAM contents are still consistent. The RAM check itself must be provided by the integrator.	
Particularities and Limitations	
▪ This function has to be filled with code by the integrator.	
Call Context	
▪ Invoked in task context ▪ Invoked just before setting the ECU into a sleep mode where the ECU is halted	

Table 5-57 EcuM_GenerateRamHash

5.8.2.15 EcuM_CheckRamHash

Prototype	
uint8 **EcuM_CheckRamHash** (void)	
Parameter	
void	none
Return code	
0	Integrity test failed
1…255	Integrity test passed
Functional Description	
This callout is intended to provide a RAM integrity check previously done with EcuM_GenerateRamHash().	
Particularities and Limitations	
▪ This function has to be filled with code by the integrator.	
Call Context	
▪ Invoked in task context ▪ Directly called after the wake-up of the ECU.	

Table 5-58 EcuM_CheckRamHash

5.8.2.16 EcuM_SleepActivity

Prototype	
void **EcuM_SleepActivity** (void)	
Parameter	
void	none
Return code	
void	none
Functional Description	
The ECU State Manager invokes this callout periodically during the Poll Sequence if the MCU is not halted. The EcuM polls periodically all sources that need polling and are active during the configured Sleep mode. After all sources are polled EcuM_SleepActivity is called once. **Caution** The EcuM_SleepActivity is called in a blocking loop at maximum frequency. If a lower period is preferred, the integrator has to implement this behavior.	
Particularities and Limitations	
▪ This function has to be filled with code by the integrator.	
Call Context	
▪ Expected to be called in task context.	

Table 5-59 EcuM_SleepActivity

5.8.2.17 EcuM_EnableWakeupSources

Prototype	
void **EcuM_EnableWakeupSources** (EcuM_WakeupSourceType wakeupSource)	
Parameter	
wakeupSource	Every bit set in the parameter indicates a wake-up source which should be enabled in the current sleep mode.
Return code	
void	none
Functional Description	
This callout will be invoked when the EcuM enters a sleep state. The EcuM calls this callout for every bit that is set as an active source for the current Sleep mode. The integrator has to take care to implement the necessary activities to enable the corresponding wake-up sources.	
Particularities and Limitations	
▪ This function has to be filled with code by the integrator.	
Call Context	
▪ Invoked in task context ▪ Invoked just before setting the ECU into a sleep mode	

Table 5-60 EcuM_EnableWakeupSources

5.8.2.18 EcuM_DisableWakeupSources

Prototype	
void **EcuM_DisableWakeupSources** (EcuM_WakeupSourceType wakeupSource)	
Parameter	
wakeupSource	Every bit set in the parameter indicates a wake-up source which should be enabled in the current sleep mode.
Return code	
void	none
Functional Description	
This callout will be invoked when the EcuM leaves a sleep state. The EcuM disables all wake-up sources that have occurred during the recent sleep phase. The not occurred sources remain active till the EcuM transits to ECUM_STATE_RUN after the successful validation of a wake-up source. The integrator has to take care to implement the necessary activities to disable the corresponding wake-up sources.	
Particularities and Limitations	
▪ This function has to be filled with code by the integrator.	
Call Context	
▪ Invoked in task context ▪ Called just before RUN state is entered after a sleep OR ▪ Called just before WAKEUP_VALIDATION state is entered	

Table 5-61 EcuM_DisableWakeupSources

5.8.2.19 EcuM_StartWakeupSources

Prototype	
void **EcuM_StartWakeupSources** (EcuM_WakeupSourceType wakeupSource)	
Parameter	
wakeupSource	Every bit set in the parameter indicates a wake-up source which is enabled in the current sleep mode.
Return code	
void	none
Functional Description	
The callout shall start the given wake-up source(s) so that they are ready to perform wake-up validation.	
Particularities and Limitations	
▪ This function has to be filled with code by the integrator.	
Call Context	
▪ Expected to be called in task context.	

Table 5-62 EcuM_StartWakeupSources

5.8.2.20 EcuM_StopWakeupSources

Prototype	
void **EcuM_StopWakeupSources** (EcuM_WakeupSourceType wakeupSource)	
Parameter	
wakeupSource	Every bit set in the parameter indicates a wake-up source which should be stopped after unsuccessful wake-up validation.
Return code	
void	none
Functional Description	
This callout shall stop the given wake-up source(s) after unsuccessful wake-up validation.	
Particularities and Limitations	
▪ This function has to be filled with code by the integrator.	
Call Context	
▪ Expected to be called in task context.	

Table 5-63 EcuM_StopWakeupSources

5.8.2.21 EcuM_CheckWakeup

Prototype	
void **EcuM_CheckWakeup** (EcuM_WakeupSourceType wakeupSource)	
Parameter	
wakeupSource	ID of the wake-up source to be checked
Return code	
void	none
Functional Description	
This callout shall be called by the ISR of a wake-up source to set up the PLL and check wake-up sources that may be connected to the same interrupt.	
Particularities and Limitations	
▪ This function has to be filled with code by the integrator.	
Call Context	
▪ Expected to be called in interrupt context.	

Table 5-64 EcuM_CheckWakeup

5.8.2.22 EcuM_CheckValidation

Prototype	
void **EcuM_CheckValidation** (EcuM_WakeupSourceType wakeupSource)	
Parameter	
wakeupSource	Wake-up IDs of pending wake-up events.
Return code	
void	none
Functional Description	
This callout is called by the EcuM when wake-up validation of a wake-up event is necessary. The pending wake-up event(s) are passed by the parameter in order to allow the necessary reaction depending on the wake-up source.	
Particularities and Limitations	
▪ This function has to be filled with code by the integrator.	
Call Context	
▪ Invoked in task context ▪ Called in WAKE-UP VALIDATION state	

Table 5-65 EcuM_CheckValidation

5.8.2.23 EcuM_DeterminePbConfiguration

Prototype	
EcuM_ConfigRefType **EcuM_DeterminePbConfiguration** (void)	
Parameter	
void	none
Return code	
EcuM_ConfigRefType	Pointer to the Post-Build structure
Functional Description	
In the case of Post-Build Loadable or Selectable the EcuM gets the global configuration pointer via this callout. **Note** In case of a MultiCore ECU this callout is only called on the core which starts up first.	
Particularities and Limitations	
▪ This function has to be filled with code by the integrator.	
Call Context	
▪ Expected to be called in application context.	

Table 5-66 EcuM_DeterminePbConfiguration

5.8.2.24 EcuM_BswErrorHook

Prototype	
void **EcuM_BswErrorHook** (uint16 BswModuleId, uint8 ErrorId)	
Parameter	
BswModuleId	The reporting BSW module
ErrorId	The Id of the reported error
Return code	
void	none
Functional Description	
This API can be called by Basic Software Modules to notify corrupted Postbuild configuration data. Specified ErrorIds are: ▪ ECUM_BSWERROR_NULLPTR ▪ ECUM_BSWERROR_COMPATIBILITYVERSION ▪ ECUM_BSWERROR_MAGICNUMBER	
Particularities and Limitations	
▪ The handling of an occurred error has to be specified by the integrator.	
Call Context	
▪ Invoked in task context.	

Table 5-67 EcuM_BswErrorHook

5.8.3 Callout Functions by EcuM flex

5.8.3.1 EcuM_GptStartClock

Prototype	
void **EcuM_GptStartClock** (Gpt_ChannelType GptChannel, Gpt_ModeType Mode, Gpt_ValueType Value)	
Parameter	
GptChannel	The configured Gpt channel which serves as time base for alarm clock
Mode	The Gpt normal mode
Value	The value to start the Gpt timer for second based notification / wake up
Return code	
Void	none
Functional Description	
This callout prepares the Gpt for calling the callback EcuM_AlarmCheckWakeup every second to increment the system time. **Note** This callout is only active if the EcuM alarm clock is enabled	
Particularities and Limitations	
▪ This function is filled by the configuration tool. It can be adapted by the integrator.	
Call Context	
▪ Expected to be called by EcuM_StartupTwo().	

Table 5-68 EcuM_GptStartClock

5.8.3.2 EcuM_GptSetSleep

Prototype	
void **EcuM_GptSetSleep** (Gpt_ChannelType GptChannel, Gpt_ModeType Mode)	
Parameter	
GptChannel Mode	The configured Gpt channel which serves as time base for alarm clock The Gpt sleep mode
Return code	
Void	none
Functional Description	
This callout sets the Gpt to sleep mode and enables the wake up functionality of the Gpt.  **Note** This callout is only active if the EcuM alarm clock is enabled	
Particularities and Limitations	
▪ This function is filled by the configuration tool. It can be adapted by the integrator.	
Call Context	
▪ Expected to be called by EcuM_GoHalt() or EcuM_GoPoll()	

Table 5-69 EcuM_GptSetSleep

5.8.3.3 EcuM_GptSetNormal

Prototype	
void **EcuM_GptSetNormal** (Gpt_ChannelType GptChannel, Gpt_ModeType Mode)	
Parameter	
GptChannel	The configured Gpt channel which serves as time base for alarm clock
Value	The Gpt normal mode
Return code	
Void	none
Functional Description	
This callout sets the Gpt back to normal mode after the ECU has woken up from a sleep mode. **Note** This callout is only active if the EcuM alarm clock is enabled	
Particularities and Limitations	
▪ This function is filled by the configuration tool. It can be adapted by the integrator.	
Call Context	
▪ Expected to be called by EcuM_GoHalt() or EcuM_GoPoll()	

Table 5-70 EcuM_GptSetNormal

5.8.3.4 EcuM_AL_DriverInitBswM_<ID>

Prototype	
void **EcuM_AL_DriverInitBswM_<ID>** (const EcuM_ConfigType *ConfigPtr)	
Parameter	
ConfigPtr	Pointer to global module configuration structure.
Return code	
void	none
Functional Description	
This callout can be invoked by the BswM to initialize the stack of the ECU. **Note** The ID and the count of this callout depends on the configuration. The integrator can configure multiple driver init lists of this type.	
Particularities and Limitations	
▪ This function is filled by the configuration tool. It can be extended by the integrator by using the Userblocks.	
Call Context	
▪ Invoked in BswM_Init(), task context	

Table 5-71 EcuM_AL_DriverInitBswM

5.8.4 Callout Functions by EcuM fixed

5.8.4.1 EcuM_AL_DriverInitTwo

Prototype	
void **EcuM_AL_DriverInitTwo** (void)	
Parameter	
void	none
Return code	
void	none
Functional Description	
This callout is invoked during EcuM_StartupTwo(), prior the Rte is started.	
Particularities and Limitations	
▪ This function is filled by the configuration tool. It can be extended by the integrator by using the Userblocks. **Note** PostBuild data can be accessed via the global pointer EcuM_GlobalPBConfig_Ptr, example: EcuM_GlobalPBConfig_Ptr->CfgPtr_Com_Init. **Note** Variant data can be accessed via the global pointer EcuM_GlobalPCConfig_Ptr, example: EcuM_GlobalPCConfig_Ptr->CfgPtr_ComM_Init.	
Call Context	
▪ Invoked in EcuM_StartupTwo(), task context	

Table 5-72 EcuM_AL_DriverInitTwo

5.8.4.2 EcuM_AL_DriverInitThree

Prototype	
void **EcuM_AL_DriverInitThree** (void)	
Parameter	
void	none
Return code	
void	none
Functional Description	
This callout is invoked during EcuM_StartupTwo(), after the Rte is started.	
Particularities and Limitations	
▪ This function is filled by the configuration tool. It can be extended by the integrator by using the Userblocks. **Note** PostBuild data can be accessed via the global pointer EcuM_GlobalPBConfig_Ptr, example: EcuM_GlobalPBConfig_Ptr->CfgPtr_Com_Init. **Note** Variant data can be accessed via the global pointer EcuM_GlobalPCConfig_Ptr, example: EcuM_GlobalPCConfig_Ptr->CfgPtr_ComM_Init.	
Call Context	
▪ Invoked in EcuM_StartupTwo(), task context	

Table 5-73 EcuM_AL_DriverInitThree

5.8.4.3 EcuM_OnEnterRun

Prototype	
void **EcuM_OnEnterRun** (void)	
Parameter	
void	none
Return code	
void	none
Functional Description	
Allows the execution of activities before entering RUN state.	
Particularities and Limitations	
▪ This function has to be filled with code by the integrator.	
Call Context	
▪ Invoked in task context ▪ Called just before entering RUN state.	

Table 5-74 EcuM_OnEnterRun

5.8.4.4 EcuM_OnExitRun

Prototype	
void **EcuM_OnExitRun** (void)	
Parameter	
void	none
Return code	
void	none
Functional Description	
Allows the execution of activities before leaving RUN state.	
Particularities and Limitations	
▪ This function has to be filled with code by the integrator.	
Call Context	
▪ Invoked in task context ▪ Called just before leaving RUN state.	

Table 5-75 EcuM_OnExitRun

5.8.4.5 EcuM_OnGoSleep

Prototype	
void **EcuM_OnGoSleep** (void)	
Parameter	
void	none
Return code	
void	none
Functional Description	
Allows the execution of additional activities while module is in GO SLEEP state.	
Particularities and Limitations	
▪ This function has to be filled with code by the integrator.	
Call Context	
▪ Invoked in task context ▪ Called after entering GO SLEEP state.	

Table 5-76 EcuM_OnGoSleep

5.8.4.6 EcuM_OnPrepShutdown

Prototype	
void **EcuM_OnPrepShutdown** (void)	
Parameter	
void	none
Return code	
void	none
Functional Description	
Allows the execution of additional activities in PREP SHUTDOWN state.	
Particularities and Limitations	
▪ This function has to be filled with code by the integrator.	
Call Context	
▪ Invoked in task context ▪ Called just after entering PREP SHUTDOWN state.	

Table 5-77 EcuM_OnPrepShutdown

5.8.4.7 EcuM_OnExitPostRun

Prototype	
void **EcuM_OnExitPostRun** (void)	
Parameter	
void	none
Return code	
void	none
Functional Description	
Allows the execution of activities while leaving POST RUN state.	
Particularities and Limitations	
▪ This function has to be filled with code by the integrator.	
Call Context	
▪ Invoked in task context ▪ Called while leaving POST RUN state.	

Table 5-78 EcuM_OnExitPostRun

5.8.4.8 EcuM_OnFailedNvmWriteAllJobReaction

Prototype	
void **EcuM_OnFailedNvmWriteAllJobReaction** (void)	
Parameter	
void	none
Return code	
void	none
Functional Description	
The ECU State Manager will call this function in case that a Nvm_WriteAll() job was not finished in time.	
Particularities and Limitations	
▪ This function has to be filled with code by the integrator.	
Call Context	
▪ Invoked in task context	

Table 5-79 EcuM_OnFailedNvmWriteAllJobReaction

5.8.4.9 EcuM_OnWakeupReaction

Prototype	
void **EcuM_OnWakeupReaction** (void)	
Parameter	
void	none
Return code	
void	none
Functional Description	
Allows the execution of additional activities in WAKEUP_REACTION state.	
Particularities and Limitations	
▪ This function has to be filled with code by the integrator.	
Call Context	
▪ Invoked in task context ▪ Called in ECUM_STATE_WAKEUP_REACTION state.	

Table 5-80 EcuM_OnFailedNvmWriteAllJobReaction

5.8.4.10 EcuM_OnRTEStartup

Prototype	
void **EcuM_OnRTEStartup** (void)	
Parameter	
void	none
Return code	
void	none
Functional Description	
Allows the execution of activities before starting the RTE.	
Particularities and Limitations	
▪ This function has to be filled with code by the integrator.	
Call Context	
▪ Invoked in task context ▪ Called before Rte_Start() is executed. Module state: STARTUP	

Table 5-81 EcuM_OnRTEStartup

5.9 Service Ports

5.9.1 Client Server Interface

A client server interface is related to a Provide Port at the server side and a Require Port at client side.

5.9.1.1 Provide Ports on EcuM Side

At the Provide Ports of the EcuM the API functions described in 5.2 are available as Runnable Entities. The Runnable Entities are invoked via Operations. The mapping from a SWC client call to an Operation is performed by the RTE. In this mapping the RTE adds Port Defined Argument Values to the client call of the SWC, if configured.

The following sub-chapters present the Provide Ports defined for the EcuM and the Operations defined for the Provide Ports, the API functions related to the Operations to be added by the RTE.

5.9.1.1.1 ShutdownTarget Port

Operation	API Function
SelectShutdownTarget	EcuM_SelectShutdownTarget()
GetLastShutdownTarget	EcuM_GetLastShutdownTarget()
GetShutdownTarget	EcuM_GetShutdownTarget()
SelectShutdownCause	EcuM_SelectShutdownCause()
GetShutdownCause	EcuM_GetShutdownCause()

Table 5-82 ShutdownTarget Port

5.9.1.1.2 BootTarget Port

Operation	API Function
SelectBootTarget	EcuM_SelectBootTarget()
GetBootTarget	EcuM_GetBootTarget()

Table 5-83 BootTarget Port

5.9.1.1.3 AlarmClock Port

Operation	API Function
SelectRelWakeupAlarm	EcuM_SelectRelWakeupAlarm()
SelectAbsWakeupAlarm	EcuM_SelectAbsWakeupAlarm()
AbortWakeupAlarm	EcuM_AbortWakeupAlarm()
GetCurrentTime	EcuM_GetCurrentTime()
GetWakeupTime	EcuM_GetWakeupTime()
SetClock	EcuM_SetClock()

Table 5-84 AlarmClock Port

Caution
The AlarmClock Port is only available in case of EcuM flex.

5.9.1.1.4 StateRequest Port

Operation	API Function	Port Defined Argument Value
RequestRUN	EcuM_RequestRUN()	EcuM_UserType UserId
ReleaseRUN	EcuM_ReleaseRUN()	EcuM_UserType UserId
RequestPOST_RUN	EcuM_RequestPOST_RUN()	EcuM_UserType UserId
ReleasePOST_RUN	EcuM_ReleasePOST_RUN()	EcuM_UserType UserId
GetState	EcuM_GetStateWrapper()	EcuM_UserType UserId

Table 5-85 StateRequest Port

Info
The GetState operation above is mapped to an additional API function `EcuM_GetStateWrapper()` which has to be introduced to be compliant with ASR3 Microsar EcuM. This API is not described in chapter 5.2 because the functionality is the same as `EcuM_GetState()`.

5.9.1.1.5 currentMode Port

Operation	RTE Interface	Mode Declaration Group
currentMode	`Rte_Switch_currentMode_currentMode`	STARTUP RUN POST_RUN SLEEP WAKE_SLEEP SHUTDOWN

Table 5-86 currentMode Port

Caution
The Ports CurrentMode and StateRequest are only available in case of EcuM fixed.

6 AUTOSAR Standard Compliance

6.1 Deviations

6.1.1 Deviation in the Naming of API Parameters

6.1.1.1 ResetSleepMode

The parameter “mode” has been changed to “resetSleepMode” for the following APIs:

- EcuM_GetLastShutdownTarget()
- EcuM_GetShutdownTarget()
- EcuM_SelectShutdownTarget()

6.1.1.2 TargetState

The parameter “target” has been changed to “targetState” for the following API:

- EcuM_SelectShutdownTarget()

6.1.1.3 ShutdownTarget

The parameter “shutdownTarget” has been changed to “target” for the following API:

- EcuM_GetShutdownTarget()
- EcuM_GetLastShutdownTarget()

6.1.1.4 Target (ShutdownTarget)

The parameter “target” has been changed to “shutdownCause” for the following API:

- EcuM_SelectShutdownCause()

6.1.1.5 Target (BootTarget)

The parameter “target” has been changed to “BootTarget” for the following API:

- EcuM_SelectBootTarget()
- EcuM_GetBootTarget()

6.1.1.6 Sources

The parameter “sources” has been changed to “WakeupSource” for the following API:

- EcuM_ClearWakeupEvent()
- EcuM_SetWakeupEvent()
- EcuM_ValidateWakeupEvent()

6.1.2 Starting of the Validation Timer

The validation timer is not started by calling EcuM_SetWakeupEvent(), instead it is started with the next MainFunctionCycle.

6.1.3 Multiplicity of Parameters

6.1.3.1 EcuMResetReasonRef

The parameter has been changed to optional so that not every wake-up source must have configured an Mcu reset reason.

6.1.3.2 EcuMSleepMode

The parameter has been changed to optional to allow code optimization on ECUs without the possibility to switch ECUM_STATE_OFF.

6.1.3.3 EcuMConfigConsistencyHash

The parameter has been changed to optional because it is only necessary in the case of variant post build.

6.1.3.4 Removed parameter ConfigPtr from DriverInit Lists

Removed the parameter ConfigPtr from the prototypes of the following Callouts:

- EcuM_AL_DriverInitOne()
- EcuM_AL_DriverInitTwo()
- EcuM_AL_DriverInitThree()

6.2 Additions/ Extensions

6.2.1 Additional Configuration Parameters

To fulfill the jobs of the EcuM some more parameters beyond the AUTOSAR specification are needed. The description of these parameters can be found in the BSWMD file which is part of the delivery.

The following containers are added:

- EcuMDriverInitListBswM

The following parameters are added:

- EcuMAdditionalInitCode
- EcuMGoDownRequestID
- EcuMAdditionalIncludes
- EcuMUserConfigurationFile
- EcuMCheckWakeupTimeout
- EcuMDeferredBswMNotification
- EcuMGptChannelRef
- EcuMSlaveCoreHandling
- EcuMGenModeSwitchPort
- EcuMIncludeDem
- EcuMModeSwitchRteAck
- EcuMGenModeSwitchPort
- EcuMNvmCancelWriteAllTimeout
- EcuMEnableFixBehavior
- EcuMBswCoreId
- EcuMPNCEcuMComMPNCRef

6.2.2 Buffering of Wake ups if the BswM is Not Initialized

In early phases of the ECU, wake-up events can occur and should not be missed. The EcuM detects these Wake-up Events and if the BswM is not initialized the Event will be buffered and reported to the BswM as soon as the BswM is initialized by the EcuM.

6.2.3 Buffering of Wake ups if the ComM is Not Initialized

In early phases of the ECU, wake-up events can occur and should not be missed. The EcuM checks if the ComM is active by the routine ComM_GetStatus(), if the ComM is not active in this phase the Wake-up Event is also buffered. In the EcuM_MainFunction the EcuM checks if the ComM is still uninitialized and the Wake-up Event is reported as soon as possible to the ComM.

6.2.4 Additional API EcuM_ClearValidatedWakeupEvent

The EcuM implements an API to clear only the validated wakeup events. A call of the regular API EcuM_ClearWakeupEvent leads to a clear of all events, pending wakeup events will be lost in this case.

It is necessary to clear the validated wakeup events to enter a sleep mode or shutdown the Ecu.

6.2.5 Support of Asynchronous Transceiver Handling

To support asynchronous transceiver handling a check-wakeup validation timeout was introduced for wake-up sources which cannot be checked in the context of EcuM_CheckWakeup.

6.2.6 Deferred notification of the BswM about wake-up events

To prevent that the notification via BswM_EcuM_CurrentWakeup() is executed in context of an interrupt (via EcuM_SetWakeupEvent or EcuM_ValidateWakeupEvent), the notification can be deferred to the next cycle of the EcuM_MainFunction. If the notification is executed deferred or not can be configured via the parameter EcuMDeferredBswMNotification.

6.2.7 Additional Callback EcuM_AlarmCheckWakeup

This callback is called by the Gpt every second to increment the EcuM clock which is provided by the alarm clock feature.

6.2.8 Additional API EcuM_GoToSelectedShutdownTarget

This API can be called e.g. from the BswM without knowledge about the currently configured shutdown target. The EcuM decides if EcuM_GoHalt(), EcuM_GoPoll() or EcuM_GoDown() has to be called.

6.2.9 Additional Callout EcuM_WaitForSlaveCores

This callout is only active in case of MultiCore and if the parameter EcuMSlaveCoreHandling is set true. In this case, the EcuM Master Core calls cyclically this callout. It can be used to initiate that the shutdown is also entered on the slave core.

6.2.10 Support of EcuM fixed

The EcuM supports the EcuM with fixed state machine. The EcuM fixed can be configured without EcuM flex or combined.

6.2.10.1 Shutdown Target ECUM_STATE_RESET

The shutdown target ECUM_STATE_RESET is available and the callout EcuM_AL_Reset is available, independent of EcuM_Flex configuration. The ResetMode parameter will be passed to EcuM_AL_Reset but EcuM does not check if the parameter is valid, because this is a EcuM flex parameter.

6.2.10.2 Synchronization of EcuM and RTE modes

Some transitions in the EcuM state machine lead to RTE mode switch notifications via the API Rte_Switch_currentMode_currentMode().

If the acknowledge mechanism of the EcuM is configured active, EcuM remains in its state until the RTE has acknowledged the current mode switch.

6.3 Limitations

6.3.1 Inter Module Checks

The EcuM does not check the AUTOSAR version of included external modules.

6.3.2 Recording of Shutdown Causes

The EcuM does not support the facility to record recent shutdown causes. Therefore the following two APIs are not supported:

- EcuM_GetMostRecentShutdown()
- EcuM_GetNextRecentShutdown()

6.3.3 Not Supported Configuration Parameters and Containers

Some of the specified configuration parameters are not supported. These parameters are marked with the addition “Not used” in the corresponding parameter description. The description is located within the module’s BSWMD file which is part of the delivery.
The following containers (including the parameters) are not supported in this release:

- EcuMShutdownTarget
- EcuMTTII

The following parameters are not supported in this release:

- EcuMSleepModeSuspend
- EcuMAlarmClockTimeOut
- EcuMResetLoopDetection
- EcuMIncludeDem
- EcuMIncludeDet
- EcuMNvramReadallTimeout
- EcuMIncludeNvM
- EcuMTTIIEnabled
- EcuMTTIIWakeupSourceRef

6.3.4 Wake-up Events after Reset Reason Translation are not Validated

During the initialization the EcuM get the reason for the current startup via the Mcu reset reason translation. For this translated events the wake-up validation is not performed.

6.3.5 EcuM Fixed Limitations

- NvM_ReadAll() is not started by the EcuM. This can be done by the integrator e.g. in DriverInitListTwo().
- EcuM_AL_Reset is available, independent of EcuM_Flex configuration. ResetMode parameter will be passed to EcuM_AL_Reset, but EcuM checks not if the parameter is valid.
- TTII is not supported. As a consequence, the callout EcuM_OnWakeupReaction has no parameter and no return value.

- EcuM_WakeupReactionType is not supported.
- EcuM_GetStatusOfWakeupSource is not supported.
- The following APIs are not available if EcuM flex and EcuM fixed are both configured:
  - EcuM_GoHalt
  - EcuM_GoPoll
  - EcuM_GoDown
  - EcuM_GoToSelectedShutdownTarget

7 Glossary and Abbreviations

7.1 Glossary

Term	Description
Configuration Tool	Tool for generation like DaVinci Configurator Pro
MSN	Module Short Name, the AUTOSAR short name of the module, e.g. Can, CanIf, EcuM, etc.

Table 7-1 Glossary

7.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
BSWMD	Basic Software Module Description
BswM	Basis Software Mode Manager
CAN	Controller Area Network
ComM	Communication Manager
DEM	Diagnostic Event Manager
DET	Development Error Tracer
ECU	Electronic Control Unit
EcuC	ECU configuration description
HIS	Hersteller Initiative Software
Gpt	General Purpose Timer
ICU	Input Capture Unit
ISR	Interrupt Service Routine
MCU	Microcontroller Unit
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
MSN	Module Short Name
PLL	Phase Locked Loop
RTE	Runtime Environment
SchM	Scheduling Manager
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification

Table 7-2 Abbreviations

8 Contact

Visit our website for more information on

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

www.vector.com