MICROSAR ECUM

Technical Reference

Version 2.07.02

Authors	Christian Marchl, Bethina Mausz
Status	Released

1 Document Information

1.1 History

Author	Date	Version	Remarks
Christian Marchl	2006-10-12	0.9	Initial setup
Christian Marchl	2006-12-12	1.0	Rework of review findings, first release
Christian Marchl	2007-07-30	1.1	Update chapter for AUTOSAR 2.1 release
Christian Marchl	2007-09-26	1.2	Description of EcuM_Init() function corrected
Christian Marchl	2008-04-03	1.03.00	Added AUTOSAR 3 changes.
Christian Marchl	2008-05-07	1.04.00	File renamed according to new standard name.
Christian Marchl, Heike Bischof	2008-05-07	2.00.00	Conversion to MICROSAR Technical Reference.
Christian Marchl	2008-07-23	2.00.01	ESCAN00028261: Added TTII in abbreviation table.
Christian Marchl	2008-10-22	2.00.02	Added description for PreCompile implementation variant.
Christian Marchl	2008-12-08	2.01.00	Added description for currentMode port, Solved ESCAN00031271, Added description new callout function for selection the boot target.
Christian Marchl	2009-03-20	2.02.00	ESCAN00032687: Added chapter for BSW initialization example. ESCAN00032038: Added chapter how to handle wakeup events after ShutdownOS() Added Chapters 4.7, 4.8, 4.9, 4.10
Christian Marchl	2009-05-19	2.03.00	ESCAN00034455: Update Figure 2-1 AUTOSAR architecture, ESCAN00034997: Document Restriction and deviations for multiple configuration ECUs, Ch 4.11.
Christian Marchl	2009-07-20	2.04.00	ESCAN00035749: Add

			restriction for usage of InitItems regarding modules without standard AUTOSAR naming (Ch 4.8.1) ESCAN00036018: Add chapter configuration variant. (Ch 6.1) ESCAN00036019: Add changed SWC generation behavior (Ch 5.8) ESCAN00036020: Add description for UserBlockCode generation (Ch 4.12)
Christian Marchl	2009-09-10	2.04.01	ESCAN00037245: corrected description of Compiler/Memory Abstraction in Ch 4.3.
Christian Marchl	2009-11-23	2.04.02	ESCAN00039291: Added Ch 7.3.6
Christian Marchl	2010-04-13	2.04.03	ESCAN00037828: Corrected execution sequence regarding WdgM_SetMode() (Ch 4.6) Rework of review findings: (Ch 3.6.3, 7.2.4) ESCAN00042035: updated chapter 4.6 ESCAN00039984: added chapter 7.2.5. ESCAN00042228: extended chapter 4.4.7 ESCAN00042314: extended chapter 5.6.2.1, 3.6.1. ESCAN00042505: added chapter 4.10.4.
Bethina Mausz	2010-07-30	2.04.04	ESCAN00043965: Corrected ECUM_SUBSTATE_MASK description in chapter 5.2 ESCAN00044273: Inserted more description in chapter 4.4.7
Bethina Mausz	2010-09-20	2.04.05	ESCAN00045520: inserted Mcu_PerformReset call into chapter 4.6
Bethina Mausz	2010-10-15	2.05.00	ESCAN00045755: added Nvm_GetErrorStatus call into NvM_ReadAll loop chapter

			4.8.2 ESCAN00042662: added information about CAT1 ISRs in chapter 4.7 ESCAN00046048 adapted the description to new Sleep Mode ISR handling in chapter 4.7 ESCAN00046058 modified the description for Mcu_SetMode in chapter 4.10.4
Bethina Mausz	2010-11-09	2.05.00	Update after review
Bethina Mausz	2011-01-24	2.07.00	ESCAN00046327 more description about EcuM_KillAllRUNRequests added ESCAN00045114 more description about NvM_CancelWriteAll usage added ESCAN00046158 more description about Rte_Feedback usage added Figure 2-2 update Added the description for new configuration parameters: “RTE Acknowledgment Mechanism” and “NvM_CancelWriteAll Timout”
Bethina Mausz	2011-02-09	2.07.01	Architecture Overview updated, issues from last review fixed and ESCAN00048281
Bethina Mausz	2011-06-17	2.07.02	ESCAN00051734 modifications in chapter 4.7

Table 1-1 History of the document

1.2 Reference Documents

No.	Title	Version
[1]	AUTOSAR_SWS_ECU_StateManager.pdf	V1.2.0
[2]	AUTOSAR_SWS_DET.pdf	V2.2.0
[3]	AUTOSAR_SWS_DEM.pdf	V2.2.1
[4]	AUTOSAR_BasicSoftwareModules.pdf	V1.2.0

Table 1-2 Reference documents

Please note
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

1 Document Information 2
1.1 History 2
1.2 Reference Documents 4

2 Introduction 15
2.1 Architecture Overview 16

3 Functional Description 18
3.1 Features 18
3.2 Initialization 19
3.3 Startup into wakeup validation state 19
3.4 States 20
3.5 Main Functions 21
3.5.1 Run Request Protocol 21
3.5.2 Time Triggered Increased Inoperation Protocol 22
3.5.3 Wakeup Validation Protocol 22
3.6 Error Handling 24
3.6.1 Development Error Reporting 24
3.6.1.1 Parameter Checking 26
3.6.2 Production Code Error Reporting 28
3.6.3 Vector specific error reporting 28

4 Integration 29
4.1 Scope of Delivery 29
4.1.1 Static Files 29
4.1.2 Dynamic Files 29
4.2 Include Structure 30
4.3 Compiler Abstraction and Memory Mapping 30
4.4 Dependencies on SW modules 31
4.4.1 AUTOSAR OS 31
4.4.2 MCU 31
4.4.3 DEM 32
4.4.4 DET 32
4.4.5 WDGM 32
4.4.6 COMM 32
4.4.7 NVM 32
4.4.8 RTE 33
4.4.9 SCHM 33

4.4.10 Other BSW Modules ... 33
4.5 Successive Adding of Modules ... 34
4.6 Callout execution sequences ... 34
4.7 Critical code sections ... 40
4.7.1 Wakeup Interrupt Source Handling – General Description ... 40
4.7.2 Wakeup Interrupt Source Handling – Cautions and further Information ... 41
4.8 AUTOSAR stack initialization ... 41
4.8.1 Configuration set selection ... 41
4.8.2 Initialization order ... 43
4.9 Handling of wakeup events after ShutdownOS() ... 44
4.10 Wakeup event handling and Wakeup Validation ... 44
4.10.1 Wakeup after a physical sleep mode ... 45
4.10.1.1 Use case description ... 45
4.10.1.2 Execution flow ... 45
4.10.1.3 Callout implementation examples ... 46
4.10.2 Handling of wakeup events while ECUM is in RUN state ... 47
4.10.2.1 Use case description ... 47
4.10.2.2 Execution flow ... 47
4.10.2.3 Callout implementation examples ... 47
4.10.3 Wakeup validation of communication channels (ECUM in RUN state) ... 47
4.10.3.1 Use case description ... 47
4.10.3.2 Execution flow ... 47
4.10.3.3 Callout implementation examples ... 49
4.10.4 Notes to Mcu_SetMode() ... 50
4.11 Multiple Identity ECUs ... 50
4.11.1 Restrictions when implementation variant PreCompile is used ... 50
4.11.1.1 Initialization ... 51
4.11.1.2 EcuM configuration ... 51
4.12 Generated Template Files ... 51

5 API Description ... 53
5.1 Interfaces Overview ... 53
5.2 Type Definitions ... 53
5.3 Services provided by ECUM ... 55
5.3.1 EcuM_Init ... 55
5.3.2 EcuM_Shutdown ... 56
5.3.3 EcuM_GetVersionInfo ... 57
5.3.4 EcuM_RequestRUN ... 57
5.3.5 EcuM_ReleaseRUN ... 58
5.3.6 EcuM_ComM_RequestRUN ... 58
5.3.7 EcuM_ComM_ReleaseRUN ... 59

5.3.8 EcuM_ComM_HasRequestedRUN ... 59
5.3.9 EcuM_RequestPOST_RUN ... 60
5.3.10 EcuM_ReleasePOST_RUN ... 60
5.3.11 EcuM_KillAllRUNRequests ... 61
5.3.12 EcuM_SelectShutdownTarget ... 62
5.3.13 EcuM_GetState ... 62
5.3.14 EcuM_GetShutdownTarget ... 63
5.3.15 EcuM_GetLastShutdownTarget ... 63
5.3.16 EcuM_GetPendingWakeupEvents ... 64
5.3.17 EcuM_ClearWakeupEvent ... 65
5.3.18 EcuM_GetValidatedWakeupEvents ... 65
5.3.19 EcuM_GetExpiredWakeupEvents ... 65
5.3.20 EcuM_GetStatusOfWakeupSource ... 66
5.3.21 EcuM_SelectApplicationMode ... 67
5.3.22 EcuM_GetApplicationMode ... 67
5.3.23 EcuM_SelectBootTarget ... 68
5.3.24 EcuM_GetBootTarget ... 68
5.3.25 EcuM_MainFunction ... 69
5.3.26 EcuM_StartupTwo ... 69
5.4 Services used by ECUM ... 70
5.5 Callback Functions ... 70
5.5.1 EcuM_CB_NfyNvMJobEnd ... 70
5.5.2 EcuM_SetWakeupEvent ... 71
5.5.3 EcuM_ValidateWakeupEvent ... 71
5.6 Configurable Interfaces ... 72
5.6.1 Notifications ... 72
5.6.2 Callout Functions ... 72
5.6.2.1 EcuM_AL_DriverInitZero ... 72
5.6.2.2 EcuM_AL_DriverInitOne ... 73
5.6.2.3 EcuM_AL_DriverInitTwo ... 74
5.6.2.4 EcuM_AL_DriverInitThree ... 75
5.6.2.5 EcuM_DeterminePbConfiguration ... 76
5.6.2.6 EcuM_AL_SwitchOff ... 76
5.6.2.7 EcuM_AL_DriverRestart ... 77
5.6.2.8 EcuM_GenerateRamHash ... 77
5.6.2.9 EcuM_CheckRamHash ... 78
5.6.2.10 EcuM_OnRTEStartup ... 78
5.6.2.11 EcuM_OnEnterRun ... 79
5.6.2.12 EcuM_OnExitRun ... 79
5.6.2.13 EcuM_OnExitPostRun ... 80
5.6.2.14 EcuM_OnPrepShutdown ... 80

5.6.2.15 EcuM_OnGoSleep ... 81
5.6.2.16 EcuM_OnGoOffOne ... 81
5.6.2.17 EcuM_OnGoOffTwo ... 82
5.6.2.18 EcuM_OnWakeupReaction ... 82
5.6.2.19 EcuM_EnableWakeupSources ... 83
5.6.2.20 EcuM_DisableWakeupSources ... 84
5.6.2.21 EcuM_StartWakeupSources ... 85
5.6.2.22 EcuM_StopWakeupSources ... 86
5.6.2.23 EcuM_CheckWakeup ... 86
5.6.2.24 EcuM_CheckValidation ... 87
5.6.2.25 EcuM_GeneratorCompatibilityError ... 87
5.6.2.26 EcuM_Appl_SelectBootTarget ... 88
5.6.2.27 EcuM_Appl_GetBootTarget ... 89
5.6.2.28 EcuM_McuSetMode ... 89
5.6.3 Mode switch notification functions ... 90
5.6.3.1 Appl_EcuM_currentMode_currentMode ... 90
5.6.3.2 Rte_Switch_currentMode_currentMode ... 91
5.7 Service Ports ... 91
5.7.1 Client Server Interface ... 91
5.7.1.1 Provide Ports on ECUM side ... 91
5.7.1.1.1 StateRequest Port ... 92
5.7.1.1.2 ShutdownTarget Port ... 92
5.7.1.1.3 BootTarget Port ... 92
5.7.1.1.4 ApplicationMode Port ... 92
5.7.1.2 Require Ports on ECUM side ... 93
5.7.1.2.1 currentMode Port ... 93
5.7.2 Sender Receiver Interface ... 93
5.8 Software Component Template ... 93
5.8.1 Generation by EAD ... 93
5.8.2 Generation by DaVinci Configurator ... 94
5.8.3 Import into DaVinci Developer ... 95

6 Configuration ... 96
6.1 Configuration Variants ... 96
6.2 Configuration of ECUM with DaVinci Configurator/EAD ... 96
6.2.1 Start configuration of the ECUM ... 96
6.2.2 ECUM Configuration Perspective ... 96
6.2.2.1 Node "Startup" ... 96
6.2.2.1.1 Node "Used Module Configuration" ... 96
6.2.2.1.2 Node "Mcu" ... 97
6.2.2.1.3 Node "WdgM" ... 97

6.2.2.1.4 Node "Nvm" 97
6.2.2.1.5 Node "Dem" 98
6.2.2.1.6 Node "Det" 98
6.2.2.1.7 Node "ComM" 98
6.2.2.1.8 Node "Rte" 99
6.2.2.1.9 Node "Os" 99
6.2.2.1.10 Node "SchM" 100
6.2.2.1.11 Node “Init Configuration Sets” 100
6.2.2.1.12 Nodes "EcuMDriverInitListOne", "EcuMDriverInitListTwo", “EcuMDriverInitListThree”, "EcuMDriverRestartList" 100
6.2.2.1.13 Node Additional Includes 101
6.2.2.2 Node "EcuM User" 102
6.2.2.3 Node "Time settings" 102
6.2.2.4 Node "Wakeup" 102
6.2.2.5 WdgM Modes 103
6.2.2.6 Node “Sleep Modes” 104
6.2.2.7 Child Nodes of “Sleep Modes” 104
6.2.2.8 Node “TTII” 105
6.2.2.9 Node “Shutdown” 105
6.2.3 General Settings Perspective 106
6.2.4 Module API Perspective 109
6.2.5 Module Callouts Perspective 109

7 AUTOSAR Standard Compliance 111
7.1 Deviations 111
7.1.1 Service Port Generation 111
7.1.2 Supervised Entity in WDGM 111
7.1.3 Parameter “EcuMWdgMStartupModeRef” not supported 111
7.1.4 Signature and name of DriverInitLists in configuration variant “post-build” 111
7.1.5 Enabling/Disabling wakeup sources 111
7.1.6 Execution flow in shutdown sequence 111
7.2 Additions/ Extensions 112
7.2.1 Parameter Checking 112
7.2.2 State Request service port 112
7.2.3 Wakeup validation in each state 112
7.2.4 Callout EcuM_GeneratorCompatibilityError() 112
7.2.5 Buffering of wakeup events until COMM is initialized 112
7.3 Limitations 113
7.3.1 Link-time Configuration not supported 113
7.3.2 Configuration Check via Consistency Hash not implemented 113
7.3.3 Not implemented Callouts 113

7.3.4 References in ECUC File 113
7.3.5 Not supported configuration parameters 113
7.3.6 Polling of Wakeup Sources Is Not Supported 113

8 Glossary and Abbreviations 114
8.1 Glossary 114
8.2 Abbreviations 115

9 Contact 117

Illustrations

Figure 2-1 AUTOSAR architecture ... 16
Figure 2-2 Interfaces to adjacent modules of the EcuM ... 17
Figure 3-1 State diagram ... 21
Figure 4-1 Include structure ... 30
Figure 5-1 Generate an ECUM software component template ... 94
Figure 5-2 Import a new software component into DaVinci Developer ... 95

Tables

Table 1-1 History of the document ... 4
Table 1-2 Reference documents ... 4
Table 3-1 Supported SWS features ... 18
Table 3-2 Not supported SWS features ... 19
Table 3-3 States ... 20
Table 3-4 Mapping of service IDs to services ... 25
Table 3-5 Errors reported to DET ... 26
Table 3-6 Development Error detection: Assignment of checks to services ... 27
Table 3-7 Errors reported to DEM ... 28
Table 4-1 Static files ... 29
Table 4-2 Generated files ... 29
Table 4-3 Compiler abstraction and memory mapping ... 31
Table 4-4 initialization order ... 44
Table 5-1 Type definitions ... 55
Table 5-2 EcuM_Init ... 56
Table 5-3 EcuM_Shutdown ... 56
Table 5-4 EcuM_GetVersionInfo ... 57
Table 5-5 EcuM_RequestRUN ... 57
Table 5-6 EcuM_ReleaseRUN ... 58
Table 5-7 EcuM_ComM_RequestRUN ... 58
Table 5-8 EcuM_ComM_ReleaseRUN ... 59
Table 5-9 EcuM_ComM_HasRequestedRUN ... 59
Table 5-10 EcuM_RequestPOST_RUN ... 60
Table 5-11 EcuM_ReleasePOST_RUN ... 60
Table 5-12 EcuM_KillAllRUNRequests ... 61
Table 5-13 EcuM_SelectShutdownTarget ... 62
Table 5-14 EcuM_GetState ... 63
Table 5-15 EcuM_GetShutdownTarget ... 63
Table 5-16 EcuM_GetLastShutdownTarget ... 64
Table 5-17 EcuM_GetPendingWakeupEvents ... 64
Table 5-18 EcuM_GetPendingWakeupEvents ... 65
Table 5-19 EcuM_GetValidatedWakeupEvents ... 65
Table 5-20 EcuM_GetExpiredWakeupEvents ... 66
Table 5-21 EcuM_GetStatusOfWakeupSource ... 66
Table 5-22 EcuM_SelectApplicationMode ... 67
Table 5-23 EcuM_GetApplicationMode ... 68
Table 5-24 EcuM_SelectBootTarget ... 68
Table 5-25 EcuM_GetBootTarget ... 69
Table 5-26 EcuM_MainFunction ... 69
Table 5-27 EcuM_StartupTwo ... 70
Table 5-28 EcuM_CB_NfyNvMJobEnd ... 71

Table 5-29 EcuM_SetWakeupEvent ... 71
Table 5-30 EcuM_ValidateWakeupEvent ... 72
Table 5-31 EcuM_AL_DriverInitOne ... 74
Table 5-32 EcuM_AL_DriverInitTwo ... 75
Table 5-33 EcuM_AL_DriverInitThree ... 75
Table 5-34 EcuM_DeterminePbConfiguration ... 76
Table 5-35 EcuM_AL_SwitchOff ... 77
Table 5-36 EcuM_AL_DriverRestart ... 77
Table 5-37 EcuM_GenerateRamHash ... 78
Table 5-38 EcuM_CheckRamHash ... 78
Table 5-39 EcuM_OnRTEStartup ... 79
Table 5-40 EcuM_OnEnterRun ... 79
Table 5-41 EcuM_OnExitRun ... 80
Table 5-42 EcuM_OnExitPostRun ... 80
Table 5-43 EcuM_OnPrepShutdown ... 81
Table 5-44 EcuM_OnGoSleep ... 81
Table 5-45 EcuM_OnGoOffOne ... 82
Table 5-46 EcuM_OnGoOffTwo ... 82
Table 5-47 EcuM_OnWakeupReaction ... 83
Table 5-48 EcuM_EnableWakeupSources ... 84
Table 5-49 EcuM_DisableWakeupSources ... 85
Table 5-50 EcuM_StartWakeupSources ... 85
Table 5-51 EcuM_StopWakeupSources ... 86
Table 5-52 EcuM_CheckWakeup ... 86
Table 5-53 EcuM_CheckValidation ... 87
Table 5-54 EcuM_GeneratorCompatibilityError ... 88
Table 5-55 EcuM_Appl_SelectBootTarget ... 89
Table 5-56 EcuM_Appl_GetBootTarget ... 89
Table 5-57 EcuM_McuSetMode ... 90
Table 5-58 Appl_EcuM_currentMode_currentMode ... 90
Table 5-59 Rte_Switch_currentMode_currentMode ... 91
Table 5-60 StateRequest Port ... 92
Table 5-61 ShutdownTarget Port ... 92
Table 5-62 BootTarget Port ... 92
Table 5-63 ApplicationMode Port ... 93
Table 5-64 currentMode Port ... 93
Table 6-1 Node "Mcu" ... 97
Table 6-2 Node "WdgM" ... 97
Table 6-3 Node "Nvm" ... 97
Table 6-4 Node "Dem" ... 98
Table 6-5 Node "Det" ... 98
Table 6-6 Node "ComM" ... 99
Table 6-7 Node "Rte" ... 99
Table 6-8 Node "Os" ... 99
Table 6-9 Node "SchM" ... 100
Table 6-10 Node “Init Configuration Sets” ... 100
Table 6-11 Nodes “EcuMDriverInitListOne”, “EcuMDriverInitListTwo”, “EcuMDriverInitListThree”, “EcuMDriverRestartList” ... 101
Table 6-12 Node "Additional Includes" ... 101
Table 6-13 Node "EcuM User" ... 102
Table 6-14 Node "Time settings" ... 102
Table 6-15 Node "Wakeup Sources" ... 103
Table 6-16 Node “WdgM Modes” ... 104
Table 6-17 Child Nodes of “Sleep Modes” ... 104

Table 6-18 Node “TTII” 105
Table 6-19 Node “Shutdown” 106
Table 6-20 General Settings 109
Table 6-21 Module API 109
Table 6-22 Module Callouts 110
Table 8-1 Glossary 115
Table 8-2 Abbreviations 116

2 Introduction

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

Supported AUTOSAR Release:	3	
Supported Configuration Variants:	post-build, pre-compile	
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 3.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 all BSW (Basis Software) modules in the startup phase of the microcontroller including OS and RTE
- Preparation of the microcontroller for a sleep phase and the following wakeup
- Performing an ordered shutdown of the ECU

To fulfill this tasks the ECUM offers several mechanism which will be described later in this document, e.g.:

- RUN Request Protocol
- Time Triggered Increased Inoperation Protocol (TTII)
- Wakeup validation protocol

2.1 Architecture Overview

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

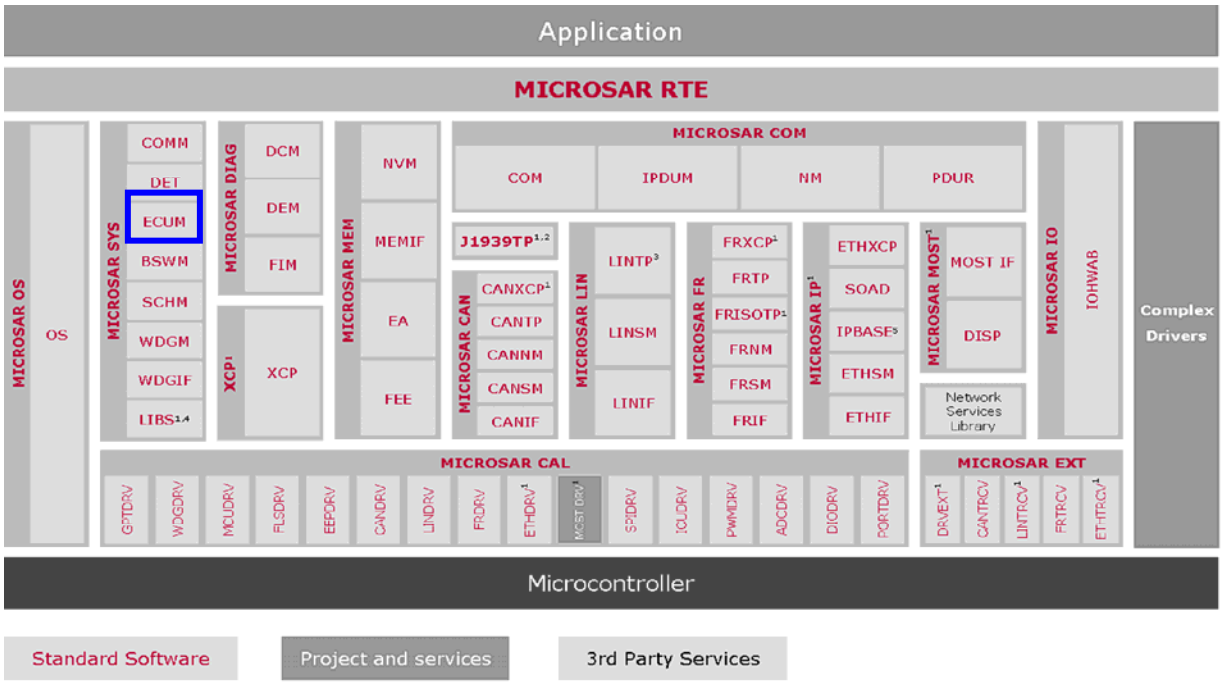

Figure 2-1 AUTOSAR architecture

The next figure shows the interfaces to adjacent modules of the ECUM. These interfaces are described in chapter 5.

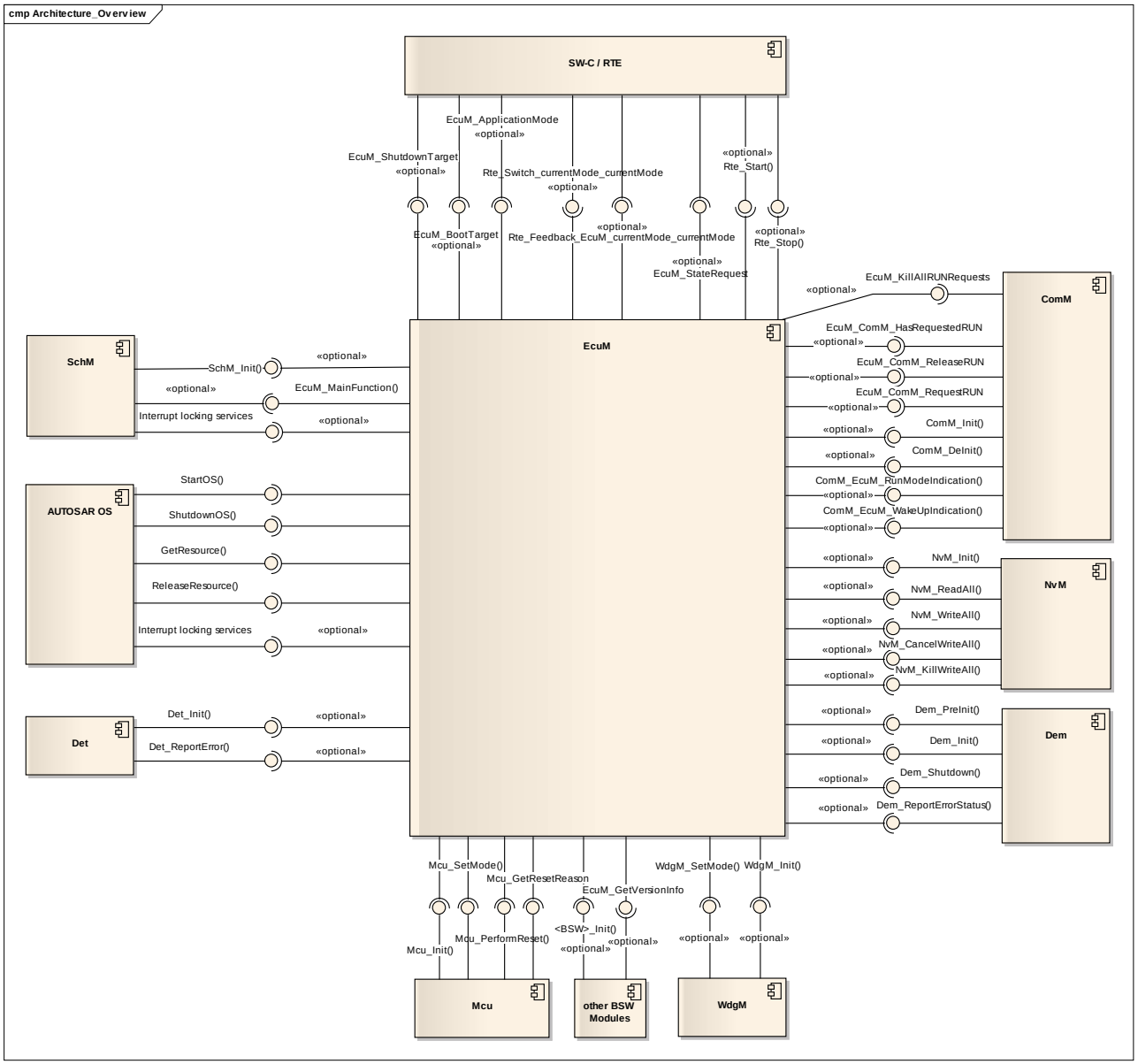

Figure 2-2 Interfaces to adjacent modules of the EcuM

Applications do not access the services of the BSW modules directly. They use the service ports provided by the BSW modules via the RTE. The service ports provided by the ECUM are listed in chapter 5.7 and are defined in [1].

3 Functional Description

3.1 Features

The features listed in this chapter cover the complete functionality specified in [1].

The "supported" and "not supported" features are presented in the following two tables. For further information of not supported features also see chapter 7.

The following features described in [1] are supported:

Feature
Support of Run Request Protocol.
Support of time triggered increased inoperation protocol (TTII).
Configuration of different wakeup sources.
Configuration of EcuM users.
Configuration of software component name.
Configurable startup sequence of the complete BSW stack.
Possibility to add additional initialization code into the initialization lists.
Notification of the Communication Manager every time when run state is entered.
Notification of the Communication Manager if a wakeup event occurs on communication channels.
Assignment of communication channels to wakeup sources.
Configuration of different sleep modes.
Selection of different shutdown targets.
Configurable mechanism to protect critical sections is supported (e.g. by OS functions, by SCHM, assembler statements, etc.).
Support of different kinds of parameter checks which can be switched on or off.
Generation of the SW-C description file needed for the generation of the RTE.
Additional port interface operation GetState provided.
Service Port: EcuM_StateRequest
Service Port: EcuM_CurrentMode
Service Port: EcuM_ShutdownTarget
Service Port: EcuM_BootTarget
Service Port: EcuM_ApplicationMode

Table 3-1 Supported SWS features

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

Feature
Callout EcuMDriverInitListZero is not in post-build structure.

Feature
Callout `EcuM_SleepActivity()` is not supported.
Callout `EcuM_Errorhook()` is not supported.
Consistency hash checking according to AUTOSAR specification is not supported.
Handling of ECUM as supervised entity of WDGM is not supported.

Table 3-2 Not supported SWS features

3.2 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.

The first part will be performed by the function `EcuM_Init()`. 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()`.

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. Within this function the DriverInitLists “EcuMDriverInitListTwo” and “EcuMDriverInitListThree” are executed. When `EcuM_StartupTwo()` is left and the reset reason which caused the startup needs no validation the ECU is in RUN state.

3.3 Startup into wakeup validation state

The startup procedure of the ECUM provides the possibility to enter wakeup validation state out of the startup state (refer to chapter 3.4) To use this feature some background information is necessary which is described in the following chapters.

At startup time when `EcuM_Init()` is executed the ECUM queries the MCU module which reset reason was responsible for that startup. These reset reasons are mapped to wakeup sources. The mapping process has to be done by the user via the parameter EcuMResetReason (6.2.2.4 Node "Wakeup", GUI element “Mcu reset reason”). With this mapping the ECUM has usually a valid reset reason or wakeup source respectively and therefore the transitions will be from STARTUP to RUN.

Caution
If a MCU reset reason is mapped to one of the preconfigured wakeup sources (refer to 5.2) the ECUM will transit always to RUN state if the user code, implemented in one of the DriverInitLists, does not change this behavior.

This behavior is probably not what is expected for ECU layouts which use a CAN transceiver with integrated power control and where wakeup events on the according bus must be validated. In that cases the user must change the default behavior with use of the API functions `EcuM_GetValidatedWakeupEvents()`, `EcuM_ClearWakeupEvent()`, `EcuM_GetPendingWakeupEvents()`, etc. by clearing the wakeup source from the reset reason mapping if a pending wakeup event is on the hand.

3.4 States

Module State	Activities	Point in Time
ECUM_STATE_STARTUP_ONE	Initialization of driver modules which need no OS. Drivers could be initialized by configuration (DriverInitOne)	Executed by `EcuM_Init()`.
ECUM_STATE_STARTUP_TWO	Initialization of driver or BSW modules which need a running OS	Executed by `EcuM_StartupTwo()`.
ECUM_STATE_APP_RUN	Check for RUN requests, ECUM performs a self run request when entering this state	Normal operation of the ECU.
ECUM_STATE_APP_POST_ RUN	Check for POST RUN requests	After RUN State, allows the SW-C to do some preparation for the following sleep mode.
ECUM_STATE_PREP_ SHUTDOWN	Shutdown DEM.	All POST RUN request are released.
ECUM_STATE_GO_OFF_ONE	Write back non-volatile memory, prepare system for the following shutdown or reset	
ECUM_STATE_GO_SLEEP	write back non-volatile memory contents	
ECUM_STATE_SLEEP	ECU is in physically sleep mode	
ECUM_STATE_WAKEUP_ONE	Set wakeup sources in normal mode in callout `EcuM_AL_RestartDrivers()`	
ECUM_STATE_WAKEUP_ VALIDATION	Wait for the validation of a wakeup event	A wakeup event occurred.
ECUM_STATE_WAKEUP_ WAKESLEEP	Result of `WAKEUP_REACTION` state was none of TTII or RUN.	
ECUM_STATE_WAKEUP_ REACTION	Computes the reaction of the ECU after a valid wakeup (TTII, switch to RUN state)	
ECUM_STATE_TTII	Execute TTII protocol	
ECUM_STATE_WAKEUP_TWO	Initialize DEM after wakeup phase.	
ECUM_STATE_GO_OFF_TWO	Activities implemented in API `EcuM_Shutdown()`.	Executed within the `ShutdownHook()`.

Table 3-3 States

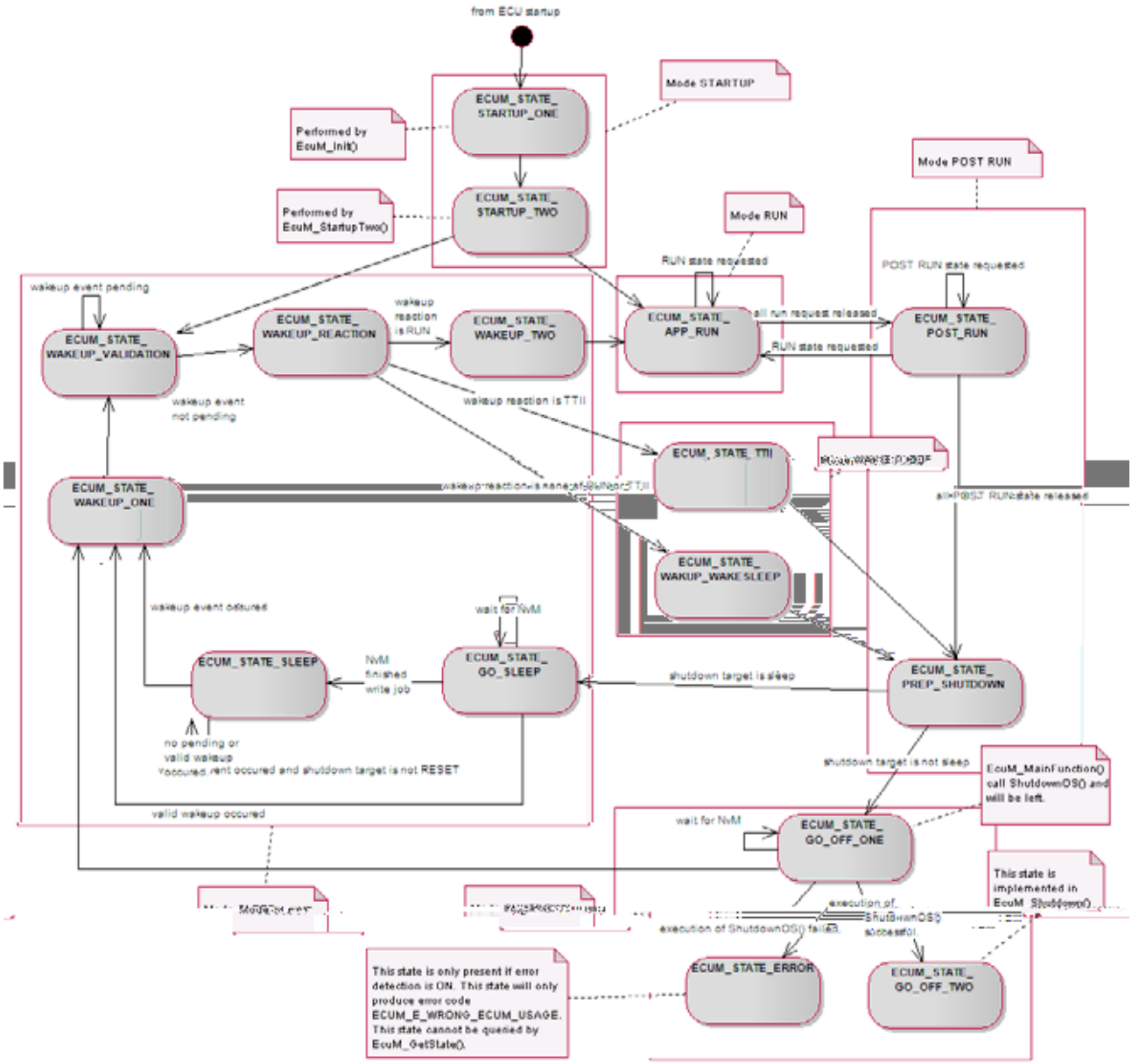

Figure 3-1 State diagram

3.5 Main Functions

3.5.1 Run Request Protocol

The run request protocol is a mechanism where applications or Software Components (SW-C) have to request RUN state explicitly. 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 decides to initiate the shutdown process of the ECU.

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, which means that 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.

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 run requests.

Info
In production mode (EcuMDevErrorDetect equals to Off) the caller must assure that the matching function pairs (requesting/releasing RUN) are called in the expected order. I.e. RUN shall be requested only once per user until the RUN is released and release RUN must also be executed once to release a previously requested RUN state to prevent unexpected behavior of the ECUM.

3.5.2 Time Triggered Increased Inoperation Protocol

This mechanism provides a way to enter a step by step decreasing power consumption mode. To use the TTII feature the ECU has to fulfill the following prerequisite:

- The ECU must support several sleep modes where each sleep mode presents a different level of low-power consumption.
- The TTII is timer based therefore the available timers on the ECU must be wakeup capable.
- A timer has to be configured as wakeup source for TTII

3.5.3 Wakeup Validation Protocol

The wakeup validation protocol provides a standardized way to recognize valid controller wakeup after a sleep phase.

Info
When the ECUM transits to SLEEP state the service `NvM_WriteAll()` is executed before the controller is set into sleep. If a subsequent wakeup validation fails and the ECUM transits into SLEEP state again, the service `NvM_WriteAll()` will not be executed again.

Caution
The validation of wakeup events is not done by the ECU State Manager. Only the management of the validation is done by ECUM. This means that the `EcuM_SetWakeupEvent()` must be called when a wakeup event was detected. If the wakeup event needs validation the module which performs this validation has to call

	`EcuM_ValidateWakeupEvent()` to indicate the ECU State Manager about a valid wakeup event and to bring the ECU into RUN state.

Example

- ECU is in sleep mode
- Wakeup source needs to be validated

1. Wakeup event occurs, the driver or the validating module calls `EcuM_CheckWakeup(wakeupsourceID)`
2. In this callout the user/integrator of the module must implement a code sequence which performs the correct action depending on the wakeup event ID. For example, if the wake event ID belongs to a CAN channel the code must execute the `CanIf_CheckWakeup()` API function which should call the `EcuM_SetWakeupEvent()`.
3. The ECUM checks whether this wakeup source needs validation => no validation required, ECUM marks this wakeup event as valid
4. ECUM proceed to RUN state

Example

- ECU is in sleep mode
- Wakeup source needs validation

1. Wakeup event occurs, the driver or the validation module calls `EcuM_CheckWakeup(wakeupSourceID)`
2. The implementation of the callout must call the appropriate service to check the wakeup event by “asking” the corresponding driver.
3. The ECUM checks whether this wakeup event needs validation => validation required, EcuM starts timeout which limits the duration of wakeup validation (if the timeout expires the EcuM sets the ECU in sleep mode again)
4. The ECUM executes the callout EcuM_StartWakeupSources(), where activities must be implemented to start BSW modules for the wakeup validation, e.g. setting the CAN transceiver in normal mode.
5. The ECUM executes the callout EcuM_CheckValidation(), where activities may be implemented to perform the wakeup validation, e.g. querying the CANIF by executing CanIf_CheckValidation().
6. If the validation was successful the validating module has to call `EcuM_ValidateWakeupEvent(wakeupSourceID)`
7. If the validation was not successful the ECUM executes the callout

	EcuM_StopWakeupSources(), where activities must be implemented to stop BSW modules which where necessary for the wakeup validation, e.g. setting the CAN transceiver in sleep mode. 8. The ECUM recognizes an valid wakeup 9. ECUM proceeds to RUN state

Info
The wakeup validation protocol applies also for communication channels while the ECUM is in RUN state.

3.6 Error Handling

3.6.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()`as specified in [2], if development error reporting is enabled (i.e. pre-compile parameter `ECUM_DEV_ERROR_DETECT == STD_ON`).

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

The reported ECUM ID can be found in chapter 2.

The reported service IDs identify the services which are described in 5.3. 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()
0x0E	EcuM_ComM_RequestRUN()

Service ID	Service
0x0F	EcuM_SelectApplicationMode()
0x10	EcuM_ComM_ReleaseRUN()
0x11	EcuM_GetApplicationMode()
0x12	EcuM_SelectBootTarget()
0x13	EcuM_GetBootTarget()
0x14	EcuM_ValidateWakeupEvent()
0x15	EcuM_GetValidatedWakeupEvents()
0x16	EcuM_ClearWakeupEvent()
0x17	EcuM_GetStatusOfWakeupSource()
0x18	EcuM_MainFunction()
0x19	EcuM_GetExpiredWakeupEvents()
0x1A	EcuM_StartupTwo()
0x1B	EcuM_ComM_HasRequestedRUN()
0x65	EcuM_CB_NfyNvMJobEnd()
0xCA	EcuM_GeneratorCompatibilityError()
0xCB	EcuM_AL_DriverInitZero()

Table 3-4 Mapping of service IDs to services

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

Error Code		Description
0x10	ECUM_E_NOT_INITED	API service used in wrong context (without module initialization)
0x11	ECUM_E_SERVICE_DISABLED	Error code defined by AUTOSAR SWS (see [1]) but not used in this implementation.
0x12	ECUM_E_NULL_POINTER	API service called where pointer parameter is equal to NULL_PTR
0x13	ECUM_E_INVALID_PAR	API service called with wrong parameter
0x14	ECUM_E_MULTIPLE_RUN_REQUESTS	Multiple request of RUN or POST RUN state by the same user ID
0x15	ECUM_E_MISMATCHED_RUN_RELEASE	Run release with an user ID without a matching request with the same Id before
0x16	ECUM_E_STATE_PAR_OUT_OF_RANGE	API services EcuM_SelectShutdownTarget() or EcuM_GetShutdownTarget() called where parameter is not in expected range
0x17	ECUM_E_UNKNOWN_WAKEUP_SOURCE	Wakeup source ID is not known by ECU State Manager
0x18	ECUM_E_PARAM_VINFO	A null pointer has been given as parameter to the API service EcuM_GetVersionInfo().
0x30	ECUM_E_OS_NOT_STARTED_SUCCESFULLY	This error will be reported if the invocation of StartOS() at the end of EcuM_Init() was not successful (this means EcuM_Init() returns.
0x31	ECUM_E_WRONG_ECUM_USAGE	This error will be reported if setting the ECU in OFF mode or performing a reset does not lead in the expected behaviour.

Error Code		Description
0x32	ECUM_E_WAKEUP_VALIDATION_PROT_ERROR	This error will be reported if EcuM_ValidateWakeupEvent() was invoked without calling EcuM_SetWakeupEvent().
0x33	ECUM_E_OFF_STATE_EXPECTED	This error will be reported if the callout EcuM_AL_SwitchOff() does not switch off the ECU. This means EcuM_AL_SwitchOff() returns.
0x34	ECUM_E_MODULE_NOT_IN_RUN_STATE	This error will be reported if EcuM_SelectShutdownTarget() is called while the ECUM is not in RUN state.
0x35	ECUM_E_GENERATOR_COMPATIBILITY	This error will be reported if the ECUM or another module executes the EcuM_GeneratorCompatibilityError() callout. This error code is additionally to AUTOSAR specification.
0x36	ECUM_E_EXTENDED_LIB_VERSION_CHECK_FAILED	This error will be reported if the delivery integrity check function VLibVersionCheck() fails. This error code is not specified by AUTOASR. It is an additional one used for error reporting in library checks.

Table 3-5 Errors reported to DET

3.6.1.1 Parameter Checking

AUTOSAR requires that API functions check the validity of their parameters. The checks in Table 3-6 are internal parameter checks of the API functions. These checks are for development error reporting and can be en-/disabled separately. The configuration of en-/disabling the checks is described in chapter 6.2.3. En-/disabling of single checks are an addition to the AUTOSAR standard which requires to en-/disable the complete parameter checking via the parameter ECUM_DEV_ERROR_DETECT.

The following table shows which parameter checks are performed on which services:

Service \ Check	ECUM_E_GENERATOR_COMPATIBILITY	ECUM_E_NOT_INITED	ECUM_E_PARAM_VINFO	ECUM_E_STATE_PAR_OUT_OF_RANGE	ECUM_E_NULL_POINTER	ECUM_E_INVALID_PAR	ECUM_E_UNKNOWN_ WAKEUP_SOURCE	ECUM_E_MULTIPLE_RUN_ REQUESTS	ECUM_E_MISMATCHED_RUN_RELEASE	ECUM_E_OS_NOT_STARTED_SUCCESFULLY	ECUM_E_WRONG_ECUM_USAGE	ECUM_E_WAKEUP_VALIDATION_PROT_ERROR	ECUM_E_OFF_STATE_EXPECTED	ECUM_E_MODULE_NOT_IN_RUN_STATE
EcuM_Init					■					■				
EcuM_Shutdown		■									■		■	

Service \ Check	ECUM_E_GENERATOR_COMPATIBILITY	ECUM_E_NOT_INITED	ECUM_E_PARAM_VINFO	ECUM_E_STATE_PAR_OUT_OF_RANGE	ECUM_E_NULL_POINTER	ECUM_E_INVALID_PAR	ECUM_E_UNKNOWN_ WAKEUP_SOURCE	ECUM_E_MULTIPLE_RUN_ REQUESTS	ECUM_E_MISMATCHED_RUN_RELEASE	ECUM_E_OS_NOT_STARTED_SUCCESFULLY	ECUM_E_WRONG_ECUM_USAGE	ECUM_E_WAKEUP_VALIDATION_PROT_ERROR	ECUM_E_OFF_STATE_EXPECTED	ECUM_E_MODULE_NOT_IN_RUN_STATE
EcuM_GetVersionInfo			■											
EcuM_RequestRUN		■				■		■						
EcuM_ReleaseRUN		■				■			■					
EcuM_ComM_RequestRUN		■				■		■						
EcuM_ComM_ReleaseRUN		■				■			■					
EcuM_ComM_HasRequestedRUN		■				■								
EcuM_RequestPOST_RUN		■				■								
EcuM_ReleasePOST_RUN		■				■								
EcuM_KillAllRUNRequests		■												
EcuM_SelectShutdownTarget		■		■		■								■
EcuM_GetState		■												
EcuM_GetShutdownTarget		■			■									
EcuM_GetLastShutdownTarget		■			■									
EcuM_SetWakeupEvent		■					■							
EcuM_GetPendingWakeupEvents		■												
EcuM_ClearWakeupEvent		■					■							
EcuM_ValidateWakeupEvent		■					■					■		
EcuM_GetValidatedWakeupEvents		■					■							
EcuM_GetExpiredWakeupEvents		■												
EcuM_GetStatusOfWakeupSource		■												
EcuM_SelectApplicationMode		■												
EcuM_GetApplicationMode		■			■									
EcuM_SelectBootTarget						■								
EcuM_MainFunction		■									■			
EcuM_StatupTwo		■												
EcuM_CB_NvyNvMJobEnd		■												
EcuM_GeneratorCompatibility-Error()	■													

Table 3-6 Development Error detection: Assignment of checks to services

3.6.2 Production Code Error Reporting

By default, production code related errors are reported to the DEM using the service `Dem_ReportErrorStatus()` as specified in [3], if production error reporting is enabled (i.e. pre-compile parameter `ECUM_PROD_ERROR_DETECT == STD_ON`).

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	RAM Hash check after a sleep phase failed
ECUM_E_ALL_RUN_REQUESTS_KILLED	API service `EcuM_KillAllRUNRequest()` was called

Table 3-7 Errors reported to DEM

Info
The production error codes listed above must be defined by the DEM and are included into the ECU State Manager by the file `Dem.h` (by default).

3.6.3 Vector specific error reporting

The ECUM module provides an optional consistency check that can detect incompatible configuration data. Eventual incompatibilities are detected during `EcuM_Init()` and are reported by the API `EcuM_GeneratorCompatibilityError()`. After calling `EcuM_GeneratorCompatibilityError()` the ECUM module will stop the initialization causing the ECUM to remain uninitialized.

The following check is provided:

- Generator versions are compatible to the implementation and compatible to the versions of previous configuration phases.

4 Integration

This chapter gives necessary information for the integration of the MICROSAR 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 chapter's 4.1.1 and 4.1.2:

4.1.1 Static Files

File Name	Description
EcuM.h	Declares the interface of the MICROSAR module ECUM.
EcuM.c	Contains the implementation of the interfaces of the ECUM.
EcuM.lib	This is the library file of the ECUM (optional).

Table 4-1 Static files

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool.

File Name	Description
EcuM_Cfg.h	Contains the static configuration part of this module.
EcuM_Callout_Stubs.c	Template for the callout code which has to be filled by the integrator.
EcuM_Generated_Types.h	Contains all provided types of the ECUM.
EcuM_Cbk.h	Contains the prototypes of the provided callback and callout functions.
EcuM_PBcfg.c	Contains the post build configurations.
EcuM_PrivateCfg.h	Contains configuration data which is only relevant for the ECUM implementation. This file must be only included in the ECUM implementation files.

Table 4-2 Generated files

4.2 Include Structure

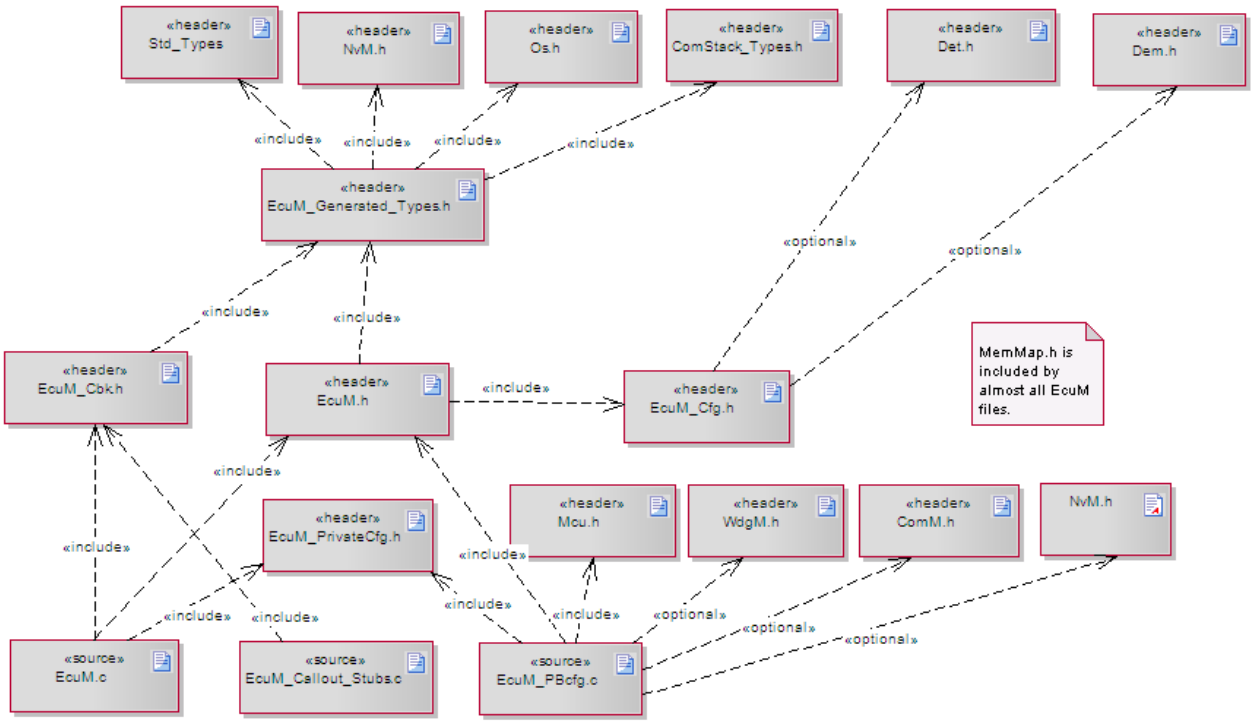

Figure 4-1 Include structure

Info

- A BSW module which uses services for wakeup event reporting shall include `EcuM_Cbk.h`
- Other BSW module shall include `EcuM.h`
- Software Components shall include the generated file by Rte to use the services of the ECUM (e.g. `Rte_EcuM.h`)

4.3 Compiler Abstraction and Memory Mapping

The objects (e.g. variables, functions, constants) are declared by compiler independent definitions – the compiler abstraction definitions. Each compiler abstraction definition is assigned to a memory section.

The following table contains the memory section names and the compiler abstraction definitions defined for the ECUM and illustrate their assignment among each other.

Compiler Abstraction Definitions / Memory Mapping Sections	ECUM_CODE	ECUM_API_CODE	ECUM_CODE_BOOT_TARGET	ECUM_APPL_DATA	ECUM_APPL_CONFIG	ECUM_CONST	ECUM_VAR_BOOT	ECUM_VAR_NOINIT	ECUM_VAR
ECUM_START_SEC_CODE	■	■							
ECUM_START_SEC_CODE_SET_BOOT_TARGET			■						
ECUM_START_SEC_CODE_GET_BOOT_TARGET			■				■		
ECUM_START_SEC_VAR_NOINIT_UNSPECIFIED					■			■	
ECUM_START_SEC_VAR_NOINIT_8BIT								■	
ECUM_START_SEC_VAR_NOINIT_16BIT								■	
ECUM_START_SEC_VAR_UNSPECIFIED									■
ECUM_START_SEC_CONST_UNSPECIFIED						■			
ECUM_START_SEC_CONST_PBCFG						■			
ECUM_START_SEC_CONST_8BIT						■			
Application buffer memory section - access from ECUM into application memory				■					

Table 4-3 Compiler abstraction and memory mapping

4.4 Dependencies on SW modules

4.4.1 AUTOSAR OS

This module depends on the AUTOSAR OS. It starts and performs the shutdown of OS. The ECUM needs a valid reference within the EcuC file to the configured application modes.

The usage cannot be switched off.

4.4.2 MCU

This module depends on the MCU driver. The ECUM uses the mode setting functionality and the service to perform an ECU reset. If a sleep mode is configured within the ECUM a valid configuration reference to MCU mode settings must exist within the EcuC.

This usage cannot be switched off.

4.4.3 DEM

The ECUM depends on the DEM. The ECUM performs the initialization and reports production errors to the DEM.

Its initialization can be enabled or disabled by the switch “Use DEM”. The error reporting can be en-/disabled by switching the configuration switch “Enable Production Error Reporting” on.

4.4.4 DET

The ECUM depends on the DET. The ECUM performs the initialization and reports development errors for diagnostic purposes.

Its initialization can be enabled or disabled by the configuration switch “Use DET”. The error reporting can be en-/disabled by switching the configuration switch “Enable Development Error Detection”.

4.4.5 WDGM

This module depends on the WDGM. The ECUM performs the initialization and via callouts it might be necessary to interact with the WDGM while entering sleep mode. Furthermore the ECUM sets the mode of the WDGM in different ECU states.

Its usage can be enabled or disabled by the switch “Use Watchdog Manager”.

4.4.6 COMM

This module depends on the COMM. The ECUM performs the initialization and manages the wakeup of communication channels. Additional it indicates to this module when RUN state is reached.

Its usage can be en-/disabled by the switch “Use ComM”.

4.4.7 NVM

The module depends on the NVM. The ECUM performs the initialization and calls `NvM_ReadAll()` and `NvM_WriteAll()` in the start up respectively in the shutdown phase.

Caution

Note that the `NvM_ReadAll()`, `NvM_Init()` and the triggering of the memory stack has to be implemented manually in one of the driver init lists. Refer to chapter 4.8.2 for an example of how to initialize the memory stack.

Its usage can be enabled or disabled by the switch “Use NVRAM Manger”. This switch disables the execution of `NvM_WriteAll()` by ECUM module but does not prevent the user to add module initialization in one of the driver init list.

If `NvM_WriteAll()` is enabled, the user is able to configure a timeout for `NvM_CancelWriteAll()`. This timeout parameter specifies the time which EcuM waits for before it calls the `NvM_KillWriteAll()` function (if available) or switches into ECU

RUN state. With an adequate value for this timeout parameter, the synchronization between NvM and EcuM is guaranteed.

Caution

NvM_CancelWriteAll Usage:

Note that a wait mechanism until NvM is finished has to be implemented manually in case `NvM_CancelWriteAll()` is called.

This caution is only valid for **EcuMS.c SW version number** smaller or equal to **05.03.00**.

In case `NvM_CancelWriteAll()` is called by EcuM because of incoming wakeup event during `NvM_WriteAll` processing, the NvM component do not have to cancel destructive the currently running `NvM_WriteAll` job. Thus the cancel mechanism can take some time.

The EcuM does not wait until the NvM finishes, instead it transits immediately into ECU RUN state.

To avoid this, a wait mechanism for NvM shall be implemented in `EcuM_DriverRestartList()`.

4.4.8 RTE

The module depends on the RTE.

The ECUM performs the start and stop of the RTE. Its usage can be enabled or disabled by the switch “Use Rte”.

The ECUM ensures the synchronization to RTE at each important mode switch.

In case the ECUM leaves the RUN state or enters SLEEP/OFF state it waits for the feedback from RTE before the mode switch is performed. This mechanism is activated by enabling the mode switch notification and the usage of acknowledgement mechanism to RTE.

4.4.9 SCHM

The module depends on the SCHM. The ECUM performs the initialization of that module.

Its usage can be enabled or disabled by the switch “Use SchM”.

4.4.10 Other BSW Modules

The module depends on other BSW modules. It performs their initialization. The configuration which modules should be initialized could be done over the configuration Node “EcuMDriverInitListOne”, “EcuMDriverInitListTwo”, “EcuMDriverInitListThree” and “EcuMDriverRestartList”.

Caution
The initialization of the modules described above must be configured by the integrator in one of the DriverInitLists.

4.5 Successive Adding of Modules

The MICROSAR ECUM allows the successive adding of modules during the integration phase. This means, that the integrator has the possibility to start integration with a minimum subset of modules, e.g. ECUM, MCU and OS and if the used configuration is working, to add other modules like GPT, DEM, FIM etc..

4.6 Callout execution sequences

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

STARTUP – RUN
▪ `EcuM_AL_DriverInitZero()`
▪ `EcuM_DeterminePbConfiguration()`
▪ `EcuM_AL_DriverInitOne()`
▪ `StartOS()`
▪ `SchM_Init()`
▪ `EcuM_AL_DriverInitTwo()`
▪ `Rte_Start()`
▪ `EcuM_AL_DriverInitThree()`
▪ `Rte_Switch_currentMode_currentMode(RTE_MODE_EcuM_Mode_RUN)`
▪ `EcuM_OnEnterRun()`
▪ `WdgM_SetMode()` (Run mode)
▪ `ComM_RunModeIndication()`

RUN – RESET

- `EcuM_OnExitRun()`
- `WdgM_SetMode()` (Post Run mode)
- `Rte_Switch_currentMode_currentMode(RTE_MODE_EcuM_Mode_POST_RUN)`
- `EcuM_OnExitPostRun()`
- `EcuM_OnPrepShutdown()`
- `Dem_Shutdown()`
- `Rte_Switch_currentMode_currentMode(RTE_MODE_EcuM_Mode_SHUTDOWN)`
- `EcuM_OnGoOffOne()`
- `NvM_WriteAll()`
- `WdgM_SetMode()` (Shutdown mode)
- System waits for finishing the `NvM_WriteAll()`
- `Rte_Stop()`
- `ComM_DeInit()`
- `ShutdownOS()`
- `EcuM_OnGoOffTwo()`
- `Mcu_PerformReset()`

RUN – OFF

- `EcuM_OnExitRun()`
- `WdgM_SetMode()` (Post Run mode)
- `Rte_Switch_currentMode_currentMode(RTE_MODE_EcuM_Mode_POST_RUN)`
- `EcuM_EnableWakeupSources()`
- `EcuM_OnExitPostRun()`
- `EcuM_OnPrepShutdown()`
- `Dem_Shutdown()`
- `Rte_Switch_currentMode_currentMode(RTE_MODE_EcuM_Mode_SHUTDOWN)`
- `EcuM_OnGoOffOne()`
- `NvM_WriteAll()`
- `WdgM_SetMode()` (Shutdown mode)
- System waits for finishing the `NvM_WriteAll()`
- `Rte_Stop()`
- `ComM_DeInit()`
- `ShutdownOS()`
- `EcuM_OnGoOffTwo()`
- `EcuM_AL_SwitchOff()`
- `Mcu_PerformReset()`

RUN – SLEEP – RUN

- EcuM_OnExitRun()
- WdgM_SetMode() (Post Run mode)
- Rte_Switch_currentMode_currentMode(RTE_MODE_EcuM_Mode_POST_RUN)
- EcuM_EnableWakeupSources()
- EcuM_OnExitPostRun()
- EcuM_OnPrepShutdown()
- Dem_Shutdown()
- Rte_Switch_currentMode_currentMode(RTE_MODE_EcuM_Mode_SLEEP)
- EcuM_OnGoSleep()
- NvM_WriteAll()
- WdgM_SetMode() (Shutdown mode)
- System waits for finishing the NvM_WriteAll()
- GetResource(RES_SCHEDULER)
- EcuM_GenerateRamHash()
- Mcu_SetMode()
- System is in sleep mode
- Occurrence of a valid wakeup event
- WdgM_SetMode() (Wakeup mode)
- EcuM_CheckRamHash()
- EcuM_AL_DriverRestart()
- ReleaseResource(RES_SCHEDULER)
- EcuM_OnWakeupReaction()
- EcuM_DisableWakeupSources()
- Dem_Init()
- EcuM_OnEnterRun()
- WdgM_SetMode() (Run mode)
- ComM_RunModeIndication()

RUN – SLEEP – WAKEUP VALIDATION – RUN

- `EcuM_OnExitRun()`
- `WdgM_SetMode()` (Post Run mode)
- `Rte_Switch_currentMode_currentMode(RTE_MODE_EcuM_Mode_POST_RUN)`
- `EcuM_EnableWakeupSources()`
- `EcuM_OnExitPostRun()`
- `EcuM_OnPrepShutdown()`
- `Dem_Shutdown()`
- `Rte_Switch_currentMode_currentMode(RTE_MODE_EcuM_Mode_SLEEP)`
- `EcuM_OnGoSleep()`
- `NvM_WriteAll()`
- `WdgM_SetMode()` (Shutdown mode)
- System waits for finishing the `NvM_WriteAll()`
- `WdgM_SetMode()` (Sleep mode)
- `GetResource(RES_SCHEDULER)`
- `EcuM_GenerateRamHash()`
- `Mcu_SetMode()`
- System is in sleep mode
- Occurrence of a wakeup event
- `WdgM_SetMode()` (Wakeup mode)
- `EcuM_CheckRamHash()`
- `EcuM_AL_DriverRestart()`
- `ReleaseResource(RES_SCHEDULER)`
- Wakeup detected as valid
- `EcuM_OnWakeupReaction()`
- `EcuM_DisableWakeupSources()`
- `Dem_Init()`
- `EcuM_OnEnterRun()`
- `WdgM_SetMode()` (Run mode)
- `ComM_RunModeIndication()`

RUN – SLEEP – WAKEUP VALIDATION – SLEEP

- EcuM_OnExitRun()
- WdgM_SetMode() (Post Run mode)
- Rte_Switch_currentMode_currentMode(RTE_MODE_EcuM_Mode_POST_RUN)
- EcuM_EnableWakeupSources()
- EcuM_OnExitPostRun()
- WdgM_SetMode() (Post Run mode)
- EcuM_OnPrepShutdown()
- Dem_Shutdown()
- Rte_Switch_currentMode_currentMode(RTE_MODE_EcuM_Mode_SLEEP)
- EcuM_OnGoSleep()
- NvM_WriteAll()
- WdgM_SetMode() (Shutdown mode)
- System waits for finishing the NvM_WriteAll()
- WdgM_SetMode() (Sleep mode)
- GetResource(RES_SCHEDULER)
- EcuM_GenerateRamHash()
- Mcu_SetMode() (Sleep mode)
- System is in sleep mode
- Occurrence of a wakeup event
- WdgM_SetMode() (Wakeup mode)
- EcuM_CheckRamHash()
- EcuM_AL_DriverRestart()
- ReleaseResource(RES_SCHEDULER)
- Wakeup detected as not valid
- EcuM_OnWakeupReaction()
- Rte_Switch_currentMode_currentMode(RTE_MODE_EcuM_Mode_WAKE_SLEEP)
- Rte_Switch_currentMode_currentMode(RTE_MODE_EcuM_Mode_SLEEP)

- `EcuM_OnGoSleep()`
- `GetResource(RES_SCHEDULER)`
- `EcuM_GenerateRamHash()`
- `Mcu_SetMode()` (Sleep mode)

4.7 Critical code sections

The ECUM has the following defined critical code sections:

- ECUM_EXCLUSIVE_AREA_0: Must lock task interrupts and interrupt sources during normal ECU modes like RUN and POST_RUN.
- ECUM_EXCLUSIVE_AREA_1: Must lock SLEEP mode Interrupts. Special interrupt control functionality has to be implemented for ECU SLEEP mode. Therefore the interrupt function set shall be used within the DaVinci Configuration tool to disable and enable these interrupts. The reason why this second exclusive are is defined is that, if the ECUM switches into SLEEP state not all interrupts are allowed to get disabled and the instructions to do this are HW dependent.

4.7.1 Wakeup Interrupt Source Handling – General Description

Wakeup interrupt sources are disabled and enabled.

Before `Mcu_SetMode` is called, because the ECU will change in SLEEP mode, the EcuM checks whether pending wakeups are existent. The call of Mcu API and this check shall be done while interrupts are locked.

Before it is checked whether pending wakeups are existent the wakeup interrupt sources shall get disabled in order that no wakeup interrupts are detected, until ECU goes into sleep mode.

It must also be assured that in case a wakeup event is detected as PENDING the wakeup interrupt sources are enabled again.

In case hardware specific instructions shall be used to disable or enable the wakeup interrupt sources the following procedure can be used. In other case the SchM functionality can be used for example.

To be able to use hardware specific source code, an ‘interrupt function set’ shall be defined as “Default Interrupt Settings” in configuration tool, for example `InterruptFunctionSetEcuMWkupIsr`*. For this ‘function set’ the suspend and restore functions must be defined, for example* `SuspendEcuMWkupInterrupts()` *and* `ResumeEcuMWkpInterrupts()`*. The corresponding header file must get inserted into “Include File” list.*

Within the EcuM configuration this defined interrupt function set shall be chosen in the drop down list “Sleep mode ISR handling” within the configuration tool.

4.7.2 Interrupt Source Handling – Cautions and further Information

Caution

- OS functionalities must not be used for handling ECUM_EXCLUSIVE_AREA_1.
- In case the ISRs are configured to category 1, it is not allowed to call OS API services, except functions `DisableAllInterrupts`, `EnableAllInterrupts`, `SuspendAllInterrupts` and `ResumeAllInterrupts`.
- If ‘interrupt function set’ is used the corresponding resume and restore function shall be filled otherwise the system will not running.

Info

The critical section handling apply the following parameter:

6.2.3 EcuMIrqHandling (Critical section handling) for ECUM_EXCLUSIVE_AREA_0 – SchM dependent

6.2.3 EcuMWkupIsrHandling (Sleep mode ISR handling) is used for ECUM_EXCLUSIVE_AREA_1 – SchM independent

4.8 AUTOSAR stack initialization

4.8.1 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 a Init-function signature with configuration pointer but make no use of it and needs to be initialized with a NULL_PTR

To use this mechanism within ECUM, the user must distinguish between the aspects given above and make the correct configuration decision (see Ch. 6.2.2.1.11)

- Module uses a configuration pointer for its initialization:

- Select the MSN to be initialized (e.g. Can)
- Select the corresponding configuration set (e.g. CanConfigSet)
- Select in one of the DriverInitList the MSN and the Init Function to be used (e.g. `Can_Init()`
- Result: The ECUM generates: "`Can_Init(&CanConfigSet);`" in the specified DriverInitList.

- Module has a void Init-function signature
  - The configuration items described in Ch. 6.2.2.1.11 must not be used
  - Select in one of the DriverInitList the MSN and the Init Function to be used (e.g. `ComM_Init()`
  - Result: The ECUM generates: `ComM_Init();`

- Module needs to be initialized with NULL_PTR
  - The configuration items described in Ch. 6.2.2.1.11 must not be used
  - Select in one of the DriverInitList the MSN and the Init Function to be used (e.g. `Mcu_Init())`
  - Result: The ECUM generates: "`Mcu_Init(NULL_PTR);`" in the specified DriverInitList

Additional to the AUTOSAR mechanism the MICROSAR implementation provides the possibility to insert any kind of function into a DriverInitList by using the "Code-Segment" in the driver init lists:

- The function `Mcu_InitClock()` must be executed in one of the DriverInitLists:
  - Select or type "Code" into "Module ID"
  - The code which should be executed must be inserted into "EcuMAdditionalInitCode" parameter, e.g. `Mcu_InitClock(0);`
  - Result: The ECUM generates: "`Mcu_InitClock(0);`" in the specified DriverInitList.

Info

Generally, every function which does not match to the standard AUTOSAR naming and behavior must be initialized with use of a "Code-Segement". For example:

- `Fee_30_Inst2_Init()`
- `Fls_30_ADBus01_Init()`
- `CanTrcv_30_Tja1041_Init()`

4.8.2 Initialization order

Depending on the modules which are used within the BSW stack the following example gives an overview of a possible initialization order.

Initialization Group	Comment
DriverInitListZero	
Det_Init()	
DriverInitListOne	
Mcu_Init()	
Dem_PreInit()	
Mcu_InitClock(0); while(MCU_PLL_LOCKED != Mcu_GetPllStatus()); Mcu_DistributePllClock();	Sequence to initialize PLL clock
Port_Init()	
DriverInitListTwo	
Adc, Pwm, Icu, Spi, etc.	MCAL driver initialization
Eep	Memory stack initialization
Fls	
Ea	
Fee	
NvM_Init()	

NvM_ReadAll()
{
NvM_RequestResultType blockStatus;
do
{
NvM_MainFunction();
Ea_MainFunction();
Fee_MainFunction();
Eep_MainFunction();
Fls_MainFunction();
NvM_GetErrorStatus(MultiBlockID,&blockStatus);
} while (blockStatus == NVM_REQ_PENDING);
}
	Read all sequence

CanTrcv	CAN stack initialization
CanDrv	
CanIf	
LinTrcv	LIN stack initialization
LinDrv	
LinIf	

LinIf_GotoSleepInternal	
FrTrcv	Flexray stack initialization
FrDrv	
FrIf	
CanSM	Bus State Manager initialization
LinSM	
FrSM	
Com	
NmIf	
NmCan	
NmFr	
ComM	
DCM	
DEM	Could also be initialized in DriverInitListThree
DriverInitListThree	
DEM	Could also be initialized in DriverInitListTwo

Table 4-4 initialization order

4.9 Handling of wakeup events after ShutdownOS()

When shutdown target ECUM_STATE_OFF is selected the ECUM waits in state ECUM_STATE_GO_OFF_ONE until the NVM has finished with writing NVRAM data. In this state the ECUM will react on occurring wakeup events. After the data is written the function ShutdownOS() will be executed and EcuM_AL_SwitchOff() will switch off the ECU, in this time space between wakeup events may occur and a ECU reset should be the desired reaction in most cases. The integrator is responsible to implement the desired behavior into EcuM_AL_SwitchOff().

Caution
The integrator is responsible to implement the wakeup event reaction handling after ECUM_STATE_GO_OFF_ONE is left.

4.10 Wakeup event handling and Wakeup Validation

The handling of wakeup sources and wakeup validation has to be configured and implemented specific for every ECU. The following list provides a short overview which callouts are affected:

- EcuM_EnableWakeupSources(), refer to Ch. 5.6.2.19
- EcuM_DisableWakeupSources(), refer to Ch. 5.6.2.20
- EcuM_CheckWakeup(), refer to Ch. 5.6.2.23
- EcuM_ValidateWakeup(), refer to Ch. 5.6.2.24

- EcuM_StartWakeupSources(), refer to Ch. 5.6.2.21
- EcuM_StopWakeupSources(), refer to Ch. 5.6.2.22

The integration task is to fill these callouts with code which match the ECU specific requirements. To give an idea what must be implemented, the following paragraphs covers three use cases:

- Wakeup after a physical sleep mode
- Handling of wakeup events while ECUM is in RUN state
- Wakeup validation of communication channels (ECUM in RUN state)

4.10.1 Wakeup after a physical sleep mode

4.10.1.1 Use case description

A raising edge on an ICU channel shall bring the ECUM into RUN state. A wakeup source “ECUM_WKSOURCE_ICU_CH0” is configured for that. The name of the configured ICU channel is Icu_Channel0. No wakeup validation shall be performed on that wakeup event. This wakeup event is the only active wakeup event for the desired sleep mode.

4.10.1.2 Execution flow

- ECUM is in ECUM_APP_RUN no further RUN requests are existent
- ECUM transits from ECUM_APP_RUN to ECUM_APP_POST_RUN
    - Callout EcuM_EnableWakeupSources() is executed.
- ECUM transits further to ECUM_STATE_SLEEP (ECU is physically in sleep mode)
- 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 wakeup event is instantly valid
- ECUM transits from ECUM_STATE_SLEEP until ECUM_STATE_WAKEUP_TWO
- ECUM transits from ECUM_STATE_WAKEUP_TWO to ECUM_STATE_APP_RUN
    - Callout EcuM_DisableWakeupSources() is executed

4.10.1.3 Callout implementation examples

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

void 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);
  }
  /* … */
}

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

4.10.2 Handling of wakeup events while ECUM is in RUN state

For this use case it is assumed that wakeup events which occur while ECUM is in RUN state are caused by communication channels.

4.10.2.1 Use case description

A wakeup 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 wakeup source ECUM_WKSOURCE_CAN0 is configured for that. No wakeup validation will be performed.

4.10.2.2 Execution flow

- ECUM is in RUN state and the CAN channel is in sleep state and is able to detect wakeup events
- Callout EcuM_CheckWakeup() is executed by ISR
- API EcuM_SetWakeupEvent() is executed
- ECUM executes implicitly executed ComM_WakeUpIndication() and EcuM_ValidateWakeupEvent() because wakeup event is instantly valid.

4.10.2.3 Callout implementation examples

```
void EcuM_CheckWakeup(EcuM_WakeupSourceType wakeupSource)
{
  if ((wakeupSource & ECUM_WKSOURCE_CAN0) != 0)
  {
     /* check if the underlaying driver has a wakeup event detected.
      * CanIf will execute EcuM_SetWakeupEvent() in this case.
      */
     CanIf_CheckWakeup(ECUM_WKSOURCE_CAN0);
  }
}
```

4.10.3 Wakeup validation of communication channels (ECUM in RUN state)

4.10.3.1 Use case description

The same prerequisites as specified in Ch. 4.10.2.1 apply, but the wakeup event must be validated.

4.10.3.2 Execution flow

- ECUM is in RUN state and the CAN channel is in sleep state and is able to detect wakeup events
- Callout EcuM_CheckWakeup() is executed by ISR
- API EcuM_SetWakeupEvent() is executed, ECUM starts wakeup validation timeout

- EcuM_MainFunction() triggered by SCHM
  - (a) ECUM detects a pending wakeup 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 wakeup event is validated or expired, but EcuM_StartWakeupSources() is executed only once.
- Case Validation successful:
  - API EcuM_ValidateWakeupEvent() is executed, ECUM implicitly executed ComM_WakeUpIndication()
  - EcuM_MainFunction() triggered by SCHM
    - ECUM stops validation timeout
- Case Validation failed:
  - ECUM executes callout EcuM_StopWakeupSources()
  - No further wakeup related actions performed

©2011, Vector Informatik GmbH

4.10.3.3 Callout implementation examples

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

void EcuM_StartWakeupSources(EcuM_WakeupSourceType wakeupSource)
{
  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_SetTransceiverMode(0, CANIF_TRCV_MODE_NORMAL);
        CanIf_SetControllerMode(0, CANIF_CS_STOPPED);
        CanIf_SetControllerMode(0, CANIF_CS_STARTED);
     }
     else
     {
         /* Stack already up and running */
     }
   }
}

void EcuM_CheckValidation(EcuM_WakeupSourceType wakeupSource)
{
  if ((wakeupSource & ECUM_WKSOURCE_CAN0) != 0)
  {
     /* Query the driver if the wakeup event was valid */
     CanIf_CheckValidation(ECUM_WKSOURCE_CAN0);
  }
}
```

```
void 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_SetTransceiverMode(0, CANIF_TRCV_MODE_STANDBY);
  }
}
```

4.10.4 Notes to Mcu_SetMode()

The execution of `Mcu_SetMode()` to enter sleep mode is one of the crucial actions which are performed by ECUM. The AUTOSAR specifications are not so exact specified so that every MCU driver supplier has the same knowledge which implicit requirements have to be supported by the MCU driver.

Important to know is that the ECUM should call `Mcu_SetMode()` with global locked interrupts, this can be configured with the second critical section described in 4.7.

If MCU driver supplier specific action are necessary, a possibility must be given to add them before and after the `Mcu_SetMode()` call. Therefore the `Mcu_SetMode()` is called within the callout `EcuM_McuSetMode`. This callout is always available, it can not be disabled.

Caution

Use this functionality with care.

It is not allowed to remove the `Mcu_SetMode` function call from callout.

4.11 Multiple Identity ECUs

4.11.1 Restrictions when implementation variant PreCompile is used

The following restrictions exists if the ECUM shall be used for initializing multiple identity ECUs when the implementation variant PreCompile is used. To use the ECUM in implementation variant PreCompile is not the normal use case for such ECUs but might be considered to reduce ECU resources.

4.11.1.1 Initialization

The description in Ch 4.8 does not apply. The user is responsible to implement the distinction which identity is currently selected and which modules initialization applies. It is recommended to use the “Code” segment which can be selected in the DriverInitLists for this task.

4.11.1.2 EcuM configuration

If several multiple configuration containers (“/MICROSAR/EcuM/EcuMConfiguration”) within the ECUM configuration exist, the ECUM uses the first multiple configuration container as reference for its generation.

This implies that some configuration parameters cannot differ between the configuration sets, e.g.:

- Sleep modes
- Wakeup Sources
- ECUM Users
- ComM channels
- TTII successors and divisors
- DriverInitLists
- All configuration parameters which are specified within the “/MICROSAR/EcuM/EcuMConfiguration” – container.

4.12 Generated Template Files

Info
This chapter is only applicable if DaVinci Configurator is used for ECUM generation.

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, where the user must insert his code in between. The comments have the following format:

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

/*********************************************************************************
```

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

Caution
Do not modify or delete the comments.

Example:

The following example explains where code could be implemented:

```
FUNC(void, ECUM_CODE) EcuM_OnEnterRun(void)
{
  /* A */
/**********************************************************************************
 * DO NOT CHANGE THIS COMMENT!  <USERBLOCK EcuM_OnEnterRun>  DO NOT CHANGE THIS COMMENT!
 **********************************************************************************/
  /* B */
  return;
/**********************************************************************************
 * DO NOT CHANGE THIS COMMENT!  </USERBLOCK>              DO NOT CHANGE THIS COMMENT!
 **********************************************************************************/

  /* C */
} /* End of EcuM_OnEnterRun() */
```

A and C: all modifications before and after will be deleted at the next generation process

B: all modifications will not be deleted

The ECUM provides the following generated template files:

- EcuM_Callout_Stubs.c

5 API Description

5.1 Interfaces Overview

See chapter 2.1.

5.2 Type Definitions

ECUM types are described in the following tables:

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 ANDing 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 WAKEUP super state ECUM_STATE_WAKEUP_ONE Reinitializing of drivers for normal operation ECUM_STATE_WAKEUP_VALIDATION Waits for validation of a wakeup event ECUM_STATE_WAKEUP_REACTION Computes the appropriate wakeup reaction ECUM_STATE_WAKEUP_TWO Prepares the ECU for RUN state ECUM_STATE_WAKEUP_WAKESLEEP A short system phase where the ECU transit from a wakeup directly to sleep again ECUM_STATE_WAKEUP_TTII Performs the TTII protocol ECUM_STATE_RUN Normal ECU operation super state

Type Name	C-Type	Description	Value Range
			ECUM_STATE_APP_RUN ECU is in normal operation state ECUM_STATE_APP_POST_RUN ECU performs POST RUN activities ECUM_STATE_SHUTDOWN Shutdown super state ECUM_STATE_PREP_SHUTDOWN Prepares the ECU for the following shutdown sequence ECUM_STATE_GO_SLEEP Activation of the wakeup sources ECUM_STATE_GO_OFF_ONE Shutdown of system services ECUM_STATE_GO_OFF_TWO Performs a RESET or switches of 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_UserType	uint8	ID of the User 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_WakeupSourceType	uint32	Each bit in this type identifies a wakeup 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) 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) ... Can be extended by configuration

Type Name	C-Type	Description	Value Range
			ECUM_WKSOURCE_ALL_SOURCES Identifies each wakeup source ECUM_WKSOURCE_NONE Value 0. This is a MICROSAR ECUM extension and identifies an invalid wakeup source.
EcuM_WakeupStatusType	uint8	The type describes possible outcomes of the WAKEUP VALIDATION state.	ECUM_WKSTATUS_NONE No pending wakeup was detected ECUM_WKSTATUS_PENDING The wakeup event was detected but not yet validated ECUM_WKSTATUS_VALIDATED The wakeup event is valid ECUM_WKSTATUS_EXPIRED The wakeup event has not been validated and has expired before.
EcuM_WakeupReactionType	uint8	The describes the possible outcomes of the WAKEUP REACTION state	ECUM_WKACT_RUN Transit into RUN state ECUM_WKACT_TTII Execute the TTII protocol ECUM_WKACT_SHUTDOWN Directly transit to shutdown
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

Table 5-1 Type definitions

5.3 Services provided by ECUM

The ECUM API consists of services, which are realized by function calls.

Info
Most of the following API functions report errors as listed in chapter 3.6.1. If an error is detected the concerning API function will be left without any further actions.

5.3.1 EcuM_Init

Prototype
`void EcuM_Init ( void )`

Parameter	
--	--
Return code	
Void	--
Functional Description	
This function initializes the ECUM and initializes basic drivers. In this function drivers which are configured in DriverInitListZero and DriverInitListOne will be initialized. At the end of this function `StartOS()` is called and therefore this function never returns.	
Particularities and Limitations	
■ This service is synchronous. ■ This service is non re-entrant. ■ This service is always available.	
Expected Caller Context	
■ Expected to be called by an application.	

Table 5-2 EcuM_Init

5.3.2 EcuM_Shutdown

Prototype	
`void EcuM_Shutdown ( void )`	
Parameter	
--	--
Return code	
Void	--
Functional Description	
This function performs a reset or switches off the ECU (depending on which shutdown target is currently chosen). If the ECU should be switched off the callout `EcuM_AL_SwitchOff()` must implement the specific code. **Info** This function shall be called inside the OS `ShutdownHook()` routine. The integrator is responsible to perform this task.	
Particularities and Limitations	
■ This service is synchronous. ■ This service is non re-entrant. ■ This service is always available.	
Expected Caller Context	
■ Expected to be called by an application.	

Table 5-3 EcuM_Shutdown

5.3.3 EcuM_GetVersionInfo

Prototype	
`void EcuM_GetVersionInfo ( Std_VersionInfoType *versioninfo )`	
Parameter	
versioninfo	Pointer with the version information
Return code	
void	--
Functional Description	
This service returns the version information of the ECU State Manager.	
Particularities and Limitations	
■ This service is synchronous. ■ This service is non re-entrant. ■ This service is available, dependently on `ECUM_VERSION_INFO_API`.	
Expected Caller Context	
■ Expected to be called by an application.	

Table 5-4 EcuM_GetVersionInfo

5.3.4 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 are normally application and resource manager like Communication Manager. The tracking of the requests is specific for each user. **Info** RUN request will be ignored after an API call to `EcuM_KillAllRUNRequest()`.	
Particularities and Limitations	
■ This service is synchronous. ■ This service is re-entrant. ■ This service is always available.	
Expected Caller Context	
■ Expected to be called in application context.	

Table 5-5 EcuM_RequestRUN

5.3.5 EcuM_ReleaseRUN

Prototype	
`Std_ReturnType EcuM_ReleaseRUN ( EcuM_UserType user )`	
Parameter	
user	User ID which releases the run request
Return code	
E_OK	Release accepted
E_NOT_OK	Release not accepted
Functional Description	
Releases the RUN request previously done with a call to `EcuM_RequestRUN()`.	
Particularities and Limitations	
■ This service is synchronous. ■ This service is re-entrant. ■ This service is always available.	
Expected Caller Context	
■ Expected to be called in application context.	

Table 5-6 EcuM_ReleaseRUN

5.3.6 EcuM_ComM_RequestRUN

Prototype	
`Std_ReturnType EcuM_ComM_RequestRUN ( NetworkHandleType channel )`	
Parameter	
channel	ID of the communication channel requesting the RUN state.
Return code	
E_OK	Request accepted
E_NOT_OK	Request not accepted
Functional Description	
This function shall be used by the Com Manager to request RUN state. The behavior is identical to `EcuM_RequestRUN()`.RUN request will be ignored after an API call to `EcuM_KillAllRUNRequest()`.	
Particularities and Limitations	
■ This service is synchronous. ■ This service is re-entrant. ■ This service is always available. ■ The configured communication channels are obtained from the ECUC file. This means that a valid COMM configuration must be available at generation time of the ECUM.	
Expected Caller Context	
■ Expected to be called in application context.	

Table 5-7 EcuM_ComM_RequestRUN

5.3.7 EcuM_ComM_ReleaseRUN

Prototype	
`Std_ReturnType` **`EcuM_ComM_ReleaseRUN`** `( NetworkHandleType channel )`	
Parameter	
channel	ID of the communication channel requesting 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_ComM_RequestRUN()`.	
Particularities and Limitations	
■ This service is synchronous. ■ This service is re-entrant. ■ This service is always available. ■ The configured communication channels are obtained from the ECUC file. This means that a valid COMM configuration must be available at generation time of the ECUM.	
Expected Caller Context	
■ Expected to be called in application context.	

Table 5-8 EcuM_ComM_ReleaseRUN

5.3.8 EcuM_ComM_HasRequestedRUN

Prototype	
`Boolean` **`EcuM_ComM_HasRequestedRUN`** `( EcuM_ChannelHandleType channel )`	
Parameter	
channel	ID of the communication channel requesting the RUN state.
Return code	
TRUE	the channel has requested RUN state
FALSE	the channel has not requested RUN state
Functional Description	
This service returns TRUE if the channel given as parameter has actually requested RUN state, the service returns FALSE in the other case.	
Particularities and Limitations	
■ This service is synchronous. ■ This service is re-entrant. ■ This service is always available. ■ The configured communication channels are obtained from the ECUC file. This means that a valid COMM configuration must be available at generation time of the ECUM.	
Expected Caller Context	
■ Expected to be called in application context.	

Table 5-9 EcuM_ComM_HasRequestedRUN

5.3.9 EcuM_RequestPOST_RUN

Prototype	
`Std_ReturnType` **`EcuM_RequestPOST_RUN`** `( EcuM_UserType user )`	
Parameter	
User	User ID which requests the post run request.
Return code	
E_OK	Request accepted
E_NOT_OK	Request not accepted
Functional Description	
Places a POST RUN request for this user. Users are normally application and resource manager like COM Manager. The tracking of the requests are specific for each user. POST RUN request will be ignored after an API call to `EcuM_KillAllRUNRequest()`.	
Particularities and Limitations	
■ This service is synchronous. ■ This service is re-entrant. ■ This service is always available.	
Expected Caller Context	
■ Expected to be called in application context.	

Table 5-10 EcuM_RequestPOST_RUN

5.3.10 EcuM_ReleasePOST_RUN

Prototype	
`Std_ReturnType` **`EcuM_ReleasePOST_RUN`** `( EcuM_UserType user )`	
Parameter	
User	User ID which releases the post run request
Return code	
E_OK	Release accepted
E_NOT_OK	Release not accepted
Functional Description	
Releases POST RUN requests previously done with a call to `EcuM_RequestPOST_RUN()`.	
Particularities and Limitations	
■ This service is synchronous. ■ This service is re-entrant. ■ This service is always available.	
Expected Caller Context	
■ Expected to be called in application context.	

Table 5-11 EcuM_ReleasePOST_RUN

5.3.11 EcuM_KillAllRUNRequests

Prototype	
`void EcuM_KillAllRUNRequests ( void )`	
Parameter	
--	--
Return code	
void	--
Functional Description	

Deletes all RUN and POST RUN requests and initiates an ECU reset.

Info

- This service was introduced for the Com Manager to perform software resets in the case of network errors. Only the Com Manager should use this service.
- After a call to this function all RUN requests will be rejected and wakeup events will be ignored.

Caution

EcuM_KillAllRunRequests:

- its usage is only intended for emergency shutdown
- it sets the shutdown target to RESET
- this shutdown target cannot be modified because:

  `EcuM_SelectShutDownTarget` checks whether `EcuM_KillAllRunRequest` was called before and in this case the function is left with NOT_OK

  `EcuM_SelectShutDownTarget` call before `EcuM_KillAllRunRequest` is called makes no sense, because the shutdown target will be modified to RESET within EcuM`_KillAllRunRequest.`

Particularities and Limitations
■ This service is synchronous.
■ This service is re-entrant.
■ This service is available, dependently on `ECUM_KILL_ALL_RUNREQUEST_API`.
Expected Caller Context
■ Expected to be called in application context.

Table 5-12 EcuM_KillAllRUNRequests

5.3.12 EcuM_SelectShutdownTarget

Prototype	
`Std_ReturnType EcuM_SelectShutdownTarget ( EcuM_StateType target, uint8 mode )`	
Parameter	
target	The selected shutdown target must be one of: ■ ECUM_STATE_SLEEP ■ ECUM_STATE_RESET ■ ECUM_STATE_OFF
mode	If the selected target is ECUM_STATE_SLEEP this parameter is an index into the selected sleep mode. If the selected target is not ECUM_STATE_SLEEP this parameter will be ignored, so the value of the parameter does not matter.
Return code	
E_OK	the shutdown target was accepted
E_NOT_OK	the shutdown target was not accepted
Functional Description	

This service selects a shutdown target in which the shutdown sequence should switch.

Caution
This service can only be used while the ECUM is in RUN state in order to enable the wakeup sources. Please refer to 5.6.2.19 EcuM_EnableWakeupSources for more information.

Functional Description
Returns the current module state of the ECUM.
Particularities and Limitations
■ This service is synchronous. ■ This service is re-entrant. ■ This service is always available.
Expected Caller Context
■ Expected to be called by an application.

Table 5-14 EcuM_GetState

5.3.14 EcuM_GetShutdownTarget

Prototype	
`Std_ReturnType EcuM_GetShutdownTarget ( EcuM_StateType *target, uint8 *sleepmode )`	
Parameter	
target	One of these values: ■ ECUM_STATE_SLEEP ■ ECUM_STATE_RESET ■ ECUM_STATE_OFF
sleepmode	If the target is ECUM_STATE_SLEEP the index of the chosen sleep mode is returned else sleep mode is always set to 0.
Return code	
E_OK	both parameters where not NULL_PTR
E_NOT_OK	both parameters where NULL_PTR
Functional Description	
Returns the current selected shutdown target.	
Particularities and Limitations	
■ This service is synchronous. ■ This service is re-entrant. ■ This service is always available.	
Expected Caller Context	
■ Expected to be called in application context.	

Table 5-15 EcuM_GetShutdownTarget

5.3.15 EcuM_GetLastShutdownTarget

Prototype
`Std_ReturnType EcuM_GetLastShutdownTarget ( EcuM_StateType *target, uint8 *sleepmode )`

Parameter	
target	One of these values: ■ ECUM_STATE_SLEEP ■ ECUM_STATE_RESET ■ ECUM_STATE_OFF
sleepmode	If the target is ECUM_STATE_SLEEP the index of the chosen sleep mode is returned else sleep mode is always set to 0.
Return code	
E_OK	both parameters where not NULL_PTR
E_NOT_OK	both parameters where NULL_PTR
Functional Description	
The return value describes the ECU state from which the last wakeup or power up occurred. This function always shall return the same value until the next shutdown.	
Particularities and Limitations	
■ This service is synchronous. ■ This service is re-entrant. ■ This service is always available.	
Expected Caller Context	
■ Expected to be called in application context.	

Table 5-16 EcuM_GetLastShutdownTarget

5.3.16 EcuM_GetPendingWakeupEvents

Prototype	
`EcuM_WakeupSourceType EcuM_GetPendingWakeupEvents ( void )`	
Parameter	
--	--
Return code	
EcuM_WakeupSourceType	Every bit set in the return value indicates a wakeup source where the validation is in progress.
Functional Description	
Returns all wakeup events where the validation is in progress.	
Particularities and Limitations	
■ This service is synchronous. ■ This service is non re-entrant and non interruptible. ■ This service is always available.	
Expected Caller Context	
■ Expected to be called in interrupt context.	

Table 5-17 EcuM_GetPendingWakeupEvents

5.3.17 EcuM_ClearWakeupEvent

Prototype	
`void EcuM_ClearWakeupEvent ( EcuM_WakeupSourceType sources )`	
Parameter	
Sources	Every wakeup source set in the value will be cleared
Return code	
void	--
Functional Description	
Clears the pending, validated and expired wakeup events which are passed by the parameter.	
Particularities and Limitations	
■ This service is synchronous. ■ This service is non re-entrant and non interruptible. ■ This service is always available.	
Expected Caller Context	
■ Expected to be called in interrupt context.	

Table 5-18 EcuM_GetPendingWakeupEvents

5.3.18 EcuM_GetValidatedWakeupEvents

Prototype	
`EcuM_WakeupSourceType EcuM_GetValidatedWakeupEvents ( void )`	
Parameter	
--	--
Return code	
EcuM_WakeupSourceType	ID of the wakeup source which caused the wakeup of the ECU.
Functional Description	
This function returns wakeup event which causes the wakeup of the microcontroller from the previous sleep mode.	
Particularities and Limitations	
■ This service is synchronous. ■ This service is non re-entrant and non interruptible. ■ This service is available, dependently on `ECUM_GET_VALIDATED_WAKEUP_EVENTS_API`.	
Expected Caller Context	
■ Expected to be called in application context.	

Table 5-19 EcuM_GetValidatedWakeupEvents

5.3.19 EcuM_GetExpiredWakeupEvents

Prototype
`EcuM_WakeupSourceType EcuM_GetExpiredWakeupEvents ( void )`

Parameter

5.3.21 EcuM_SelectApplicationMode

Prototype	
`Std_ReturnType EcuM_SelectApplicationMode ( AppModeType appmode )`	
Parameter	
appmode	Application mode for the next OS restart. Values depend on the OS configuration.
Return code	
E_OK	The service always succeeds, because no parameter checking can be performed.
Functional Description	
Sets the application mode of the OS for the next restart. The application mode will be set after an intended reset. **Info** Deviation to AUTOSAR: return value E_NOT_OK is not provided.	
Particularities and Limitations	
■ This service is synchronous. ■ This service is re-entrant. ■ This service is always available.	
Expected Caller Context	
■ Expected to be called in application context.	

Table 5-22 EcuM_SelectApplicationMode

5.3.22 EcuM_GetApplicationMode

Prototype	
`Std_ReturnType EcuM_GetApplicationMode ( AppModeType *appmode )`	
Parameter	
appmode	Currently selected application mode of the OS.
Return code	
E_OK	Call was successful.
E_NOT_OK	Call was rejected (in the case if a `NULL_PTR` has been passed as argument).
Functional Description	
Returns the current selected application mode.	
Particularities and Limitations	
■ This service is synchronous. ■ This service is re-entrant. ■ This service is always available.	
Expected Caller Context	

■ Expected to be called in application context.

Table 5-23 EcuM_GetApplicationMode

5.3.23 EcuM_SelectBootTarget

Prototype	
`Std_ReturnType EcuM_SelectBootTarget ( EcuM_BootTargetType target )`	
Parameter	
target	The selected boot target.
Return code	
E_OK	Call was accepted.
E_NOT_OK	Call was rejected.
Functional Description	
This service sets the boot target to the specified value. Whenever this service is executed the callout function EcuM_Appl_SelectBootTarget() will be executed implicitly. In this callout the desired behavior has to be implemented. **Caution** This service does not perform any NULL_PTR check or UNINIT check. When such functionality is required it must be implemented by the user in the specific callouts.	
Particularities and Limitations	
■ This service is synchronous. ■ This service is re-entrant. ■ This service is always available.	
Expected Caller Context	
■ Expected to be called in application context.	

Table 5-24 EcuM_SelectBootTarget

5.3.24 EcuM_GetBootTarget

Prototype	
`Std_ReturnType EcuM_GetBootTarget ( EcuM_BootTargetType *target )`	
Parameter	
target	The current selected boot target.
Return code	
E_OK	Call was accepted.
E_NOT_OK	Call was rejected (in the case if a `NULL_PTR` has been passed as argument).

Functional Description
This service returns the current selected boot target. Whenever this service is executed the callout function EcuM_Appl_GetBootTarget() will be executed implicitly. In this callout the desired behavior has to be implemented. (See also description of 5.3.23).
Particularities and Limitations
■ This service is synchronous. ■ This service is re-entrant. ■ This service is always available.
Expected Caller Context
■ Expected to be called in application context.

Table 5-25 EcuM_GetBootTarget

5.3.25 EcuM_MainFunction

Prototype	
`void EcuM_MainFunction ( void )`	
Parameter	
--	--
Return code	
void	--
Functional Description	
This service implements all activities of the ECU State Manager while OS is up and running. This service must be called on a periodic basis from an adequate OS task. To determine the period the SWS suggests the following points: ■ The period directly results in a possible latency for testing RUN requests. The largest acceptable reaction time will therefore limit the maximum period for invocation. ■ The service also carries out the wakeup 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	
Particularities and Limitations	
■ This service is synchronous. ■ This service is re-entrant. ■ This service is always available.	
Expected Caller Context	
■ Expected to be called in application context.	

Table 5-26 EcuM_MainFunction

5.3.26 EcuM_StartupTwo

Prototype
`void EcuM_StartupTwo ( void )`

Parameter	
--	--
Return code	
void	--
Functional Description	
This function initializes drivers and manager modules which are not initialized in `EcuM_Init()` function, like Nvm, ComM, etc. The activities performed in this function are described in the ECU State Manager specification as STARTUP II phase. The DriverInitListTwo and DriverInitListThree will be executed in this function.	
Particularities and Limitations	
■ This service is synchronous. ■ This service is non re-entrant. ■ This service is always available.	
Expected Caller Context	
■ Expected to be called in application context.	

Table 5-27 EcuM_StartupTwo

5.4 Services used by ECUM

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

Refer to chapter 2.1

5.5 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.

5.5.1 EcuM_CB_NfyNvMJobEnd

Prototype	
`void EcuM_CB_NfyNvMJobEnd ( uint8 ServiceID, NvM_RequestResultType JobResult )`	
Parameter	
ServiceID	Unique service ID of NVRAM manger service.
JobResult	Covers the job result of the previous processed multi block job.
Return code	
void	--
Functional Description	
Used to notify about the end of NVRAM jobs initiated by ECUM.	

Particularities and Limitations
■ This function is synchronous.
■ This function is non re-entrant.
■ This service is available, dependently on ECUM_INCLUDE_NVRAM_MGR.
■ The NVRAM manager must be configured to call this callback as a multiple block job end notification.
Expected Caller Context
■ This function is expected to be called in application context.

Table 5-28 EcuM_CB_NfyNvMJobEnd

5.5.2 EcuM_SetWakeupEvent

Prototype	
`void EcuM_SetWakeupEvent ( EcuM_WakeupSourceType event )`	
Parameter	
event	ID of the wakeup source
Return code	
void	--
Functional Description	
The driver or the module which validates the appropriate wakeup event must call this service to start the wakeup validation protocol. If the wakeup source needs validation the corresponding timeout will be started. If no validation is required this function implicitly calls `EcuM_ValidateWakeupEvent()`.	
Particularities and Limitations	
■ This service is synchronous.	
■ This service is non re-entrant and non interruptible.	
■ This service is always available.	
Expected Caller Context	
■ Expected to be called in interrupt context.	

Table 5-29 EcuM_SetWakeupEvent

5.5.3 EcuM_ValidateWakeupEvent

Prototype	
`void EcuM_ValidateWakeupEvent ( EcuM_WakeupSourceType event )`	
Parameter	
event	ID of wakeup source
Return code	
void	--
Functional Description	
If the validation of the wakeup event was successful this service has to be called by the module which performs the validation. If no validation is required the function will be called implicitly.	

Particularities and Limitations
■ This service is synchronous. ■ This service is re-entrant. ■ This service is always available.
Expected Caller Context
■ Expected to be called in application context.

Table 5-30 EcuM_ValidateWakeupEvent

5.6 Configurable Interfaces

5.6.1 Notifications

The ECUM does not provide notifications.

5.6.2 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:

5.6.2.1 EcuM_AL_DriverInitZero

Prototype	
`void EcuM_AL_DriverInitZero ( void )`	
Parameter	
--	--
Return code	
void	--

Functional Description
This callout shall provide driver initialization and other hardware-related startup activities for loading the post-build configuration data.

Caution
If the pre-compile switch V_SUPRESS_EXTENDED_VERSION_CHECK is defined the default implementation of this callout executes the Vector specific function `VLibVersionCheck()`. This function ensures the integrity of the delivery.

If the function call was not successful, one of the two following will occur:

- development is activated: the error module (usually DET module) is executed
- production mode is activated: an endless loop will be entered. This is part of template code and shall be adapted by user to meet current system needs.

Info
Only pre-compile and link-time configurable modules may be used.

Particularities and Limitations
■ This function is mandatory.
■ This function is non re-entrant.
■ This function is always available.
■ This function has to be filled with code by the integrator.
Call Context
■ Invoked in EcuM_Init(), task context

5.6.2.2 EcuM_AL_DriverInitOne

Prototype	
`void EcuM_AL_DriverInitOne (const EcuM_ConfigType *ConfigPtr )`	
Parameter	
`ConfigPtr`	Pointer to ECUM configuration structure. **Caution** The ECUM executes this callout with a NULL_PTR as argument.

Return code	
void	--

Functional Description

This callout initializes all driver modules which should be initialized before the OS is up and running.

Info
If the ECUM is delivered in post-build variant. The signature of this callout changes into void EcuM_DriverInitOne_<0..n>(void), in order to provide the post-build capability of the ECUM.

Particularities and Limitations

- This function is mandatory.
- This function is non re-entrant.
- This function is always available.
- This function is configured via configuration tool.

Call Context

- Invoked in `EcuM_Init()`, task context

Table 5-31 EcuM_AL_DriverInitOne

5.6.2.3 EcuM_AL_DriverInitTwo

Prototype

```
void EcuM_AL_DriverInitTwo (const EcuM_ConfigType *ConfigPtr )
```

Parameter

`ConfigPtr`	Pointer to ECUM configuration structure. **Caution** The ECUM executes this callout with a NULL_PTR as argument.

Return code	
void	--

Functional Description

Initializes driver modules which should be initialized after the OS has been started.

Info
If the ECUM is delivered in post-build variant. The signature of this callout changes into void EcuM_DriverInitTwo_<0..n>(void), in order to provide the post-build capability of the ECUM.

Particularities and Limitations
■ This function is mandatory.
■ This function is non re-entrant.
■ This function is always available.
■ This function is configured via configuration tool.
Call Context
■ invoked in `EcuM_StartupTwo()`, task context
■ Invoked when module state = STARTUPII

Table 5-32 EcuM_AL_DriverInitTwo

5.6.2.4 EcuM_AL_DriverInitThree

Prototype	
`void` **`EcuM_AL_DriverInitThree`** `(const EcuM_ConfigType *ConfigPtr )`	
Parameter	
`ConfigPtr`	Pointer to ECUM configuration structure. **Caution** The ECUM executes this callout with a NULL_PTR as argument.
Return code	
void	--
Functional Description	
This callout shall provide driver initialization of drivers which need OS and need valid (if the `NvM_ReadAll()` job was successful) NVRAM data. **Info** If the ECUM is delivered in post-build variant. The signature of this callout changes into void EcuM_DriverInitOne_<0..n>(void), in order to provide the post-build capability of the ECUM.	
Particularities and Limitations	
■ This function is mandatory. ■ This function is non re-entrant. ■ This function is always available. ■ This function is configured via configuration tool.	
Call Context	
■ Invoked in EcuM_StartupTwo(), task context ■ Invoked after Rte_Start() is executed	

Table 5-33 EcuM_AL_DriverInitThree

5.6.2.5 EcuM_DeterminePbConfiguration

Prototype	
`EcuM_ConfigType* EcuM_DeterminePbConfiguration ( void )`	
Parameter	
--	--
Return code	
EcuM_ConfigType	Pointer to the used configuration set.
Functional Description	
This callout should determine which post-build configuration should be used by the ECUM. This has to be done by the integrator. **Info** This callout will only be activated when the implementation of the module is delivered as post-build variant.	
Particularities and Limitations	
■ This function is mandatory. ■ This function is non re-entrant. ■ This function is always available. ■ This function has to be filled with code by the integrator.	
Call Context	
■ Invoked in `EcuM_Init()`, task context ■ Invoked between `EcuM_AL_DriverInitZero()` and `EcuM_AL_DriverInitOne()`	

Table 5-34 EcuM_DeterminePbConfiguration

5.6.2.6 EcuM_AL_SwitchOff

Prototype	
`void EcuM_AL_SwitchOff ( void )`	
Parameter	
--	--
Return code	
void	--
Functional Description	
This callout shall take the code for shutting off the power supply of the ECU.	
Particularities and Limitations	
■ This function is mandatory. ■ This function is non re-entrant. ■ This function is always available. ■ This function has to be filled with code by the integrator.	

Call Context
■ Invoked in `EcuM_Shutdown()`, task context

Table 5-35 EcuM_AL_SwitchOff

5.6.2.7 EcuM_AL_DriverRestart

Prototype	
`void EcuM_AL_DriverRestart ( void )`	
Parameter	
--	--
Return code	
void	--
Functional Description	
This callout shall provide driver initialization and other hardware related startup activities after a wakeup event from SLEEP state. **Info** If the ECUM is delivered in post-build variant. The signature of this callout changes into void EcuM_DriverRestart_<0..n>(void), in order to provide the post-build capability of the ECUM.	
Particularities and Limitations	
■ This function is mandatory. ■ This function is non re-entrant. ■ This function is always available. ■ This function is configured via configuration tool.	
Call Context	
■ Invoked in `EcuM_MainFunction()`, task context ■ Invoked directly after the wakeup phase	

Table 5-36 EcuM_AL_DriverRestart

5.6.2.8 EcuM_GenerateRamHash

Prototype	
`void EcuM_GenerateRamHash ( void )`	
Parameter	
--	--
Return code	
void	--
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 is still consistent. The RAM check itself must be provided by the integrator.	

Particularities and Limitations
■ This function is mandatory. ■ This function is non re-entrant. ■ This function is always available. ■ This function has to be filled with code by the integrator.
Call Context
■ Invoked in task context ■ Invoked just before putting the ECU into sleep mode

Table 5-37 EcuM_GenerateRamHash

5.6.2.9 EcuM_CheckRamHash

Prototype	
`uint8 EcuM_CheckRamHash ( void )`	
Parameter	
--	--
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 is mandatory. ■ This function is non re-entrant. ■ This function is always available. ■ This function has to be filled with code by the integrator.	
Call Context	
■ Invoked in task context ■ Directly called after the wakeup of the ECU.	

Table 5-38 EcuM_CheckRamHash

5.6.2.10 EcuM_OnRTEStartup

Prototype	
`void EcuM_OnRTEStartup ( void )`	
Parameter	
--	--
Return code	
void	--
Functional Description	
Allows the execution of activities before starting the RTE.	

Particularities and Limitations
■ This function is optional. ■ This function is non re-entrant. ■ This function is available dependently on ECUM_ON_RTESTARTUP_CALLOUT. ■ 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: STARTUPII

Table 5-39 EcuM_OnRTEStartup

5.6.2.11 EcuM_OnEnterRun

Prototype	
void **EcuM_OnEnterRun** (void)	
Parameter	
--	--
Return code	
void	--
Functional Description	
Allows the execution of activities before entering RUN state.	
Particularities and Limitations	
■ This function is optional. ■ This function is non re-entrant. ■ This function is available dependently on ECUM_ON_ENTER_RUN_CALLOUT. ■ This function has to be filled with code by the integrator.	
Call Context	
■ Invoked in task context ■ Called just before entering RUN state.	

Table 5-40 EcuM_OnEnterRun

5.6.2.12 EcuM_OnExitRun

Prototype	
void **EcuM_OnExitRun** (void)	
Parameter	
--	--
Return code	
void	--
Functional Description	
Allows the execution of activities while leaving RUN state.	

Particularities and Limitations
■ This function is optional. ■ This function is non re-entrant. ■ This function is available dependently on ECUM_ON_EXIT_RUN_CALLOUT. ■ This function has to be filled with code by the integrator.
Call Context
■ Invoked in task context ■ Called at the transition from RUN to POST RUN state.

Table 5-41 EcuM_OnExitRun

5.6.2.13 EcuM_OnExitPostRun

Prototype	
`void EcuM_OnExitPostRun ( void )`	
Parameter	
--	--
Return code	
void	--
Functional Description	
Allows the execution of activities while leaving POST RUN state.	
Particularities and Limitations	
■ This function is optional. ■ This function is non re-entrant. ■ This function is available dependently on ECUM_ON_EXIT_RUN_CALLOUT. ■ This function has to be filled with code by the integrator.	
Call Context	
■ Invoked in task context ■ Called while leaving POST RUN state.	

Table 5-42 EcuM_OnExitPostRun

5.6.2.14 EcuM_OnPrepShutdown

Prototype	
`void EcuM_OnPrepShutdown ( void )`	
Parameter	
--	--
Return code	
void	--
Functional Description	
Allows the execution of additional activities in PREP SHUTDOWN state.	

Particularities and Limitations
■ This function is optional. ■ This function is non re-entrant. ■ This function is available dependently on ECUM_ON_PREP_SHUTDOWN_CALLOUT. ■ 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 5-43 EcuM_OnPrepShutdown

5.6.2.15 EcuM_OnGoSleep

Prototype	
`void EcuM_OnGoSleep ( void )`	
Parameter	
--	--
Return code	
void	--
Functional Description	
Allows the execution of additional activities while module is in GO SLEEP state.	
Particularities and Limitations	
■ This function is optional. ■ This function is non re-entrant. ■ This function is available dependently on ECUM_ON_GO_SLEEP_CALLOUT. ■ This function has to be filled with code by the integrator.	
Call Context	
■ Invoked in task context ■ Called after entering GO SLEEP state.	

Table 5-44 EcuM_OnGoSleep

5.6.2.16 EcuM_OnGoOffOne

Prototype	
`void EcuM_OnGoOffOne ( void )`	
Parameter	
--	--
Return code	
void	--
Functional Description	
Allows the execution of additional activities in GO OFF I state.	

Particularities and Limitations
■ This function is optional. ■ This function is non re-entrant. ■ This function is available dependently on ECUM_ON_GO_OFF_ONE_CALLOUT. ■ This function has to be filled with code by the integrator.
Call Context
■ Invoked in task context ■ Called right after entering GO OFF I state.

Table 5-45 EcuM_OnGoOffOne

5.6.2.17 EcuM_OnGoOffTwo

Prototype	
`void EcuM_OnGoOffTwo ( void )`	
Parameter	
--	--
Return code	
void	--
Functional Description	
Allows the execution of additional activities in GO OFF II state.	
Particularities and Limitations	
■ This function is optional. ■ This function is non re-entrant. ■ This function is available dependently on ECUM_ON_GO_OFF_TWO_CALLOUT. ■ This function has to be filled with code by the integrator.	
Call Context	
■ Invoked in task context ■ Called right after entering GO OFF II state.	

Table 5-46 EcuM_OnGoOffTwo

5.6.2.18 EcuM_OnWakeupReaction

Prototype	
`EcuM_WakeupReactionType EcuM_OnWakeupReaction ( EcuM_WakeupReactionType wact )`	
Parameter	
wact	Default wakeup reaction after a controller wakeup.
Return code	
EcuM_WakeupReactionType	ECUM_WKACT_RUN The ECUM should transit to RUN state.

EcuM_WakeupReactionType	ECUM_WKACT_TTII The ECUM should perform the TTII protocol (only possible if globally enabled).
EcuM_WakeupReactionType	ECUM_WKACT_SHUTDOWN The ECUM should transit into SHUTDOWN sequence.

Functional Description

Within this callout the System designer has the possibility to influence the default behavior of the ECU State Manager after a wakeup phase. If desired the integrator has the ability to set the input parameter to a value which is different as the default mechanism.

Caution
It is not recommended to redefine the default values computed by the ECUM.

Particularities and Limitations

- This function is optional.
- This function is non re-entrant.
- This function is available dependently on ECUM_ON_WAKEUP_REACTION_CALLOUT.
- 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 5-47 EcuM_OnWakeupReaction

5.6.2.19 EcuM_EnableWakeupSources

Prototype

```
void EcuM_EnableWakeupSources ( EcuM_WakeupSourceType wakeupSource )
```

Parameter

wakeupSource	Every bit set in the parameter indicates a wakeup source which should be enabled in the current sleep mode. **Info** If the shutdown target is ECUM_STATE_OFF, this callout is executed with parameter value ECUM_WKSOURCE_ALL_SOURCES.

Return code

void	--

Functional Description
This callout will be called whenever RUN state is left. Every bit set in the parameter indicates the wakeup source which should be enabled. The integrator has to take care to implement the necessary activities to enable the corresponding wakeup source. **Info** [1] proposed to execute this callout in GO OFF SLEEP II state. The time between leaving RUN state and the possibility to detect wakeup event may take a relative long long time where wakeup events can be lost. Therefore this implementation executes this callout right after leaving RUN state.
Particularities and Limitations
■ This function is mandatory. ■ This function is non re-entrant. ■ This function is always available. ■ This function has to be filled with code by the integrator.
Call Context
■ Invoked in task context ■ Called when RUN state is left.

Table 5-48 EcuM_EnableWakeupSources

5.6.2.20 EcuM_DisableWakeupSources

Prototype	
`void EcuM_DisableWakeupSources ( EcuM_WakeupSourceType wakeupSource )`	
Parameter	
`wakeupSource`	Every bit set in the parameter indicates a wakeup source which was active in the previous sleep mode. **Info** If the shutdown target is ECUM_STATE_OFF, this callout is executed with parameter value ECUM_WKSOURCE_ALL_SOURCES.
Return code	
void	--
Functional Description	
This callout will be called whenever RUN state is entered from a previous sleep state. Every bit set in the parameter indicates the wakeup source which was enabled in the previous sleep mode.	

Particularities and Limitations
■ This function is mandatory. ■ This function is non re-entrant. ■ This function is always available. ■ This function has to be filled with code by the integrator.
Call Context
■ Invoked in task context ■ Called just before RUN state is entered

Table 5-49 EcuM_DisableWakeupSources

5.6.2.21 EcuM_StartWakeupSources

Prototype	
`void EcuM_StartWakeupSources ( EcuM_WakeupSourceType wakeupSource )`	
Parameter	
wakeupSource	Every bit set in the parameter indicates a wakeup source which is enabled in the current sleep mode.
Return code	
void	--
Functional Description	
The callout shall start the given wakeup source(s) so that they are ready to perform wakeup validation.	

Info
Within [1] is not clear how the parameter which is passed within the ECUM should be derived, hence this implementation passed the enabled wakeup sources for the actual chosen sleep mode. The integrator has to take care about to implement the necessary code in the corresponding wakeup sources.

Particularities and Limitations
■ This function is mandatory. ■ This function is non re-entrant. ■ This function is always available. ■ This function has to be filled with code by the integrator.
Call Context
■ Invoked in task context ■ Called in WAKEUP ONE state

Table 5-50 EcuM_StartWakeupSources

5.6.2.22 EcuM_StopWakeupSources

Prototype	
`void EcuM_StopWakeupSources ( EcuM_WakeupSourceType wakeupSource )`	
Parameter	
wakeupSource	Every bit set in the parameter indicates a wakeup source which is enabled in the current sleep mode.
Return code	
void	
Functional Description	
This callout shall stop the given wakeup source(s) after unsuccessful wakeup validation. Refer to 5.6.2.21 for the information how the parameter is derived.	
Particularities and Limitations	
■ This function is mandatory. ■ This function is non re-entrant. ■ This function is always available. ■ This function has to be filled with code by the integrator.	
Call Context	
■ Invoked in task context ■ Called in WAKEUP WAKESLEEP state	

Table 5-51 EcuM_StopWakeupSources

5.6.2.23 EcuM_CheckWakeup

Prototype	
`void EcuM_CheckWakeup ( EcuM_WakeupSourceType wakeupSource )`	
Parameter	
wakeupSource	ID of a wakeup source.
Return code	
void	--
Functional Description	
This callout shall be called by the ISR of a wakeup source to set up the PLL and check other wakeup sources that may be connected to the same interrupt. Refer to [1] for more information.	
Particularities and Limitations	
■ This function is mandatory. ■ This function is non re-entrant. ■ This function is always available. ■ This function has to be filled with code by the integrator.	
Call Context	
■ Expected to be called in interrupt context.	

Table 5-52 EcuM_CheckWakeup

5.6.2.24 EcuM_CheckValidation

Prototype	
`void EcuM_CheckValidation ( EcuM_WakeupSourceType wakeupSource )`	
Parameter	
wakeupSource	Wakeup IDs of pending wakeup events.
Return code	
void	--
Functional Description	
This callout is called by the ECUM when wakeup validation of a wakeup event is necessary. The pending wakeup event(s) are passed by the parameter in order to allow the necessary reaction depending on the wakeup source.	
Particularities and Limitations	
■ This function is mandatory. ■ This function is non re-entrant. ■ This function is always available. ■ This function has to be filled with code by the integrator.	
Call Context	
■ Invoked in task context ■ Called in WAKEUP VALIDATION state	

Table 5-53 EcuM_CheckValidation

5.6.2.25 EcuM_GeneratorCompatibilityError

Prototype	
`void EcuM_GeneratorCompatibilityError ( uint16 ModuleId, uint8 InstanceId )`	
Parameter	
ModuleId	Module ID where the generator compatibility check failed
InstanceId	Instance ID where the generator compatibility check failed
Return code	
void	--

Functional Description
This callout was introduced additionally to the AUTOSAR standard. Every Vector AUTOSAR component which has the generator compatibility check implemented calls this callout if for example the expected generator version does not match the current used generator version. A detailed description when this callout is executed can be found in the corresponding documentation of the BSW module. The generator compatibility check was introduced by Vector to recognize version inconsistency in post-build configuration data. The check is performed in the modules' Init-function. The integrator has to implement the necessary actions to resolve such version conflicts. **Caution** The default implementation of this callout calls the development error module (usually DET module) if the ECUM is in development mode or enters an endless loop if the ECUM is in production mode. The endless loop is part of template code and shall be adapted by user to meet current system needs.
Particularities and Limitations
■ This function is non re-entrant. ■ The availability of this callout depends on the customer license or the configuration parameter EcuMGeneratorCompatibilityError. ■ This function has to be filled with code by the integrator.
Call Context
■ Expected to be called in application context.

Table 5-54 EcuM_GeneratorCompatibilityError

5.6.2.26 EcuM_Appl_SelectBootTarget

Prototype	
`Std_ReturnType` **`EcuM_Appl_SelectBootTarget`** `( EcuM_BootTargetType target )`	
Parameter	
target	The selected boot target.
Return code	
E_OK	The implementation must return E_OK if no error occurs.
E_NOT_OK	The implementation must return E_NOT_OK if an error occurs.
Functional Description	
This callout was introduced additionally to the AUTOSAR standard. Its intention is to be able to write the necessary code to select the boot target in which the ECU shall boot after the next reset. **Info** The implementation of this callout depends strongly on the used boot loader and memory stack.	

Particularities and Limitations
■ This function has to be filled with code by the integrator.
Call Context
■ Expected to be called in application context.

Table 5-55 EcuM_Appl_SelectBootTarget

5.6.2.27 EcuM_Appl_GetBootTarget

Prototype	
`Std_ReturnType EcuM_Appl_GetBootTarget ( EcuM_BootTargetType *target )`	
Parameter	
target	The current selected boot target.
Return code	
E_OK	The implementation must return E_OK if no error occurs.
E_NOT_OK	The implementation must return E_NOT_OK if an error occurs.
Functional Description	
This callout was introduced additionally to the AUTOSAR standard. Its intention is to be able to write the necessary code to retrieve the selected boot target.	

Info
The implementation of this callout depends strongly on the used boot loader and memory stack.

Particularities and Limitations
■ This function has to be filled with code by the integrator.
Call Context
■ Expected to be called in application context.

Table 5-56 EcuM_Appl_GetBootTarget

5.6.2.28 EcuM_McuSetMode

Prototype	
`void EcuM_McuSetMode ( uint8 mode_u8 )`	
Parameter	
mode_u8	The desired MCU mode.
Return code	
void	-

Functional Description
This callout was introduced additionally to the AUTOSAR standard. Its intention is to be able to implement the Sleep Mode ISR dependent from used HW. **Info** The implementation of this callout depends strongly on the used microcontroller.
Particularities and Limitations
■ This function has to be filled with code by the integrator.
Call Context
■ Expected to be called in application context.

Table 5-57 EcuM_McuSetMode

5.6.3 Mode switch notification functions

5.6.3.1 Appl_EcuM_currentMode_currentMode

Prototype	
`Std_ReturnType Appl_EcuM_currentMode_currentMode(EcuM_ModeType mode)`	
Parameter	
mode	■ ECUM_MODE_POST_RUN ■ ECUM_MODE_RUN ■ ECUM_MODE_SHUTDOWN ■ ECUM_MODE_SLEEP ■ ECUM_MODE_STARTUP ■ ECUM_MODE_WAKE_SLEEP
Return code	
E_OK	The SW-C notified.
E_NOT_OK	The SW-C does not notified the mode.
Functional Description	
This function informs the SW-C about a ECUM mode change.	
Particularities and Limitations	
■ This function is only available if no RTE is used. ■ This function is only available if current mode port is enabled.	
Call context	
■ Expected to be called in application context.	

Table 5-58 Appl_EcuM_currentMode_currentMode

5.6.3.2 Rte_Switch_currentMode_currentMode

Prototype	
`Std_ReturnType Rte_Switch_currentMode_currentMode(Rte_ModeType_EcuM_Mode mode)`	
Parameter	
mode	■ RTE_MODE_EcuM_Mode_POST_RUN ■ RTE_MODE_EcuM_Mode_RUN ■ RTE_MODE_EcuM_Mode_SHUTDOWN ■ RTE_MODE_EcuM_Mode_SLEEP ■ RTE_MODE_EcuM_Mode_STARTUP ■ RTE_MODE_EcuM_Mode_WAKE_SLEEP
Return code	
E_OK	The SW-C notified.
E_NOT_OK	The SW-C does not notified the mode.
Functional Description	
This function informs the SW-C about a ECUM mode change.	
Particularities and Limitations	
■ This function is only available if RTE is used. ■ This function is only available if current mode port is enabled.	
Call context	
■ Expected to be called in application context.	

Table 5-59 Rte_Switch_currentMode_currentMode

5.7 Service Ports

Via Service Ports the software components (SWC) have the possibility to execute services of the ECUM with an abstract RTE interface. Hence, the software components are independent from the underlying basic software stack.

5.7.1 Client Server Interface

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

5.7.1.1 Provide Ports on ECUM side

At the Provide Ports of the ECUM the API functions described in 5.3 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 and the Port Defined Argument Values to be added by the RTE.

5.7.1.1.1 StateRequest Port

Operation	API Function	Port Defined Argument Values
RequestRUN	`EcuM_RequestRUN()`	EcuM_UserType UserId
ReleaseRUN	`EcuM_ReleaseRUN()`	EcuM_UserType UserId
RequestPOSTRUN	`EcuM_RequestPOST_RUN()`	EcuM_UserType UserId
ReleasePOSTRUN	`EcuM_ReleasePOST_RUN()`	EcuM_UserType UserId
GetState	`EcuM_GetStateWrapper()`	EcuM_UserType UserId

Table 5-60 StateRequest Port

Info
The GetState runnable above is mapped to an additional API function `EcuM_GetStateWrapper()` which has to be introduced because of a specification error of the ECUM. This API is not described in chapter 5.3 because the functionality is the same as `EcuM_GetState()`, the only difference is how the current state is given back to the caller. `EcuM_GetStateWrapper()` returns the current state by reference in its arguments.

The user does not have to care about which API name should be used. The generation tools handle the correct API usage.

5.7.1.1.2 ShutdownTarget Port

Operation	API Function	Port Defined Argument Values
SelectShutdownTarget	`EcuM_SelectShutdownTarget()`	-
GetLastShutdownTarget	`EcuM_GetLastShutdownTarget()`	-
GetShutdownTarget	`EcuM_GetShutdownTarget()`	-

Table 5-61 ShutdownTarget Port

5.7.1.1.3 BootTarget Port

Operation	API Function	Port Defined Argument Values
SelectBootTarget	`EcuM_SelectBootTarget()`	-
GetBootTarget	`EcuM_GetBootTarget()`	-

Table 5-62 BootTarget Port

5.7.1.1.4 ApplicationMode Port

Operation	API Function	Port Defined Argument Values
SelectApplicationMode	`EcuM_SelectApplicationMode()`	-

Operation	API Function	Port Defined Argument Values
GetApplicationMode	EcuM_GetApplicationMode()	-

Table 5-63 ApplicationMode Port

Caution
For generating this service port the configuration tool needs a valid OS configuration within the ECUC file. If no information about an OS is found, no service Port of this kind will be generated even though the generation of ApplicationMode port is enabled in the configuration tool (see 6.2.3).

5.7.1.2 Require Ports on ECUM side

At its Require Ports the ECUM calls Operations. These Operations have to be provided by the SWCs by means of Runnable Entities. These Runnable Entities implement the callback functions expected by the ECUM.

The following sub-chapters present the Require Ports defined for the ECUM, the Operations that are called from the ECUM and the related notifications, which are described in chapter 5.6.3.

5.7.1.2.1 currentMode Port

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

Table 5-64 currentMode Port

5.7.2 Sender Receiver Interface

The ECUM does not provides any sender receiver interface

5.8 Software Component Template

The definition of the Provide Ports is described in an XML file. This file describes the ECUM as a software component with ports to which other applications can connect.

5.8.1 Generation by EAD

The XML file can be generated by MICROSAR EAD via the Menu “Tools-> AUTOSAR->SW-C->EcuM”.

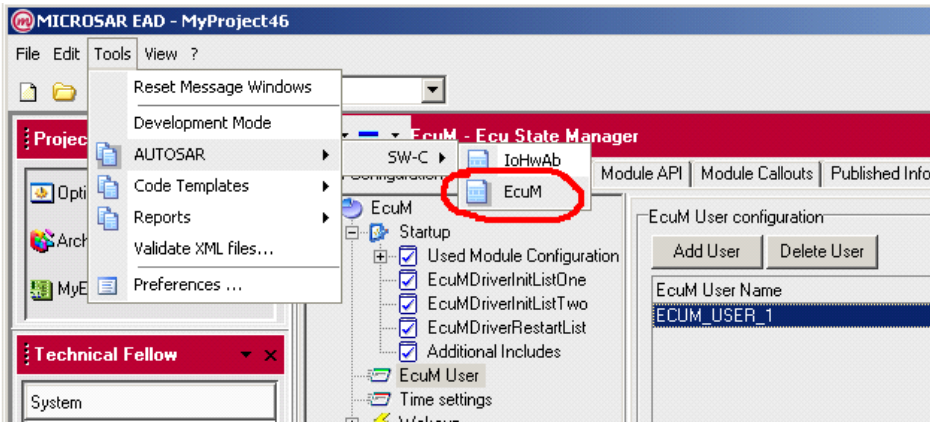

Figure 5-1 Generate an ECUM software component template

5.8.2 Generation by DaVinci Configurator

The XML file will be generated in each generation process and is located at an user specified location. Please consult the documentation of the generation tool how to configure this path.

5.8.3 Import into DaVinci Developer

For further processing the generated software component template file has to be imported into DaVinci Developer. This can be done while a DaVinci-project is open by clicking “File->Import XML File...”. Choose the correct file for the import.

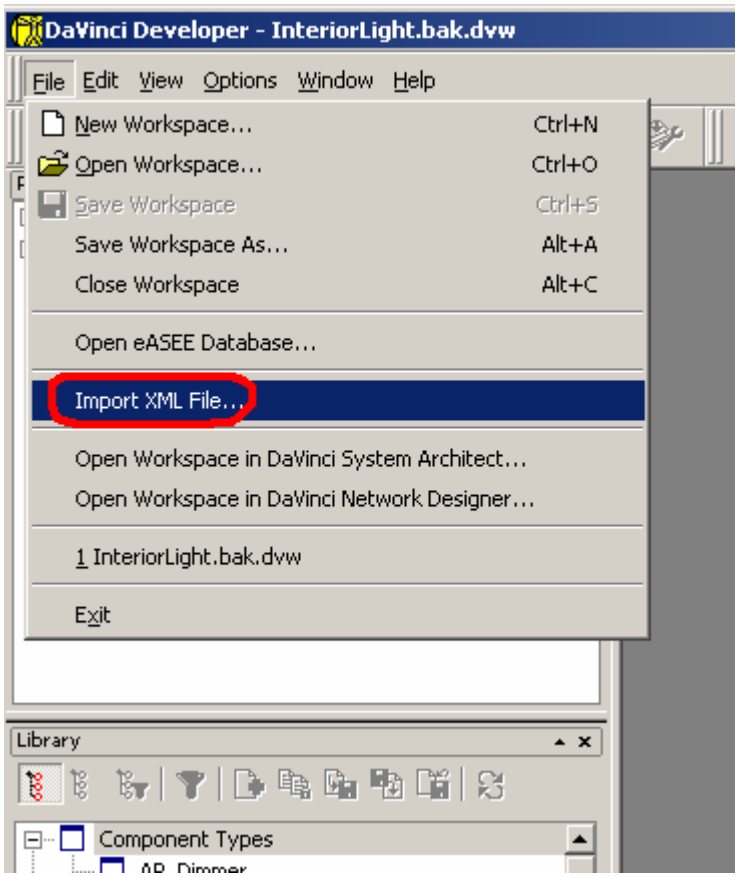

Figure 5-2 Import a new software component into DaVinci Developer

After importing the ECUM as software component there is a new component type in the library view available. After a double click on that component ECUM all configured ports are presented.

6 Configuration

In the ECUM the attributes can be configured with the following tools:

- Configuration in DaVinci Configurator/EAD; for a detailed description see 6.2
- Configuration with a Generic Configuration Editor.
- Configuration in GENy.

6.1 Configuration Variants

The ECUM supports the configuration variants

- `VARIANT-PRE-COMPILE`
- `VARIANT-POST-BUILD`

The configuration classes of the ECUM parameters depend on the supported configuration variants. For their definitions please refer to the modules' bswmd file which is part of the delivered package.

6.2 Configuration of ECUM with DaVinci Configurator/EAD

The ECUM is configured with the help of the configuration tool DaVinci Configurator/EAD.

Info
In case of object delivery modifications are without effect, if a parameter is specified as a pre-compile value. Different object code is needed for different settings. Other object files can be obtained at Vector Informatik.

6.2.1 Start configuration of the ECUM

The component name of the ECU State Manager in DaVinci Configurator/EAD is “EcuM”. In the “Architecture view” (initial page) of the DaVinci Configurator/EAD, the EcuM can be opened by its context menu to start its configuration. Optionally, the EcuM can be opened for configuration with the component list under the “System” tab located at the left side of the DaVinci Configurator/EAD.

6.2.2 ECUM Configuration Perspective

6.2.2.1 Node "Startup"

6.2.2.1.1 Node "Used Module Configuration"

This node represents the configuration options for modules which provide services used by the ECUM. With the following child nodes the integrator has the possibility to specify the header files of the used modules and if the module should be used by the ECUM.

6.2.2.1.2 Node "Mcu"

Attribute Name	Value Type	Values The default value is written in bold	Description
Mcu Include File	header file	include file; default: **Mcu.h**	In this FileEdit element select the MCU driver include file.

Table 6-1 Node "Mcu"

6.2.2.1.3 Node "WdgM"

Attribute Name	Value Type	Values The default value is written in bold	Description
Use WdgM	boolean	ON **OFF**	Enable/disable the use of the Watchdog Manager.
WdgM Include File	header file	include file; default: **WdgM.h**	In this FileEdit element select the WDGM include file.

Table 6-2 Node "WdgM"

6.2.2.1.4 Node "Nvm"

Attribute Name	Value Type	Values The default value is written in bold	Description
Use NVRAM Manager	boolean	ON **OFF**	Enable/disable the use of the NVRAM Manager.
Nvm Include File	header file	include file; default: **Nvm.h**	In this FileEdit element select the NVM include file.
Nvm Types File	header file	include file; default: **Nvm.h**	Select the include file where the types of the NVM are defined.
NvM_CancelWriteAll Timeout	float	0.001..60 **60**	Contains the waiting time for ECUM in seconds. It is the time which defines how long the ECUM will wait for NVM before it cancels the `NvM_WriteAll` job and transits into ECU RUN or power save state (depending from the wakeup validation result).

Table 6-3 Node "Nvm"

6.2.2.1.5 Node "Dem"

Attribute Name	Value Type	Values The default value is written in bold	Description
Use Dem	boolean	ON **OFF**	Enable/disable the use of the DEM.
Dem Include File	header file	include file; default: **Dem.h**	In this FileEdit element select the DEM include file.

Table 6-4 Node "Dem"

6.2.2.1.6 Node "Det"

Attribute Name	Value Type	Values The default value is written in bold	Description
Initialize Det	boolean	ON **OFF**	Enable/disable the use of the DET.
Det Include File	header file	include file; default: **Det.h**	In this FileEdit element select the DET include file.

Table 6-5 Node "Det"

6.2.2.1.7 Node "ComM"

Caution
If COMM is enabled, then a valid configuration of the COMM must be present within the ECUC file. The configuration/generation of the ECUM will not work as expected if no configuration is available.

Attribute Name	Value Type	Values The default value is written in bold	Description
Use COM Manager	boolean	ON **OFF**	This checkbox enables the use of the COM Manager.
ComM Include File	header file	include file; default: **ComM.h**	In this FileEdit element select the COM Manager include file.

Attribute Name	Value Type	Values The default value is written in bold	Description
ComM Callback File	header file	include file; default: **ComM_EcuM.h**	In this FileEdit element select the call back header file of the COM Manager.
Com Stack Types Include File	header file	Include file; default: ComStack_Types.h	This parameter specifies the name of the include file of the basic type definitions for the communication stack.

Table 6-6 Node "ComM"

6.2.2.1.8 Node "Rte"

Attribute Name	Value Type	Values The default value is written in bold	Description
Use Rte	boolean	**ON** OFF	Enable/disable the use of the Rte.
Rte Include File	header file	include file; default: **Rte_Main.h**	In this FileEdit element select the Rte include file.

Table 6-7 Node "Rte"

6.2.2.1.9 Node "Os"

Attribute Name	Value Type	Values The default value is written in bold	Description
EcuMDefaultApp Mode	AppModeType	Depends on OS configuration; default: **OSDEFAULTAPPMODE**	This Edit field must be filled with the application mode which should be used as default.
Os include name	header file	include file; default: **Os.h**	Select with this FileEdit the name of the header file of the OS.

Table 6-8 Node "Os"

6.2.2.1.10 Node "SchM"

Attribute Name	Value Type	Values The default value is written in bold	Description
Use SchM	boolean	ON **OFF**	Enables/disable the use of the SCHM.
SchM include File	header file	include file; default: **SchM.h**	In this FileEdit element select the SCHM include file.

Table 6-9 Node "SchM"

6.2.2.1.11 Node “Init Configuration Sets”

Attribute Name	Value Type	Values The default value is written in bold	Description
Insert	--	--	Inserts a new configuration set item.
Delete	--	--	Deletes the selected configuration set item.
Module	String parameter	Depends on the available modules within the configuration	The Module short name of the module which should be initialized.
ModuleRuntime	Reference parameter to a configuration set	Depends on the available modules within the configuration	The corresponding configuration set which should be used for the module initialization.

Table 6-10 Node “Init Configuration Sets”

6.2.2.1.12 Nodes "EcuMDriverInitListOne", "EcuMDriverInitListTwo", “EcuMDriverInitListThree”, "EcuMDriverRestartList"

Attribute Name	Value Type	Values The default value is written in bold	Description
Insert	--	**--**	This button inserts a new BSW module into the table.
Delete	--	**--**	This button deletes the current selected BSW module

Attribute Name	Value Type	Values The default value is written in bold	Description
Move Up	--	**--**	This button moves the current selected BSW module one row up. The order of the modules influences the order in which the Init function will be executed.
Move Down	--	**--**	This button moves the current selected BSW module one row below. The order of the modules influences the order in which the Init function will be executed.
ModuleID (Table column)	--	**--** **Code**	In this combobox modules which are present in the ecuc configuration file are displayed. Enter the short name of the module to be initialized. Enter the keyword “Code” here to enable the field Code below.
ModuleID (Edit field)	--	**--**	Display of the current edited table item.
Init Function	C-function identifier	**--**	This parameter specifies the Init function which should be used for initialization of the corresponding module.
Code	String parameter	**--**	This field allows adding extra code which is necessary for the system initialization. **Info** It is not indented to use this field as typical source code editor. The insertion of one line is proposed.

Table 6-11 Nodes “EcuMDriverInitListOne”, “EcuMDriverInitListTwo”, “EcuMDriverInitListThree”, “EcuMDriverRestartList”

6.2.2.1.13 Node Additional Includes

Attribute Name	Value Type	Values The default value is written in bold	Description
Include file list	String Parameter	**--**	This parameter specifies additional files to be included into EcuM_PBcfg.c, e.g. header files where the Init() – functions are declared must be specified here.

Table 6-12 Node "Additional Includes"

6.2.2.2 Node "EcuM User"

Attribute Name	Value Type	Values The default value is written in bold	Description
Add User	--	**--**	This button adds a user of the ECU State Manager which is able to request and release RUN or POST RUN state.
Delete User	--	--	This button deletes the selected user of the ECU State Manager.
Container Name	String Parameter	**EcuMUserConfig**	Fill this field with unique ECU State Manager user names.
EcuMUserId	Integer	1..255 **1**	The ID of the corresponding user, this ID is the port defined argument described in 5.7.1.1.1.

Table 6-13 Node "EcuM User"

Info
For each configured user one service port will be defined. The name of the service port corresponds exactly to the name provided here.

6.2.2.3 Node "Time settings"

Attribute Name	Value Type	Values The default value is written in bold	Description
Run Self Request Period	Float Unit is second	**0.100**	How long the self run request should last must be filled in this Edit field expressed in seconds.
EcuM_ MainFunction() Trigger Time	Float Unit is second	**0.010**	Select with this Edit the cycle time of the `EcuM_MainFunction()` in seconds.

Table 6-14 Node "Time settings"

6.2.2.4 Node "Wakeup"

Attribute Name	Value Type	Values The default value is written in bold	Description
Insert	--	--	This button adds a new wakeup source
Delete	--	--	This button deletes the selected wakeup source. Per default five wakeup sources are already inserted which it is not possible to delete them.

Attribute Name	Value Type	Values The default value is written in bold	Description
Move Up	--	--	This button moves the selected entry one row up.
Move Down	--	--	This button moves the selected entry down.
Wakeup Source Name	String parameter	--	This field should be filled with a valid name for the wakeup source.
Wakeup Source Id	Integer	--	The ID of the corresponding wakeup source.
Validation Timeout	Float, unit is second	**0.000**	This parameter specifies the maximum validation timeout for a wakeup source. If the timeout expires the ECU is going into sleep state. Wakeup sources which don't need validation must have a value of 0.
ComM Channel handle	--		Select the ComM channel handle if the wakeup source if a ComM channel. If the wakeup source is no communication channel don't select any value.
Mcu Reset Reason	Integer	**-1** (these values depend on Mcu module implementation)	This parameter describes the mapping of reset reasons detected by the MCU driver into wakeup sources. Please enter the value returned by Mcu_GetResetReason() which should mapped to the according wakeup source. Let this field free or enter "-1" if the value is unknown. If no value is configured or "-1" is inserted the reset reason will be mapped to ECUM_WKSOURCE_RESET.

Table 6-15 Node "Wakeup Sources"

6.2.2.5 WdgM Modes

Attribute Name	Value Type	Values The default value is written in bold	Description
WdgM Post Run Mode	Reference to WDGM mode	**--**	This WDGM mode will be selected whenever POST RUN state is entered.
WdgM Run Mode	Reference to WDGM mode	**--**	This WDGM mode will be selected whenever RUN state is entered.

Attribute Name	Value Type	Values The default value is written in bold	Description
WdgM Shutdown Mode	Reference to WDGM mode	**--**	This WDGM mode will be selected whenever SHUTDOWN phase is entered.
WdgM Wakeup Mode	Reference to WDGM mode	**--**	This WDGM mode will be selected whenever WAKEUP VALIDATION state is entered.

Table 6-16 Node “WdgM Modes”

6.2.2.6 Node “Sleep Modes”

This Node does not directly display any configuration possibilities. A right click on this node will open a context menu, where sleep modes can be configured. The new configuration will be shown as a child node of “Sleep Modes”.

A right click on the child node opens a context menu, where items can be deleted or renamed.

6.2.2.7 Child Nodes of “Sleep Modes”

Attribute Name	Value Type	Values The default value is written in bold	Description
Name	--	**--**	Display the name of the current sleep mode configuration
EcuMSleepModeId	Integer	**0** 0..255	This parameter specifies the ID of the sleep mode.
Mcu Mode	--	Depends on MCU configuration	This parameter specifies the MCU mode which should the ECUM select when the sleep mode is entered.
WdgM mode	--	Depends on WDGM configuration	This parameter specifies the WDGM mode which should be selected by the ECUM when the sleep mode is entered.
Enabled Wakeup Sources	--	Depends on configured wakeup sources	This parameter specifies the wakeup sources which should be enabled when the sleep mode is selected.

Table 6-17 Child Nodes of “Sleep Modes”

6.2.2.8 Node “TTII”

Attribute Name	Value Type	Values The default value is written in bold	Description
Enable TTII	--	ON **OFF**	This checkbox enables the use of the TTII protocol.
TTII Wakeup Source	Reference parameter to configured wakeup sources	Depends on the wakeup source configuration. Default: **ECUM_WKSOURCE_POWER**	Select in this DropDownList the wakeup source which should act as wakeup source for TTII. **Info** The selected wakeup source should be a timer.
Insert	--	--	This Button inserts a new successor sleep mode for TTII use.
Delete	--	**--**	This Button deletes the selected sleep mode from the table.
EcuMSuccessor	Reference parameter to configured sleep modes.	Depends on the sleep mode configuration.	A click into the field opens a DropDownList of the configured sleep mode. Select one from the list. The TTII will use each sleep mode from top to bottom in this list. Therefore order this list from more power consuming to less power consuming. **Caution** The integrator has the responsibility to order this list from more power consuming to less power consuming.
EcuMDivisor	Integer	1	This field must be filled with the divisor value for this sleep mode.

Table 6-18 Node “TTII”

6.2.2.9 Node “Shutdown”

Attribute Name	Value Type	Values The default value is written in bold	Description
Default State	EcuM_StateType	Sleep **Reset** Off	This parameter specifies the default shutdown target state.

Attribute Name	Value Type	Values The default value is written in bold	Description
Default Sleep Mode	--		This parameter specifies the sleep mode which should be selected when the default shutdown target state is “Sleep”.

Table 6-19 Node “Shutdown”

6.2.3 General Settings Perspective

Attribute Name	Value Type	Values The default value is written in bold	Description
Enable Development Error Detection	boolean	**ON** OFF	It is the main switch over following buttons: ■ Check Parameter ■ Check Uninit ■ Det Errorhook include file ■ Det Errorhook Function The checks simplify the search for errors. In production mode, this switch should be disabled to save RAM and to speed up the module.
Check Parameter	boolean	**ON** OFF	This check is the main switch over: ■ Check Pointer Parameter ■ Check State Parameter ■ Check Wakeup Source Parameter ■ Check BootTarget Parameter ■ Check SleepMode Parameter ■ Check User Parameter ■ Check ConfigType
Check Pointer Parameter	boolean	**ON** OFF	If the check is activated, it will be verified that a pointer parameter is not equal to `NULL_PTR`.
Check State Parameter	boolean	**ON** OFF	If the check is activated, it will be verified that a parameter of type EcuM_StateType is within the expected range.
Check Wakeup Source Parameter	boolean	**ON** OFF	If the check is activated, it will be verified that a parameter of Type EcuM_WakeupSourceType is within the expected range.
Check BootTarget Parameter	boolean	**ON** OFF	If the check is activated, it will be verified that a parameter of Type EcuM_BootTargetType is within the expected range.
Check SleepMode Parameter	boolean	**ON** OFF	If the check is activated, it will be verified that a parameter of Type EcuM_SleepModeType is within the expected range.
Check User Parameter	boolean	**ON** OFF	If the check is activated, it will be verified that a parameter of Type EcuM_UserType is within the expected range.
Check ConfigType	boolean	**ON**	If the check is activated, it will be verified that the

Attribute Name	Value Type	Values The default value is written in bold	Description
		OFF	configuration structure of ECUM is not a NULL_PTR.
Det Errorhook Function	C-function identifier	Valid C-function identifier, default: **Det_ReportError**	Defines the function that is to be called in case of a Development Error.
Det Include File	header file	Valid header file, default: **Det.h**	Include this header to get the declaration of Det_ReportError.
Enable Production Error Reporting	boolean	**ON** OFF	Enable this identifier in production mode, for reporting production's errors.
Dem Errorhook function	C-function identifier	Valid C-function identifier, default: **Dem_ReportErrorStatus**	Defines the function that is to be called in case of a production error.
Dem Errorhook include file	header file	Valid header file, default: **Dem.h**	Select the name of the include file where the used DEM is declared.
Critical Section Handling	--	**UseSuspendFunctions** UseOSFunctions UseEnableFunctions	Specifies the function set that shall be called when critical sections are entered or left. **Info** This dropdown list is disabled if the BSW Scheduler is enabled within the OS configuration tab of the ECU configuration. Hence, the functions of the Schedule Manager are used by the ECU State Manager to protect the critical sections.
Generate StateRequest Ports	--	**ON** OFF	Enables the generation of StateRequest Ports. If enabled for each ECUM user a StateRequest port will be generated.
Generate ApplicationMode Port	--	**ON** OFF	Enables the generation of the ApplicationMode Port. If enabled one Port of this kind will be generated.
Generate BootTarget Port	--	**ON** OFF	Enables the generation of the BootTarget Port. If enabled one Port of this kind will be generated.

Attribute Name	Value Type	Values The default value is written in bold	Description
Generate ShutdownTarget Port	--	**ON** OFF	Enables the generation of the ShutdownTarget Port. If enabled one Port of this kind will be generated.
Generate Current Mode Port	--	ON **OFF**	This parameter enables the generation of the EcuM_CurrentMode sender-receiver interface within the SW-C description and the usage of a callback function for mode switch notification. The used notification function depends on the ECUM parameter EcuMIncludeRte: ■ If this parameter is set to ON the ECUM will use the provided RTE function Rte_Switch_currentMode_currentMode(). ■ If the parameter EcuMIncludeRte is set to OFF the callback function EcuM_Appl_currentMode_currentMode() will be used to notify mode switch event.
Enables the Acknowledge Mechanism	--	**ON** OFF	This parameter enables the feedback mechanism from RTE. The ECUM leaves its main states (RUN, POST_RUN, GO_SLEEP and GO_OFF_ONE) only in case the RTE achieves the current state and has informed the ECUM about it by sending a feedback. This parameter is dependent from “Generate Current Mode Port” parameter. **Info** This parameter is used for synchronization between RTE and ECUM.
Service Component Name	--	service component 's name; default: **EcuM**	In this field specify the Name of the service component which will be displayed in the Rte generation tool. **Info** Please choose different names of the service component ECUM if one DaVinci project has more than one ECUM.
Sleep mode ISR handling	--		This parameter should reference an appropriate function set which disables/enables the global interrupts. **Caution** Do not use an OS function for this parameter

Attribute Name	Value Type	Values The default value is written in bold	

Attribute Name	Value Type	Values The default value is written in bold	Description
Enable EcuM_OnExitRun	boolean	**ON** OFF	Use of callout `EcuM_OnExitRun()` will be enabled.
Enable EcuM_OnExitPost Run	boolean	**ON** OFF	Use of callout `EcuM_OnExitPostRun()` will be enabled.
Enable EcuM_OnPrep Shutdown	boolean	**ON** OFF	Use of callout `EcuM_OnPrepShutdown()` will be enabled.
Enable EcuM_OnGoSleep	boolean	**ON** OFF	Use of callout `EcuM_OnGoSleep()` will be enabled.
Enable EcuM_OnGoOffOne	boolean	**ON** OFF	Use of callout `EcuM_OnGoOffOne()` will be enabled.
Enable EcuM_OnGoOffTwo	boolean	**ON** OFF	Use of callout `EcuM_OnGoOffTwo()` will be enabled.
Enable EcuM_OnWakeupRe action	boolean	**ON** OFF	Use of callout `EcuM_OnWakeupReaction()` will be enabled.
Enable EcuM_GeneratorCo mpatibilityError	boolean	ON **OFF**	Use of callout `EcuM_GeneratorCompatibilityError()` will be enabled.

Table 6-22 Module Callouts

7 AUTOSAR Standard Compliance

7.1 Deviations

7.1.1 Service Port Generation

Export of SWC Description assumes an AppModeType with 8 bit data width (uint8).

7.1.2 Supervised Entity in WDGM

The configuration of the ECUM as a supervised entity of the WDGM is not supported.

7.1.3 Parameter “EcuMWdgMStartupModeRef” not supported

The configuration parameter EcuMWdgMStartupModeRef is not supported.

7.1.4 Signature and name of DriverInitLists in configuration variant “post-build”

The method signature of the callouts

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

are different as specified in [1] to provide more flexibility and library capability.

These callouts are replaced by function pointers stored in the EcuM_ConfigType. The configuration is transparent to the user.

7.1.5 Enabling/Disabling wakeup sources

The callout EcuM_EnableWakeupSources() is executed whenever RUN state is left and shutdown target is ECUM_STATE_SLEEP or ECUM_STATE_OFF.

The callout EcuM_DisableWakeupSources() is executed whenever RUN state is entered and the shutdown target is ECUM_STATE_SLEEP or ECUM_STATE_OFF.

7.1.6 Execution flow in shutdown sequence

The ECUM SWS specifies the following execution flow when shutdown target ECUM_-STATE_OFF is selected:

- Dem_Shutdown()
- Rte_Stop()
- ComM_DeInit()
- NvM_WriteAll()
- Wait until NvM_WriteAll() has finished.

- ShutdownOS()

To enable a fast reaction time for wakeup events while NVRAM data is written this ECUM implementation changed this execution flow into the following sequence:

- Dem_Shutdown()
- NvM_WriteAll()
- Wait until NvM_WriteAll() has finished
- Rte_Stop()
- ComM_DeInit()
- ShutdownOS()

7.2 Additions/ Extensions

7.2.1 Parameter Checking

The internal parameter checks of the API functions can be en-/disabled separately. The AUTOSAR standard requires en-/disabling of the complete parameter checking only. For details see chapter 3.6.1.1.

7.2.2 State Request service port

The ECUM provides an additional operation in the StateRequest port. The current state of the ECUM can be retrieved by the GetState operation.

7.2.3 Wakeup validation in each state

This ECUM implementation provides the possibility to perform a wakeup validation in each state, especially the validation of communication channels while ECUM is in RUN state.

7.2.4 Callout EcuM_GeneratorCompatibilityError()

This callout is additional to the AUTOSAR standard. Further information can be found in chapter 5.6.2.25.

7.2.5 Buffering of wakeup events until COMM is initialized

The ECUM buffers wakeup events until it is assured that COMM is initialized. Wakeup events on communication channels are propagated to COMM module. The AUTOSAR standard can not require that COMM module is already initialized when a wakeup event occur in startup phase, because of the different initialization phases and component modularization. After finishing `DriverInitListThree` all wakeup events on communication channels are reported to COMM.

7.3 Limitations

7.3.1 Link-time Configuration not supported

The ECUM module supports only post-build and pre-compile configuration.

7.3.2 Configuration Check via Consistency Hash not implemented

The consistency check as proposed by AUTOSAR is not implemented because this feature is not clearly stated enough.

7.3.3 Not implemented Callouts

The callouts `EcuM_Errorhook()` and `EcuM_SleepActivity()` are currently not supported.

7.3.4 References in ECUC File

The ECUM uses references within the ECUC file. If one of the referenced modules does not support the ECUC file the DaVinci Configurator/EAD generates a warning. Referenced modules are: OS, COMM, WDGM, MCU and BSW modules which should be initialized.

7.3.5 Not supported configuration parameters

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 modules bswmd file which is part of the delivery.

7.3.6 Polling of Wakeup Sources Is Not Supported

This ECUM implementation does not support the polling of wakeup sources, i.e. the callout EcuM_CheckWakeup() will not be executed by the ECUM itself to query the wakeup sources if a wakeup event has occurred.

8 Glossary and Abbreviations

8.1 Glossary

<table>
<tr><th>Term</th><th>Description</th></tr>
<tr><td>EAD</td><td>Embedded Architecture Designer; generation tool for MICROSAR components</td></tr>
<tr><td>DaVinci</td><td>The DaVinci Tool Suite is a design environment for the development of distributed electronic systems in the automotive domain. It consists of several tools, which may be used standalone or in combination. The DaVinci Tool Suite provides you with a seamless support throughout the development process. The functionality goes from design of distributed functions to the definition of network communication to the application development, configuration and generation (since DaVinci Configurator Pro) of networked ECUs.
For each working step a dedicated tool is available:
<table>
<tr><th>Tool</th><th>Description</th></tr>
<tr><td>DaVinci System Architect</td><td>With DaVinci System Architect you may define the distributed SW architecture of vehicles according to the AUTOSAR methodology.</td></tr>
<tr><td>DaVinci Network Designer</td><td>With the DaVinci Network Designer tools you design the network architecture of the vehicles and the data communication on the networks. The networking data are the base for subsequent development steps like system simulation, configuration of the ECU software as well as ECU and system integration tests.</td></tr>
<tr><td>DaVinci Developer</td><td>During the ECU development you may use DaVinci Developer to design AUTOSAR compliant applications and to integrate them based on an AUTOSAR RTE (runtime environment).</td></tr>
<tr><td>DaVinci Configurator Pro</td><td>For MICROSAR components configuration and generation you may use the DaVinci Configurator Pro. It is the successor from EAD.</td></tr>
</table>
</td></tr>
<tr><td>DriverInitList</td><td>A list of basic software init functions which are executed by the EcuM in the startup phase. The ECUM provides four DriverInitList’s:
■ EcuMDriverInitListZero
■ EcuMDriverInitListOne
■ EcuMDriverInitListTwo
■ EcuMDriverInitListThree</td></tr>
</table>

MSN	Module Short name, the AUTOSAR short name of the module, e.g. Can, CanIf, EcuM, etc…
GENy	Generation tool for CANbedded and MICROSAR components
Configuration Tool	Could be EAD, DaVinci Configurator or GENy in this document.

Table 8-1 Glossary

8.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
CAN	Controller Area Network
COMM	Communication Manager
DEM	Diagnostic Event Manager
DET	Development Error Tracer
EAD	Embedded Architecture Designer
ECU	Electronic Control Unit
EcuC	ECU configuration description
ECUM	ECU Manager
GPT	General Purpose Timer
HIS	Hersteller Initiative Software
ICU	Input Capture Unit
ISR	Interrupt Service Routine
MCU	Microprocessor Control Unit
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
NVM	NVRAM Manager
NVRAM	Non Volatile Random Access Memory
OS	Operating System
Pport	Provide Port
Rport	Require Port
RTE	Runtime Environment
SCHM	Scheduling Manager
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification
WDGM	Watchdog Manager
PLL	Phase-locked loop
TTII	Time Triggered Increased Inoperation

MSN	Module Short Name

Table 8-2 Abbreviations

9 Contact

Visit our website for more information on

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

www.vector-informatik.com