| **Document Title** | Specification of ECU State Manager |
|---|---|
| **Document Owner** | AUTOSAR GbR |
| **Document Responsibility** | AUTOSAR GbR |
| **Document Version** | 1.1.0 |
| **Document Status** | Draft |
| **Part of Release** | 2.1 |
| **Revision** | 0014 |

# Document Change History

| Date | Ver. | Changed by | Change Description |
|---|---|---|---|
| 31.01.2007 | 1.1.0 | AUTOSAR Administration | • Corrected startup flow and wakeup concept.<br>• Added specification for AUTOSAR ports.<br>• Modified configuration for compliance with variant management.<br>• Added new API services.<br><br>• Legal disclaimer revised<br>• Release Notes added<br>• "Advice for users" revised<br>• "Revision Information" added |
| 28.06.2006 | 1.0.0 | AUTOSAR Administration | Initial release |

# Release Notes

## Errata and known deficiencies

The wakeup concept is currently neither harmonized nor consistent throughout all wakeup related specification documents and therefore subject to change.

## Known and potential problems resulting from known deficiencies

Due to the fact that the wakeup concept is not harmonized, inconsistent assumptions may lead to

- the duplication of functionalities across multiple modules
- proprietary implementation extensions
- difficulties during integration

## Changes planned for next release

The harmonized wakeup concept throughout all wakeup related documents will result in

- adapted specification texts in Chapter 7 of the specification documents
- adapted APIs in Chapter 8 of the specification documents
- adapted wakeup sequences in Chapter 9 of the specification documents

**Disclaimer**

**Any use** of these specifications requires membership within the AUTOSAR Development Partnership or an agreement with the AUTOSAR Development Partnership. The AUTOSAR Development Partnership will not be liable for any use of these specifications.

Following the completion of the development of the AUTOSAR specifications commercial exploitation licenses will be made available to end users by way of written License Agreement only.

No part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the publisher.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Copyright © 2004-2006 AUTOSAR Development Partnership. All rights reserved.

**Advice to users of AUTOSAR Specification Documents:**

AUTOSAR Specification Documents may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).
Any such exemplary items are contained in the Specification Documents for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such Specification Documents, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

# Table of Contents

Release Notes ........ 2
Errata and known deficiencies ........ 2
Known and potential problems resulting from known deficiencies ........ 2
Changes planned for next release ........ 2
1 Introduction ........ 11
1.1 Functional Overview ........ 11
1.2 Conventions Used in this Specification ........ 12
1.2.1 Font Faces ........ 12
1.2.2 Figures ........ 12
2 Definitions and Acronyms ........ 14
3 Related documentation ........ 15
3.1 Input documents ........ 15
3.2 Related standards and norms ........ 15
3.3 Related AUTOSAR Software Specifications ........ 15
4 Constraints and Assumptions ........ 17
4.1 Limitations ........ 17
4.2 Hardware Requirements ........ 17
4.3 Applicability to car domains ........ 17
The ECU State Manager is applicable to all car domains. ........ 17
5 Dependencies to other modules ........ 18
5.1 Mode Management Modules ........ 18
5.1.1 Communication Manager ........ 18
5.1.2 Watchdog Manager ........ 18
5.2 SPAL Modules ........ 18
5.2.1 MCU Driver ........ 18
5.2.2 Driver Dependencies and Initialization Order ........ 19
5.3 Peripherals with Wakeup Capability ........ 19
5.4 Operating System ........ 19
5.5 Runtime Environment (RTE) ........ 20
5.6 NVRAM Manager ........ 20
5.7 Diagnostic Event Manager ........ 20
5.8 Software Components ........ 20
5.9 File Structure ........ 21
6 Requirements traceability ........ 22
7 Functional Specification ........ 29
7.1 Main States of the ECU State Manager ........ 29
7.1.1 STARTUP State ........ 30
7.1.2 RUN State ........ 30
7.1.3 SHUTDOWN State ........ 30
7.1.4 SLEEP State ........ 31
7.1.5 WAKEUP State ........ 31

7.1.6 OFF State ... 31
7.1.7 Reset Behavior and Leaving States ... 31
7.2 Structural Description of the ECU State Manager ... 32
7.2.1 Standardized AUTOSAR Software Modules ... 32
7.2.2 Software Components ... 33
7.2.3 Resource Managers ... 33
7.3 STARTUP State ... 34
7.3.1 High Level Sequence Diagram ... 34
7.3.2 Activities before EcuM_Init ... 35
7.3.3 STARTUP Activity Overview ... 35
7.3.4 Sub-State Descriptions ... 37
7.3.4.1 STARTUP I ... 37
7.3.4.2 STARTUP II ... 39
7.3.5 Driver Initialization ... 40
7.3.6 DET Initialization ... 41
7.4 RUN State ... 41
7.4.1 State Breakdown Structure ... 41
7.4.2 High Level Sequence Diagram ... 42
7.4.3 Sub-State Description ... 43
7.4.3.1 RUN II ... 43
7.4.3.2 Entering RUN II State ... 44
7.4.3.3 Leaving RUN II State ... 44
7.4.3.4 RUN III ... 45
7.4.3.5 Leaving RUN III State ... 45
7.5 SHUTDOWN State ... 46
7.5.1 State Breakdown Structure ... 46
7.5.2 High Level Sequence Diagram ... 47
7.5.3 SHUTDOWN Activity Overview ... 48
7.5.4 Sub-State Descriptions ... 49
7.5.4.1 PREP SHUTDOWN ... 49
7.5.4.2 GO SLEEP ... 50
7.5.4.3 GO OFF I ... 52
7.5.4.4 GO OFF II ... 53
7.6 SLEEP State ... 53
7.6.1 Shutdown Targets ... 53
7.6.2 Leaving SLEEP State ... 54
7.7 WAKEUP State ... 55
7.7.1 High Level Sequence Diagram ... 55
7.7.2 State Breakdown Structure ... 56
7.7.3 WAKEUP Activity Overview ... 57
7.7.4 Sub-State Descriptions ... 58
7.7.4.1 WAKEUP I ... 58
7.7.4.2 WAKEUP VALIDATION ... 59
7.7.4.3 WAKEUP REACTION ... 60
7.7.4.4 WAKEUP II ... 61
7.8 Wakeup Validation Protocol ... 61
7.8.1 Wakeup of Communication Channels ... 61
7.8.2 Wakeup of the Entire ECU ... 62
7.8.3 Interaction of Wakeup Sources and the ECU State Manager ... 62
7.8.4 WakeupValidation Timout Timer ... 63

7.8.5 Requirements for Drivers with Wakeup Sources ................................ 63
7.8.6 Requirements for Wakeup Validation ................................ 64
7.8.7 Wakeup Sources and Reset Reason ................................ 64
7.8.8 Wakeup Sources with Integrated Power Control ................................ 64
7.8.9 Activity Diagram ................................ 65
7.8.10 Example: Application of Wakeup Validation Protocol for GPT ............ 66
7.8.10.1 Example 1: Wakeup from CAN ................................ 68
7.8.10.2 Example 2: Wakeup From ICU ................................ 69
7.8.10.3 Example 3: Wakeup from ICU ................................ 70
7.9 Time Triggered Increased Inoperation ................................ 71
7.10 AUTOSAR Ports ................................ 72
7.10.1 Scope of this Chapter ................................ 72
7.10.2 Overview ................................ 72
7.10.3 Use Cases ................................ 72
7.10.4 Specification of the Port Interfaces ................................ 73
7.10.4.1 Port Interface for Interface State Request ................................ 73
7.10.4.1.1 General Approach ................................ 73
7.10.4.1.2 Data Types ................................ 73
7.10.4.1.3 Port Interface ................................ 74
7.10.4.2 Port Interface for Interface Current Mode ................................ 74
7.10.4.2.1 General Approach ................................ 74
7.10.4.2.2 Data Types ................................ 76
7.10.4.2.3 Port Interface ................................ 76
7.10.4.3 Ports and Port Interface for Interface Shutdown Target ............... 76
7.10.4.3.1 General Approach ................................ 76
7.10.4.3.2 Data Types ................................ 76
7.10.4.3.3 Port Interface ................................ 77
7.10.4.4 Port Interface for Interface Boot Target ................................ 77
7.10.4.4.1 General Approach ................................ 77
7.10.4.4.2 Data Types ................................ 77
7.10.4.4.3 Port Interface ................................ 78
7.10.4.5 Port Interface for Interface Application Mode ................................ 78
7.10.4.5.1 General Approach ................................ 78
7.10.4.5.2 Data Types ................................ 78
7.10.4.5.3 Port Interface ................................ 78
7.10.5 Summary of ports ................................ 78
7.10.5.1 Definitions of interfaces ................................ 78
7.10.5.2 Definition of the Service ECU State Manager ................................ 81
7.10.6 Runnables and Entry points ................................ 81
7.10.6.1 Internal behavior ................................ 81
7.10.6.2 Entry points of Runnables ................................ 82
7.11 Advanced Topics ................................ 83
7.11.1 Application Modes ................................ 83
7.11.2 Relation to Bootloader ................................ 84
7.11.3 Relation to Complex Drivers ................................ 84
7.11.4 Handling Errors during Startup and Shutdown ................................ 84
7.11.5 Configuration Alternative for Providing Wake-Sleep Operation ........... 84
7.11.6 Selecting Scheduling Schemes for Startup and Shutdown ................. 85
7.12 Error Classification ................................ 86

8 API specification ................................ 87

8.1 Imported Types ... 87
8.1.1 Standard Types ... 87
8.1.2 Communication Manager Types ... 87
8.1.3 Watchdog Manager Types ... 87
8.1.4 NVRAM Manager Types ... 87
8.1.5 MCU Driver Types ... 87
8.1.6 OS Types ... 87
8.2 Type definitions ... 88
8.2.1 EcuM_ConfigType ... 88
8.2.2 EcuM_StateType ... 88
8.2.3 EcuM_UserType ... 89
8.2.4 EcuM_WakeupSourceType ... 89
8.2.5 EcuM_WakeupStatusType ... 90
8.2.6 EcuM_WakeupReactionType ... 90
8.2.7 EcuM_SleepModeConfigType ... 91
8.2.8 EcuM_BootTargetType ... 91
8.2.9 EcuM_ChannelHandleType ... 91
8.3 Function Definitions ... 91
8.3.1 Initialization and Shutdown ... 91
8.3.1.1 EcuM_Init ... 91
8.3.1.2 EcuM_Shutdown ... 92
8.3.1.3 EcuM_GetVersionInfo ... 92
8.3.2 State Management ... 93
8.3.2.1 EcuM_RequestRUN ... 93
8.3.2.2 EcuM_ReleaseRUN ... 93
8.3.2.3 EcuM_ComM_RequestRUN ... 94
8.3.2.4 EcuM_ComM_ReleaseRUN ... 94
8.3.2.5 EcuM_ComM_HasRequestedRUN ... 95
8.3.2.6 EcuM_RequestPOST_RUN ... 95
8.3.2.7 EcuM_ReleasePOST_RUN ... 96
8.3.2.8 EcuM_KillAllRUNRequests ... 96
8.3.2.9 EcuM_SelectShutdownTarget ... 97
8.3.2.10 EcuM_GetState ... 97
8.3.2.11 EcuM_GetShutdownTarget ... 98
8.3.2.12 EcuM_GetLastShutdownTarget ... 98
8.3.3 Wakeup Notifications ... 99
8.3.3.1 EcuM_SetWakeupEvent ... 99
8.3.3.2 EcuM_GetPendingWakeupEvents ... 100
8.3.3.3 EcuM_ClearWakeupEvent ... 100
8.3.3.4 EcuM_ValidateWakeupEvent ... 100
8.3.3.5 EcuM_GetValidatedWakeupEvents ... 101
8.3.3.6 EcuM_GetExpiredWakeupEvents ... 102
8.3.3.7 EcuM_GetStatusOfWakeupSource ... 102
8.3.4 Miscellaneous ... 103
8.3.4.1 EcuM_SelectApplicationMode ... 103
8.3.4.2 EcuM_GetApplicationMode ... 103
8.3.4.3 EcuM_SelectBootTarget ... 104
8.3.4.4 EcuM_GetBootTarget ... 104
8.4 Scheduled Functions ... 104
8.4.1 EcuM_MainFunction ... 105

8.4.2 EcuM_StartupTwo ..... 105
8.5 Callback Definitions ..... 105
8.5.1 Callbacks for NVRAM Manager ..... 105
8.5.1.1 EcuM_CB_NfyNvMJobEnd ..... 105
8.6 Callout Definitions ..... 106
8.6.1 Generic Callouts ..... 106
8.6.1.1 EcuM_ErrorHook ..... 106
8.6.2 Callouts from STARTUP ..... 106
8.6.2.1 EcuM_AL_DriverInitOne ..... 106
8.6.2.2 EcuM_AL_DriverInitTwo ..... 107
8.6.2.3 EcuM_OnRTEStartup ..... 107
8.6.3 Callouts from RUN State ..... 108
8.6.3.1 EcuM_OnEnterRun ..... 108
8.6.3.2 EcuM_OnExitRun ..... 108
8.6.4 EcuM_OnExitPostRun ..... 108
8.6.5 Callouts from SHUTDOWN ..... 109
8.6.5.1 EcuM_OnPrepShutdown ..... 109
8.6.5.2 EcuM_OnGoSleep ..... 109
8.6.5.3 EcuM_OnGoOffOne ..... 109
8.6.5.4 EcuM_OnGoOffTwo ..... 110
8.6.5.5 EcuM_PutWakeupSourcesToSleep ..... 110
8.6.5.6 EcuM_GenerateRamHash ..... 111
8.6.5.7 EcuM_AL_SwitchOff ..... 111
8.6.6 Callouts from WAKEUP ..... 111
8.6.6.1 EcuM_CheckRamHash ..... 111
8.6.6.2 EcuM_AL_DriverRestart ..... 112
8.6.6.3 EcuM_OnWakeupReaction ..... 112
8.6.7 Callouts from SLEEP State ..... 113
8.6.7.1 EcuM_SleepActivity ..... 113
8.7 Expected Interfaces ..... 113
8.7.1 Mandatory Interfaces ..... 113
8.7.2 Optional Interfaces ..... 114
8.7.3 Configurable interfaces ..... 115
8.8 API Parameter Checking ..... 115

9 Sequence Charts ..... 116

10 Configuration specification ..... 117

10.1 Configuration Variants ..... 117
10.2 Type definitions ..... 117
10.2.1 EcuM_LabelType ..... 117
10.3 Configurable Parameters ..... 118
10.3.1 Overview ..... 118
10.3.2 EcuM_Configuration ..... 118
10.3.3 EcuM_DriverInitListOne ..... 124
10.3.4 EcuM_DriverInitListTwo ..... 124
10.3.5 EcuM_DriverRestartList ..... 125
10.3.6 EcuM_DriverInitItem ..... 125
10.3.7 EcuM_WakeupSource ..... 125
10.3.8 EcuM_User ..... 127
10.3.9 EcuM_SleepMode ..... 128



- AUTOSAR confidential -

10.3.10 EcuM_TTII ........................................................................... 128
10.4 Published Parameters ........................................................................... 130
10.5 Checking Configuration Consistency ........................................................................... 131
10.5.1 The Necessity for Checking Configuration Consistency in the ECU State Manager ........................................................................... 131
10.5.2 Example Hash Computation Algorithm ........................................................................... 133
10.6 Generated Callouts ........................................................................... 134
10.6.1 Managing Wakeup Sources ........................................................................... 134

11 Changes to Release 1 ........................................................................... 135

# List of Figures

Figure 1 - ECU Main States (top level diagram) ........ 29
Figure 2 - Module Relationship (top level diagram) ........ 32
Figure 3 - Startup Sequence (high level diagram) ........ 34
Figure 4 - Init Sequence I (STARTUP I) ........ 37
Figure 5 - Init Sequence II (STARTUP II) ........ 39
Figure 6 - RUN State Breakdown ........ 41
Figure 7 - RUN State Sequence (high level diagram) ........ 42
Figure 8 - RUN II State Sequence ........ 43
Figure 9 - RUN III State Sequence ........ 45
Figure 10 - Fine Structure of SHUTDOWN ........ 46
Figure 11 - Shutdown Sequence (high level diagram) ........ 47
Figure 12 - Deinitialization Sequence I (PREP SHUTDOWN) ........ 50
Figure 13 - Deinitialization Sequence IIa (GOSLEEP) ........ 51
Figure 14 - Deinitialization Sequence IIb (GO OFF I) ........ 52
Figure 15 - Deinitialization Sequence III (GO OFF II) ........ 53
Figure 16 - Wakeup Sequence (high level diagram) ........ 55
Figure 17 - WAKEUP State Breakdown ........ 56
Figure 18 - Wakeup Sequence I ........ 58
Figure 19 - Wakeup Validation Sequence ........ 59
Figure 20 - Activity Diagram of WAKEUP REACTION ........ 60
Figure 21 - Wakeup Sequence II ........ 61
Figure 22- Wakeup Validation Protocol ........ 66
Figure 23 - Example: Wakeup from CAN ........ 68
Figure 24 - Wakeup from ICU with ISR ........ 69
Figure 25 - Wakeup from ICU without ISR ........ 70
Figure 26 - Activity Diagram of TTII ........ 71
Figure 27 - ARPackage EcuM ........ 72
Figure 28 - Mapping of Declared Modes to ECU State Manager States ........ 75
Figure 29 - Selection of Boot Targets ........ 84
Figure 30 - Configuration Container Diagram ........ 118
Figure 31 - BSW Configuration Steps ........ 131

# List of Tables

Table 1 - Initialization Activities ........ 36
Table 2 - Driver Initialization Details, Sample Configuration ........ 40
Table 3 - Shutdown Activities ........ 49
Table 4 - Wakeup Activities ........ 57
Table 5 - Error Classification ........ 86
Table 6 - Mandatory interfaces ........ 114
Table 7 - Optional Interfaces ........ 114
Table 8 - Published parameters ........ 130


- AUTOSAR confidential -

# 1 Introduction

## 1.1 Functional Overview

The ECU State Manager is a basic software module (see [1]). It manages all aspects of the ECU related to the OFF, RUN, and SLEEP states of that ECU and the transitions (transient states) between these states like STARTUP and SHUTDOWN.
In detail, the ECU State Manager

- is responsible for the initialization and de-initialization of all basic software modules including OS and RTE,
- cooperates with the COM Manager, and hence indirectly with network management, to shut down the ECU when needed,
- manages all wakeup events and configures the ECU for SLEEP when requested.

In order to fulfill all these tasks, the ECU State Manager provides some important protocols:

- the RUN request protocol, which is needed to coordinate whether the ECU must be kept alive or is ready to shut down,
- the wakeup validation protocol to distinguish 'real' wakeup events from 'erratic' ones,
- the time triggered increased inoperation protocol (TTII), which allows to put the ECU into an increasingly energy saving sleep state over time.

These protocols were specified with the following underlying constraints:

- standardization at the API side, to allow applicability to all kinds of ECUs and portability of AUTOSAR applications
- high degree of flexibility to the low side interface, mainly reached by a set of callouts
- quick startup times
- consistent programming paradigm across all mode managing modules (rubber band model[1])

Summarizing all this, the ECU State Manager will be one of the principal state machines of an AUTOSAR compliant ECU, namely that one around states with the highest priority: RUN, SLEEP, and OFF. However, it does not and shall not in future contain functionality which might be related to terms like 'vehicle modes', 'error modes', or any other kind of application related kind of states or modes. These topics shall be addressed by other state machines (application mode managers).

[1] As long as some entity requests run, the rubber band is stretched to the RUN state, and it snaps back when it is released. Since there is only one state (namely the RUN state) to which the rubber band applies, this term is not used any further in this specification. However, it is important to understand that, if applied to resource managers, the result is a powerful and consistent concept for enhancing state machines. The COM manager is a module which picks up the idea of the resource manager and of the rubber band model and henceforce fits well into landscape spawn by the ECU State Manager.

## 1.2 Conventions Used in this Specification

### 1.2.1 Font Faces

**EcuM123**: Requirements are tagged with an ID in bold font.

*References*
to other documents or to other chapters within this document are printed in italic.

`Source code`
is printed in a Courier font.

*`Configuration Parameters`*
are printed in Courier Italic.

STATE
State names are written in capital letters.

### 1.2.2 Figures

**Figure X - Title (diagram type)**

Figures are typically drawn in UML. To capture the hierarchical organization of the UML diagrams, some diagrams are classified in the title (diagram type). The following types are used:

- *top level*
  An entry diagram to the structural or behavioral domain
- *high level*
  First degree of break down below the top level
- *SUB-STATE*
  The diagram describes the behavior of the given sub-state, the diagram type is the name of the sub-state
- no class
  All other diagrams, typically detail information

In the present version of this documentation, there is only one top level diagram: The main state machine.
The next level is covered by high level diagrams. There are four notable high level sequence diagrams:

- 
- *Figure 3 - Startup Sequence (high level* diagram)
- 
- *Figure 7 - RUN State Sequence (high level* diagram)
- 
- *Figure 11 - Shutdown Sequence (high level* diagram
- 
- *Figure 16 - Wakeup Sequence (high level* diagram)

These high level diagrams give an overview of the major activities in the main state and explain how the state transitions occur. High level sequence diagrams always start with a diagram reference to the preceding sequence and end with a diagram reference to the following sequence.
High level diagrams are typically broken down into SUB-STATE diagrams. They show details which are irrelevant at the high level.

# 2 Definitions and Acronyms

| **Term:** | **Description:** |
|---|---|
| Inoperation | An artificial word to describe the ECU when it is not operational, i.e. not running. Comprises all meanings of *off, sleeping, frozen*, etc. Using this definition is beneficial since it has no predefined meaning. |
| Shutdown Target | The shutdown of an ECU may end up in different states, depending on what application requires or desires for the next shutdown. By selecting a shutdown target, the application can communicate its wishes to the ECU State Manager. SLEEP, OFF, and Reset are shutdown targets. |
| Callback | Within this document, the term 'callback' is used for API services which are intended for notifications to other BSW modules. |
| Callout | Within this document, the term 'callout' is used for function stubs which can be filled by the system designer, usually at configuration time, with the purpose to add functionality to the ECU State Manager. Callouts are separated into two classes, where one class is optional to be filled. The other class is mandatory and serves as a hardware abstraction layer. |
| Passive Wakeup | A wakeup caused from an attached bus rather than an internal event like a timer or sensor activity. |
| Post run | Post run is the period from when the application detects a reason to start the shutdown until the shutdown actually occurs. Typically this period starts when all network communication is put to sleep and lasts until the ECU is put to sleep. |
| Vital Data | Any kind of data (RAM or NVRAM) that must stay consistent to ensure correct operation of the ECU. E.g. stacks, important state variables, etc. |
| Wakeup Event | A physical event which causes a wakeup. A CAN message or a toggling IO line can be wakeup events.<br>Similarly, the internal SW representation, e.g. an interrupt, may also be called a wakeup event. |
| Wakeup Reason | The wakeup reason is the wakeup event being the actual cause of the last wakeup. |
| Wakeup Source | The peripheral or ECU component which deals with wakeup events is called a wakeup source. |

| **Acronym:** | **Description:** |
|---|---|
| BSW | Basic Software |
| BSWM | Basic Software Module |
| ISR | Interrupt Service Routine |
| RTE | Runtime Environment (WP4.2.1.1) |
| SW-C | Software Component (above RTE) |
| TTII | Time-Triggered Increased Inoperation |

# 3 Related documentation

## 3.1 Input documents

[1] List of Basic Software Modules
https://svn.autosar.org/repos/10Releases
AUTOSAR_BasicSoftwareModules.pdf

[2] Layered Software Architecture
https://svn.autosar.org/repos/10Releases
AUTOSAR_LayeredSoftwareArchitecture.pdf

[3] General Requirements on Basic Software Modules
https://svn.autosar.org/repos/10Releases
AUTOSAR_SRS_General.pdf

[4] Requirements on Mode Management
https://svn.autosar.org/repos/10Releases
AUTOSAR_SRS_ModeManagement.pdf

## 3.2 Related standards and norms

[5] D1.5-General Architecture; ITEA/EAST-EEA, Version 1.0; ch. 3, pp 72

[6] D2.1-Embedded Basic Software Structure Requirements; ITEA/EAST-EEA

[7] D2.2-Description of existing solutions; ITEA/EAST-EEA, Version 1.0 or higher.

## 3.3 Related AUTOSAR Software Specifications

[8] Glossary
https://svn.autosar.org/repos/10Releases
AUTOSAR_Glossary.pdf

[9] Specification of Communication Manager
https://svn.autosar.org/repos/10Releases
AUTOSAR_SWS_ComManager.pdf

[10] Specification of Watchdog Manager
https://svn.autosar.org/repos/10Releases
AUTOSAR_SWS_WatchdogManager.pdf

[11] Specification of CAN Interface
https://svn.autosar.org/repos/10Releases
AUTOSAR_SWS_CAN_Interface.pdf

[12] Specification of FlexRay Interface
https://svn.autosar.org/repos/10Releases

AUTOSAR_SWS_Flexray_Interface.pdf

[13] Specification of Generic Network Management
https://svn.autosar.org/repos/10Releases
AUTOSAR_SWS_Generic_NM.pdf

[14] Specification of NVRAM Manager
https://svn.autosar.org/repos/10Releases
AUTOSAR_SWS_NVRAM_Manager.pdf

[15] Specification of MCU Driver
https://svn.autosar.org/repos/10Releases
AUTOSAR_SWS_MCU_Driver.pdf

[16] Specification of SPI Handler/Driver
https://svn.autosar.org/repos/10Releases
AUTOSAR_SWS_SPIHandlerDriver.pdf

[17] Specification of EEPROM Interface
https://svn.autosar.org/repos/10Releases
AUTOSAR_SWS_EEPROM_Driver.pdf

[18] Specification of Flash Interface
https://svn.autosar.org/repos/10Releases
AUTOSAR_SWS_Flash_Driver.pdf

[19] Specification of Operating System
https://svn.autosar.org/repos/10Releases
AUTOSAR_SWS_OS.pdf

[20] Specification of RTE Software
https://svn.autosar.org/repos/10Releases
AUTOSAR_SWS_RTE.pdf

[21] Specification of Diagnostics Event Manager
https://svn.autosar.org/repos/10Releases
AUTOSAR_SWS_DEM.pdf

[22] Specification of Development Error Tracer
https://svn.autosar.org/repos/10Releases
AUTOSAR_SWS_DET.pdf

[23] Specification of CAN Transceiver Driver
https://svn.autosar.org/repos/10Releases
AUTOSAR_SWS_CAN_TransceiverDriver.pdf

[24] Specification of C Implementation Rules
https://svn.autosar.org/repos/10Releases
AUTOSAR_SWS_C_ImplementationRules.pdf


- AUTOSAR confidential -

# 4 Constraints and Assumptions

## 4.1 Limitations

The shutdown target OFF requires special hardware on the ECU so that it can actually be reached (e.g. a power hold circuit). If this hardware is not available, this specification proposes to issue a reset instead but other default behaviors can be defined.

**EcuM2522**: Applications (SW-C's) shall not assume that it is actually possible to switch off ECUs (i.e. power consumption is zero).

## 4.2 Hardware Requirements

The following requirements are needed to support switching the application mode (see *7.11.1 Application Modes*). Other basic software modules also need this requirement.

**EcuM2261**: ECU RAM must keep contents of vital data while ECU clock is switched off. This requirement is needed to implement sleep states as required in *7.6 SLEEP State*.

**EcuM2262**: ECU RAM must provide a no init area which keeps contents over a reset cycle. A no init area shall only be initialized on a power on event (clamp 30). The system designer is responsible for establishing an initialization strategy.

## 4.3 Applicability to car domains

The ECU State Manager is applicable to all car domains.


- AUTOSAR confidential -

# 5 Dependencies to other modules

The following sections outline the important relationships to other modules. They also contain some requirements that these modules have to fulfill to collaborate correctly with ECU State Manager.

## 5.1 Mode Management Modules

### 5.1.1 Communication Manager

The Communication Manager is a so-called 'Resource Manager'[2] and thus requests RUN state. Resource Managers are described in chapter *7.2.3 Resource Managers*.

The Communication Manager requests RUN state when it is leaving the 'no communication' state and it releases RUN when it is returning to this state.

### 5.1.2 Watchdog Manager

The Watchdog Manager is initialized by the ECU State Manager.
Triggering the watchdog during STARTUP I and GO OFF II state (see *7.1 Main States of the ECU State Manager*) is a currently only partially solved. Currently, the behavior is as follows:

- Typically, the watchdog is armed by the initialization of the watchdog driver.
- With its initialization, the Watchdog Manager takes over the responsibility for triggering. For the time in between, it is responsibility of the ECU State Manager, to trigger the watchdog. This is a short period during STARTUP I.
- When the Watchdog Manager is de-initialized, the responsibility to trigger the watchdog is handed back to the ECU State Manager. This is a short period during GO OFF II.
- For a typical ECU design, the time spent in STARTUP I or GO OFF II is considered so short, that triggering is not needed during these state.
- If triggering is needed, then it is left up to the system designer to trigger the watchdog from one of the ECU State Manager's callouts directly. It is the system designer's responsibility to ensure that the Watchdog Manager is not active meanwhile.

## 5.2 SPAL Modules

### 5.2.1 MCU Driver

The MCU Driver is the first basic software module initialized by the ECU State Manager. However, returning `MCU_Init`, the MCU and the MCU driver are not

---

[2] 'Resource Manager' is invented in this specification to classify BSW modules which interact with Ecu State Manager.

necessarily fully initialized. Additional, MCU specific steps may be needed. The ECU State Manager provides a callout where this additional code can be placed. For details on how this code should look like refer to [15].

### 5.2.2 Driver Dependencies and Initialization Order

BSW drivers may depend on each other. A typical example is the watchdog driver which needs the SPI driver to access an external watchdog. This means on the one hand, that drivers may be stacked (not relevant to EcuM) but on the other hand that the underlying driver needs to be initialized first.

**EcuM2502**: The system designer is responsible for defining the initialization order at configuration time.

## 5.3 Peripherals with Wakeup Capability

Wakeup sources have to be handled and encapsulated by drivers. The implementation must follow the protocols and requirements presented in this document to ensure a seamless integration into AUTOSAR BSW.
To support the wakeup and validation protocol, the driver has to fulfill the following requirements:

The driver has to notify ECU State Manager by invoking the `EcuM_SetWakeupEvent` service once when a wakeup event is detected. The same service should also be invoked during initialization of the driver if a pending wakeup event is detected during the initialization. Preferably, the invocation is done from a callout or function stub of the caller, to decouple driver modules and ECU State Manager.

The driver shall provide an explicit service to put the wakeup source to sleep. This service shall put the wakeup source into a energy saving and inert operation mode and re-arm the wakeup notification mechanism.

If the wakeup source is capable of generating faulty events[3] then the driver or the software stack consuming the driver or another appropriate BSW module shall either provide a validation callout for the wakeup event under validation or directly call the wakeup validation service of the ECU State Manager. If validation is not necessary, then this requirement is not applicable for the according wakeup source.

## 5.4 Operating System

ECU State Manager starts and shuts down the AUTOSAR OS. It also defines the protocol how control is handed over to the OS after its startup and how it is handed back to the ECU State Manager when it is shut down.

[3] Faulty wakeup events may result from EMV spikes, bouncing effects on wakeup lines etc.


- AUTOSAR confidential -

## 5.5 Runtime Environment (RTE)

The initialization and de-initialization functions of RTE are assumed to return.

ECU State Manager shall use the mode port feature of RTE to notify about state changes. See chapter *7.10 AUTOSAR Ports* for more information 7.10.

## 5.6 NVRAM Manager

The following operations of the NVRAM Manager [14] are executed by the ECU State Manager.

- Initialization of NVRAM Manager after a power up or reset of the ECU
- Read-back of non-volatile data from NVRAM to ECU RAM during the initialization of the ECU
- Storing of non-volatile data to NVRAM in all shutdown paths. The storing process is prematurely terminated upon wakeup events to ensure a quick restart of the ECU.

NVRAM is not read during the wakeup sequence since RAM contents is assumed to be still valid from the previous cycle. To verify this, RAM integrity is checked[4]. NVRAM is only read during the STARTUP.

The NVRAM Manager shall call the callbacks defined in chapter *8.5.1 Callbacks for NVRAM Manager* to notify the ECU State Manager about job status.

## 5.7 Diagnostic Event Manager

The DEM requires NVRAM Manager to be operational. The DEM is aware if NVRAM Manager is operational or provides limited functionality. These differences are handled within the DEM.

## 5.8 Software Components

The ECU State Manager handles two ECU-wide settings/variables:

- Application modes[5]
- Setting of shutdown targets

It is assumed in this specification that these properties are set by the application (through AUTOSAR ports), typically by some ECU specific part of the application. The ECU State Manager does not prohibit two application overriding each other's settings. The policy must be defined at a higher level.
The following two requirements formulate an attempt to resolve this issue.

[4] See *8.6.5.6 EcuM_GenerateRamHash* and *8.6.6.1 EcuM_CheckRamHash* for details.
[5] In this context, 'application mode' is a technical term which is defined by the AUTOSAR OS specification.

The SW-C Template may specify a field whether the SW-C sets the application mode or the shutdown target.

The generation tool may only allow configuration which have only one SW-C accessing application mode or shutdown target.

## 5.9 File Structure

**EcuM2675**: The file structure shall be as follows:

- One or more C file `EcuM_xxx.c` containing the entire or parts of ECU State Manager code
- One C file `EcuM_Lcfg.c` containing link time configuration.
  This file is candidate for deletion because there is no link time configuration currently.
- One C file `EcuM_PBcfg.c` containing post build time configuration.
- One file `EcuM_Callout_Stubs.c` containing the stubs of the defined callouts. Whether this file shall be modified directly or includes other generated files is specific to the implementation.
- An API interface `EcuM.h` providing the fix type declarations, forward declaration to generated types, and function prototypes
- A type header `EcuM_Generated_Types.h` providing generated types and fulfills the forward declarations from `EcuM`.h.
- A type header `EcuM_Cfg.h` providing the configuration parameters
- A callback/callout interface `EcuM_Cbk.h` providing the callback/callout function prototype

**EcuM2676**: It shall only be necessary to include EcuM.h to use all services of the ECU State Manager.

**EcuM2677**: It shall only be necessary to include EcuM_Cbk.h to interact with the callbacks and callouts of the ECU State Manager.

Also refer to *8.7 Expected Interfaces* for dependencies to other modules.

# 6 Requirements traceability

Document: General Requirements on Basic Software Modules [4], v1.11.0

| *Requirement* | | *Satisfied by* |
|---|---|---|
| [BSW00344] | Reference to link-time configuration | EcuM2500<br>EcuM does not define configuration sets but references the init configuration, e.g. for driver initialization |
| [BSW00404] | Reference to post build time configuration | EcuM2500<br>EcuM does not define configuration sets but references the init configuration, e.g. for driver initialization |
| [BSW00405] | Reference to multiple configuration sets | EcuM2500<br>EcuM does not define configuration sets but references the init configuration, e.g. for driver initialization |
| [BSW00345] | Pre-compile-time configuration | *10.3 Configurable Parameters*<br>*5.9 File Structure* |
| [BSW159] | Tool-based configuration | not applicable<br>(EcuM does not specify the configuration tool) |
| [BSW167] | Static configuration checking | *10.3 Configurable Parameters* |
| [BSW171] | Configurability of optional functionality | *10.3 Configurable Parameters* |
| [BSW00380] | Separate C-files for configuration parameters | *5.9 File Structure* |
| [BSW00419] | Separate C-files for pre-compile-time configuration parameters | *5.9 File Structure* |
| [BSW00381] | Separate configuration header files for pre-compile-time parameters | *5.9 File Structure* |
| [BSW00412] | Separate H-file for configuration parameters | *5.9 File Structure* |
| [BSW00383] | List dependencies to other configuration files | *10.3 Configurable Parameters* |
| [BSW00384] | List dependencies to other modules | *5 Dependencies to other modules*<br>*8.7Expected Interfaces* |
| [BSW00387] | Specify the configuration class of a callback function | *8.5 Callback Definitions* |
| [BSW00388] - [BSW00400] | | *10.3 Configurable Parameters* |
| [BSW00402] | Published information | *10.4 Published Parameters* |
| [BSW00375] | Notification of wakeup reason | *8.3.3 Wakeup Notifications* |
| [BSW101] | Initialization interface | *8.3.1.1 EcuM_Init* |

| | | |
|---|---|---|
| [BSW00416] | Sequence of initialization | EcuM2559 |
| [BSW00406] | Check module initialization | not applicable<br>(EcuM initializes the BSW, hence EcuM is always initialized from the point of view of any other BSW module.) |
| ***Normal operation*** | | |
| [BSW168] | Diagnostic Interface of SW components | not applicable<br>(EcuM has no testing requirements) |
| [BSW00407] | Function to read out published parameters | *8.3.1.3 EcuM_GetVersionInfo* |
| [BSW00423] | Usage of SW-C template to describe BSW modules with AUTOSAR interfaces | *7.10 AUTOSAR Ports* |
| [BSW00424] | BSW main processing function task allocation | Implementation of `EcuM_MainFunction` according to this specification does not require extended task mechanisms. |
| [BSW00425] | Trigger conditions for schedulable objects | *8.4.1 EcuM_MainFunction* |
| [BSW00426] | Exclusive areas in BSW modules | not applicable<br>(EcuM does not specify directly accessible global data.) |
| [BSW00427] | ISR description for BSW modules | not applicable<br>(EcuM does not specify ISRs.) |
| [BSW00428] | Execution order dependencies of main processing functions | There are no requirements of this sort. |
| [BSW00429] | Restricted BSW OS functionality access | EcuM does not use any other than the allowed OS services. |
| [BSW00431] | The BSW Scheduler module implements task bodies | EcuM does not define any task body. |
| [BSW00432] | Modules should have separated main processing functions for a read/receive and write/transmit data path | not applicable<br>(EcuM does not specify RxTx functionality.) |
| [BSW00433] | Calling of main processing functions | EcuM does not call any main processing function. |
| [BSW00434] | The Schedule Module shall provide an API for exclusive areas | not applicable<br>(This is not an EcuM requirement) |
| [BSW00336] | Shutdown interface | *8.3.1.2 EcuM_Shutdown* |
| ***Fault Operation and Error Detection*** | | |
| [BSW00337] | Classification of errors | *Table 5 - Error* Classification |
| [BSW00338] | Detection and reporting of development errors | *Table 5 - Error* Classification |
| [BSW00369] | Do not return development error codes via API | *8 API specification* |
| [BSW00339] | Reporting of production relevant error statuses | EcuM2759 |

| | | |
|---|---|---|
| [BSW00417] | Reporting of Error Events by Non-Basic Software | not applicable |
| [BSW00323] | API parameter checking | *8.3 Function* |
| [BSW004] | Version check | *10.4 Published Parameters* |
| [BSW00409] | Header files for production code error IDs | *5.9 File Structure* |
| [BSW00385] | List possible error notifications | *Table 5 - Error* Classification |
| [BSW00386] | Configuration for detecting errors | *7.12 Error Classification* |
| [BSW161] | Microcontroller abstraction | not applicable<br>(Requirements related to layered software architecture are reflected by the EcuM SRS) |
| [BSW162] | ECU layout abstraction | not applicable<br>(Requirements related to layered software architecture are reflected by the EcuM SRS) |
| [BSW005] | No hard coded horizontal interfaces within MCAL | not applicable<br>(Requirements related to layered software architecture are reflected by the EcuM SRS) |
| [BSW00415] | User dependent include files | not applicable<br>(EcuM does not define user specific functionality) |
| [BSW164] | Implementation of ISRs | not applicable<br>(EcuM does not specify ISRs.) |
| [BSW00325] | Runtime of ISRs | not applicable<br>(EcuM does not specify ISRs.) |
| [BSW00326] | Transition from ISRs to OS task | not applicable<br>(EcuM does not specify ISRs.) |
| [BSW00342] | Usage of source code and object code. | *5.9 File Structure* |
| [BSW00343] | Specification and configuration of time | *10.3 Configurable Parameters* |
| [BSW160] | Human-readable configuration data | not applicable<br>(This specification does not define the configuration file) |
| [BSW007] | HIS MISRA C | The API definition complies with MISRA C.<br>*8 API specification* |
| [BSW00300] | Module naming conventions. | *5.9 File Structure* |
| [BSW00413] | Accessing instances of BSW modules | not applicable<br>(EcuM defines only one instance.) |
| [BSW00347] | Naming separation of different instances of BSW drivers | not applicable<br>(EcuM defines only one instance.) |
| [BSW00305] | Self-defined data types naming conventions | *8.2 Type definitions* |
| [BSW00307] | Global variables naming convention | not applicable<br>(EcuM does not specify global variables.) |

| | | |
|---|---|---|
| [BSW00310] | API naming conventions | *8 API specification* |
| [BSW00373] | Main processing function naming convention | *8.4.1 EcuM_MainFunction* |
| [BSW00327] | Error values naming convention | *Table 5 - Error* Classification |
| [BSW00335] | Status values naming convention | *8.2 Type definitions* |
| [BSW00350] | Development error detection keyword | *10.3 Configurable Parameters* |
| [BSW00408] | Configuration parameter naming convention | *10.3 Configurable Parameters* |
| [BSW00410] | Compiler switches shall have defined values | not applicable<br>(This specification does not define compiler switchers) |
| [BSW00411] | Get version info keyword | *10.4 Published Parameters* |
| [BSW00346] | Basic set of module files | *5.9 File Structure* |
| [BSW158] | Separation of configuration from implementation | *5.9 File Structure* |
| [BSW00314] | Separation of interrupt frames from service routines | not applicable<br>(EcuM does not specify ISRs.) |
| [BSW00370] | Separation of callback interface from API | *8 API specification* |
| ***Standard Header Files*** | | |
| [BSW00348] | Standard header type | not applicable<br>(EcuM does not define standard types) |
| [BSW00353] | Platform specific type header | not applicable<br>(EcuM is specified platform independent) |
| [BSW00361] | Compiler specific language extension header | not applicable<br>(EcuM does not define language extensions) |
| [BSW00301] | Limited import information | *8.1 Imported Types* |
| [BSW00302] | Limited export information | *8 API specification* |
| [BSW00328] | Avoid duplication of code | Not applicable<br>(Requirement to implementation) |
| [BSW00312] | Shared code shall be re-entrant | *8 API specification* |
| [BSW006] | Platform independency | *8 API specification* |
| [BSW00357] | Standard API return type | *8 API specification* |
| [BSW00377] | Module specific API return types | *8 API specification* |

| | | |
|---|---|---|
| [BSW00304] | AUTOSAR integer data types | *8 API specification* |
| [BSW00355] | Do not redefine AUTOSAR integer data types | *8 API specification* |
| [BSW00378] | AUTOSAR boolean type | *8 API specification* |
| [BSW00306] | Avoid direct use of compiler and platform specific keywords | *8 API specification* |
| [BSW00308] | Defintion of global data | Not applicable<br>(EcuM does not specify global data.) |
| [BSW00309] | Global data with read-only constraints | Not applicable<br>(EcuM does not specify global data.) |
| [BSW00371] | Do not pass function pointers via API | *8 API specification* |
| [BSW00358] | Return type of init() functions | *8.3.1 Initialization and Shutdown* |
| [BSW00414] | Parameter of init function | *8.3.1 Initialization and Shutdown* |
| [BSW00376] | Return type and parameters of main processing functions | *8.4.1 EcuM_MainFunction* |
| [BSW00359] | Return type of callback functions | *8.5 Callback Definitions* |
| [BSW00360] | Parameters of callback functions | *8.5 Callback Definitions* |
| [BSW00329] | Avoidance of generic interfaces | *8 API specification* |
| [BSW00330] | Usage of macros/inline functions instead of functions | not applicable<br>(Requirement to implementation) |
| [BSW00331] | Separation of error and status values | *8.2 Type definitions* |
| [BSW009] | Module user documentation | Fulfilled by usage of template/formal review |
| [BSW00401] | Documentation of multiple instances of configuration parameters | *10.3 Configurable Parameters* |
| [BSW172] | Compatibility and documentation of scheduling strategy | *0*<br><br>*Scheduled* Functions |
| [BSW010] | Memory resource documentation | not applicable<br>(requirement to implementation) |
| [BSW00333] | Documentation of callback function context | *8.5 Callback Definitions* |
| [BSW00374] | Module vendor identification | *10.4 Published Parameters* |
| [BSW00379] | Module identification | *10.4 Published Parameters* |

| | | |
|---|---|---|
| [BSW003] | Version identification | *10.4 Published Parameters* |
| [BSW00318] | Format of module version numbers | *10.4 Published Parameters* |
| [BSW00321] | Enumeration of module version numbers | *10.4 Published Parameters* |
| [BSW00341] | Microcontroller compatibility documentation | not applicable (requirement to implementation) |
| [BSW00334] | Provision of XML file | not applicable (provided by system team) |

Document: Requirements on Mode Management [4], v0.32

| ***Requirement*** | | ***Satisfied by*** |
|---|---|---|
| [BSW09120] | Configuration of initialization process of basic software | EcuM2559, EcuM2520, EcuM2521, *8.6.2 Callouts from STARTUP* |
| [BSW09147] | Configuration of de-initialization process of basic software | |
| [BSW09122] | Configuration of users of the ECU State Manager | EcuM487, *10.3 Configurable Parameters* |
| [BSW09100] | Selection of wakeup sources shall be configurable | EcuM2389, *10.3 Configurable Parameters* |
| [BSW09146] | Configuration of time triggered increased inoperation | EcuM2654, EcuM2223, *10.3 Configurable Parameters* |
| [BSW09001] | Standardization of state relations | EcuM2664 |
| [BSW09116] | Requesting and releasing the RUN state | *8.3.2.1 EcuM_RequestRUN*, *8.3.2.2 EcuM_ReleaseRUN* |
| [BSW09114] | Starting/invoking the shutdown process | EcuM2311 |
| [BSW09104] | ECU State Manager shall take over control after OS shutdown | EcuM2328 |
| [BSW09113] | Initialization of Basic Software modules | *Table 1 - Initialization* Activities |
| [BSW09127] | De-initialization of BSW | Table 3 - Shutdown Activities |
| [BSW09128] | Support of several shutdown targets | *7.6.1 Shutdown Targets* |
| [BSW09119] | Support of several sleep modes | EcuM2363 |
| [BSW09102] | API for selecting the sleep mode | *8.3.2.9 EcuM_SelectShutdownTarget* |
| [BSW09072] | Force ECU shutdown | |

| | | |
|---|---|---|
| [BSW09009] | Activation of software when entering/leaving ECU states | *8.6 Callout Definitions* |
| [BSW09017] | Provide ECU state information | *8.3.2.10 EcuM_GetState* |
| [BSW09138] | Selection of application modes of OS | EcuM2141, *7.11.1 Application Modes* |
| [BSW09136] | Centralized Wakeup Management | *7.8 Wakeup Validation Protocol* |
| [BSW09098] | Registration of wakeup reasons | *8.3.3 Wakeup Notifications* |
| [BSW09097] | Validation of physical channel wakeup | *7.8 Wakeup Validation Protocol* |
| [BSW09118] | Time Triggered Increased Inoperation | *7.9 Time Triggered Increased Inoperation* |
| [BSW09145] | Support of wake-sleep operation | *7.11.5 Configuration Alternative* for Providing Wake-Sleep Operation |
| [BSW09126] | Provide an API for querying of wakeup reason | *8.3.3 Wakeup Notifications* |
| [BSW09145] | Evaluate condition to stay in the RUN state | EcuM2311 |
| [BSW09164] | Shutdown synchronization support for SW-Components | *7.4.3.4 RUN* |
| [BSW09165] | Requesting and releasing the POST RUN state | 8.3.2State Management |
| [BSW09166] | Evaludate condition to stay in POST RUN state | |
| [BSW09148] | Notify ECU State Manager about wakeup reason | EcuM2483, EcuM2484 |

# 7 Functional Specification

## 7.1 Main States of the ECU State Manager

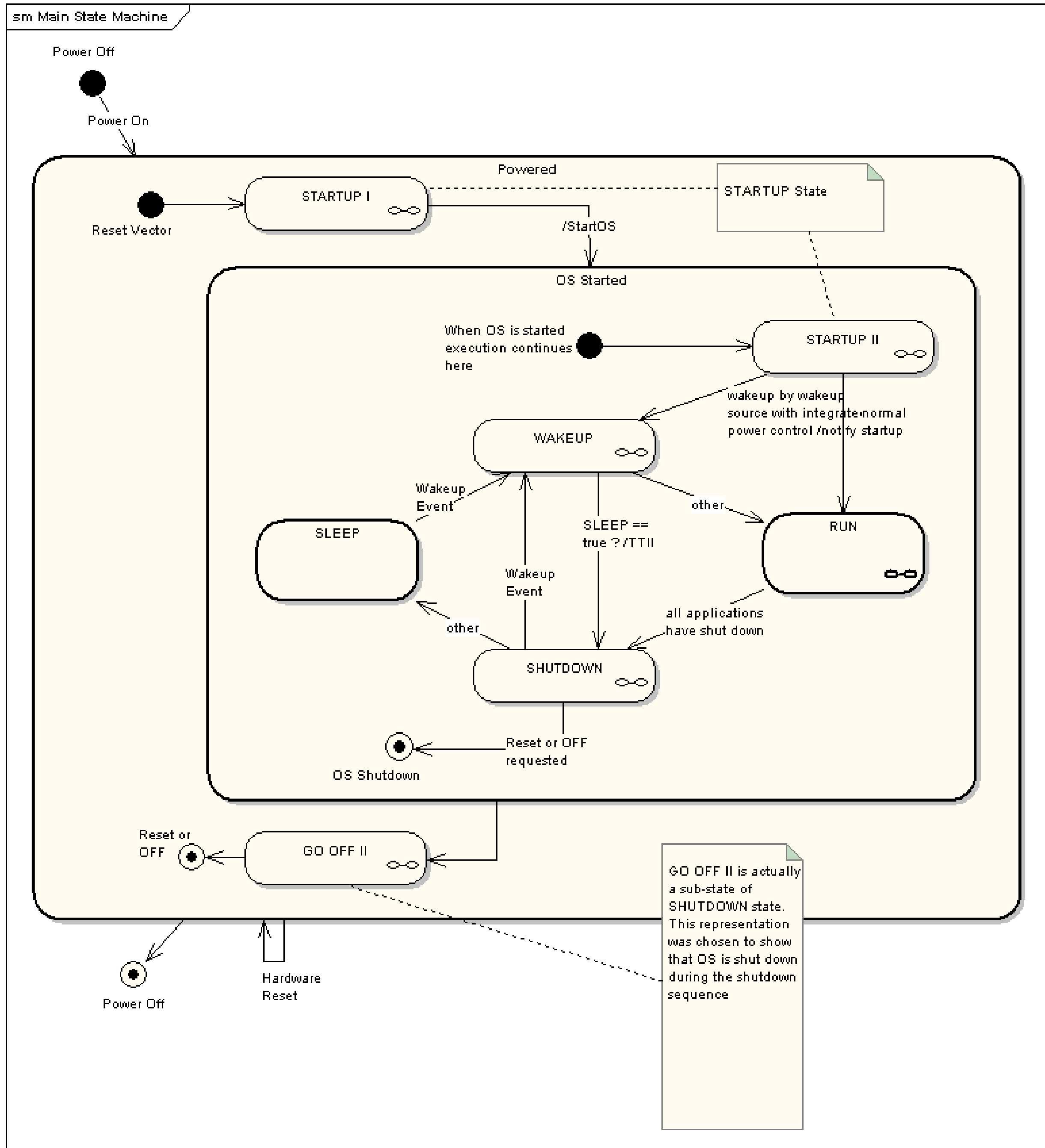


**Figure 1 - ECU Main States (top level diagram)[6]**

[6] All UML diagrams were drawn with Enterprise Architect.

*Figure 1 - ECU Main States (top level* diagram) shows the main state machine provided by the ECU State Manager. This state machine manages the 'life cycle' of an ECU from STARTUP over RUN to SLEEP or OFF.

### 7.1.1 STARTUP State

The purpose of the STARTUP state is to initialize the basic software modules. The STARTUP state is divided into two parts, the first being the part before OS startup, the second part after OS startup (and therefore with a running OS). More details about the initialization are given in chapter *7.3 STARTUP State*.

### 7.1.2 RUN State

The RUN State is entered by the ECU State Manager after all modules of basic software including OS and RTE have been initialized by the ECU State Manager.
The RUN State indicates to the SW-C's above RTE that BSW has initialized and applications start operating. Further, the RUN state provides a mechanism for synchronized shutdown of application software.
RUN state must be requested by the application explicitly or implicitly[7] whenever it is needed to keep the ECU awake. Otherwise, the ECU State Manager will commence shutdown. In other words: SW-C shall request the RUN state from the ECU State Manager when the ECU needs to stay awake.
The RUN State falls into two sub-states: The regular RUN state and a POST RUN state. The POST RUN state can be requested by SW-C's to indicate that the need to execute cleanup or saving activities before the ECU goes to sleep. The POST RUN state can be requested independently from the RUN state with a separate API or from AUTOSAR ports accordingly[8].
SW-C's shall react on state changes by interfaceing with the mode port of the ECU State Manager.
If the SW-C's primary intent is to communicate with other SW-C's, SW-C's shall request a communication state from the Communication State Manager instead.

### 7.1.3 SHUTDOWN State

The shutdown state handles the controlled shutdown of basic software modules and finally results in the selected shutdown target for the ECU: SLEEP, OFF, or Reset. Important activities in this state are to write non-volatile data back to NVRAM.

[7] RUN state is requested implicitly if a non-idle state is requested from a Resource Manager. E.g. requesting any state but 'no communication' from the Communication Manager will have the Communication Manager requesting RUN state from the ECU State Manager in turn. This is a request for communication which implicitly results in a request for RUN state. See also [8].
[8] In this specification RUN and POST RUN sub-states are called RUN II and RUN III.

### 7.1.4 SLEEP State

The SLEEP state is an energy saving state. Typically, no code is executed but power is still supplied, and if configured accordingly, the ECU is wakeable in this state[9]. The SLEEP state provides a configurable set of sleep modes which typically are a trade off between power consumption and time to restart the ECU. In terms of the API, the sleep modes are referred to as *shutdown targets*.

### 7.1.5 WAKEUP State

The WAKEUP State is entered when the ECU comes out of the SLEEP state, due to intended or unintended wakeup.
The WAKEUP State provides a protocol to support validation of wakeup events. This is necessary to differentiate between intended und unintended wakeups. The validation itself is a cooperative process between the driver which handles the wakeup source and the ECU State Manager (see *7.8 Wakeup Validation Protocol*).

### 7.1.6 OFF State

The OFF state describes the unpowered ECU. Wakeability is not required in this state, but the ECU must be startable (e.g. by reset events).

### 7.1.7 Reset Behavior and Leaving States

The diagram shows two major states which are important to keep in mind but not explained further in this specification: The POWERED state and the OS STARTED state. Transitions in and out of the OS STARTED state are managed by starting up or shutting down the OS during the STARTUP and SHUTDOWN state respectively.
The POWERED state is entered when the reset vector is invoked, this also occurs after a reset. The POWERED state is not under software control but is entirely controlled by hardware, in particular by the design of the reset logic and of the power supply.

[9] Some ECU designs actually do require code execution to implement a SLEEP state (and the wakeup capability). For these ECUs, the clock speed is typically dramatically reduced. These could be implemented with a small loop inside the SLEEP state.


- AUTOSAR confidential -

## 7.2 Structural Description of the ECU State Manager

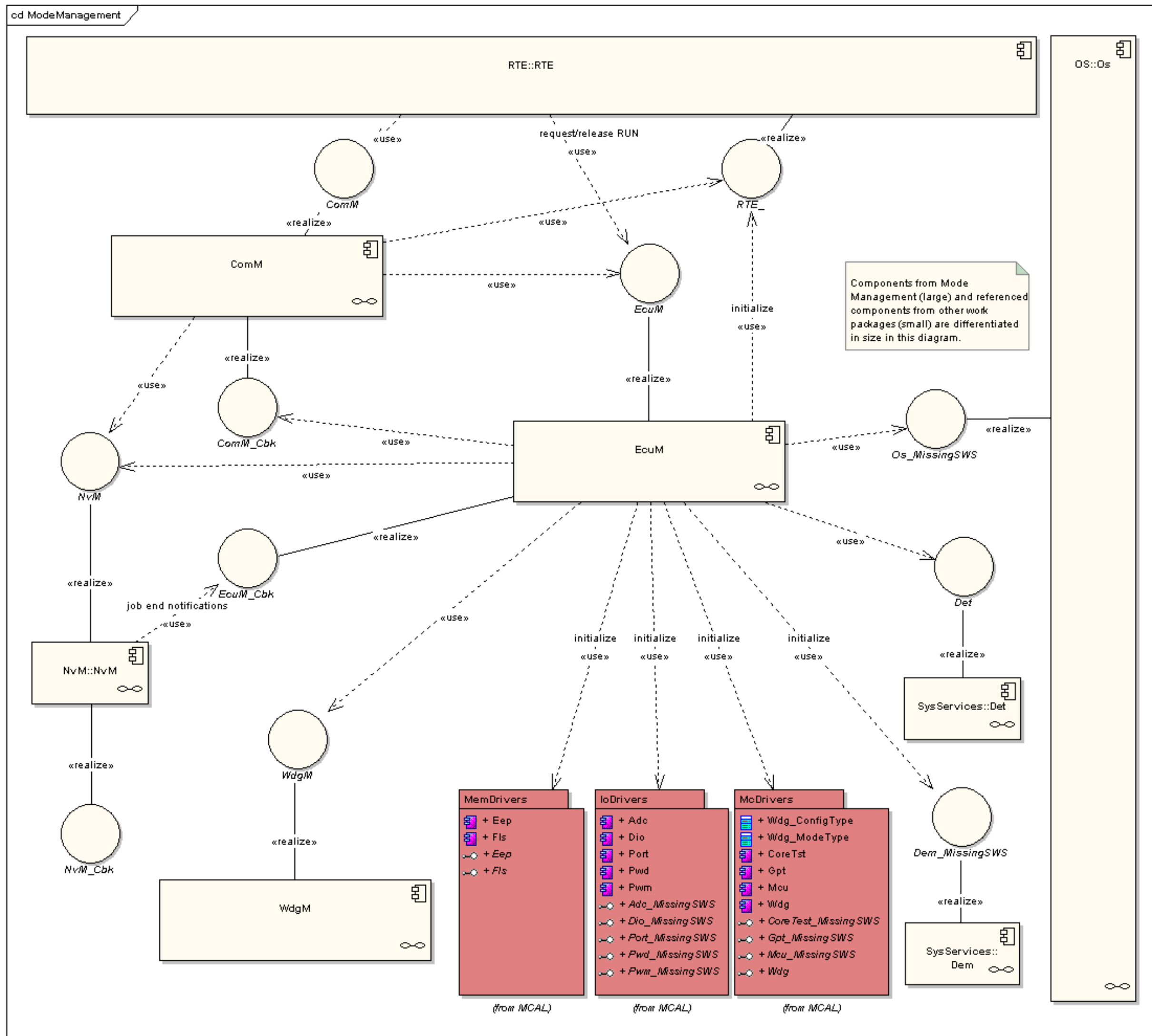


**Figure 2 - Module Relationship (top level diagram)**

The diagram shows how the ECU State Manager is related to other modules. In most cases, the ECU State Manager is simply responsible for initialization[10]. There are however some modules which have a functional relationship with ECU State Manager which are explained in the following paragraphs.

### 7.2.1 Standardized AUTOSAR Software Modules

Basic Software modules are initialized and shut down by the ECU State Manager.
The RTE is initialized and shut down by the ECU State Manager.

[10] To be precise, the «initialize» stereotype not only means initialization, but could also mean de-initialization.

The OS is initialized and shut down by the ECU State Manager. After the OS initialization, additional initialization steps are undertaken by the ECU State Manager before the RUN state is reached. Execution control is handed over to the ECU State Manager after OS shutdown. Details are provided in the chapters *7.3 STARTUP State* and *7.5 SHUTDOWN State*.

### 7.2.2 Software Components

SW Components contain the application code of an AUTOSAR ECU. Software components shall request the RUN state from the ECU State Manager when they have the need to keep the ECU alive.
If the intent of the SW-C is primarily to communicate then it should request a communication state from the Communication Manager (see [8]). This will implicitly keep the ECU alive. A SW-C should clearly separate between the need to communicate and the need to keep an ECU alive. Mixing up these two ideas may result in an instable shutdown algorithm.
A SW-C interacts with the ECU State Manager using AUTOSAR ports.

### 7.2.3 Resource Managers

The concept of resource managers allows to add new state machines to the BSW (as a part of new BSW modules) which behave like sub-state machines of the RUN state.
In order to collaborate correctly with the ECU State Manager only very few requirements must be met:

**EcuM2153**: A Resource Manager has to define exaclty one idle state which signifies the state where the Resource Manager isn't doing anything but waiting.

**EcuM2154**: A Resource Manager shall transit into its idle state after initialization. It shall request the RUN state from the ECU State Manager whenever it leaves its idle state and it shall release the RUN state when it returns back to its idle state.
The Communication Manager is a resource manager.

## 7.3 STARTUP State

See 7.1.1 STARTUP State for an overview description.

### 7.3.1 High Level Sequence Diagram

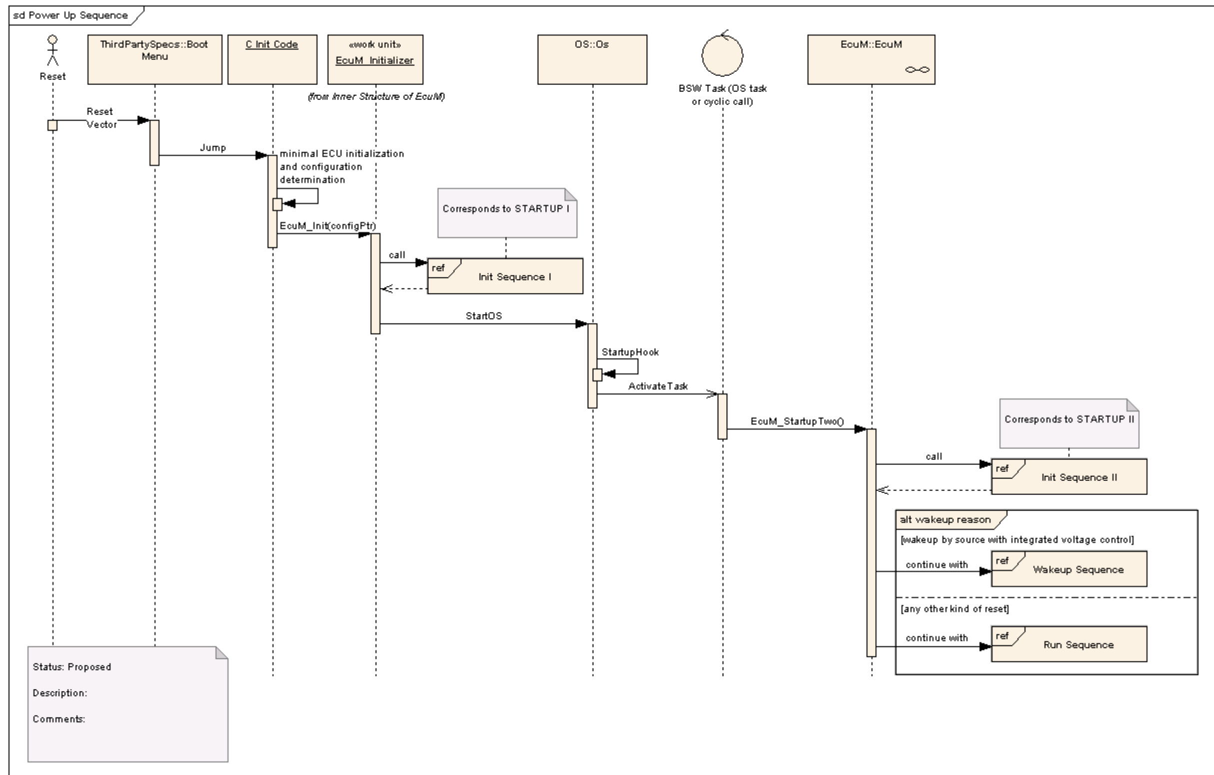


**Figure 3 - Startup Sequence (high level diagram)**

To see adjacent diagrams refer to

*Figure 7 - RUN State Sequence*

*Figure 1 - ECU Main States (top level diagram)*

The startup sequence shows the startup behavior of the ECU. With the invocation of `EcuM_Init` the ECU State Manager takes control of the startup procedure. The startup falls into two parts. The first part, init sequence I or STARTUP I is finished when the AUTOSAR OS is started. The second part, init sequence II or STARTUP II is finished when RTE is started.
To distinguish services which are called before the OS is started from those which are called afterwards and to have a cleaner visualization, the ECU State Manager is split into two parts: The EcuM_Initializer, which runs without OS, and EcuM.

### 7.3.2 Activities before EcuM_Init

The ECU State Manager assumes that before `EcuM_Init` is called a minimal initialization of the MCU has taken place, so that a stack is set up and code can be executed.
Furthermore, it is outside the scope of the ECU State Manager where post-build configuration data is located and how it is loaded. If the ECU contains multiple selectable configurations, it is also outside the scope of the ECU State Manager where these are located and how one of these configurations is selected. This might be read from a hardware pin encoding or stored in non-volatile memory.

The only assumption the ECU State Manager makes is that it is passed a pointer to a fully initialized `EcuM_ConfigType` structure in `EcuM_Init`.

### 7.3.3 STARTUP Activity Overview

**EcuM2411**: The following table shows the activities and the order in which they shall be executed.

| ***Sub-state***<br>***Initialization Activity***[11] | ***Comment*** | ***Opt.***[12] |
|---|---|---|
| **STARTUP I** | | |
| Check consistency of configuration data | If check fails the `EcuM_ErrorHook` is called. See *10.5 Checking Configuration Consistency* for details on the consistency check. | |
| MCU Driver | | no |
| Get reset reason | The reset reason is derived from `Mcu_GetResetReason` and the *EcuM_WakeupSource* configuration. See *10.3.7 EcuM_WakeupSource* for details.<br>**EcuM2623**: The wakeup source resulting from the reset reason translation shall be remembered by the ECU State Manager. See *8.3.3.1 EcuM_SetWakeupEvent* and *8.3.3.5 EcuM_GetValidatedWakeupEvents*.<br>The system designer should precise the reset reasons if the driver for other sources, e.g. the CAN transceiver driver, have been initialized. | |
| Development Error Tracer | | yes |
| Diagnostic Event Manager | Pre-initialization | yes |
| *callout* `EcuM_AL_DriverInitOne` | Init block I<br>The callout may not only contain driver initialization but any kind of pre-OS, low level initialization code.<br>See also *7.3.5 Driver Initialization* | |
| General Purpose Timer | | no |
| Select default shutdown target | | |
| Select application mode | see *7.3.4.1 STARTUP I* for a detailed description | |
| StartOS | OS service | |

[11] Activities marked with × are conditional.
[12] Optional activities can be switched on or off by configuration. See chapter *10.3 Configurable Parameters* for details.

| *Sub-state* *Initialization Activity*[11] | *Comment* | *Opt.*[12] |
|---|---|---|
| **STARTUP II** | | |
| BSW Scheduler | | no |
| *callout* `EcuM_AL_DriverInitTwo` | Init block II see *7.3.5 Driver Initialization* | |
| NVRAM Manager | | yes |
| restore persistent data from NVRAM | This step is only executed if NVRAM Manager is enabled by configured | |
| Watchdog Manager | | yes |
| Diagnostic Event Manager | Regular initialization | yes |
| COM Manager[13] | | no |
| CAN Interface | | |
| CAN Driver | | |
| CAN Transceiver | | |
| CAN TP | | |
| FlexRay Interface | | |
| FlexRay Driver | | |
| FlexRay External Time Services | | |
| FlexRay TP | | |
| LIN Interface | | |
| LIN Low Level Driver | | |
| PDU Router | | |
| Generic NM | | |
| CAN NM | | |
| FlexRay NM | | |
| COM | | |
| Diagnostic Communication Manager | | |
| *callout EcuM_OnRTEStartup* | | |
| RTE | | no |
| × *switch to WAKEUP VALIDATION state* | **EcuM2632**: If one of the wakeup sources listed in *7.8.7 Wakeup Sources and Reset Reason* is set, then exection shall continue with RUN state. In all other cases, execution shall continue with WAKEUP VALIDATION state. | |
| × *switch to RUN state* | | |
| Indicate mode change to RTE | Indicated mode is `Sleep` if next state is WAKEUP VALIDATION, indicated mode is `Run` if next state is RUN. | |

**Table 1 - Initialization Activities[14]**

[13] The initialization order of the COM related components is given in the Communication Manager SWS. The bus transceiver drivers are not initialized by the Communication Manager but must be initialized in the driver initialization callouts. See [8].
[14] Activities which are not under direct control of the ECU State Manager are shown grey-shaded.

### 7.3.4 Sub-State Descriptions

#### 7.3.4.1 STARTUP I

The STARTUP state is entered with a call of the API function `EcuM_Init`.

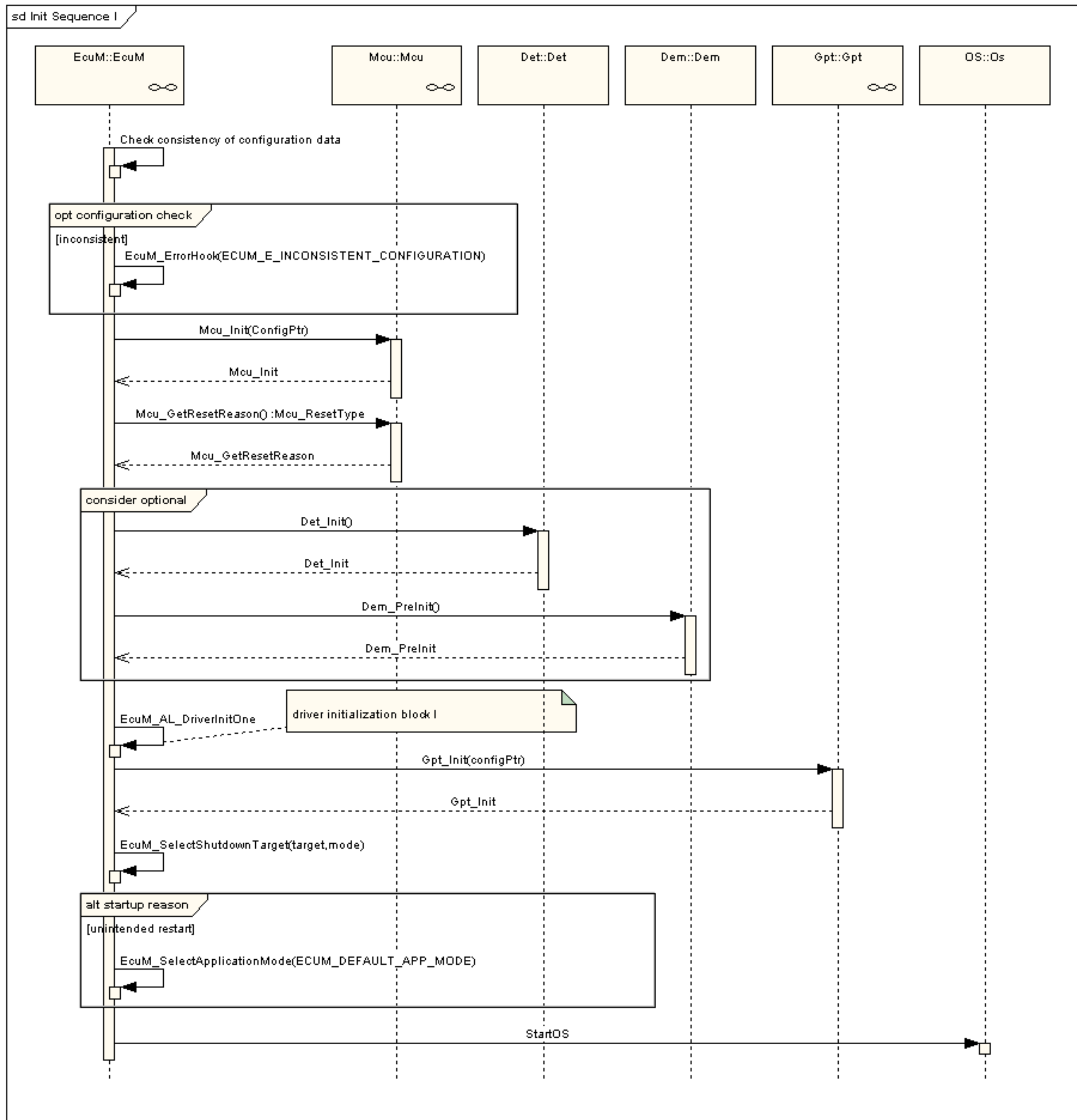


**Figure 4 - Init Sequence I (STARTUP I)**

STARTUP I is intended for preparing the ECU to initialize the OS. The phase should be kept as short as possible. This also applies to the callouts. Initialization of drivers should be done in STARTUP II whenever possible. Interrupts should not be used in

this phase. If interrupts have to be used, only category I interrupts are allowed in this context[15].
Initialization of drivers and hardware abstraction modules is not strictly defined by the ECU State Manager. A callout `EcuM_AL_DriverInitOne` is provided to define the init block I, an initialization activity during STARTUP I, where initialization can take place. Modules which need OS support can be placed into init block II (see *7.3.4.2 STARTUP II*)

**EcuM2271**: MCU_Init does not provide complete MCU initialization. Additional, hardware dependent steps have to be executed and must be defined at system design time. These steps are supposed to be taken within the `EcuM_AL_DriverInitOne` callout. Details can be found in [15].

**EcuM2181**: ECU State Manager must call `EcuM_SelectShutdownTarget` with the configured default shutdown target (see *7.6.1 Shutdown Targets*, *7.9 Time Triggered Increased Inoperation* and *10.3 Configurable Parameters*).

**EcuM2242**: If the restart was unintended the ECU State Manager must select the default application mode with a call to `EcuM_SelectApplicationMode`. Unintended restarts are power on restarts and restarts due to fault conditions like watchdog resets. In all other cases, the application mode shall not be changed. See *7.11.1 Application Modes* for details how to change the application mode.

**EcuM2603**: At the end of the STARTUP I state, it must be possible to start the OS. All basic software modules which are needed by the OS must be initialized by this time. Modules left out so far may be initialized later in STARTUP II

**EcuM2141**: The application mode parameter of the `StartOS` service shall be retrieved with the API call `EcuM_GetApplicationMode`. For more details about application modes see also *7.11.1 Application Modes*.

[15] Category II interrupts require a running OS while category I interrupts do not. AUTOSAR OS requires each interrupt vector to be exclusively put into one category.

### 7.3.4.2 STARTUP II

STARTUP II is carried out by the `EcuM_MainFunction`.
A callout `EcuM_AL_DriverInitTwo` is provided, where initialization of those basic software modules should take place, which need OS support.

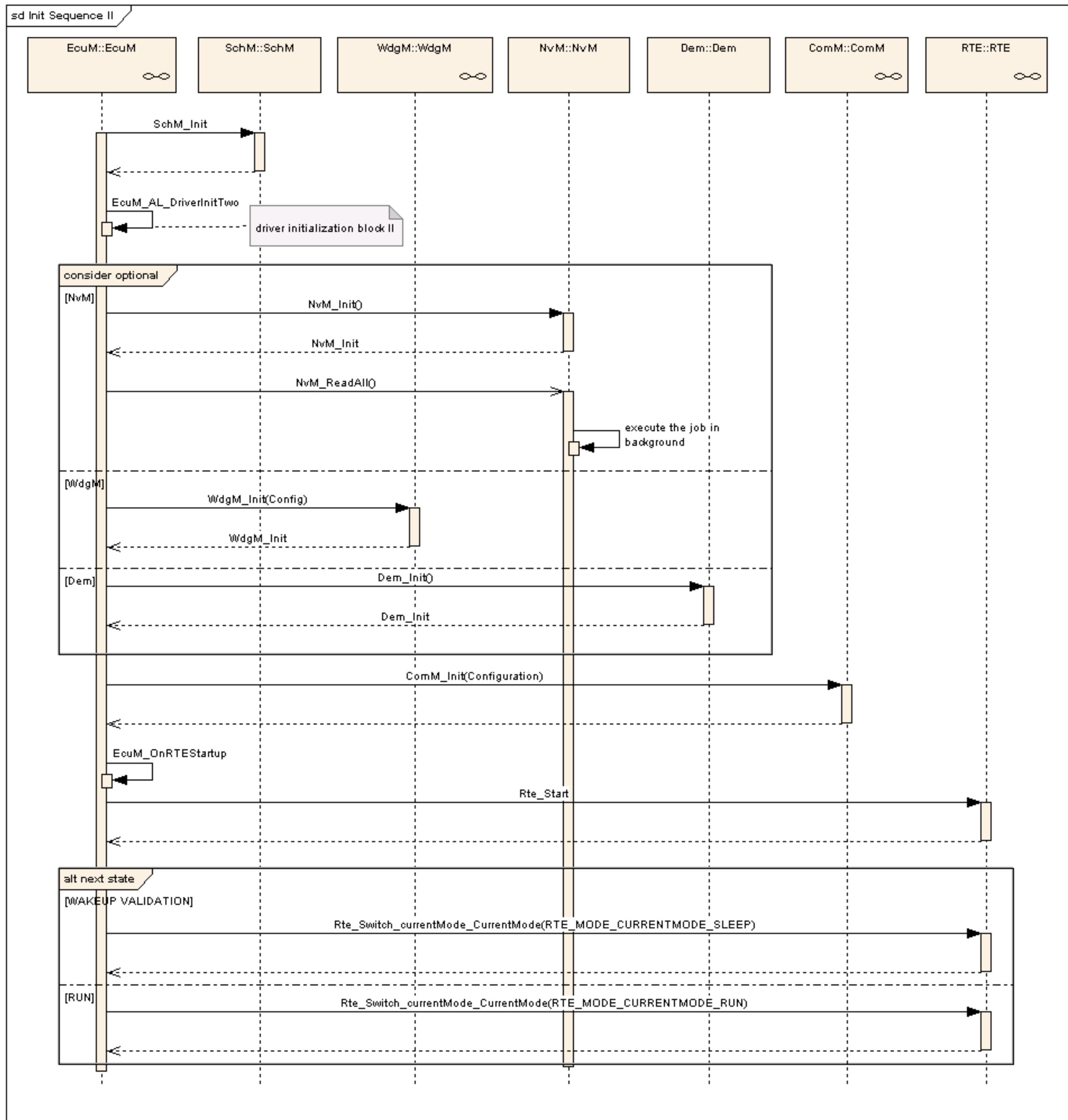


**Figure 5 - Init Sequence II (STARTUP II)**

Initialization of Communication Manager is described in [9].

### 7.3.5 Driver Initialization

This chapter applies to drivers of the AUTOSAR Basic Software which are not handled directly by the ECU State Manager.
A driver's location in the initialization process depends strongly on its implementation and the target hardware design. Drivers can be initialized from the driver init blocks I and II during STARTUP I and II respectively.

**EcuM2559**: The order inside of the blocks shall be generated from configuration information (see *10.3 Configurable Parameters*, *ECUM_DRIVER_INIT_LIST_ONE*, *ECUM_DRIVER_INIT_LIST_TWO*).

**EcuM2730**: For each driver, its init function with the configured init configuration shall be called. The init parameter for the init function shall be derived from driver's configuration. See the according driver specification for details.
Some drivers may need re-initialization when the ECU is woken up. This is especially true for drivers with wakeup sources. For re-initialization, a restart block is defined. The restart block is part of the WAKEUP state.

**EcuM2561**: The restart list will typically only contain a subset of drivers. But drivers shall appear in the same order as in the combined list of init block I and init block II (see *10.3 Configurable Parameters*, *ECUM_DRIVER_RESTART_LIST*).

**EcuM2562**: Drivers which serve wakeup sources must be re-initialized in the restart block. The driver restart shall re-arm the trigger mechanism of the 'wakeup detected' callback (see *7.7.4.1 WAKEUP I*).

**EcuM2563**: If hardware is put into a sleep mode during SHUTDOWN then this hardware must be restarted by its driver.
The restart list will be invoked in state WAKEUP I (see *7.1.5 WAKEUP State*).

| ***Recommended Init Block*** | |
|---|---|
| ***Init Activity*** | ***Comment*** |
| Init Block I[16] | |
| Watchdog Driver | The hardware watchdog should be initialized to a reasonably slow watchdog timeout. The watchdog will be triggered as soon as the Watchdog Manager has been initialized. |
| ADC Driver | |
| ICU Driver | |
| PWM Driver | |
| Init Block II[17] | |
| SPI Driver | |
| EEPROM Driver | |
| Flash Driver | |

**Table 2 - Driver Initialization Details, Sample Configuration**

[16] Drivers in Init Block I are listed in the *ECUM_DRIVER_INIT_LIST_ONE* configuration parameter. See *10.3 Configurable Parameters*.
[17] Drivers in Init Block II are listed in the *ECUM_DRIVER_INIT_LIST_TWO* configuration parameter.

### 7.3.6 DET Initialization

The Development Error Tracer is a software module for debug and bringup purpose. During the development cycle the behavior of this module may change. As a consequence, the initialization sequence is likely to change, too.

**EcuM2783**: DET shall be initialized early during STARTUP I by the ECU State Manager.

**EcuM2634**: DET is not *started* by default but the system designer has to configure the point where DET is started, preferably into one of the following callouts: `EcuM_AL_DriverInitOne`, `EcuM_AL_DriverInitTwo`. DET is started by invoking `Det_Start`.

## 7.4 RUN State

Ref. to *7.1.2 RUN State* for an overview description.
All activities in the RUN state described in this chapter are carried out in the `EcuM_MainFunction` service.

### 7.4.1 State Breakdown Structure

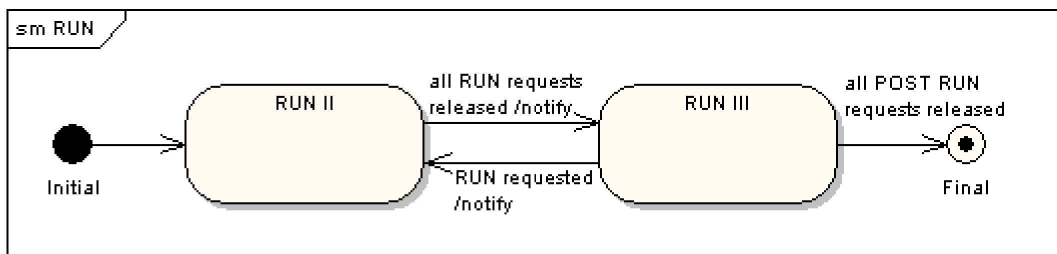


**Figure 6 - RUN State Breakdown**

### 7.4.2 High Level Sequence Diagram

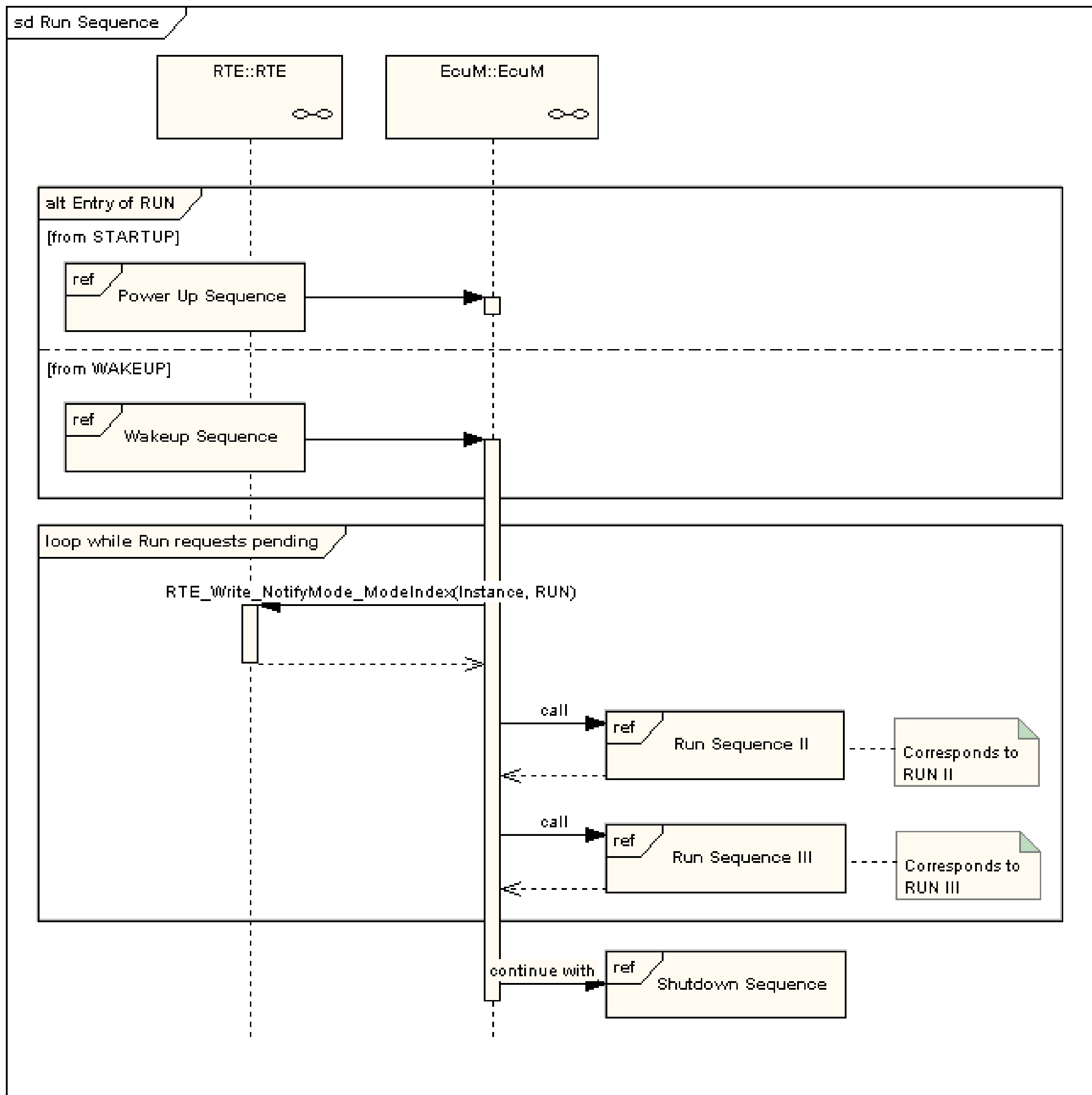


**Figure 7 - RUN State Sequence (high level diagram)**

To see adjacent diagrams refer to

*Figure 3 - Startup Sequence (high level* diagram)

*Figure 16 - Wakeup Sequence (high level* diagram)

*Figure 11 - Shutdown Sequence (high level* diagram

*Figure 1 - ECU Main States (top level* diagram)

### 7.4.3 Sub-State Description

#### 7.4.3.1 RUN II

RUN II is the state in which applications and SW-C's should execute their regular tasks.

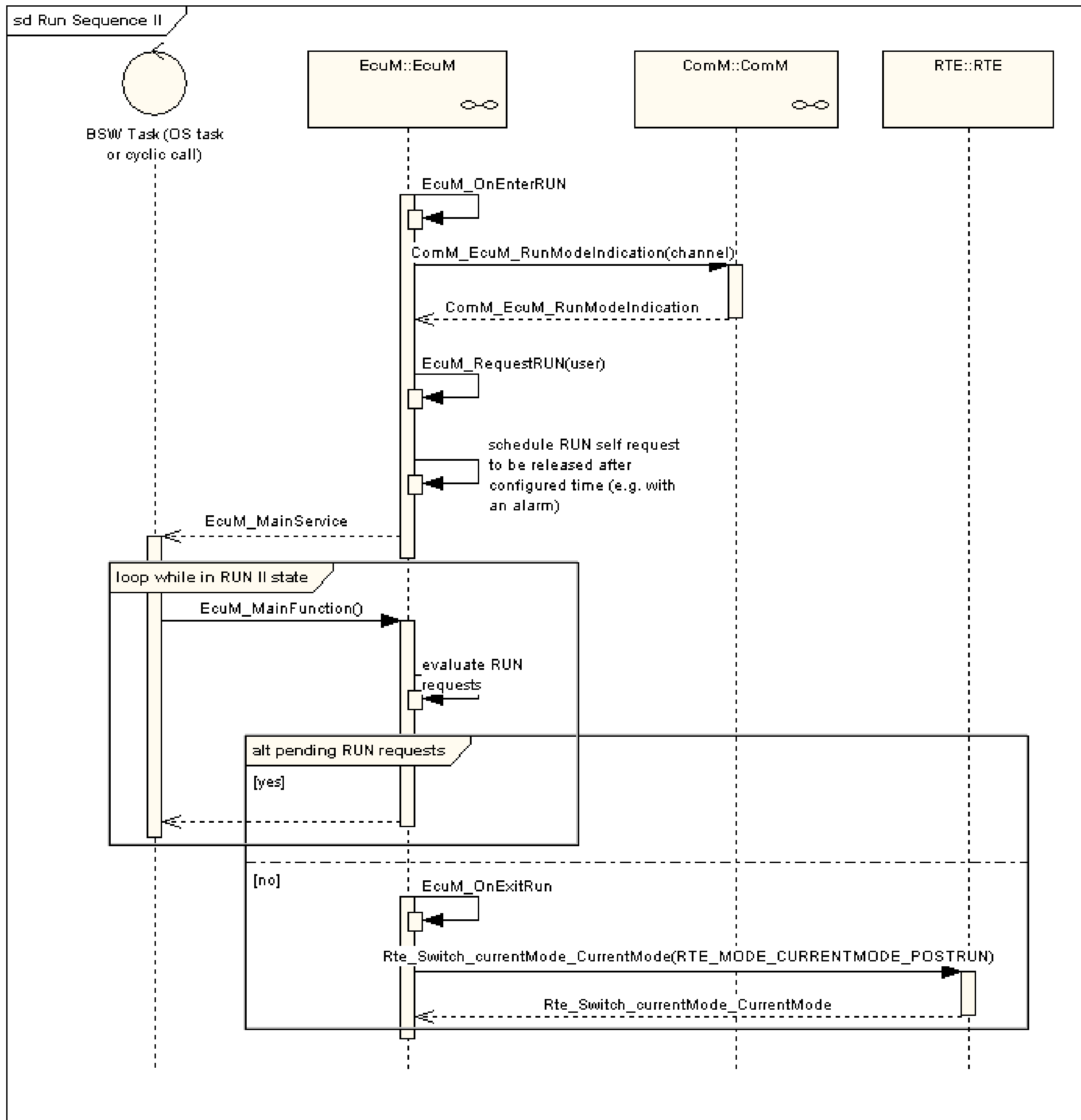


**Figure 8 - RUN II State Sequence**

#### 7.4.3.2 Entering RUN II State

On entering RUN state, the following steps must be done in the presented order:

**EcuM2308**: The callout `EcuM_OnEnterRun` must be invoked.

**EcuM2384**: RUN must be indicated to the Communication Manager by invoking `Com_Cbk_EcuM_RunModeIndication`, see [8].

**EcuM2310**: A request to RUN shall be issued by the ECU State Manager itself. The RUN request shall be released after a configurable period after the RUN state has been entered (see *10.3 Configurable Parameters*).

#### 7.4.3.3 Leaving RUN II State

**EcuM2311**: When the last RUN request has been released, ECU State Manager shall advance to the APP POST RUN state. The evaluation is done with the next cyclic invocation of `EcuM_MainFunction`.
If a SW-C needs post run activity during RUN III (e.g. shutdown preparation), then it must request POST RUN before releasing the RUN request. Otherwise it is not guaranteed that this SW-C will get a chance to run its POST RUN code.
The Communication Manager will not release RUN unless the no communication state is reached.

#### 7.4.3.4 RUN III

RUN III state provides a post run phase for SW-C's and allows them to save important data or switch off peripherals before the ECU State Manager continues with the shutdown process.

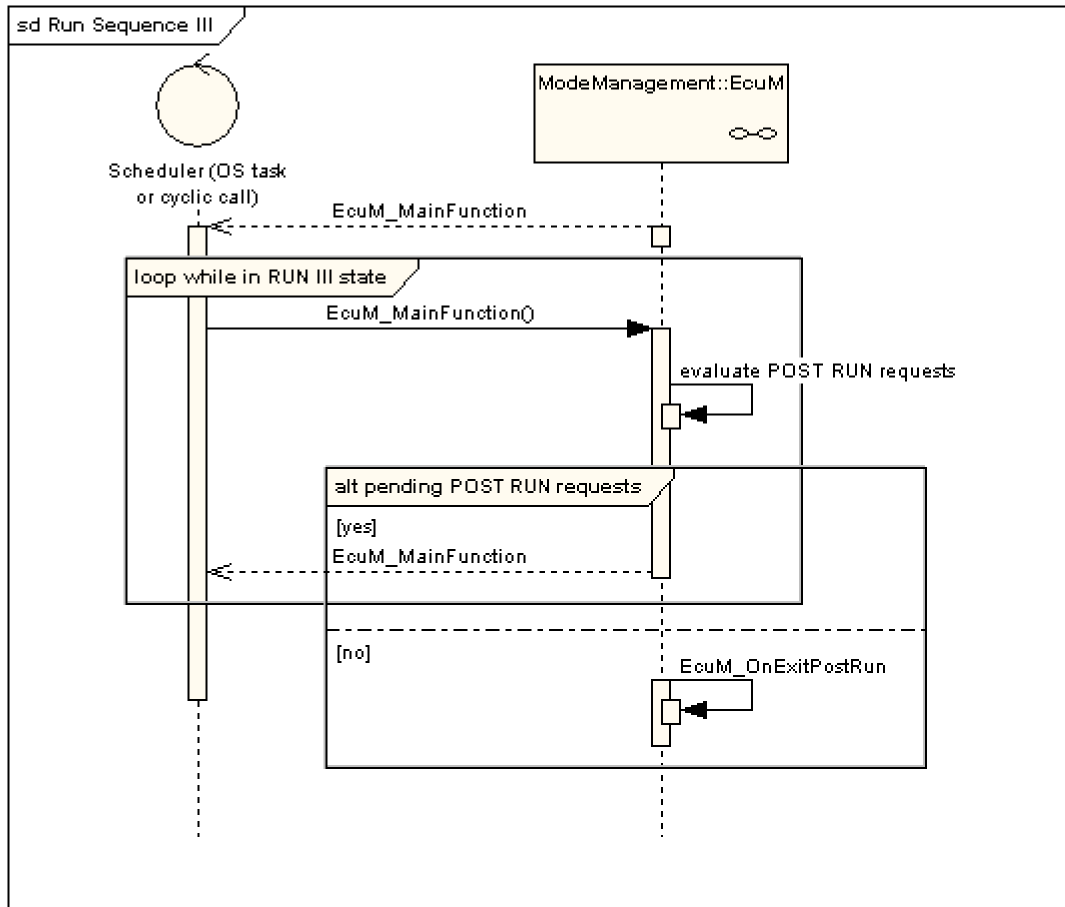


**Figure 9 - RUN III State Sequence**

#### 7.4.3.5 Leaving RUN III State

**EcuM2761**: When the last POST RUN request has been released, ECU State Manager shall advance to the shutdown state. The evaluation is done with the next cyclic invocation of `EcuM_MainFunction`.

## 7.5 SHUTDOWN State

Ref. to 7.1.3 SHUTDOWN State for an overview description.

**EcuM2188**: When SHUTDOWN state is entered and shutdown target is SLEEP, no wakeup event shall be missed. If a valid wakeup event occurs while the ECU is in transition to SLEEP the ECU shall as quickly as possible proceed to the WAKEUP state and shall not enter the SLEEP state.

**EcuM2756**: When a wakeup event occurs during the shutdown phase and the shutdown target is OFF or RESET, then the shutdown shall complete but the ECU shall restart immediately thereafter.

### 7.5.1 State Breakdown Structure

When the SHUTDOWN state is entered, applications have de-initialized and the communication stack has been put into the no communication state[18]. Ref. to *7.4.3.3 Leaving RUN II* State for details.

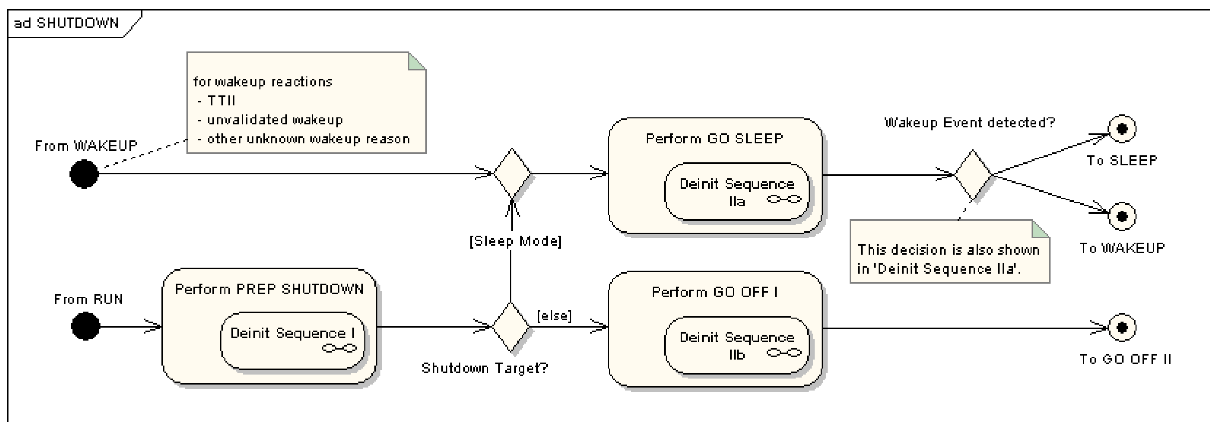


**Figure 10 - Fine Structure of SHUTDOWN**

[18] This statement is only true for SW-Cs which are registered users of the ECU State or Communication Manager. All other SW-C may be terminated by the system without warning.

### 7.5.2 High Level Sequence Diagram

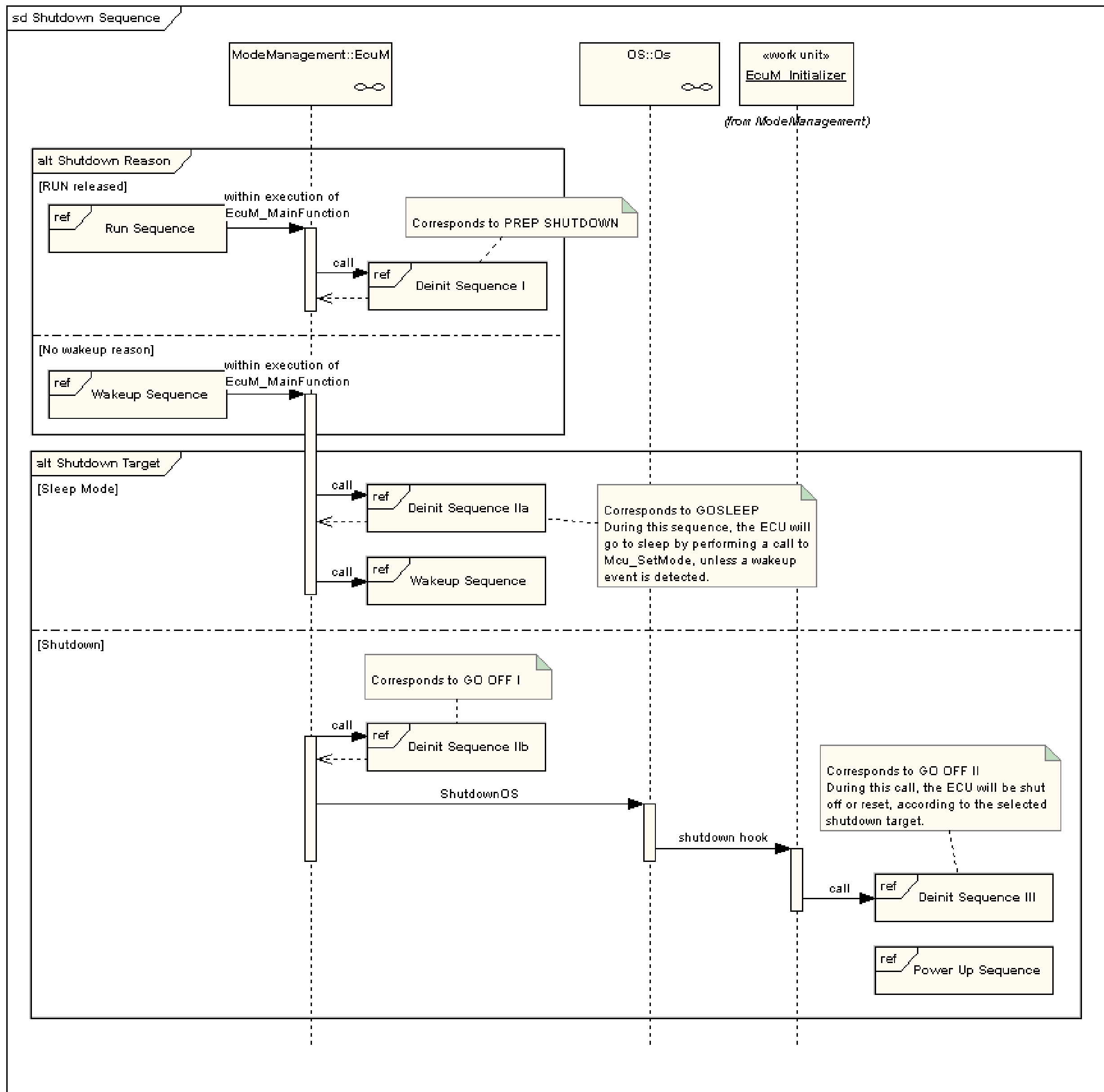


**Figure 11 - Shutdown Sequence (high level diagram)**

To see adjacent diagrams refer to

*Figure 7 - RUN State Sequence (high level* diagram)

*Figure 16 - Wakeup Sequence (high level* diagram)

*Figure 3 - Startup Sequence (high level* diagram)

*Figure 1 - ECU Main States (top level* diagram)

EcuM_Initializer is an interface of the ECU State Manager. It is only introduced here to improve readability of the diagram. See also
*Figure 3 - Startup Sequence (high level diagram)* and its comments.

### 7.5.3 SHUTDOWN Activity Overview

| ***Sub-state***[19] | ***Shutdown Activity*** | ***Comment*** | ***Optional***[20] |
|---|---|---|---|
| **PREP SHUTDOWN** | | | |
| | Clear wakeup events | | |
| | *callout* `EcuM_OnPrepShutdown` | | |
| | de-initialize Diagnostic Event Manager | | yes |
| | Indicate mode change to RTE | Indicated mode is `Sleep` if next state is GO SLEEP, indicated mode is `StartupShutdown` if next state is GO OFF I. | |
| **GO SLEEP** | | | |
| | *callout* `EcuM_OnGoSleep` | | |
| | Disable all interrupts | | |
| | *callout* `EcuM_PutWakeupSourcesToSleep` | see *10.6.1 Managing Wakeup Sources* | |
| | Enable all interrupts | | |
| | save persistent data to NVRAM | An incoming wakeup event will cancel an ongoing write job | yes |
| | Check for pending wakeup events | Purpose is to detect wakeup events that occurred while interrupts were disabled | |
| × | *Callout* `EcuM_GenerateRamHash` | critical section[21] | |
| × | Mcu_SetMode | | |
| **GO OFF I** | | | |
| | *callout* `EcuM_OnGoOffOne` | | |
| | RTE | | |
| | COM Manager | | |
| | Watchdog Manager | | yes |
| | save persistent data to NVRAM | | yes |
| | Check for pending wakeup events | Purpose is to detect wakeup events that occurred while interrupts were disabled | |
| × | Set RESET as shutdown target | This action shall only be carried out when pending wakeup events were detected | |
| | ShutdownOS | OS service | |
| **GO OFF II** | | | |
| | *Callout* `EcuM_OnGoOffTwo` | | |

[19] Rows marked with × are conditional.
[20] Optional activities can be switched on or off by configuration. It shall be the system designers choice if a module is compiled in or not for an ECU design. See chapter *10.3 Configurable Parameters* for details.
[21] The objective of the critical section is that no other code shall be able to run which could destroy the RAM hash. Which are the adequate means depends on the implementation.

| Sub-state[19] | Shutdown Activity | Comment | Optional [20] |
|---|---|---|---|
| × | Mcu_PerformReset | Switch statement. The case statement is selected by the shutdown target (reset or off) | |
| × | *Callout* `EcuM_AL_SwitchOff` | | |
| The following modules need not to be shut down: | | | |
| | NVRAM Manager | | |
| All other modules are not shutdown automatically. The following basic software modules must not be shut down at all. | | | |
| | None | | |

**Table 3 - Shutdown Activities**

### 7.5.4 Sub-State Descriptions

#### 7.5.4.1 PREP SHUTDOWN

PREP SHUTDOWN is a state common for all shutdown targets, i.e. SLEEP, OFF, reset, etc. During this state, handlers and managers of the basic software are shut down.

**EcuM2288**: If the shutdown target is not any of the sleep modes, then control has to be handed over to GO OFF I (ref. *7.5.4.3 GO OFF I*) after activities of this state have finished.

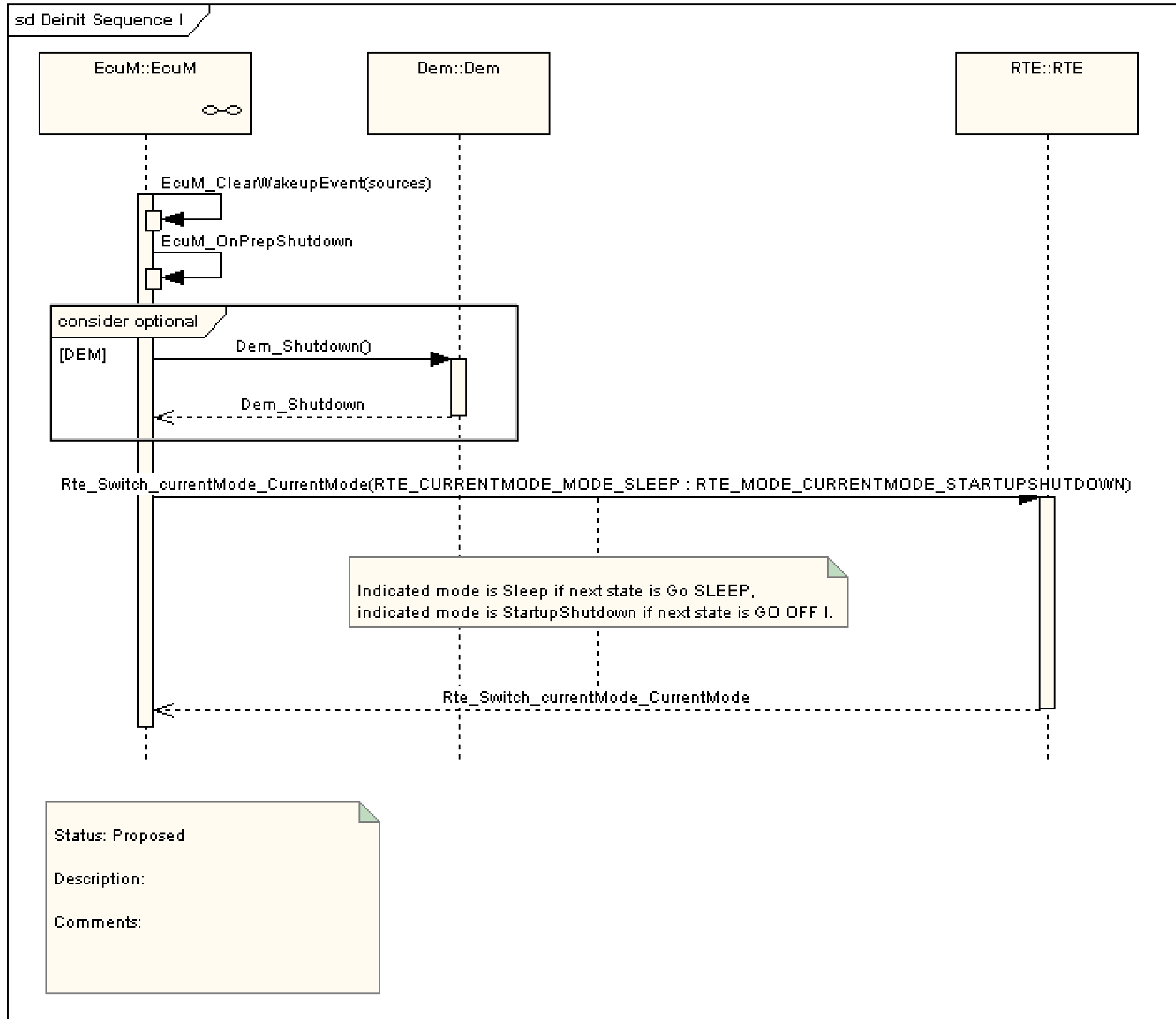


**Figure 12 - Deinitialization Sequence I (PREP SHUTDOWN)**

### 7.5.4.2 GO SLEEP

Purpose of GO SLEEP is to configure hardware for the following sleep phase and to setup the ECU for the next wakeup event.

**EcuM2389**: Wakeup sources shall be configured from the callout `EcuM_PutWakeupSourcesToSleep`. These wakeup source specific callouts are described in chapter *10.6.1 Managing Wakeup Sources*.
In contrast to shutdown, the OS is not shut down when entering the sleep state. The sleep mode shall be transparent to the OS.
If an ECU is held in sleep mode for a rather long time, memory contents might get corrupted. To prevent unforeseen behavior, a callout is invoked where the system designer can provide an adequate checksum algorithm for RAM. A similar callout is provided in WAKEUP where RAM integrity can be checked. See also *7.7.4.1 WAKEUP I* and EcuM_GenerateRamHash.

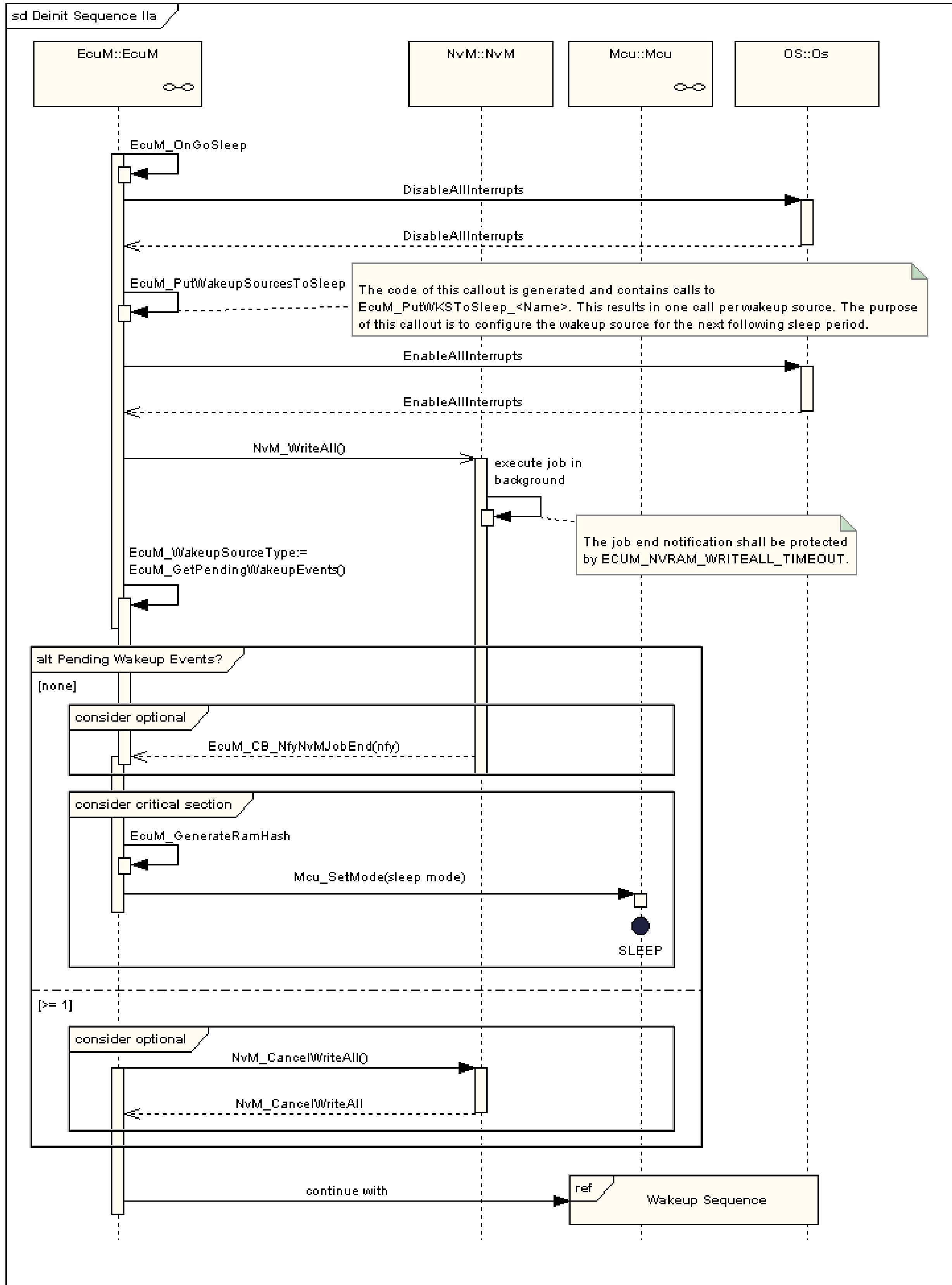


**Figure 13 - Deinitialization Sequence IIa (GOSLEEP)**

Setting up wakeup sources is explained in detail in *10.6.1 Managing Wakeup Sources*.

#### 7.5.4.3 GO OFF I

GO OFF I is carried out under OS control and is implemented by the `EcuM_MainFunction` service.

**EcuM2328**: As its last activity, the `ShutdownOS` service shall be called. This service will end up in the shutdown hook. The shutdown hook in turn shall call `EcuM_Shutdown` to terminate the shutdown process. `EcuM_Shutdown` will not return but switch off the ECU or issue a reset.

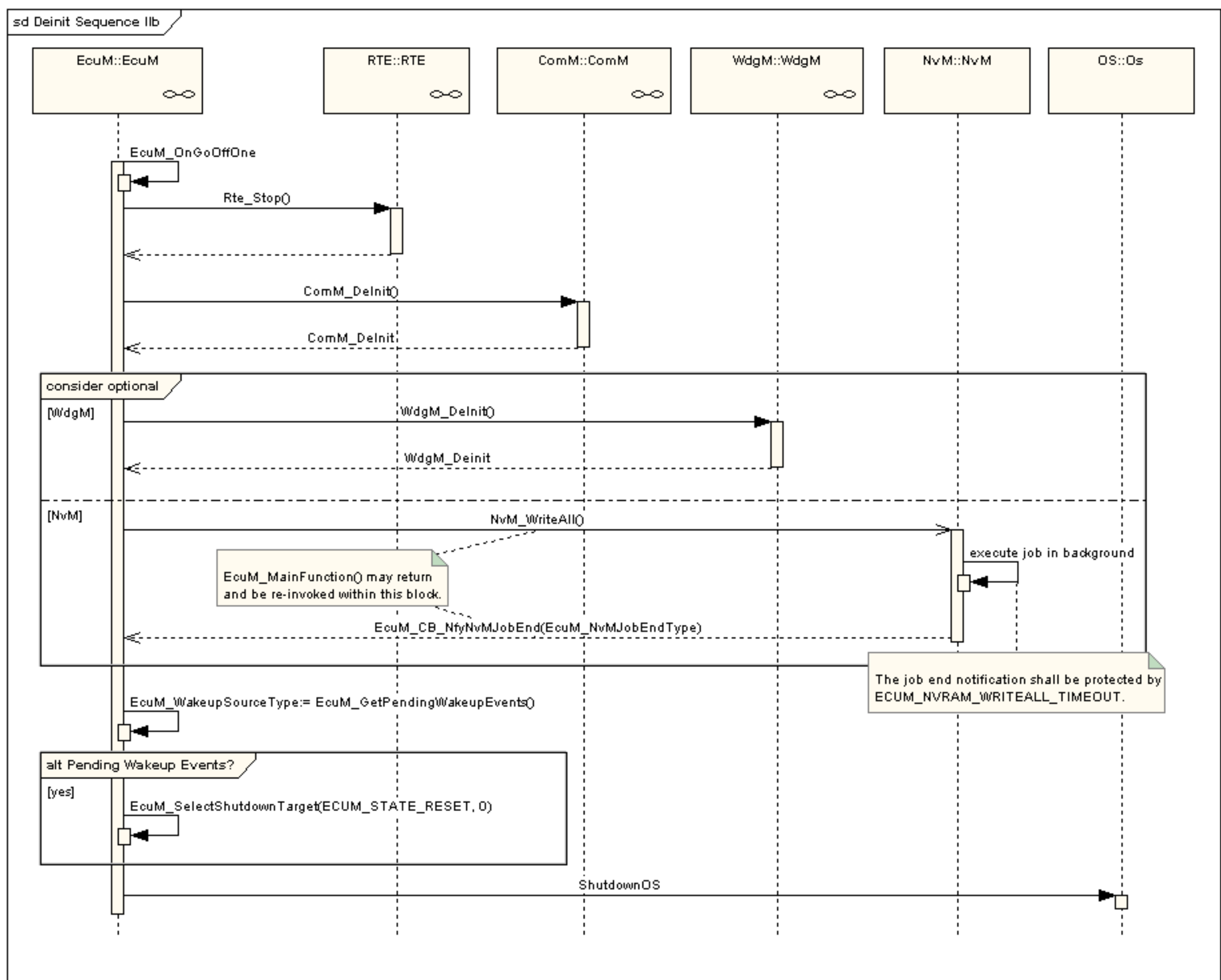


**Figure 14 - Deinitialization Sequence IIb (GO OFF I)**

#### 7.5.4.4 GO OFF II

This state implements the final steps to reach the shutdown target after the OS has been shut down.

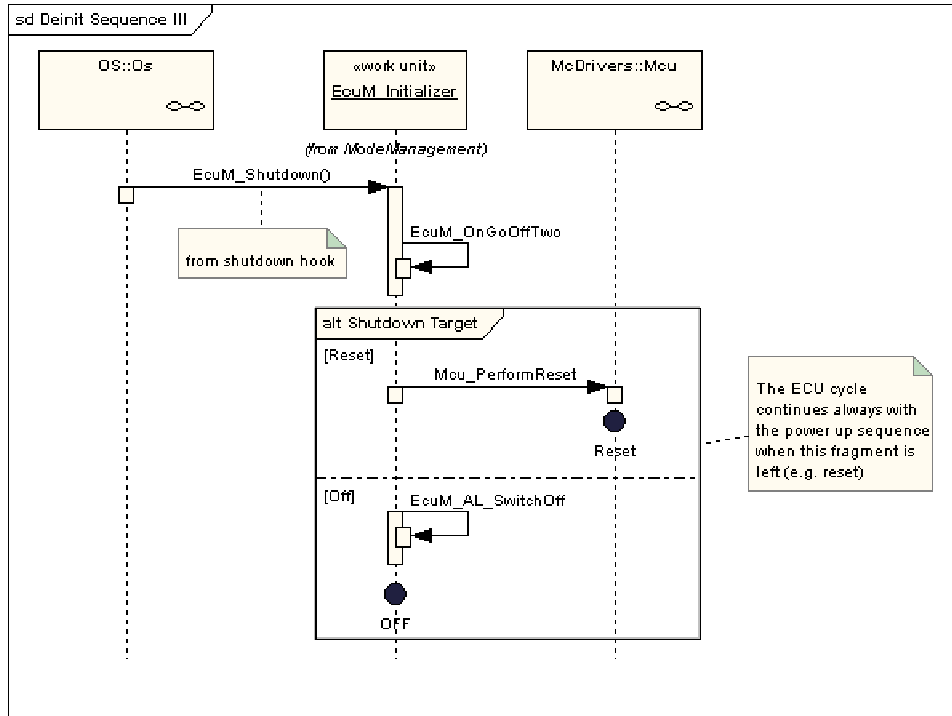


**Figure 15 - Deinitialization Sequence III (GO OFF II)**

The shutdown target RESET is reached by invoking the `Mcu_PerformReset` service of the MCU driver (see [15]).
The shutdown target OFF is implemented by the `EcuM_AL_SwitchOff` callout which must be filled at configuration time. See *8.6.5.7 EcuM_AL_SwitchOff* for details.
EcuM_Initializer is only introduced to improve readability of the diagram. See also *Figure 3 - Startup Sequence (high level* diagram) and its comments.

## 7.6 SLEEP State

Ref. to 7.1.4 SLEEP State for an overview description.

### 7.6.1 Shutdown Targets

Shutdown Targets is a descriptive term for all states and their modes or sub-states where no code is executed. They are called shutdown targets because it is the final

state where the state machine will drive to when RUN state is left. The following states are shutdown targets:

- OFF[22]
- SLEEP
- Reset
  is only a transient a state, but also can be selected as shutdown target.

**EcuM2232**: A default shutdown target can be defined by configuration. This shutdown target can be overridden by calling `EcuM_SelectShutdownTarget`.

The SLEEP state can define a configurable set of sleep modes, where each mode itself is a shutdown target (the bullet list above is a simplification). These sleep modes are hardware dependent and differ typically in clock settings or other low power features provided by the hardware. These different features are accessible through the MCU driver as so called MCU modes (see [15]). The ECU State Manager allows to map these MCU modes to ECU sleep modes and hence they are addressable as shutdown targets. Further the configuration allows to define aliases for shutdown targets to simplify portability of code across different ECUs. See *10.3.9 EcuM_SleepMode* for details.

### 7.6.2 Leaving SLEEP State

Regular exits of the SLEEP state are a result of a wakeup event (toggling a wakeup line, communication on a CAN bus etc.). An ISR may be invoked to handle the event, but this is specific to hardware and driver implementation. Finally, the `MCU_SetMode` service of the MCU driver will return and the ECU State Manager will regain control. Execution then continues with the WAKEUP state.
Irregular events are a hardware reset or a power cycle. In this case, the ECU will restart from the STARTUP state.

[22] The OFF state requires the capability of the ECU to switch off itself. This is not granted for all hardware designs.

## 7.7 WAKEUP State

### 7.7.1 High Level Sequence Diagram

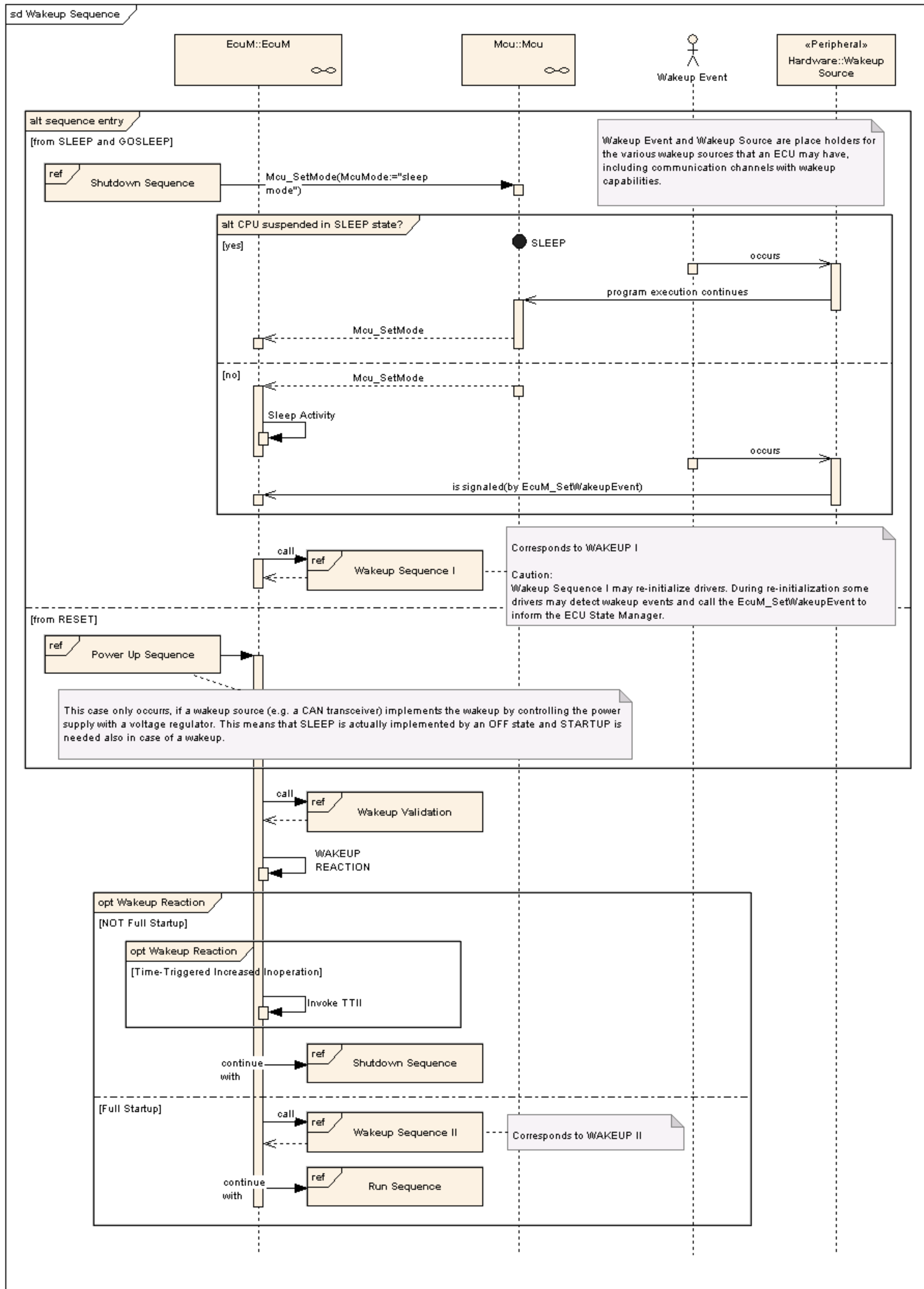


**Figure 16 - Wakeup Sequence (high level diagram)**

To see adjacent diagrams, refer to

*Figure 11 - Shutdown Sequence (high level* diagram

*Figure 7 - RUN State Sequence (high level* diagram)

*Figure 1 - ECU Main States (top level* diagram)

### 7.7.2 State Breakdown Structure

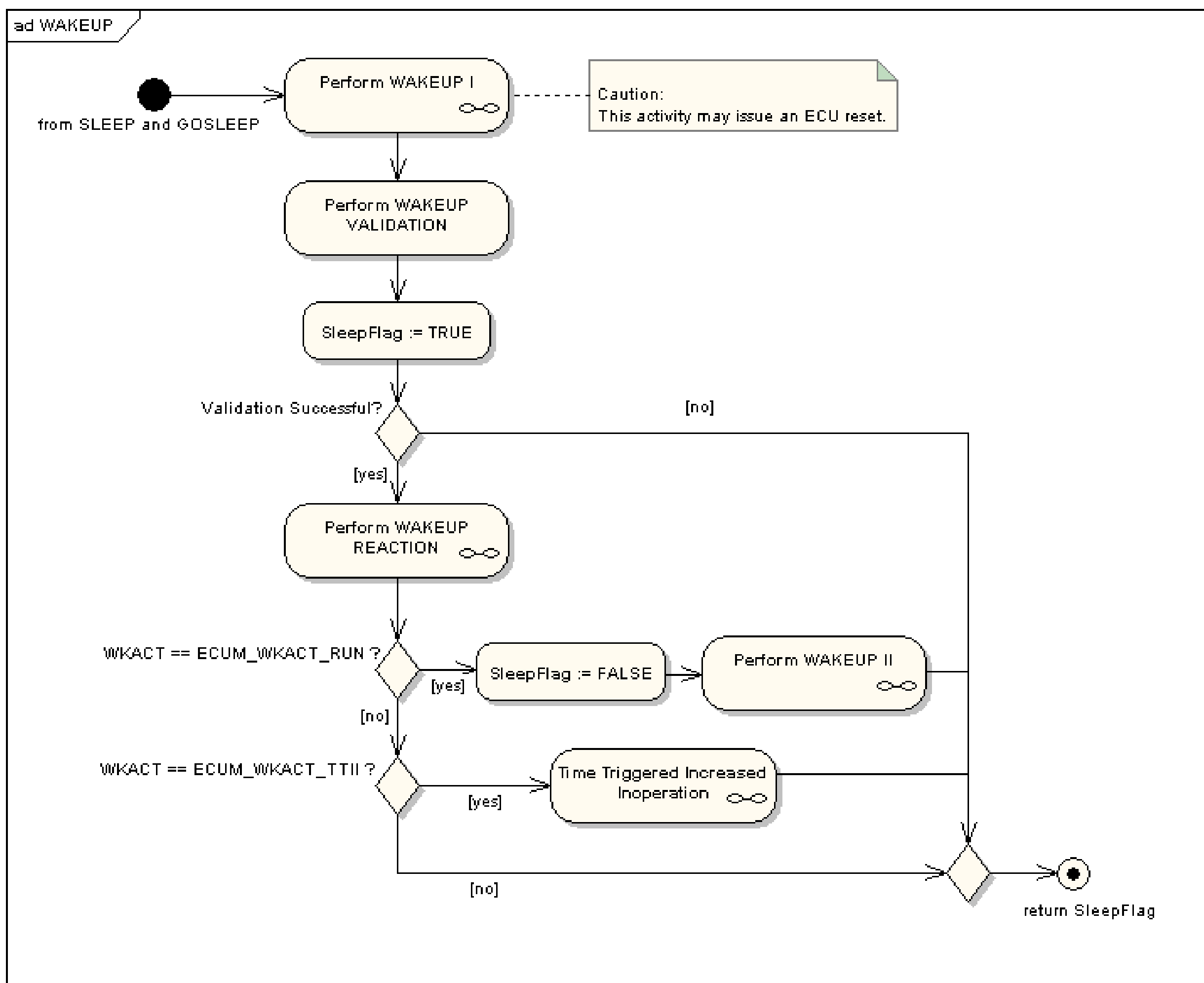


**Figure 17 - WAKEUP State Breakdown**

### 7.7.3 WAKEUP Activity Overview

| *Sub-state*[23] | *Wakeup Activity* | *Comment* | *Opt.* |
|---|---|---|---|
| **WAKEUP I** | | | |
| | *Callout EcuM_CheckRamHash* | | |
| | *Callout EcuM_AL_DriverRestart* | | |
| **WAKEUP VALIDATION** | | see chapter *7.7.4.2 WAKEUP VALIDATION* | |
| **WAKEUP REACTION** | | | |
| | Compute wakeup reaction | see chapter 7.7.4.3 below | |
| | *Callout EcuM_OnWakeupReaction* | | |
| × | Invoke TTII protocol | see chapter 7.9 below | |
| **WAKEUP II** | | | |
| | initialize Diagnostic Event Manager | | yes |
| × | indicate mode change to RTE | | |

**Table 4 - Wakeup Activities**

[23] Rows marked with × are conditional.

### 7.7.4 Sub-State Descriptions

#### 7.7.4.1 WAKEUP I

A callout is invoked where the system designer can place a RAM integrity check. See also *7.5.4.2 GO SLEEP* and `EcuM_GenerateRamHash`.
The `EcuM_AL_DriverRestart` callout is invoked. This callout is intended for re-initializing drivers. Re-initialization is typically required for drivers with wakeup sources, at least. For more details on driver initialization refer to *7.3.5 Driver Initialization*.

**EcuM2539**: During re-initialization, a driver must check if one of its assigned wakeup sources was the reason for the previous wakeup. If this test is true, it must invoke its 'wakeup detected' callback (see [23] for an example), which in turn has to call the `EcuM_SetWakeupEvent` service. As a result, when WAKEUP I has finished, ECU State Manager has a list of wakeup source candidates. These wakeup source candidates still may need validation. See also *7.8 Wakeup Validation Protocol* for more information.

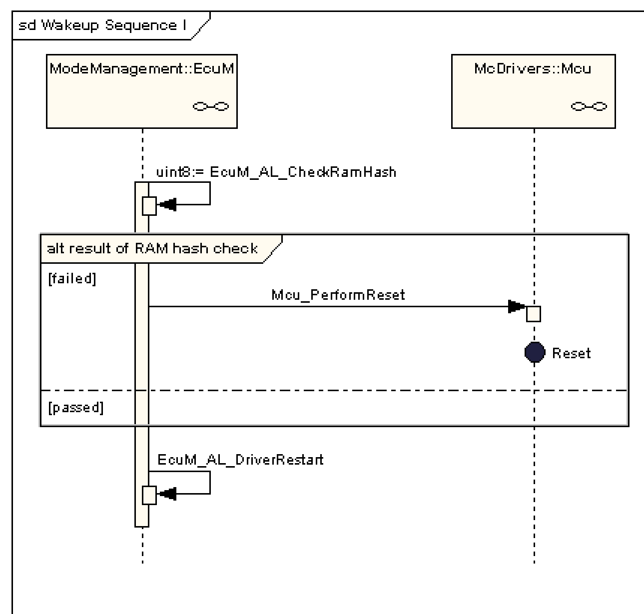


**Figure 18 - Wakeup Sequence I**

**EcuM2545**: The driver should be implemented in a way that it only invokes the wakeup callback once and then requires a dedicated service call to re-arm this mechanism. The driver then needs to be re-armed to fire the callback again.

#### 7.7.4.2 WAKEUP VALIDATION

Because wakeup events can be generated unintended (e.g. EVM spike on CAN line), it is necessary to validate wakeups before the ECU takes up its full operation. The validation mechanism is the same for all wakeup sources. When a wakeup event occurs, the ECU is woken up from its SLEEP state and execution resumes within the `MCU_SetMode` service of the MCU driver[24]. When WAKEUP I is left, the ECU State Manager will have a list of pending wakeup events which need to be validated.

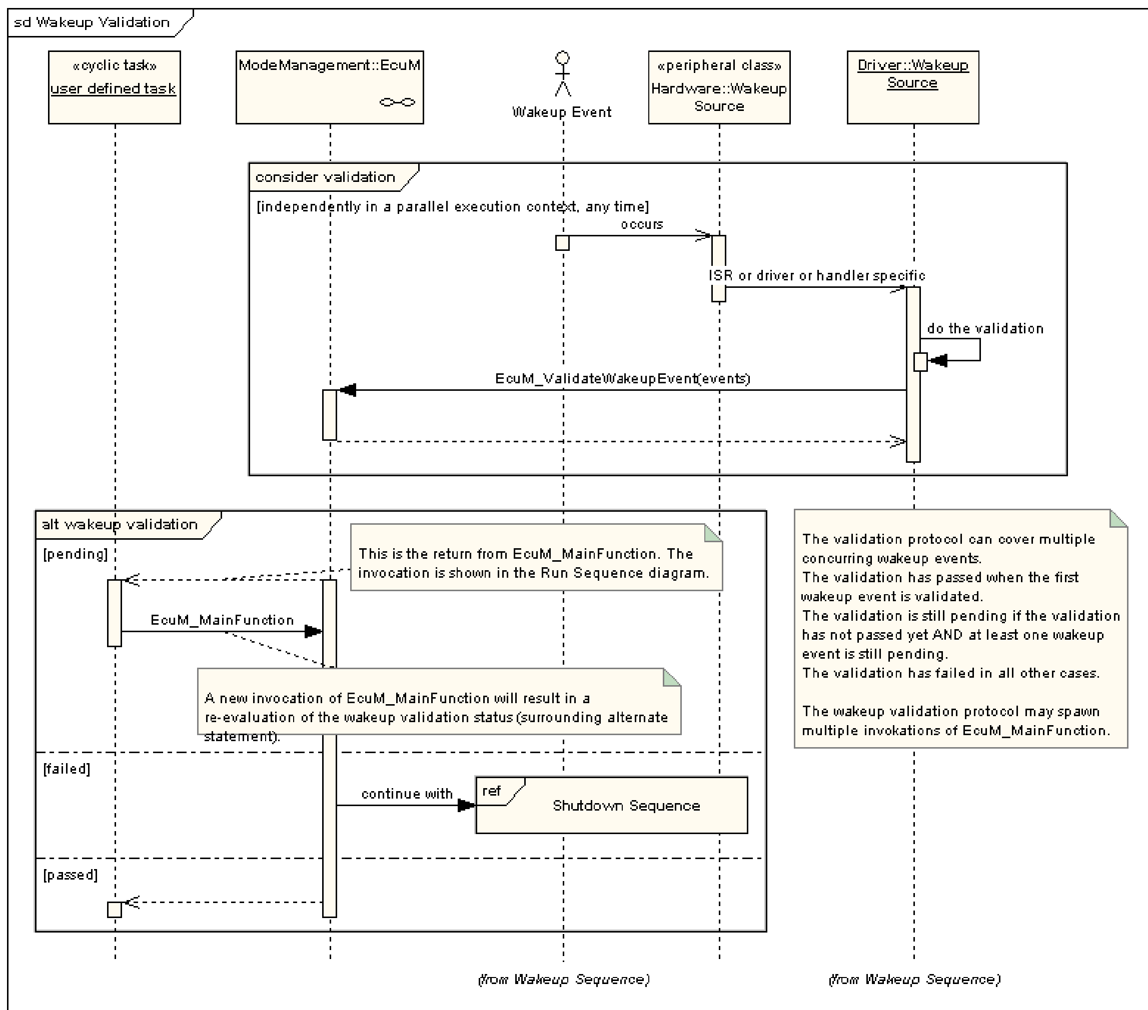


**Figure 19 - Wakeup Validation Sequence**

**EcuM2566**: Wakeup validation shall apply only to those wakeup sources where it is required by configuration. If the validation protocol is not configured, then a call to `EcuM_SetWakeupEvent` shall also imply a call to `EcuM_ValidateWakeupEvent`.

[24] Actually, the first code to be executed may be an ISR, e.g. a wakeup ISR. However, this is specific to hardware and/or driver implementation.

**EcuM2565**: For each pending wakeup event, for which validation is required, a validation timeout shall be started. The timeout is event specific and can be defined by configuration. Strictly spoken, it is sufficient for an implementation to provide only one timer, which is prolonged to the largest timeout when new wakeup events are reported.

**EcuM2567**: If the last timeout expires without validation then the wakeup validation is considered to have failed.

**EcuM2568**: If at least one of the pending events is validated then the entire validation has passed.
Pending events are validated with a call to `EcuM_ValidateWakeupEvent`. This call must be placed in the driver or the consuming stack on top of the driver (e.g. the handler). The best place to put this depends on hardware and software design. See also *7.8.5 Requirements for Drivers with Wakeup Sources*.

### 7.7.4.3 WAKEUP REACTION

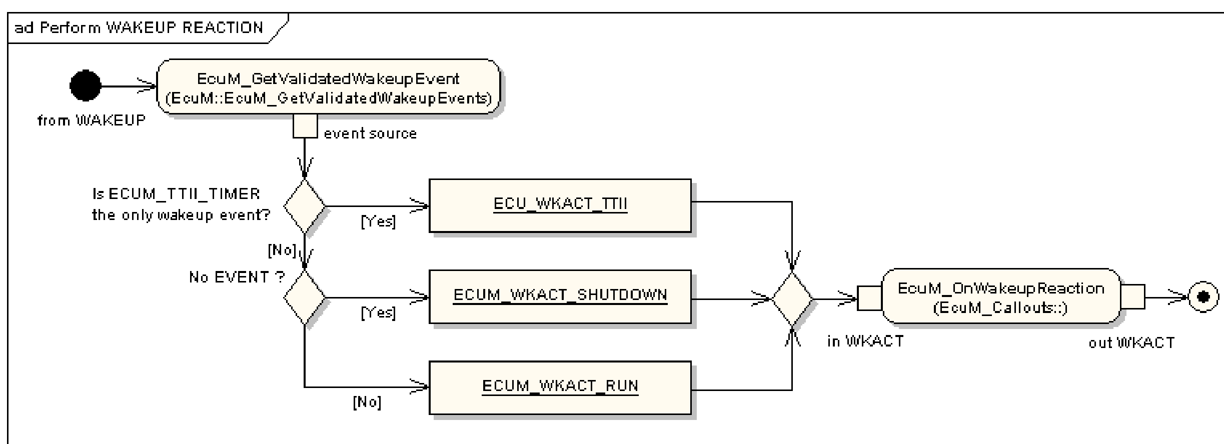


**Figure 20 - Activity Diagram of WAKEUP REACTION**

The WAKEUP REACTION state determines the appropriate wakeup reaction (see *8.2.6 EcuM_WakeupReactionType*) according to the wakeup source (see *8.2.4 EcuM_WakeupSourceType*).
As can be seen from
*Figure 20 - Activity Diagram of WAKEUP* REACTION, there are the following wakeup reactions:

- Execution of the TTII protocol (see *7.9 Time Triggered Increased Inoperation*)
- Proceed to RUN state (full startup)
- Shutdown
  If none of the above cases is chosen, the ECU will be shut down again by default. The exact behavior depends on the selected shutdown target.

The callout of this state may be used to override the wakeup reaction and provide an ECU specific algorithm.

In case of an ECU Reset, the ECU State Manager will perform a full initialization.

#### 7.7.4.4 WAKEUP II

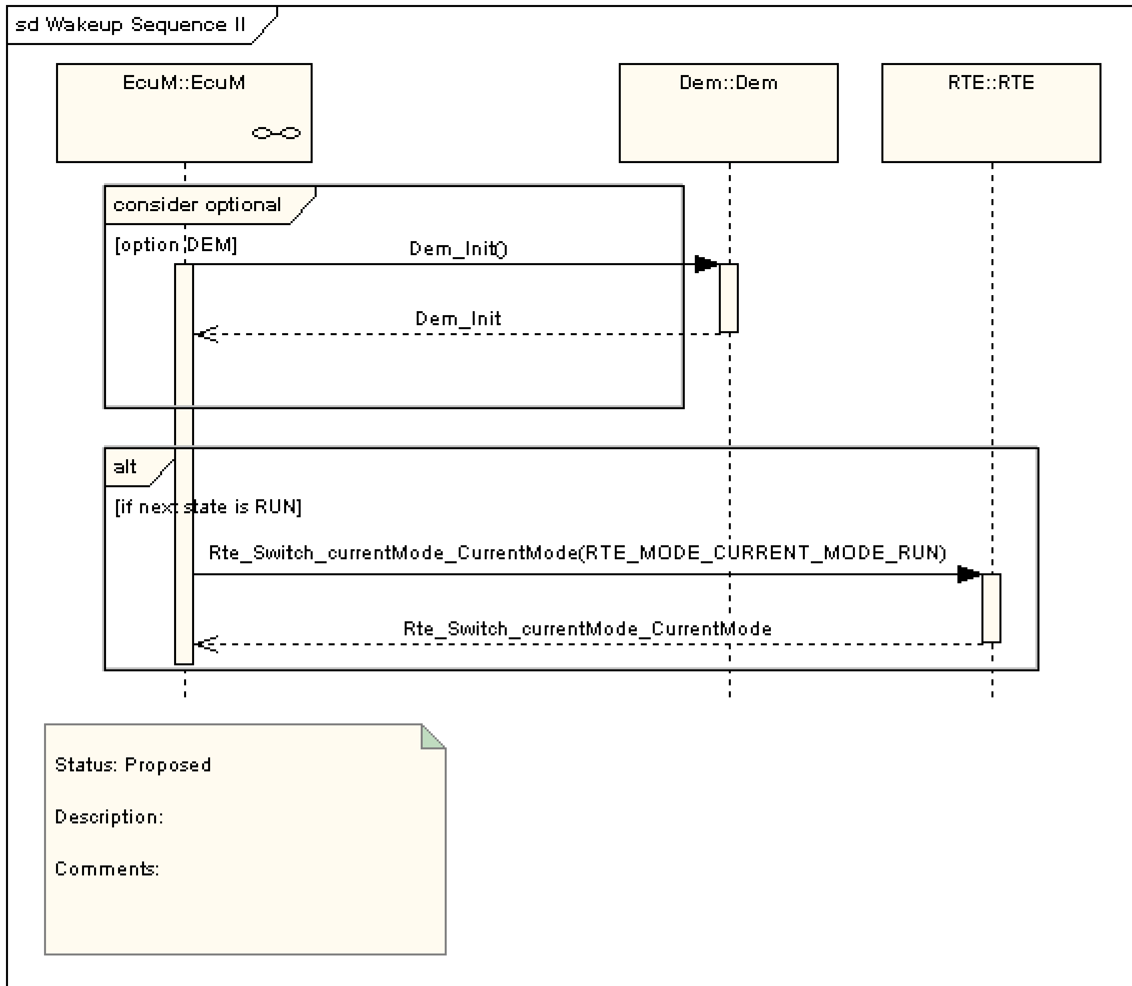


**Figure 21 - Wakeup Sequence II**

## 7.8 Wakeup Validation Protocol

### 7.8.1 Wakeup of Communication Channels

Communication channels have their own state machines including run and also sleep states.This is necessary since an ECU may have interfaces to several communication busses and busses can go to sleep independently from the ECU.
Consider the following example:

> An ECU may have two bus interfaces A and B. The ECU may be awake, bus A is in full communication state, but bus B is sleeping.

The state machines of the communication channels are completely provided by the Communication Manager, see [8] for details.
According to the specification, the Communication Manager autonomously can fulfill the following tasks:

- Drive a channel from full communication in no communication mode in collaboration with Network Management.
- Put the bus transceiver into standby mode by using the CAN Transceiver Driver or other drivers according to the bus interface type and configure the bus transceiver to generate wakeup events when bus traffic occurs.

The Communication Manager however will not drive the wakeup process since wakeup events will be directed to the ECU State Manager who in turn will notify the Communication Manager if and only if appropriate[25].

**EcuM2478**: If a wakeup occurs on a communication channel, the according bus transceiver driver shall notify the ECU State Manager by invoking the `EcuM_SetWakeupEvent` service, similar to the CAN Transceiver Driver [23]. Requirements for this notification are formulated in *5.3 Peripherals with Wakeup Capability*.

**EcuM2479**: The ECU State Manager shall execute the Wakeup Validation Protocol according to *7.8.3 Interaction of Wakeup Sources and the ECU State Manager* later in this chapter.

**EcuM2480**: If validation is successful, the ECU State Manager shall inform the Communication Manager about the wakeup event by invoking the Communication Manager's `ComM_EcuM_WakeUpIndication` service with the according channel as parameter. In turn, the Communication Manager will use this event to bring the channel into full communication mode.

### 7.8.2 Wakeup of the Entire ECU

Before the ECU State Manager can put the ECU into SLEEP state, the Communication Manager must have released all run requests[26]. This will only happen, if all communication state machines are in 'no communication' mode. But this, taking into account the previous paragraphs, implies that all communication interfaces (i.e. all bus transceivers) must have been put to standby state and the wakeup source must have been armed. Thus, when a wakeup occurs, all communication channels are in no communication state and there are no RUN requests.
The wakeup procedure is identical to the previous chapter.

### 7.8.3 Interaction of Wakeup Sources and the ECU State Manager

All wakeup sources must be treated in the same way. The procedure shall be as follows:

---

[25] See *`ECUM_WKSOURCE_COMIF_XREF_LIST`* in *10.3 Configurable Parameters*.
[26] This statement can be extended to any resource manager which may be added in future versions of the AUTOSAR Basic Software.

**EcuM2492**: Upon occurrence of a wakeup event, the responsible driver shall invoke an indication to notify the ECU State Manager about the wakeup[27].
**EcuM2494**: If wakeup validation is required for this event, then the validation protocol applies. Otherwise the event is valid immediately.

**EcuM2495**: If the valid event is a wakeup event from a communication interface then it is propagated to the Communication Manager.

**EcuM2496**: If the wakeup event is the first one after leaving the SLEEP state, it is labelled as the wakeup source and this information is made available to the application by the `EcuM_GetValidatedWakeupEvents` service.

### 7.8.4 WakeupValidation Timout Timer

It is the implementers choice whether he wants to provide a single wakeup validation timeout timer or one timer per wakeup source. The following requirements apply:

**EcuM2709**: The timer shall be started when the service `EcuM_SetWakeupEvent` is called.

**EcuM2710**: The timer shall be stopped and the validation is set to passed when the service `EcuM_ValidateWakeupEvent` is called.

**EcuM2711**: When the timer expires, validation is set to failed.

**EcuM2712**: Subsequent calls to `EcuM_SetWakeupEvent` for the same wakeup source shall not prolong the timeout.
If only one timer is used, the following approach is proposed:

**EcuM2714**: If `EcuM_SetWakeupEvent` is called for a wakeup source which did not fire yet during the same wakeup cycle then the timeout should be prolonged for the validation timeout of that wakeup source.
Wakeup timeouts are defined by configuration in chapter *10.3 Configurable Parameters*.

### 7.8.5 Requirements for Drivers with Wakeup Sources

**EcuM2571**: The driver shall invoke the `EcuM_SetWakeupEvent` service with a configurable parameter identifying the source of the wakeup once when the wakeup event is detected.

---

[27] This step can happen in several scenarios. The most likely are:

- After exiting the SLEEP state. In this scenario, the ECU State Manager would issue a re-initialization of the relevant drivers which in turn get a chance to scan their hardware e.g. for pending wakeup interrupts.
- If the wakeup source is actually in sleep mode, then the driver shall scan autonomously for wakeup events. The driver may do this interrupt driven or in polling mode, whichever is the preferred way for implementing it.

**EcuM2572**: Wakeups which occurred prior to driver initialization shall be detectable. This applies to initialization from SLEEP or from OFF state.

**EcuM2573**: The driver shall provide an API to configure the wakeup source for the SLEEP state, to enable or disable the wakeup source, and to put the related peripherals to sleep. This requirement only applies if hardware provides these capabilities.

**EcuM2574**: The callback invocation shall be enabled by calling the driver initialization service.

### 7.8.6 Requirements for Wakeup Validation

**EcuM2575**: If the wakeup source requires validation, this may be done by any but only by one appropriate module of the basic software. This may be a driver, an interface, a handler, or a manager.

Validation is done by calling the `EcuM_ValidateWakeupEvent` service.

### 7.8.7 Wakeup Sources and Reset Reason

The ECU State Manager API only provides one type (`EcuM_WakeupSourceType`) which can describe all reasons why the ECU starts or wakes up.

**EcuM2625**: The following wakeup sources shall not require validation under no circumstances:

- `ECUM_WKSOURCE_POWER`
- `ECUM_WKSOURCE_RESET`
- `ECUM_WKSOURCE_INTERNAL_RESET`
- `ECUM_WKSOURCE_INTERNAL_WDG`
- `ECUM_WKSOURCE_EXTERNAL_WDG`.

### 7.8.8 Wakeup Sources with Integrated Power Control

This section applies if the sleep state is realized by a system chip which controls the MCU's power supply. Typical examples are CAN transceivers with integrated power supplies. These transceivers switch off power upon application request and switch on power upon CAN activity.
As a consequence, the SLEEP state looks more like the OFF state. This distinction is rather philosophical and not of practical importance. The practical impact is that a passive wakeup on CAN will look like a power on reset to the ECU. Hence, the ECU will continue with the STARTUP sequence after a wakeup event. Nevertheless, wakeup validation is required. In order to make this work, the system designer has to consider the following topics:

- The CAN transceiver is initialized during one of the driver initialization blocks (init block II by default). This is configured or generated code, i.e. code which is under control of the system designer.

- The CAN transceiver driver API provides services to find out if it was the CAN transceiver, due to a passive wakeup, which started the ECU. It is the system designer's responsibility to check the CAN transceiver for wakeup reasons and give this information to the ECU State Manager by using the `EcuM_SetWakeupEvent` and
- 
- EcuM_ClearWakeupEvent services.
- If the system designer sets the CAN transceiver as the wakeup source, then the ECU State Manager will not continue with the RUN state when STARTUP II is finished. Instead it will continue with the WAKEUP VALIDATION state.

This behavior can be applied to all kinds of wakeup sources. The CAN transceiver only serves as an example here.

When waking up from a SLEEP state which is implemented by unpowering the MCU BSW modules must be brought back into a state, so that the entire sleep state is transparent to all SWCs. The system designer should use the callouts `EcuM_AL_DriverInitOne` and `EcuM_AL_DriverInitTwo` for this purpose.
When the MCU is unpowered, it is unevitable that the ECU State Manager carries out the STARTUP state. The ECU State Manager offers support that it detects this case but then branches into wakeup validation and from there (if validation is successful) into RUN state.

### 7.8.9 Activity Diagram

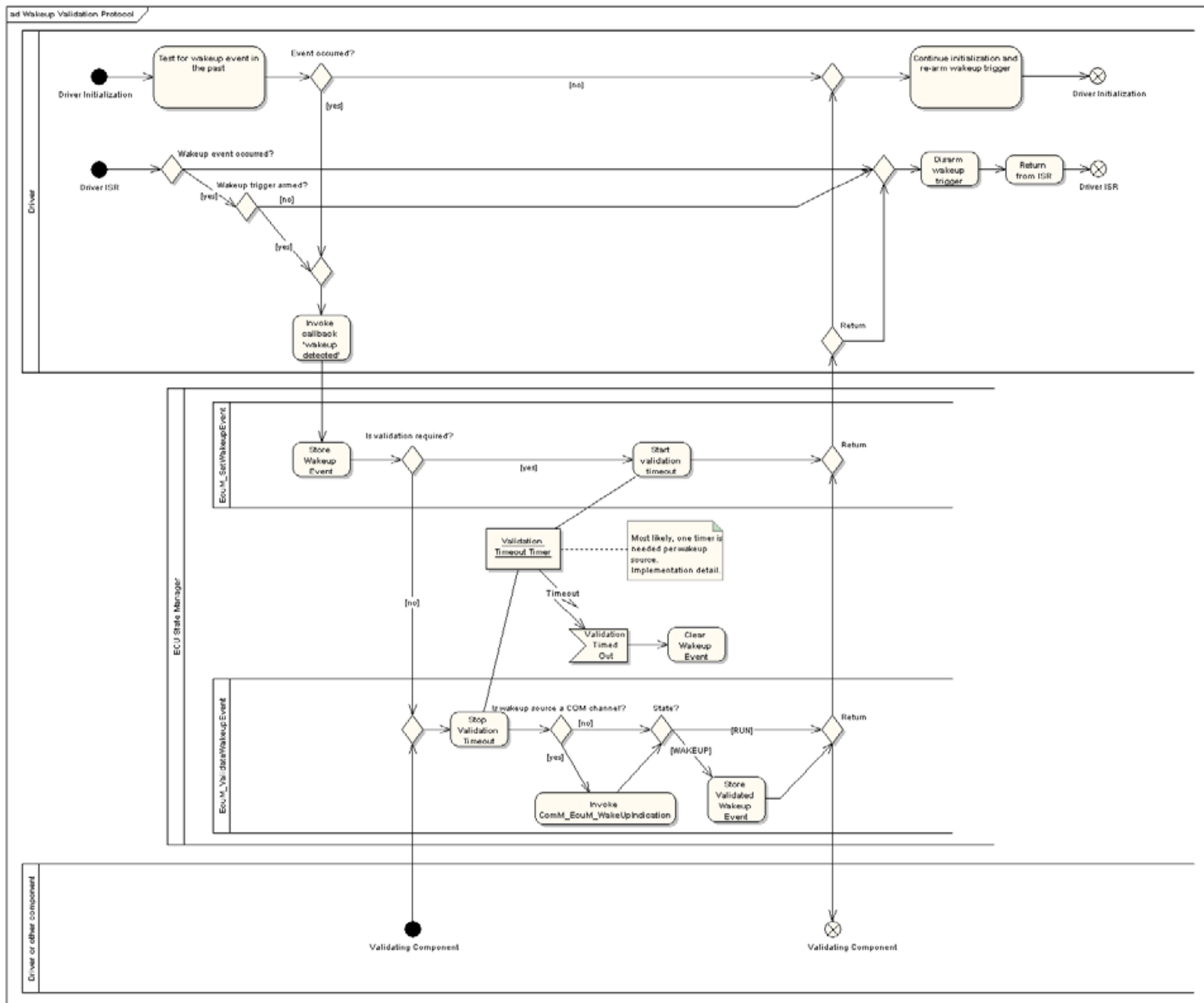

Figure 22- Wakeup Validation Protocol

### 7.8.10 Example: Application of Wakeup Validation Protocol for GPT

This chapter is for information only. It does not have normative contents.

Wakeup validation is a rather flexible protocol and can be used in various ways to achieve the desired behavior. As a general rule of thumb, wakeup validation should only be applied for peripheral wakeup sources, and there only, if hazardous events can occur (e.g. EMV spikes). The GPT can be considered as a safe source in this context. False alarms are very unlikely to occur, if not impossible.
Further, the driver designer has the possibility to do validation already inside the driver. This is reasonable if validation can be done quick and easy, e.g. during an ISR. The time base should be one OS tick. A time period can be considered short if it is less than one OS tick (a half, a tenth, etc.) or long if it is reasonabley more than one OS tick (2, 5, 10). The exact borders are chosen by the driver designer and depend on the physics of the wakeup source.
If validation can be done from within the same ISR as the handling of wakeup event then the wakeup validation protocol is not needed.
If validation needs to be split across several ISRs or across an ISR and OS tasks, then the wakeup validation protocol is needed.

The following sub-chapters present three examples:

- Example 1: CAN wakeup
  The validation of a CAN wakeup requires at least one correctly received CAN message. Therefore, the validation is split across two ISRs at least, and wakeup validation protocol is needed. Furthermore, it may take several hundred milliseconds until an event can be validated, so this is also a good reason to use the wakeup validation protocol
- Example 2: ICU wakeup with ISR
  Validation of an ICU wakeup event may be feasible simple within the ISR. Therefore the wakeup validation event is not needed.
- Example 3: ICU wakeup without ISR
  This example is similar to example 2, except that the MCU is configured not to invoke an ISR but simply to continue program execution after a wakeup event occurred.

All examples assume that the sleep state is implemented by issuing an HALT instruction to the CPU.


- AUTOSAR confidential -

### 7.8.10.1 Example 1: Wakeup from CAN

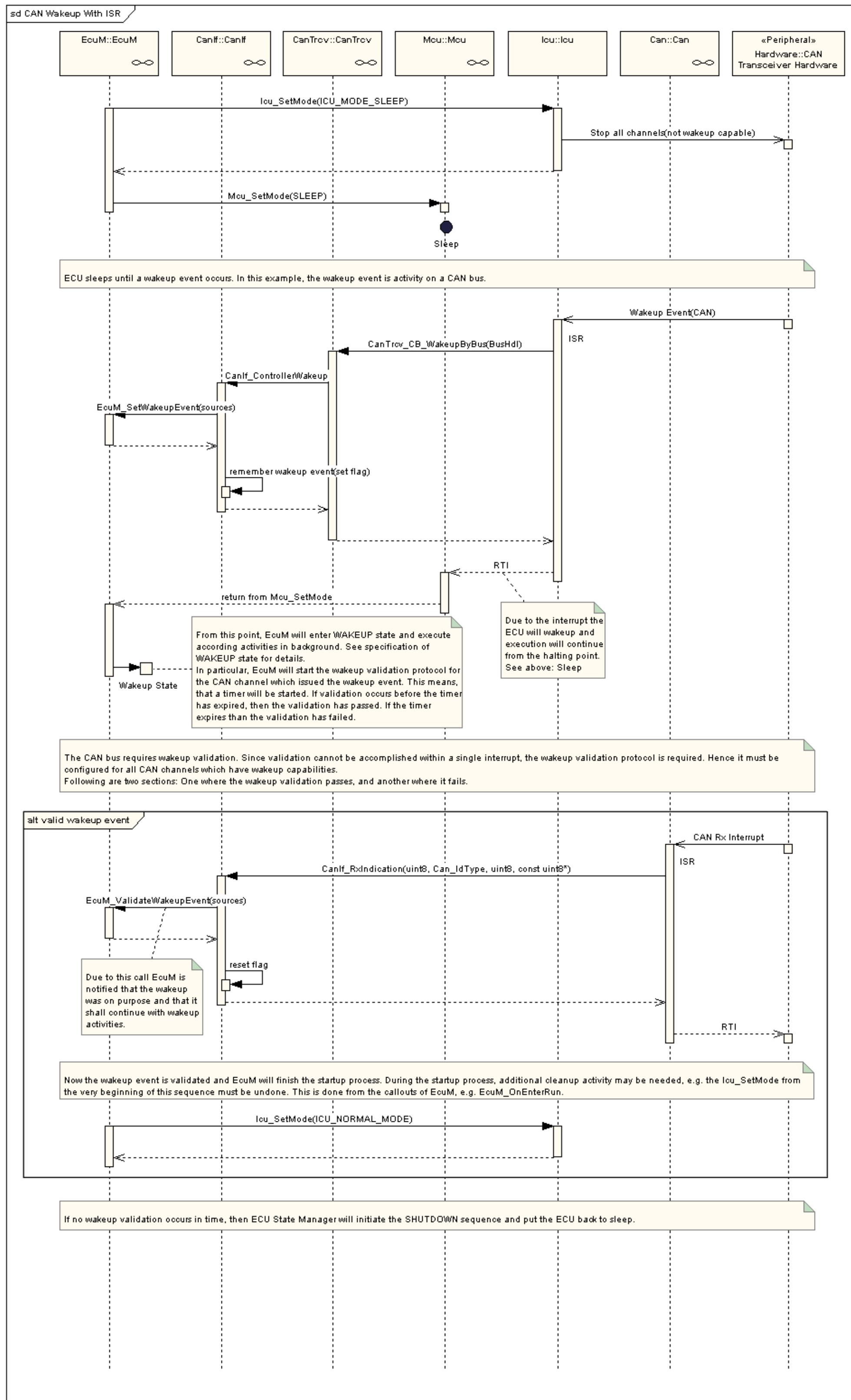


**Figure 23 - Example: Wakeup from CAN**



- AUTOSAR confidential -

#### 7.8.10.2 Example 2: Wakeup From ICU

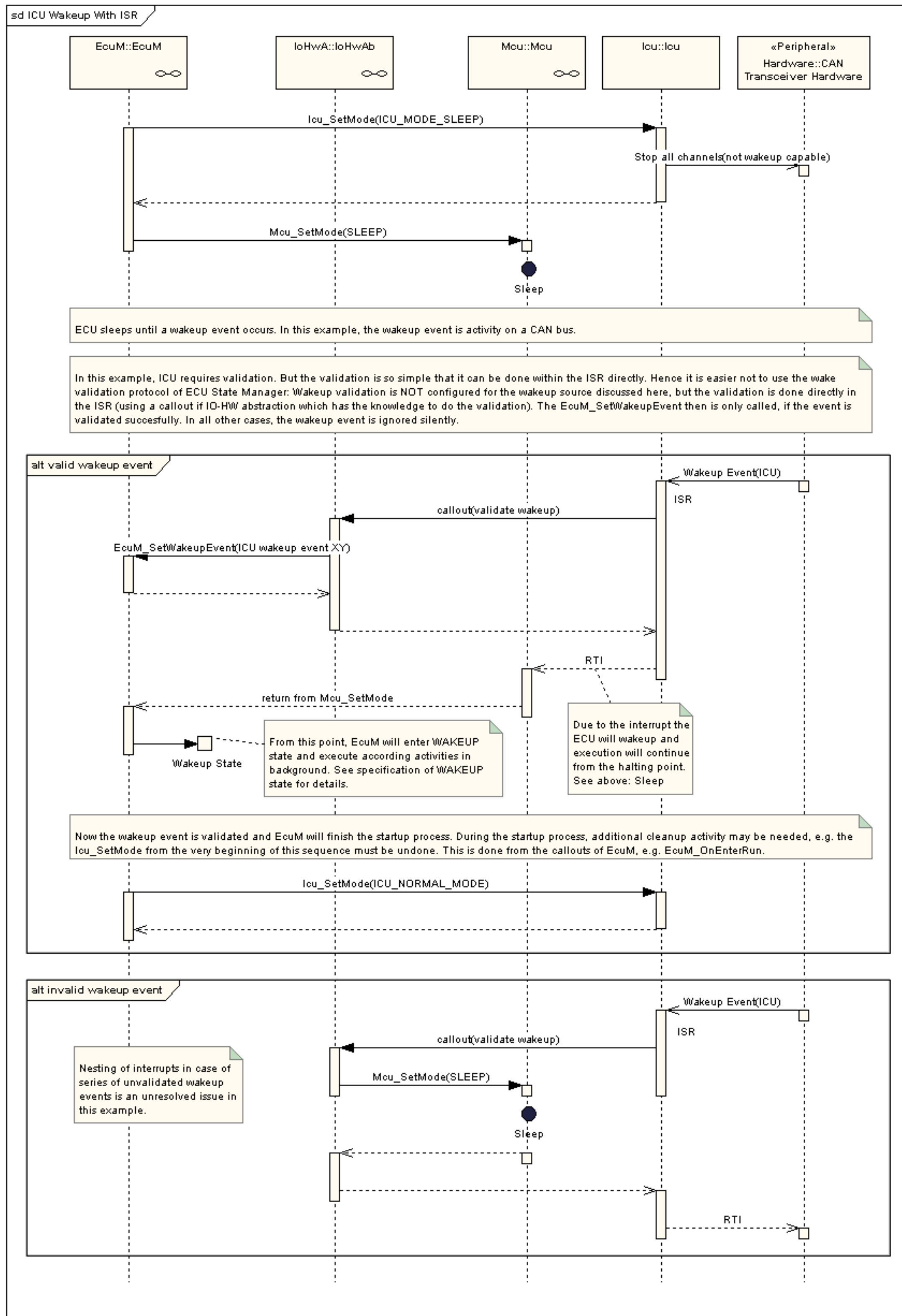


**Figure 24 - Wakeup from ICU with ISR**

#### 7.8.10.3 Example 3: Wakeup from ICU

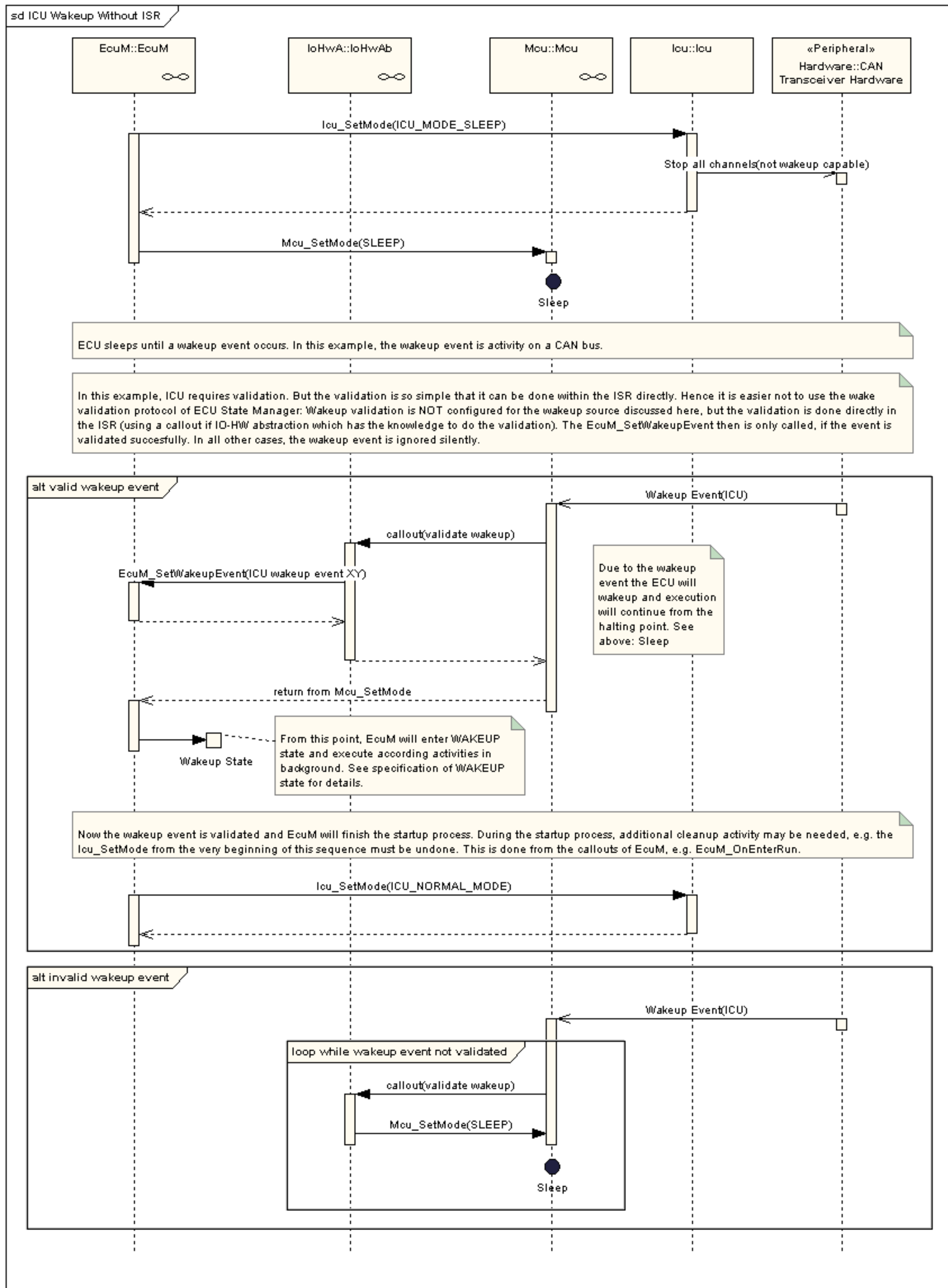


**Figure 25 - Wakeup from ICU without ISR**

## 7.9 Time Triggered Increased Inoperation

**EcuM2653**: TTII shall manage a list of sleep modes (shutdown targets). These sleep modes can be defined at configuration time. Typically the sleep modes are ordered to deepen the sleep phase of the ECU (decreased power consumption).

**EcuM2654**: An entry of the sleep mode list shall contain the following properties:

- A description of the ECU sleep mode (`EcuM_SleepModeConfigType`).
- A reference to the successor sleep mode
- A divisor counter which tells how often the ECU must be woken up before the successor sleep mode is selected.

These properties shall be defined at configuration time.

The TTII protocol is executed during the WAKEUP REACTION sub-state. Refer to chapter 7.7.4.3 WAKEUP REACTION and
*Figure 20 - Activity Diagram of WAKEUP* REACTION.

**EcuM2223**: The entire TTII feature can be completely disabled by setting the *ECUM_TTII_ENABLED* configuration parameter to false. All further described activities are only applicable if TTII is enabled.

**EcuM2222**: A wakeup source must be selected by configuration (*ECUM_TTII_WKSOURCE* configurable parameter) for use by the TTII protocol. Typically, the wakeup source will be a timer, which serves as a timebase for TTII. Whenever the ECU is woken up by this configured wakeup source, then the TTII protocol shall be executed.

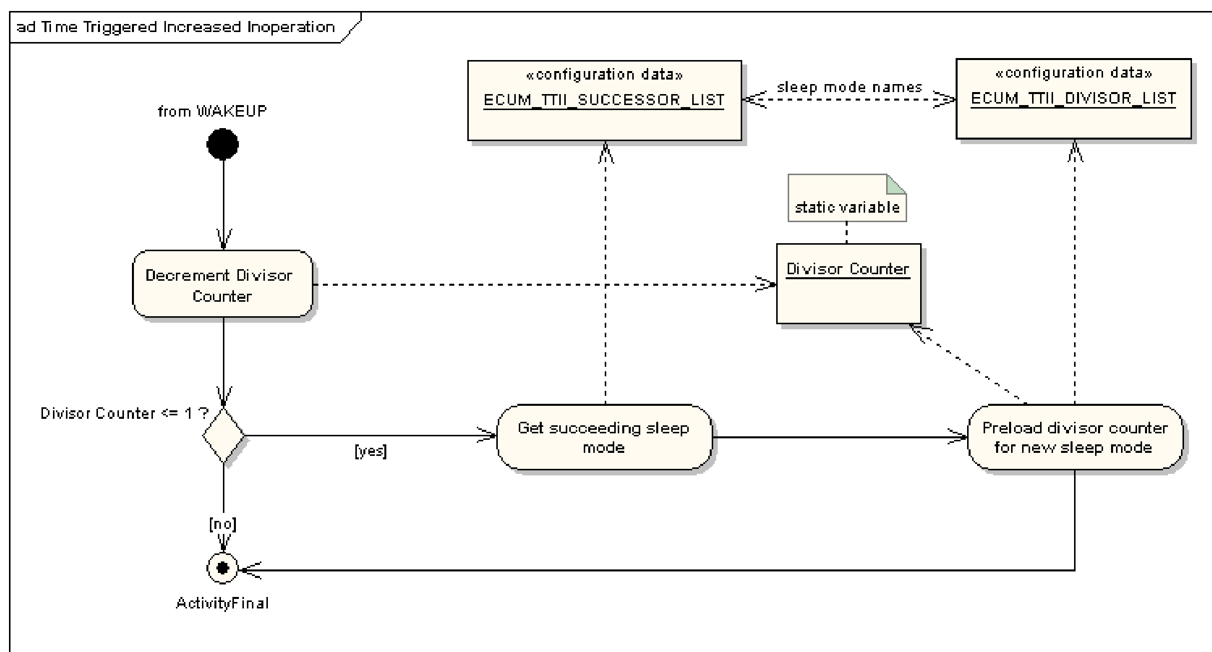


**Figure 26 - Activity Diagram of TTII**

## 7.10 AUTOSAR Ports

### 7.10.1 Scope of this Chapter

The following definitions are interpreted to be in ARPackage AUTOSAR/Services/EcuM.

### 7.10.2 Overview

The overall architecture of the ECU State Manager service is depicted in the following picture.

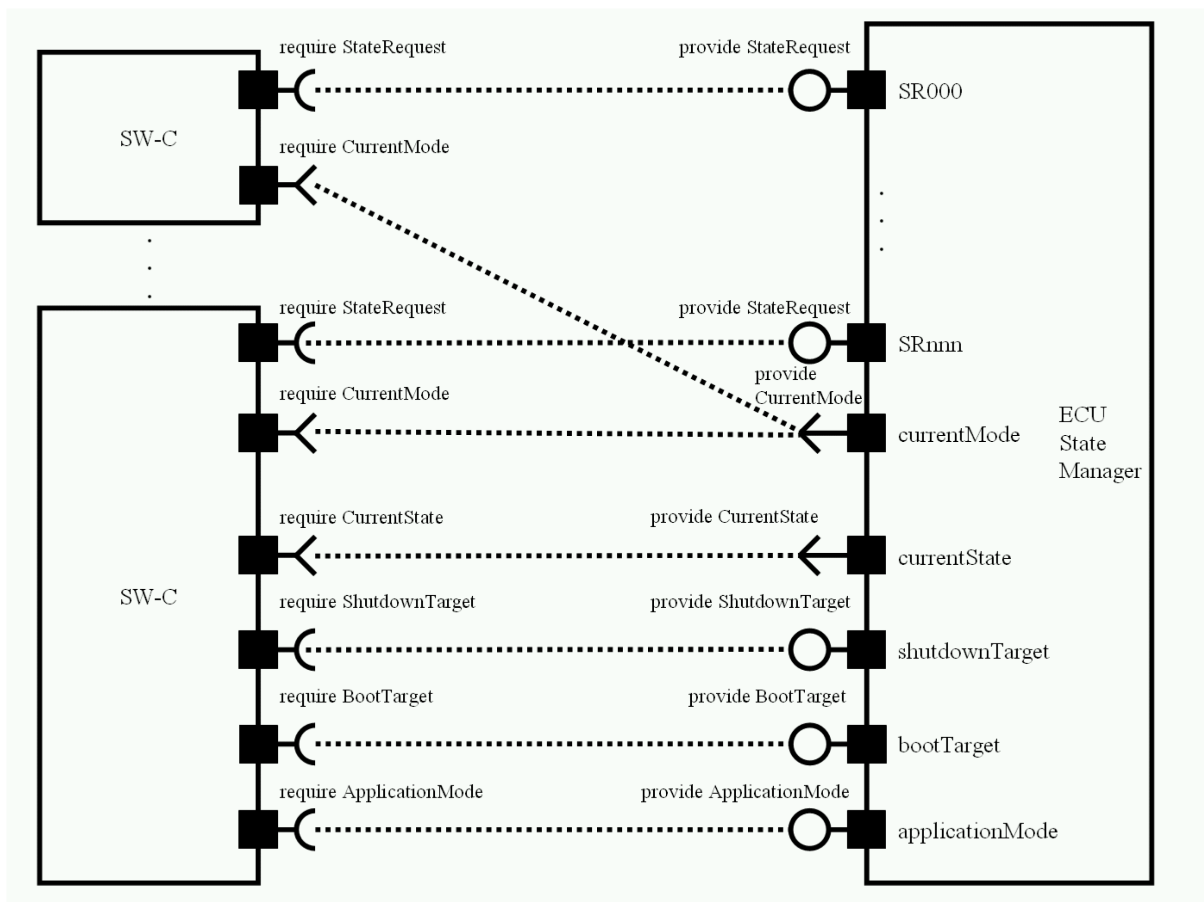


**Figure 27 - ARPackage EcuM**

### 7.10.3 Use Cases

**EcuM2762**: The ECU State Manager shall provide AUTOSAR ports for the following functionalities:

- requesting RUN
- releasing RUN

- requesting POST RUN
- releasing POST RUN

**EcuM2763**: The ECU State Manager shall provide also AUTOSAR ports for the following functionality:

- selecting and getting shutdown target
- selecting and getting application modes
- selecting and getting boot targets

The interfaces used in this chapter will be described in the following.

### 7.10.4 Specification of the Port Interfaces

This chapter specifies the Port Interfaces that are needed in order to operate the ECU State Manager functionality over the VFB. The ports implementing the Port Interfaces described in this chapter will be defined in chapter 7.10.5.

#### 7.10.4.1 Port Interface for Interface State Request

##### 7.10.4.1.1 General Approach

A SW-C which needs to keep the ECU alive or needs to execute any operations before the ECU is shut down shall require the client-server interface `StateRequest`. The SW-C can obtain the exact state of the ECU State Manager by calling the function `GetState`.

##### 7.10.4.1.2 Data Types

The data type StateType represents the current state of the ECU State Manager.

```
PrimitiveTypeWithSemantics StateType {
      IntegerType {
            LOWER-LIMIT=0x10, UPPER-LIMIT=0x80
      };
      0x10 -> ECUM_STATE_STARTUP
      0x11 -> ECUM_STATE_STARTUP_ONE
      0x12 -> ECUM_STATE_STARTUP_TWO
      0x20 -> ECUM_STATE_WAKEUP
      0x21 -> ECUM_STATE_WAKEUP_ONE
      0x22 -> ECUM_STATE_WAKEUP_VALIDATION
      0x23 -> ECUM_STATE_WAKEUP_REACTION
      0x26 -> ECUM_STATE_WAKEUP_TWO
      0x25 -> ECUM_STATE_WAKEUP_WAKESLEEP
      0x26 -> ECUM_STATE_WAKEUP_TTII
      0x30 -> ECUM_STATE_RUN
      0x32 -> ECUM_STATE_APP_RUN
      0x33 -> ECUM_STATE_APP_POST_RUN
      0x40 -> ECUM_STATE_SHUTDOWN
      0x44 -> ECUM_STATE_PREP_SHUTDOWN
```

```
        0x49 -> ECUM_STATE_GO_SLEEP
        0x4d -> ECUM_STATE_GO_OFF_ONE
        0x4e -> ECUM_STATE_GO_OFF_TWO
        0x90 -> ECUM_STATE_RESET
        0x50 -> ECUM_STATE_SLEEP
        0x80 -> ECUM_STATE_OFF
};
```

#### 7.10.4.1.3 Port Interface

```
ClientServerInterface StateRequest
{
        PossibleErrors {
        E_NOT_OK = 1 /* The request was not accepted by EcuM, a detailed
        error condition was sent to DET */
        };

        // The SW-C can request or release an ECU RUN or POSTRUN state when
        // requiring this interface
        RequestRUN(ERR{E_NOT_OK});
        ReleaseRUN(ERR{E_NOT_OK});
        RequestPOSTRUN(ERR{E_NOT_OK});
        ReleasePOSTRUN(ERR{E_NOT_OK});

        // The SW-C is informed of the current ECU state when requiring
        // this interface
        GetState(IN StateType state);
        };
```

The ECU State Manager provides additional calls which would typically be made by one management instance on the ECU as they have a global impact. The function "`EcuM_KillAllRUNRequests()`" unconditionally undoes all requests to RUN and POST RUN. Because of this, calling `EcuM_RequestRUN` does not necessarily guarantee that the ECU will stay awake until calling `EcuM_ReleaseRUN` (e.g. a `KillAllRUNRequests`-call can override the wish of individual users for the ECU to stay awake). The function "`EcuM_KillAllRUNRequests()`" is not accessible over the RTE and thus can not be used by SW-Cs.

### 7.10.4.2 Port Interface for Interface Current Mode

#### 7.10.4.2.1 General Approach

**EcuM2749**: The mode port of the ECU State Manager shall declare the following modes:

0: Startup Shutdown
1: Run
2: Post Run
3: Sleep
4: WakeSleep

This definition is a simplified view of ECU States that applications do need to know. It does not restrict or limit in any way how application states could be defined. Applications states are completely handled by the application itself.

**EcuM2750**: State changes shall be notified to SW-Cs through the RTE mode ports when the state change occurs. This shall happen by invoking

```
Rte_StatusType
Rte_Switch_currentMode_CurrentMode(Rte_ModeType_CurrentMode <mode>)
```

where `mode` is the new mode to be notified. The value range is specified by the previous requirement. The return value shall be ignored.

A SW-C which wants to be notified of mode changes should require the mode switch interface `CurrentMode`.

The following figure shows how the defined modes are mapped to the states of the ECU State Manager and when the notifications shall occur.

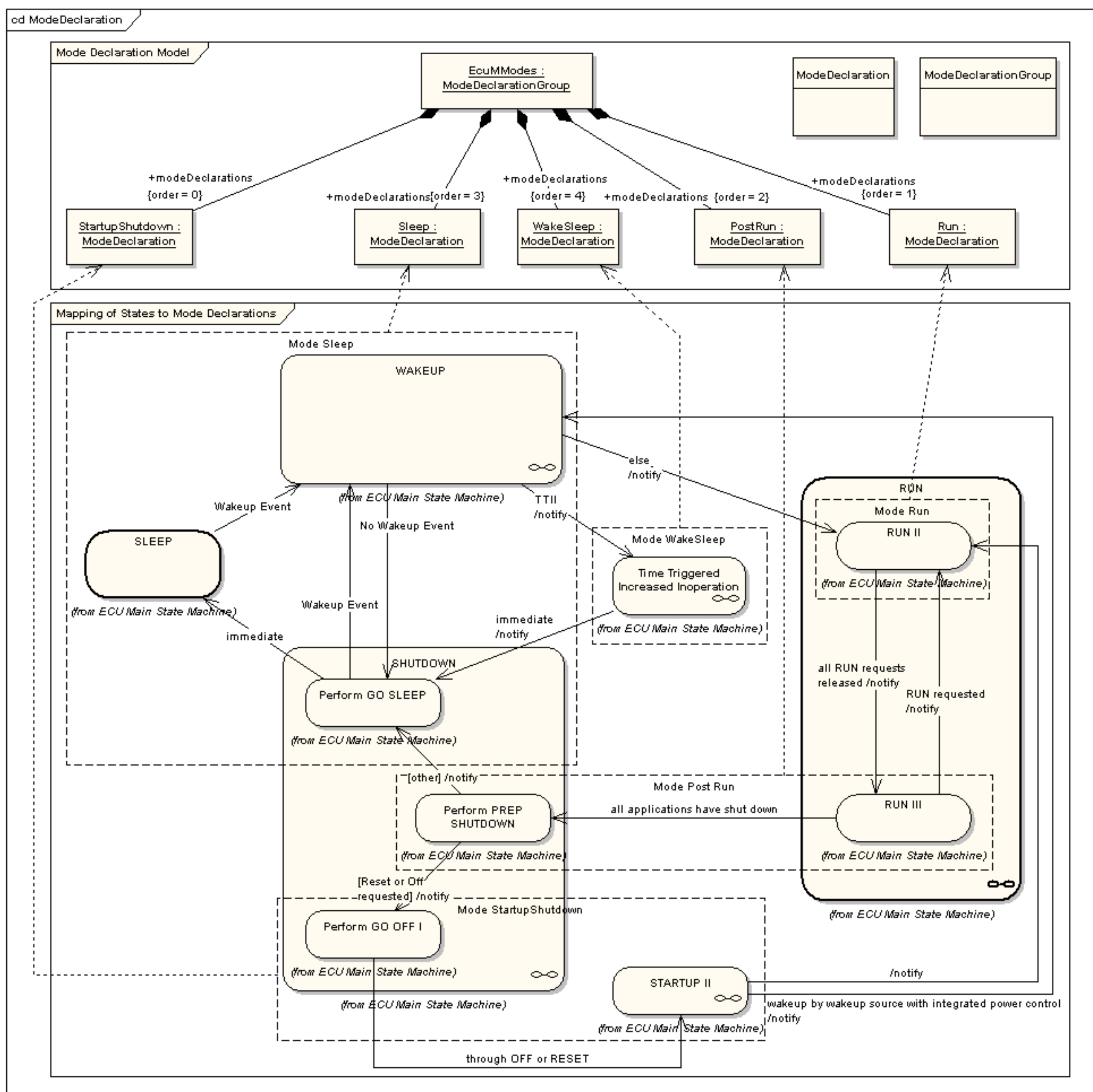


**Figure 28 - Mapping of Declared Modes to ECU State Manager States**

**EcuM2752**: The ECU State Manager shall notify WakeSleep mode and Sleep mode when transiting from WAKEUP to SHUTDOWN, but only if the selected shutdown target is SLEEP.
This allows the system designer to trigger runnables for wake-sleep operations or TTII.

#### 7.10.4.2.2 Data Types

The mode declaration group `Mode` represents the modes of the ECU State Manager that will be notified to the SW-Cs.

```
ModeDeclarationGroup Mode {
      { STARTUP_SHUTDOWN,
      RUN,
      POST_RUN,
      SLEEP,
      WAKE_SLEEP
      }
      initialMode = STARTUP_SHUTDOWN
};
```

#### 7.10.4.2.3 Port Interface

```
SenderReceiverInterface CurrentMode {
      Mode currentMode;
};
```

### 7.10.4.3 Ports and Port Interface for Interface Shutdown Target

#### 7.10.4.3.1 General Approach

A SW-C which wants to select a shutdown target should require the client-server interface `ShutdownTarget`.

#### 7.10.4.3.2 Data Types

The data type ShutdownTargetType represents the shutdown targets the ECU State Manager can be configured with.

```
PrimitiveTypeWithSemantics ShutdownTargetType {
      IntegerType {LOWER-LIMIT=0x50, UPPER-LIMIT=0x90};
      0x90 -> ECUM_STATE_RESET
      0x50 -> ECUM_STATE_SLEEP
      0x80 -> ECUM_STATE_OFF

      // The shutdown of an ECU may end up in different states,
      // depending on what application requires or desires for
      // the next shutdown. By selecting a shutdown target, the
```

```
        // application can communicate its wishes to the ECU State Manager.
        // SLEEP, OFF, and RESET are shutdown targets

        // This type is a subgroup of ECU states. We should define another
        // type since the portability of this type to SW-C Template is quite
        // inefficient
};
```

#### 7.10.4.3.3 Port Interface

```
ClientServerInterface ShutdownTarget
{
        // The SW-C can select a shutdown target when requiring
        // this interface
        PossibleErrors {
        E_NOT_OK = 1 /* The new shutdown target was not set */
        };

        // The SW-C selects a shutdown target
        SelectShutdownTarget(IN ShutdownTargetType target, IN UInt8 mode,
        ERR{E_NOT_OK});

        // The SW-C gets informed of the current shutdown target
        GetLastShutdownTarget(IN ShutdownTargetType target, IN UInt8 mode);
};
```

The parameter mode determines the concrete sleep mode. This parameter shall only be used if the target parameter equals to ECUM_STATE_SLEEP, otherwise it will be ignored.

### 7.10.4.4 Port Interface for Interface Boot Target

#### 7.10.4.4.1 General Approach

A SW-C which wants to select a boot target shall require the client-server interface `BootTarget`.

#### 7.10.4.4.2 Data Types

The data type StateType represents the boot targets the ECU State Manager can be configured with.

```
PrimitiveTypeWithSemantics BootTargetType {
        IntegerType {LOWER-LIMIT=0, UPPER-LIMIT=1};
        0 -> ECUM_BOOT_TARGET_APP
        1 -> ECUM_BOOT_TARGET_BOOTLOADER

        // ECUM_BOOT_TARGET_APP: The ECU will boot into the application
        // ECUM_BOOT_TARGET_BOOTLOADER: The ECU will boot into the bootloader
};
```

#### 7.10.4.4.3 Port Interface

```
ClientServerInterface BootTarget
{
      PossibleErrors {
      E_NOT_OK = 1 /* The new boot target was not accepted by EcuM */
      };

      // The SW-C selects a boot target
      SelectBootTarget (IN BootTargetType target, ERR{E_NOT_OK});

      // The SW-C gets informed of the current boot target
      GetBootTarget(OUT BootTargetType target);
};
```

### 7.10.4.5 Port Interface for Interface Application Mode

#### 7.10.4.5.1 General Approach

A SW-C which wants to select an application mode shall require the client-server interface `ApplicationMode`.

#### 7.10.4.5.2 Data Types

The data type `AppModeType` represents the application mode taken for next OS start. The type is defined by the operating system. See [19] for detailed information.

#### 7.10.4.5.3 Port Interface

```
ClientServerInterface ApplicationMode
{
      PossibleErrors {
      E_NOT_OK = 1 /* The new application mode was not accepted by EcuM */
      };

      // The SW-C selects an application mode
      SelectApplicationMode (IN AppModeType appMode, ERR{E_NOT_OK});

      // The SW-C gets informed of the current application mode
      GetApplicationMode (OUT AppModeType appMode);
};
```

## 7.10.5 Summary of ports

### 7.10.5.1 Definitions of interfaces

```
PrimitiveTypeWithSemantics StateType {
      IntegerType {
```

```
        LOWER-LIMIT=0x10, UPPER-LIMIT=0x80
        };
        0x10 -> ECUM_STATE_STARTUP
        0x11 -> ECUM_STATE_STARTUP_ONE
        0x12 -> ECUM_STATE_STARTUP_TWO
        0x20 -> ECUM_STATE_WAKEUP
        0x21 -> ECUM_STATE_WAKEUP_ONE
        0x22 -> ECUM_STATE_WAKEUP_VALIDATION
        0x23 -> ECUM_STATE_WAKEUP_REACTION
        0x26 -> ECUM_STATE_WAKEUP_TWO
        0x25 -> ECUM_STATE_WAKEUP_WAKESLEEP
        0x26 -> ECUM_STATE_WAKEUP_TTII
        0x30 -> ECUM_STATE_RUN
        0x32 -> ECUM_STATE_APP_RUN
        0x33 -> ECUM_STATE_APP_POST_RUN
        0x40 -> ECUM_STATE_SHUTDOWN
        0x44 -> ECUM_STATE_PREP_SHUTDOWN
        0x49 -> ECUM_STATE_GO_SLEEP
        0x4d -> ECUM_STATE_GO_OFF_ONE
        0x4e -> ECUM_STATE_GO_OFF_TWO
        0x90 -> ECUM_STATE_RESET
        0x50 -> ECUM_STATE_SLEEP
        0x80 -> ECUM_STATE_OFF
};


ClientServerInterface StateRequest
{
        PossibleErrors {
        E_NOT_OK = 1 /* The request was not accepted by EcuM, a detailed
        error condition was sent to DET */
        };

        // The SW-C can request or release an ECU RUN or POSTRUN state when
        // requiring this interface
        RequestRUN(ERR{E_NOT_OK});
        ReleaseRUN(ERR{E_NOT_OK});
        RequestPOSTRUN(ERR{E_NOT_OK});
        ReleasePOSTRUN(ERR{E_NOT_OK});

        // The SW-C is informed of the current ECU state when requiring
        // this interface
        GetState(IN StateType state);
};

ModeDeclarationGroup Mode {
        { STARTUP_SHUTDOWN,
        RUN,
        POST_RUN,
        SLEEP,
        WAKE_SLEEP
        }
        initialMode = STARTUP_SHUTDOWN
};
SenderReceiverInterface CurrentMode {
        Mode currentMode;
};

PrimitiveTypeWithSemantics ShutdownTargetType {
        IntegerType {LOWER-LIMIT=0x50, UPPER-LIMIT=0x90};
        0x90 -> ECUM_STATE_RESET
```

```
       0x50 -> ECUM_STATE_SLEEP
       0x80 -> ECUM_STATE_OFF
};

ClientServerInterface ShutdownTarget
{
       // The SW-C can select a shutdown target when requiring
       // this interface
       PossibleErrors {
       E_NOT_OK = 1 /* The new shutdown target was not set */
       };

       // The SW-C selects a shutdown target
       SelectShutdownTarget(IN ShutdownTargetType target, IN UInt8 mode,
       ERR{E_NOT_OK});

       // The SW-C gets informed of the current shutdown target
       GetLastShutdownTarget(IN ShutdownTargetType target, IN UInt8 mode);
};

PrimitiveTypeWithSemantics BootTargetType {
       IntegerType {LOWER-LIMIT=0, UPPER-LIMIT=1};
       0 -> ECUM_BOOT_TARGET_APP
       1 -> ECUM_BOOT_TARGET_BOOTLOADER

       // Bootloader and application are two separated programs which in
       // many cases even can be flashed separately. The only way to get
       // from one image to another is through reset. The boot menu will
       // branch into the one or other image depending on the selected boot
       // target
};

ClientServerInterface BootTarget
{
       PossibleErrors {
       E_NOT_OK = 1 /* The new boot target was not accepted by EcuM */
       };

       // The SW-C selects a boot target
       SelectBootTarget (IN BootTargetType target, ERR{E_NOT_OK});

       // The SW-C gets informed of the current boot target
       GetBootTarget(OUT BootTargetType target);
};

PrimitiveTypeWithSemantics AppModeType {
       // This type should be defined in the SWS OS but it is actually not
       // (Version of SWS OS 1.1.6)
       // TBD by SWS OS
};

ClientServerInterface ApplicationMode
{
       PossibleErrors {
       E_NOT_OK = 1 /* The new boot target was not accepted by EcuM */
       };

       // The SW-C selects an application mode
       SelectApplicationMode (IN AppModeType appMode, ERR{E_NOT_OK});

       // The SW-C gets informed of the current application mode
```

```
      GetApplicationMode (OUT AppModeType appMode);
};
```

#### 7.10.5.2 Definition of the Service ECU State Manager

This section provides guidance on the definition of the ECU State Manager Service. Note that these definitions can only be completed during ECU configuration (because it depends on certain configuration parameters of the ECU State Manager which determine the number of ports provided by the ECU State Manager service). Also note that the implementation of a SW-C does *not* depend on these definitions.
There are ports on both sides of the RTE: This description of the ECU State Manager service defines the ports below the RTE. Each SW-Component, which uses the Service, must contain "service ports" in its own SW-C description which will be connected to the ports of the ECU State Manager, so that the RTE can be generated.

```
/* This is the definition of the ECU State Manager as a service. This is
the "outside-view" of the ECU State Manager, which must be visible to the
SW-C's / ECU-integrator */
Service EcuStateManager {
      // For each user the ECU State Manager provides a port
      // to request/release RUN and POSTRUN states.
      // there are NU users;
      ProvidePort StateRequest SR000;
      …
      ProvidePort StateRequest SR<NU-1>;

      ProvidePort CurrentMode currentMode;

      ProvidePort ShutdownTarget shutdownTarget;

      ProvidePort BootTarget bootTarget;

      ProvidePort ApplicationMode applicationMode;
};
```

### 7.10.6 Runnables and Entry points

#### 7.10.6.1 Internal behavior

This is the inside description of the ComManager. This detailed description is only needed for the configuration of the local RTE.

```
InternalBehavior EcuStateManager {

      // Runnable entities of the EcuStateManager
      RunnableEntity RequestRUN
            symbol "EcuM_RequestRUN"
            canbeInvokedConcurrently = TRUE
      RunnableEntity ReleaseRUN
            symbol "EcuM_ReleaseRUN"
            canbeInvokedConcurrently = TRUE
      RunnableEntity RequestPOSTRUN
```

```
            symbol "EcuM_RequestPOST_RUN"
            canbeInvokedConcurrently = TRUE
      RunnableEntity ReleasePOSTRUN
            symbol "EcuM_ReleasePOST_RUN"
            canbeInvokedConcurrently = TRUE
      RunnableEntity SelectShutdownTarget
            symbol "EcuM_SelectShutdownTarget"
            canbeInvokedConcurrently = TRUE
      RunnableEntity GetLastShutdownTarget
            symbol "EcuM_GetLastShutdownTarget"
            canbeInvokedConcurrently = TRUE
      RunnableEntity GetState
            symbol "EcuM_GetState"
            canbeInvokedConcurrently = TRUE
      RunnableEntity SelectApplicationMode
            symbol "EcuM_SelectApplicationMode"
            canbeInvokedConcurrently = TRUE
      RunnableEntity GetApplicationMode
            symbol "EcuM_GetApplicationMode"
            canbeInvokedConcurrently = TRUE
      RunnableEntity SelectBootTarget
            symbol "EcuM_SelectBootTarget"
            canbeInvokedConcurrently = TRUE
      RunnableEntity GetBootTarget
            symbol "EcuM_GetBootTarget"
            canbeInvokedConcurrently = TRUE

      // Port present for each user. There are NU users
      SR000.RequestRUN -> RequestRUN
      SR000.ReleaseRUN -> ReleaseRUN
      SR000.RequestPOSTRUN -> RequestPOSTRUN
      SR000.ReleasePOSTRUN -> RequestPOSTRUN
      SR000.GetState -> GetState
      PortArgument {port=SR000, value.type=UInt8,
      value.value=EcuM_User[0].User}
      (...)
      SRnnn.RequestRUN -> RequestRUN
      SRnnn.ReleaseRUN -> ReleaseRUN
      SRnnn.RequestPOSTRUN -> RequestPOSTRUN
      SRnnn.ReleasePOSTRUN -> RequestPOSTRUN
      SRnnn.GetState -> GetState
      PortArgument {port=SRnnn, value.type=???,
      value.value=EcuM_User[nnn].User}

      shutDownTarget.SelectShutdownTarget -> SelectShutdownTarget
      shutDownTarget.GetLastShutdownTarget -> GetLastShutdownTarget
      bootTarget.SelectBootTarget -> SelectBootTarget
      bootTarget.GetBootTarget -> GetBootTarget
      applicationMode.SelectApplicationMode -> SelectApplicationMode
      applicationMode.GetApplicationMode -> GetApplicationMode
      };
```

#### 7.10.6.2 Entry points of Runnables

The following are the API's of the functions that need to be provided by an implementation of the ComManager. These functions are called by the RTE on behalf of the SW-Components.

```
typedef uint16 EcuM_StateType;

typedef uint8 EcuM_BootTargetType;

typedef ??? AppModeType;      // TBD by SWS OS

Std_ReturnType EcuM_RequestRUN(uint8 handle);

Std_ReturnType EcuM_ReleaseRUN(uint8 handle);

Std_ReturnType EcuM_RequestPOSTRUN(uint8 handle);

Std_ReturnType EcuM_ReleasePOSTRUN(uint8 handle);

Std_ReturnType EcuM_SelectShutdownTarget(uint8 target, uint8 mode);

Std_ReturnType EcuM_GetLastShutdownTarget(uint8 *target, uint8 *mode);

/* PROBLEM: see BUG# 12783 */
Std_ReturnType EcuM_GetState(EcuM_StateType *state);


Std_ReturnType EcuM_SelectApplicationMode(AppModeType appMode);

Std_ReturnType EcuMGetApplicationMode(AppModeType *appMode);


Std_ReturnType EcuM_SelectBootTarget(EcuM_BootTargetType target);

Std_ReturnType EcuM_GetBootTarget(EcuM_BootTargetType *target);
```

## 7.11 Advanced Topics

### 7.11.1 Application Modes

Application Modes is a feature of the OS which allows to define different configurations, e.g. sets of tasks which will be started initially. The application mode is an in parameter of the `StartOS` service (ref. [8]). Since the ECU State Manager is responsible for starting the OS, it has also responsibility for managing the application mode.

**EcuM2700**: An application mode change shall be accomplished by selecting the new application mode with the `EcuM_SelectApplicationMode` service (typically from RUN state) and a subsequent shutdown to the RESET shutdown target.

**EcuM2243**: The default application mode is set in the STARTUP I state in case of unintended restarts[28], see chapter *7.3.4.1 STARTUP I*. After this point, the application mode can be modified by the application itself.

[28] e.g. like watchdog reset

### 7.11.2 Relation to Bootloader

The Bootloader is not part of AUTOSAR. Still, the application needs an interface to activate the bootloader. For this purpose, two functions are provided: EcuM_SelectBootTarget and EcuM_GetBootTarget.

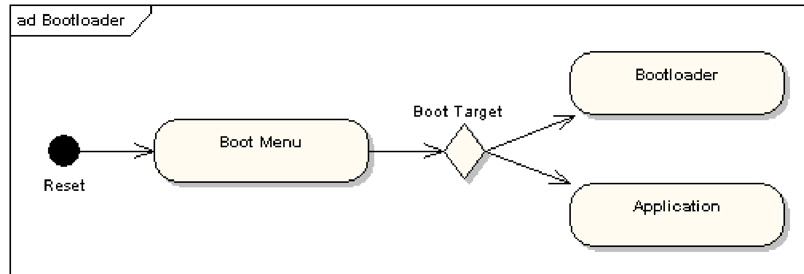


**Figure 29 - Selection of Boot Targets**

Bootloader and application are two separated programs which in many cases even can be flashed separately. The only way to get from one image to another is through reset. The boot menu will branch into the one or other image depending on the selected boot target.

### 7.11.3 Relation to Complex Drivers

**EcuM2321**: If the complex driver handles a wakeup source, it must obey all rules of this specification which are related to handling wakeup events.

**EcuM2322**: A complex driver may issue RUN requests.

### 7.11.4 Handling Errors during Startup and Shutdown

The ECU State Manager will ignore all types of errors that occur during initialization, e.g. as return values of init functions. Initialization is a configuration issue and henceforth cannot be standardized.
If errors occur during the initialization of a BSW module and this error is worthwhile being reported, then it is in the responsibility of that BSW module to report this error directly to DEM or DET and not the responsibility of the ECU State Manager.
If special error reactions are necessary, then also this is in the responsibility of the BSW module.

### 7.11.5 Configuration Alternative for Providing Wake-Sleep Operation

In rare use cases, an ECU has to wake up cyclically (e.g. each second), execute a very simple task (like blinking an LED) and go back to sleep. For most operations, the normal WAKEUP/SHUTDOWN behavior as defined by the ECU State Manager will be sufficient. Sometimes, however, the software has to be written very specific to


- AUTOSAR confidential -

maximize energy savings. Because the use case is so rare, there is no built-in feature in the ECU State Manager. However, the system designer can achieve this by using the ECU State Manager in the following way:

- Define a wakeup source to be used for the wake-sleep-operation (typically a timer)
- Check the wakeup source in the `EcuM_AL_DriverInitOne` callout and, if it was the reason, execute the necessary task
- Finally, put the ECU back to sleep or perform a startup

The code needed for this behavior is custom code which is located below the RTE.

### 7.11.6 Selecting Scheduling Schemes for Startup and Shutdown

On some ECU designs, it will be necessary to change the scheduling tables for startup and shutdown of the ECU, e.g. to improve speed for reading or writing non-volatile data. Unless other mechanisms are provided by basic software, the notification to switch the schedule table shall preferably be done from the `EcuM_OnEnterRun` and `EcuM_OnExitRun` callouts.
The system designer must consider to adapt the Watchdog Manager mode to a different scheduling scheme.

## 7.12 Error Classification

| *Type or error* | *Relevance* | *Related error code* | *Value* |
|---|---|---|---|
| A service was called prior to initialization | Development | ECUM_E_NOT_INITED | 0x10 |
| A service was called which was disabled by configuration | Development | ECUM_ E_SERVICE_DISABLED | 0x11 |
| A null pointer was passed as an argument | Development | ECUM_ E_NULL_POINTER | 0x12 |
| A parameter was invalid (unspecific) | Development | ECUM_E_INVALID_PAR | 0x13 |
| RUN was requested multiple times by the same user ID | Development | ECUM_E_MULTIPLE_RUN_REQUESTS | 0x14 |
| RUN was released though it was not requested | Development | ECUM_E_MISMATCHED_RUN_RELEASE | 0x15 |
| A state, passed as an argument to a service, was out of range (specific parameter test) | Development | ECUM_E_ STATE_PAR_OUT_OF_RANGE | 0x16 |
| An unknown wakeup source was passed as a parameter to an API | Development | ECUM_E_UNKNOWN_WAKEUP_SOURCE | 0x17 |
| The RAM check during wakeup failed | Production | ECUM_E_RAM_CHECK_FAILED | |
| The service EcuM_KillAllRUNRequests was issued | Production | ECUM_E_ALL_RUN_REQUESTS_KILLED | |
| The post build configuration parameter ECUM_COMMCONFIG_SELECTOR is out of range | Production | ECUM_E_COMMINIT_SELECTOR_OUT_OF_RANGE | |
| Configuration data is inconsistent | Production | ECUM_E_CONFIGURATION_DATA_INCONSISTENT | |

**Table 5 - Error Classification**

**EcuM2759**: All errors shall be reported as events.

**EcuM2757**: All errors shall be treated as errors immediately.

**EcuM2758**: All errors shall not be healable.

# 8 API specification

## 8.1 Imported Types

### 8.1.1 Standard Types

In this chapter all used standard types are listed.

- Std_ReturnType
- Std_VersionInfoType

### 8.1.2 Communication Manager Types

In this chapter all used types included from module ComM are listed.

- ComM_ConfigType
- ComM_ChannelHandleType

### 8.1.3 Watchdog Manager Types

In this chapter all used types included from module WdgM are listed.

- WdgM_ConfigType

### 8.1.4 NVRAM Manager Types

In this chapter all used types included from module NvM are listed.

- NvM_RequestResultType

### 8.1.5 MCU Driver Types

In this chapter all used types included from module MCU Driver are listed.

- Mcu_ModeType
- Mcu_ResetType

### 8.1.6 OS Types

In this chapter all used types included from module OS are listed.

- AppModeType (OSEK specification v2.2.3)

## 8.2 Type definitions

### 8.2.1 EcuM_ConfigType

| | | |
|---|---|---|
| ***Type:*** | EcuM_ConfigType | |
| ***Range:*** | Structure | **EcuM2801**: This structure shall hold the post-build configuration parameters for the ECU State Manager itself as well as pointers to all `ConfigType` structures of modules that have post-build parameters and are directly initialized by the ECU State Manager. Furthermore, it contains such a pointer for each module that initializes other modules (a so-called non-leaf module). |
| ***Description:*** | A pointer to such a structure shall be provided to the ECU State Manager initialization routine for configuration.<br>**EcuM2793**: The ECU State Manager Configuration Tool shall specifically generate this structure for a given set of basic software modules that comprise the ECU configuration. The contents of the structure specifically depends upon the configuration parameters in *ECUM_DRIVER_INIT_LIST_ONE* and *ECUM_DRIVER_INIT_LIST_TWO*.<br>**EcuM2794**: This structure shall contain an additional post-build configuration variant identifier (uint8/uint16/uint32 depending on algorithm to compute the identifier). See also chapter *10.5 Checking Configuration Consistency.*<br>**EcuM2795:** This structure shall contain an additional hash code with is tested against the configuration parameter *ECUM_CONFIGCONSISTENCY_HASH* for checking consistency of the configuration data. See also chapter *10.5 Checking Configuration Consistency.*<br>**EcuM2800**: The ECU State Manager Configuration Tool shall also generate for each given ECU configuration an instance of this structure that is filled with the post-build configuration parameters of the ECU State Manager as well as pointers to instances of configuration structures for the modules mentioned in EcuM2793: . The pointers are derived from the corresponding *ECUM_<Module>_INIT_CONFIG* parameters. | |

### 8.2.2 EcuM_StateType

| | | |
|---|---|---|
| ***Type:*** | uint8 | |
| ***Range:*** | ECUM_SUBSTATE_MASK | 0x0f |
| | ECUM_STATE_STARTUP | 0x10 |
| | ECUM_STATE_STARTUP_ONE | 0x11 |
| | ECUM_STATE_STARTUP_TWO | 0x12 |
| | ECUM_STATE_WAKEUP | 0x20 |
| | ECUM_STATE_WAKEUP_ONE | 0x21 |
| | ECUM_STATE_WAKEUP_VALIDATION | 0x22 |
| | ECUM_STATE_WAKEUP_REACTION | 0x23 |
| | ECUM_STATE_WAKEUP_TWO | 0x24 |
| | ECUM_STATE_WAKEUP_WAKESLEEP | 0x25 |
| | ECUM_STATE_WAKEUP_TTII | 0x26 |
| | ECUM_STATE_RUN | 0x30 |
| | ECUM_STATE_APP_RUN | 0x32 |
| | ECUM_STATE_APP_POST_RUN | 0x33 |
| | ECUM_STATE_SHUTDOWN | 0x40 |
| | ECUM_STATE_PREP_SHUTDOWN | 0x44 |
| | ECUM_STATE_GO_SLEEP | 0x49 |
| | ECUM_STATE_GO_OFF_ONE | 0x4d |

| | | |
|---|---|---|
| | ECUM_STATE_GO_OFF_TWO | 0x4e |
| | ECUM_STATE_SLEEP | 0x50 |
| | ECUM_STATE_RESET | 0x90 |
| | ECUM_STATE_OFF | 0x80 |
| ***Description:*** | **EcuM507**: Encodes states and sub-states of the ECU State Manager. States are encoded in the hi-nibble, sub-state in the lo-nibble. The sub-state can be determined be ANDing the state value with ECUM_SUBSTATE_MASK.<br>**EcuM2664**: The ECU State Manager shall define all states as listed in the EcuM_StateType. | |

### 8.2.3 EcuM_UserType

| | |
|---|---|
| ***Type:*** | uint8 |
| ***Range:*** | -- |
| ***Description:*** | **EcuM487**: For each user, a unique value must be defined at system generation time. Ref. to *10.3 Configurable Parameters*. |

### 8.2.4 EcuM_WakeupSourceType

| | | |
|---|---|---|
| ***Type:*** | uint8/uint16/uint32, used as a bitfield | |
| ***Range:*** | ECUM_WKSOURCE_POWER | Power cycle (bit 0) |
| | ECUM_WKSOURCE_RESET (default) | Hardware reset (bit 1).<br>If hardware cannot distinguish between a power cycle and a reset reason, then this shall be the default wakeup source. |
| | ECUM_WKSOURCE_INTERNAL_RESET | Internal reset of µC (bit 2)<br>The internal reset typically only resets the µC core but not peripherals or memory controllers. The exact behavior is hardware specific.<br>This source may also indicate an unhandled exception. |
| | ECUM_WKSOURCE_INTERNAL_WDG | Reset by internal watchdog (bit 3) |
| | ECUM_WKSOURCE_EXTERNAL_WDG | Reset by external watchdog (bit 4), if detection supported by hardware |
| | ECUM_WKSOURCE_ALL_SOURCES | ~0[29] |
| | … | **EcuM2165**: The list can be extended by configuration |
| ***Description:*** | The bitfield provides one bit for each wakeup source.<br>In WAKEUP state, all bits cleared indicates that no wakeup source is known.<br>In STARTUP state, all bits cleared indicates that no reason for restart or reset is known. In this case, ECUM_WKSOURCE_RESET shall be assumed.<br>**EcuM2166**: Extension values (see chapter *10.3 Configurable Parameters)* must define single additional bits. The bit assignment shall be done by the configuration tool.<br>**EcuM2601**: If hardware cannot detect a specific wakeup source, then the ECU State Manager shall report a ECUM_WKSOURCE_RESET instead. | |

[29] ~0 means bitwise NOT NULL as specified by the C language. For the specified type uint8 this results in 0xFF.

### 8.2.5 EcuM_WakeupStatusType

| | | | |
|---|---|---|---|
| ***Type:*** | uint8 | | |
| ***Range:*** | ECUM_WKSTATUS_NONE | 0 | No pending wakeup event was detected |
| | ECUM_WKSTATUS_PENDING | 1 | The wakeup event was detected but not yet validated |
| | ECUM_WKSTATUS_VALIDATED | 2 | The wakeup event is valid |
| | ECUM_WKSTATUS_EXPIRED | 3 | The wakeup event has not been validated and has expired therefore |
| ***Description:*** | The type describes the possible outcomes of the WAKEUP VALIDATION state. The type may be applied to one wakeup source or a collection of wakeup sources. See also *8.3.3.7 EcuM_GetStatusOfWakeupSource*. | | |

### 8.2.6 EcuM_WakeupReactionType

| | | | |
|---|---|---|---|
| ***Type:*** | uint8 | | |
| ***Range:*** | ECUM_WKACT_RUN | 0 | Initialization into RUN state |
| | ECUM_WKACT_TTII | 2 | Execute time triggered increased inoperation protocol and shutdown |
| | ECUM_WKACT_SHUTDOWN | 3 | Immediate shutdown |
| ***Description:*** | The type describes the possible outcomes of the WAKEUP REACTION state. | | |

### 8.2.7 EcuM_SleepModeConfigType

| | | |
|---|---|---|
| ***Type:*** | Structure | |
| ***Range:*** | `uint8 mcuMode` | Index to configuration tables of MCU driver. Describes the sleep mode of the MCU. |
| | `boolean suspendCPU` | Flag, which is set true, if the CPU is suspended, halted, or powered off in the sleep mode. If the CPU keeps running in this sleep mode, then this flag must be set to false. |
| | `EcuM_WakeupSourceType wakeupSource` | All wakeup sources where the according bit is set shall be enabled for the following sleep period. |
| ***Description:*** | This type wraps the Mcu_ModeType. This is done to avoid types of other packages in this API specification. By configuration, the system designer can define a list with elements of this type which define the different sleep modes (see *10.3 Configurable Parameters*, *8.3.2.9 EcuM_SelectShutdownTarget*, and *8.3.2.12 EcuM_GetLastShutdownTarget*) | |

### 8.2.8 EcuM_BootTargetType

| | | | |
|---|---|---|---|
| ***Type:*** | uint8 | | |
| ***Range:*** | `ECUM_BOOT_TARGET_APP` | 0 | The ECU will boot into the application |
| | `ECUM_BOOT_TARGET_BOOTLOADER` | 1 | The ECU will boot into the bootloader |
| ***Description:*** | -- | | |

### 8.2.9 EcuM_ChannelHandleType

| | | |
|---|---|---|
| ***Type:*** | ComM_ChannelHandleType | |
| ***Range:*** | `0..254` | channel handle number |
| | `255` | no channel |
| ***Description:*** | The type wraps the ComM_ChannelHandleType. See [9]. | |

## 8.3 Function Definitions

### 8.3.1 Initialization and Shutdown

#### 8.3.1.1 EcuM_Init

| | | |
|---|---|---|
| ***Service name:*** | EcuM_Init | |
| ***Syntax:*** | `void EcuM_Init(const EcuM_ConfigType *configPtr)` | |
| ***Service ID:*** | 1 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Non-Reentrant | |
| ***Parameters (in):*** | configPtr | Pointer to post build configuration of the ECU. |
| ***Parameters (out):*** | None | |
| ***Description:*** | Initializes the ECU state manager and carries out the startup procedure. The function will never return (it calls StartOS, see *5.4 Operating System*). | |

| | |
|---|---|
| ***Caveats:*** | -- |
| ***Configuration:*** | -- |

### 8.3.1.2 EcuM_Shutdown

| | | |
|---|---|---|
| ***Service name:*** | EcuM_Shutdown | |
| ***Syntax:*** | `void EcuM_Shutdown(void)` | |
| ***Service ID:*** | 2 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Non-Reentrant | |
| ***Parameters (in):*** | None | |
| ***Parameters (out):*** | None | |
| ***Return value:*** | None | |
| ***Description:*** | Typically called from the shutdown hook, this function takes over execution control and will carry out GO OFF II activities. | |
| ***Caveats:*** | -- | |
| ***Configuration:*** | -- | |

### 8.3.1.3 EcuM_GetVersionInfo

| | | |
|---|---|---|
| ***Service name:*** | EcuM_ GetVersionInfo | |
| ***Syntax:*** | `void EcuM_GetVersionInfo (     Std_VersionInfoType *versioninfo )` | |
| ***Service ID [hex]:*** | 0 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | non reentrant | |
| ***Parameters (in):*** | `none` | |
| ***Parameters (out):*** | `versioninfo` | Pointer to where to store the version information of this module. |
| ***Return value:*** | none | |
| ***Description:*** | **EcuM2728**: This service returns the version information of this module. The version information includes:<br>- Module Id<br>- Vendor Id<br>- Vendor specific version numbers (BSW00407).<br><br>**EcuM2729**: This function shall be pre compile time configurable On/Off by the configuration parameter: ECUM_VERSION_INFO_API<br><br>Hint:<br>If source code for caller and callee of this function is available this function should be realized as a macro. The macro should be defined in the modules header file. | |
| ***Caveats:*** | -- | |
| ***Configuration:*** | -- | |

### 8.3.2 State Management

#### 8.3.2.1 EcuM_RequestRUN

| | | |
|---|---|---|
| ***Service name:*** | EcuM_RequestRUN | |
| ***Syntax:*** | `Std_ReturnType EcuM_RequestRUN ( EcuM_UserType user )` | |
| ***Service ID:*** | 3 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Reentrant | |
| ***Parameters (in):*** | `user` | ID of the entity requesting the RUN state. |
| ***Return value:*** | E_OK | The request was accepted by EcuM |
| | E_NOT_OK | The request was not accepted by EcuM, a detailed error condition was sent to DET (see *Error Codes* below). |
| ***Description:*** | Places a request for the RUN state. Requests can be placed by every user made known to the state manager at configuration time.<br>**EcuM2143**: Requests cannot be nested, i.e. one user can only place one request but not more. Additional or duplicate user requests by the same user shall be ignored.<br>**EcuM2144**: An implementation must track requests for each user known on the ECU. Run requests are specific to the user.<br>**EcuM2668**: RUN requests shall be ignored after `EcuM_KillAllRUNRequests` has been executed until the shutdown has completed.<br>The service is intended for implementing AUTOSAR ports. | |
| ***Caveats:*** | -- | |
| ***Configuration:*** | Ref. to *8.2.3 EcuM_UserType* for more information about user IDs and their generation. | |
| ***Error Codes:*** | `ECUM_E_MULTIPLE_RUN_REQUESTS` | On multiple requests by the same user ID |

#### 8.3.2.2 EcuM_ReleaseRUN

| | | |
|---|---|---|
| ***Service name:*** | EcuM_ReleaseRUN | |
| ***Syntax:*** | `Std_ReturnType EcuM_ReleaseRUN ( EcuM_UserType user )` | |
| ***Service ID:*** | 4 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Reentrant | |
| ***Parameters (in):*** | `user` | ID of the entity releasing the RUN state. |
| ***Return value:*** | E_OK | The release request was accepted by EcuM |
| | E_NOT_OK | The release request was not accepted by EcuM, a detailed error condition was sent to DET (see *Error Codes* below). |
| ***Description:*** | Releases a RUN request previously done with a call to EcuM_RequestRUN. See also *8.3.2.1 EcuM_RequestRUN* for details.<br>The service is intended for implementing AUTOSAR ports. | |
| ***Caveats:*** | -- | |

| | | |
|---|---|---|
| ***Configuration:*** | Ref. to *8.2.3 EcuM_UserType* for more information about user IDs and their generation. | |
| ***Error Codes:*** | `ECUM_E_MISMATCHED_RUN_RELEASE` | On releasing without a matching request. |

### 8.3.2.3 EcuM_ComM_RequestRUN

| | | |
|---|---|---|
| ***Service name:*** | EcuM_ComM_RequestRUN | |
| ***Syntax:*** | `Std_ReturnType EcuM_ComM_RequestRUN ( EcuM_ChannelHandleType channel )` | |
| ***Service ID:*** | 14 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Reentrant | |
| ***Parameters (in):*** | `channel` | ID of the communication channel requesting the RUN state. |
| ***Return value:*** | E_OK | The request was accepted by EcuM |
| | E_NOT_OK | The request was not accepted by EcuM, a detailed error condition was sent to DET (see *Error Codes* below). |
| ***Description:*** | The behavior is identical to `EcuM_RequestRUN` except that the parameter is not a user but a communication channel.<br>**EcuM2789**: The ECU State Manager shall track requests by communication channels in exactly the same way as it tracks other users. | |
| ***Caveats:*** | -- | |
| ***Configuration:*** | | |
| ***Error Codes:*** | `ECUM_E_MULTIPLE_RUN_REQUESTS` | On multiple requests by the same user ID |

### 8.3.2.4 EcuM_ComM_ReleaseRUN

| | | |
|---|---|---|
| ***Service name:*** | EcuM_ReleaseRUN | |
| ***Syntax:*** | `Std_ReturnType EcuM_ComM_ReleaseRUN ( EcuM_ChannelHandleType channel )` | |
| ***Service ID:*** | 16 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Reentrant | |
| ***Parameters (in):*** | `channel` | ID of the communication channel releasing the RUN state. |
| ***Return value:*** | E_OK | The release request was accepted by EcuM |
| | E_NOT_OK | The release request was not accepted by EcuM, a detailed error condition was sent to DET (see *Error Codes* below). |
| ***Description:*** | Releases a RUN request previously done with a call to `EcuM_ComM_RequestRUN`. See also *8.3.2.3 EcuM_ComM_RequestRUN* for details.<br>**EcuM2792**: The service shall clear all corresponding wakeup sources corresponding to this channel. | |
| ***Caveats:*** | -- | |

| | | |
|---|---|---|
| ***Configuration:*** | | |
| ***Error Codes:*** | `ECUM_E_MISMATCHED_RUN_RELEASE` | On releasing without a matching request. |

#### 8.3.2.5 EcuM_ComM_HasRequestedRUN

| | | |
|---|---|---|
| ***Service name:*** | EcuM_ReleaseRUN | |
| ***Syntax:*** | `boolean EcuM_ComM_HasRequestedRUN ( EcuM_ChannelHandleType channel )` | |
| ***Service ID:*** | 27 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Reentrant | |
| ***Parameters (in):*** | `channel` | ID of the communication channel being tested |
| ***Return value:*** | true | The channel has reqeusted RUN state |
| | false | The channel has not requested RUN state |
| ***Description:*** | Returns if a channel has requested RUN state. | |
| ***Caveats:*** | -- | |
| ***Configuration:*** | -- | |
| ***Error Codes:*** | -- | |

#### 8.3.2.6 EcuM_RequestPOST_RUN

| | | |
|---|---|---|
| ***Service name:*** | EcuM_RequestPOST_RUN | |
| ***Syntax:*** | `Std_ReturnType EcuM_RequestPOST_RUN ( EcuM_UserType user )` | |
| ***Service ID:*** | 10 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Reentrant | |
| ***Parameters (in):*** | `user` | ID of the entity requesting the POST RUN state. |
| ***Return value:*** | E_OK | The request was accepted by EcuM |
| | E_NOT_OK | The request was not accepted by EcuM, a detailed error condition was sent to DET (see *Error Codes* below). |
| ***Description:*** | Places a request for the POST RUN state. Requests can be placed by every user made known to the state manager at configuration time.<br>All requirements of *8.3.2.1 EcuM_RequestRUN* apply accordingly.<br>Requests for RUN and POST RUN must be tracked independently (in other words: two independent variables).<br>The service is intended for implementing AUTOSAR ports. | |
| ***Caveats:*** | -- | |
| ***Configuration:*** | Ref. to *8.2.3 EcuM_UserType* for more information about user IDs and their generation. | |
| ***Error Codes:*** | `ECUM_E_MULTIPLE_RUN_REQUESTS` | On multiple requests by the same user ID |

#### 8.3.2.7 EcuM_ReleasePOST_RUN

| | | |
|---|---|---|
| ***Service name:*** | EcuM_ReleasePOST_RUN | |
| ***Syntax:*** | `Std_ReturnType EcuM_ReleasePOST_RUN ( EcuM_UserType user )` | |
| ***Service ID:*** | 11 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Reentrant | |
| ***Parameters (in):*** | `user` | ID of the entity releasing the POST RUN state. |
| ***Return value:*** | E_OK | The release request was accepted by EcuM |
| | E_NOT_OK | The release request was not accepted by EcuM, a detailed error condition was sent to DET (see *Error Codes* below). |
| ***Description:*** | Releases a POST RUN request previously done with a call to EcuM_RequestPOST_RUN. See also *8.3.2.6 EcuM_RequestPOST_RUN* for details.<br>The service is intended for implementing AUTOSAR ports. | |
| ***Caveats:*** | -- | |
| ***Configuration:*** | Ref. to *8.2.3 EcuM_UserType* for more information about user IDs and their generation. | |
| ***Error Codes:*** | `ECUM_E_MISMATCHED_RUN_RELEASE` | On releasing without a matching request. |

#### 8.3.2.8 EcuM_KillAllRUNRequests

| | | |
|---|---|---|
| ***Service name:*** | EcuM_KillAllRUNRequests | |
| ***Syntax:*** | `void EcuM_KillAllRUNRequests(void)` | |
| ***Service ID:*** | 5 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Reentrant | |
| ***Parameters (in):*** | None | |
| ***Parameters (out):*** | None | |
| ***Return value:*** | None | |
| ***Description:*** | **EcuM1872**: The function unconditionally undoes all requests to RUN and POST RUN.<br>**EcuM2600**: As a consequence `EcuM_RequestRUN` and `EcuM_RequestPOST_RUN` must not accept any new requests unless the resulting shutdown has been completed.<br>The benefit of this function over an ECU reset is that the shutdown sequence is executed, which e.g. takes care of writing back NV memory contents. | |
| ***Caveats:*** | Use this function with care. Side effects may occur in the application. If an implementation contains synchronization for more graceful shutdown a timeout must be provided to ensure that the shutdown process is initiated. | |
| ***Configuration:*** | -- | |
| ***Error Codes:*** | `ECUM_E_ALL_RUN_REQUESTS_KILLED` | On each invocation. |

### 8.3.2.9 EcuM_SelectShutdownTarget

| | | |
|---|---|---|
| ***Service name:*** | EcuM_SelectShutdownTarget | |
| ***Syntax:*** | `Std_ReturnType EcuM_SelectShutdownTarget (`<br>`    EcuM_StateType target,`<br>`    uint8 mode`<br>`)` | |
| ***Service ID:*** | 6 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Reentrant | |
| ***Parameters (in):*** | `target` | **EcuM624**: The selected shutdown target. Only the following subset of the EcuM_StateType value range is accepted:<br>• ECUM_STATE_SLEEP<br>• ECUM_STATE_RESET<br>• ECUM_STATE_OFF<br>All other values will be rejected and result in a development error message ECUM_E_STATE_PAR_OUT_OF_RANGE must be thrown. |
| | `mode` | **EcuM2185**: An index like value which can be dereferenced to a sleep mode (`EcuM_SleepModeConfigType`). Available sleep modes are defined at configuration time and are stored in the *`EcuM_SleepMode`* list. This list then contains the parameter for the `Mcu_SetMode` service, which is executed to put the ECU into SLEEP state. The mode parameter shall only be used if the target parameter equals to ECUM_STATE_SLEEP. In all other cases, it shall be ignored. |
| ***Parameters (out):*** | None | |
| ***Return value:*** | E_OK | The new shutdown target was set |
| | E_NOT_OK | The new shutdown target was not set |
| ***Description:*** | **EcuM2585**: An implementation of this service should not initiate any setup activities but only store the value for later use in the SHUTDOWN state.<br>**EcuM2228**: An implementation must preload the TTII divisor counter variable with the preload value defined in the *`ECUM_TTII_DIVISOR_LIST`*.<br>The service is intended for implementing AUTOSAR ports. | |
| ***Caveats:*** | The ECU State Manager does not define any mechanism to resolve issues arising from parallel requests. It is rather assumed that there will be one piece of application which specific to the ECU and handles these kinds of issues. | |
| ***Configuration:*** | -- | |

### 8.3.2.10 EcuM_GetState

| | |
|---|---|
| ***Service name:*** | EcuM_GetState |
| ***Syntax:*** | `EcuM_StateType EcuM_GetState(void)` |
| ***Service ID:*** | 7 |
| ***Sync/Async:*** | Synchronous |
| ***Reentrancy:*** | Reentrant |

| | |
|---|---|
| ***Parameters (in):*** | None |
| ***Parameters (out):*** | None |
| ***Return value:*** | The value of the internal state variable. |
| ***Description:*** | **EcuM2421**: The service is intended for implementing AUTOSAR ports.<br>**EcuM2423**: The service must be accessible from an OS and an OS-free context as well as from an interrupt context. |
| ***Caveats:*** | -- |
| ***Configuration:*** | -- |

#### 8.3.2.11 EcuM_GetShutdownTarget

| | | |
|---|---|---|
| ***Service name:*** | EcuM_GetShutdownTarget | |
| ***Syntax:*** | `Std_ReturnType EcuM_GetShutdownTarget`<br>`(`<br>`    EcuM_StateType *shutdownTarget,`<br>`    uint8 *sleepMode`<br>`)` | |
| ***Service ID:*** | 9 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Reentrant | |
| ***Parameters (in):*** | None | |
| ***Parameters (out):*** | `shutdownTarget` | One of these values is returned:<br>• `ECUM_STATE_SLEEP`<br>• `ECUM_STATE_RESET`<br>• ECUM_STATE_OFF |
| | `sleepMode` | If the return parameter is `ECUM_STATE_SLEEP`, this out parameter tells which of the configured sleep modes was actually chosen (index into *`EcuM_SleepMode`*).<br>**EcuM2788**: An implementation shall cope with NULL pointers by simply ignoring the out parameter in all cases. An implementation may assert the ECUM_E_NULL_POINTER development error. |
| ***Return value:*** | `E_OK` | The service always succeeds |
| ***Description:*** | This function returns always the selected shutdown target as set by `EcuM_SelectShutdownTarget` | |
| ***Caveats:*** | -- | |
| ***Configuration:*** | -- | |

#### 8.3.2.12 EcuM_GetLastShutdownTarget

| | |
|---|---|
| ***Service name:*** | EcuM_GetLastShutdownTarget |
| ***Syntax:*** | `Std_ReturnType EcuM_GetLastShutdownTarget`<br>`(`<br>`    EcuM_StateType *shutdownTarget,`<br>`    uint8 *sleepMode` |

| | | |
|---|---|---|
| | ) | |
| ***Service ID:*** | 8 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Reentrant | |
| ***Parameters (in):*** | None | |
| ***Parameters (out):*** | `shutdownTarget` | One of these values is returned:<br>• `ECUM_STATE_SLEEP`<br>• `ECUM_STATE_RESET`<br>• ECUM_STATE_OFF |
| | `sleepMode` | **EcuM2336**: If the return parameter is `ECUM_STATE_SLEEP`, this out parameter tells which of the configured sleep modes was actually chosen (index into *`EcuM_SleepMode`*).<br>**EcuM2337**: An implementation shall cope with NULL pointers by simply ignoring the out parameter in all cases. An implementation may assert the ECUM_E_NULL_POINTER development error. |
| ***Return value:*** | `E_OK` | The service always succeeds |
| ***Description:*** | **EcuM2156**: The return value describes the ECU state from which the last wakeup or power up occurred. This function shall return always the same value until the next shutdown.<br>**EcuM2157**: This function is intended for primary use in STARTUP or RUN state. Reasonable use cases exist there. To simplify implementation, it is acceptable if the value is set in late shutdown phase for use during the next startup. If so, implementation specific limitations must be clearly documented. | |
| ***Caveats:*** | -- | |
| ***Configuration:*** | -- | |

### 8.3.3 Wakeup Notifications

#### 8.3.3.1 EcuM_SetWakeupEvent

| | | |
|---|---|---|
| ***Service name:*** | EcuM_SetWakeupEvent | |
| ***Syntax:*** | `void EcuM_SetWakeupEvent`<br>`(`<br>`    EcuM_WakeupSourceType sources`<br>`)` | |
| ***Service ID:*** | 12 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Non-Reentrant, Non-Interruptible | |
| ***Parameters (in):*** | `sources` | Value to be set |
| ***Parameters (out):*** | None | |
| ***Return value:*** | None | |
| ***Description:*** | **EcuM1117**: Takes the value and stores it in an internal variable (OR-operation).<br>**EcuM2707**: The service must start the wakeup validation timeout timer according to chapter *7.8.4 WakeupValidation Timout Timer.*<br>**EcuM2171**: The function must be callable from interrupt context, from OS context and an OS-free context. | |

| | |
|---|---|
| ***Caveats:*** | -- |
| ***Configuration:*** | -- |

#### 8.3.3.2 EcuM_GetPendingWakeupEvents

| | | |
|---|---|---|
| ***Service name:*** | EcuM_GetPendingWakeupEvents | |
| ***Syntax:*** | `EcuM_WakeupSourceType EcuM_GetPendingWakeupEvents(void)` | |
| ***Service ID:*** | 13 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Non-Reentrant, Non-Interruptible | |
| ***Parameters (in):*** | None | |
| ***Parameters (out):*** | None | |
| ***Return value:*** | All wakeup events | EcuM1156: Returns wakeup events which have been set but not yet validated. |
| ***Description:*** | **EcuM2172**: The service must be callable from interrupt context, from OS context and an OS-free context. | |
| ***Caveats:*** | The wakeup events returned by this service are only pending | |
| ***Configuration:*** | -- | |

#### 8.3.3.3 EcuM_ClearWakeupEvent

| | | |
|---|---|---|
| ***Service name:*** | EcuM_ClearWakeupEvent | |
| ***Syntax:*** | `void EcuM_ClearWakeupEvent`<br>`(`<br>`    EcuM_WakeupSourceType sources`<br>`)` | |
| ***Service ID:*** | 22 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Non-Reentrant, Non-Interruptible | |
| ***Parameters (in):*** | `sources` | Events to be cleared |
| ***Parameters (out):*** | None | |
| ***Return value:*** | None | |
| ***Description:*** | **EcuM2683**: Clears all pending events passed in the in parameters from the internal variable (NAND-operation).<br>**EcuM2171**: The function must be callable from interrupt context, from OS context and an OS-free context. | |
| ***Caveats:*** | -- | |
| ***Configuration:*** | -- | |

#### 8.3.3.4 EcuM_ValidateWakeupEvent

| | |
|---|---|
| ***Service name:*** | EcuM_ValidateWakeupEvent |
| ***Syntax:*** | `void EcuM_ValidateWakeupEvent`<br>`(` |

| | |
|---|---|
| | `EcuM_WakeupSourceType sources`<br>`)` |
| ***Service ID:*** | 20 |
| ***Sync/Async:*** | Synchronous |
| ***Reentrancy:*** | Reentrant |
| ***Parameters (in):*** | `sources` Events to be validated |
| ***Parameters (out):*** | None |
| ***Return value:*** | None |
| ***Description:*** | After wakeup, the ECU State Manager will stop the process during the WAKEUP VALIDATION state to wait for validation of the wakeup event (see *Figure 16 - Wakeup Sequence (high level* diagram)). The validation is carried out with a call to this API service.<br>**EcuM2344**: The validation shall be valid when ANDing the parameter `events` with the internal variable of pending wakeup events results in a value other than null.<br>**EcuM2645**: The service shall invoke `ComM_EcuM_WakeUpIndication` of the Communication Manager for each wakeup event if the ComChannel parameter in the EcuM_WakeupSource configuration container for the according wakeup source is not 255 (see also *10.3.7 EcuM_WakeupSource*).<br>**EcuM2345**: The function must be callable from interrupt context, from OS context, and an OS-free context.<br>**EcuM2790**: The service shall return without effect for all sources except communication channels when called while ECU State Manager is NOT in one of the states: SHUTDOWN, SLEEP, WAKEUP I, WAKEUP VALIDATION.<br>**EcuM2791**: The service shall always have full effect for all those sources which correspond to a communication channel (see also ***EcuM2645:*** ). |
| ***Caveats:*** | -- |
| ***Configuration:*** | -- |

### 8.3.3.5 EcuM_GetValidatedWakeupEvents

| | |
|---|---|
| ***Service name:*** | EcuM_GetValidatedWakeupEvents |
| ***Syntax:*** | `EcuM_WakeupSourceType EcuM_GetValidatedWakeupEvents(void)` |
| ***Service ID:*** | 21 |
| ***Sync/Async:*** | Synchronous |
| ***Reentrancy:*** | Non-Reentrant, Non-Interruptible |
| ***Parameters (in):*** | None |
| ***Parameters (out):*** | None |
| ***Return value:*** | All wakeup events EcuM2533: Returns the value from the internal variable. |
| ***Description:*** | **EcuM2532:** The service must be callable from interrupt context, from OS context and an OS-free context. |
| ***Caveats:*** | -- |
| ***Configuration:*** | -- |

#### 8.3.3.6 EcuM_GetExpiredWakeupEvents

| | | |
|---|---|---|
| ***Service name:*** | EcuM_GetExpiredWakeupEvents | |
| ***Syntax:*** | `EcuM_WakeupSourceType EcuM_GetExpiredWakeupEvents(void)` | |
| ***Service ID:*** | 25 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Non-Reentrant, Non-Interruptible | |
| ***Parameters (in):*** | None | |
| ***Parameters (out):*** | None | |
| ***Return value:*** | All wakeup events | Returns all events that have been set and for which validation has failed. Events which do not need validation must never be reported by this function. |
| ***Description:*** | **EcuM2589**: The service must be callable from interrupt context, from OS context and an OS-free context. | |
| ***Caveats:*** | -- | |
| ***Configuration:*** | -- | |

#### 8.3.3.7 EcuM_GetStatusOfWakeupSource

| | | |
|---|---|---|
| ***Service name:*** | EcuM_GetStatusOfWakeupSource | |
| ***Syntax:*** | `EcuM_WakeupStatusType EcuM_GetStatusOfWakeupSource ( EcuM_WakeupSourceType sources )` | |
| ***Service ID:*** | 23 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Reentrant | |
| ***Parameters (in):*** | `sources` | EcuM2754: The sources for which the status is returned.<br>If sources equals 0, then this service returns `ECUM_WKSTATUS_NONE`.<br>If sources equals `ECUM_WKSOURCE_ALL_SOURCES`, then the service returns the sum status of all registered sources.<br>If sources contains an unknown source, then the sum status of all known sources shall be returned, but the service shall send the `ECUM_E_UNKNOWN_WAKEUP_SOURCE` error message to DET (development only). |
| ***Parameters (out):*** | None | |
| ***Return value:*** | | Sum status of all wakeup sources passed in the in parameter. |
| ***Description:*** | The sum status shall be computed according to the following algorithm:<br>• If `EcuM_GetValidatedWakeupEvents` returns not null then return `ECUM_WKSTATUS_VALIDATED`<br>• If `EcuM_GetPendingWakeupEvents` returns not null then return `ECUM_WKSTATUS_PENDING`<br>• If `EcuM_GetExpiredWakeupEvents` returns not null then return `ECUM_WKSTATUS_EXPIRED`<br>`Else return ECUM_WKSTATUS_NONE` | |

| | |
|---|---|
| ***Caveats:*** | -- |
| ***Configuration:*** | -- |

### 8.3.4 Miscellaneous

#### 8.3.4.1 EcuM_SelectApplicationMode

| | | |
|---|---|---|
| ***Service name:*** | EcuM_SelectApplicationMode | |
| ***Syntax:*** | `Std_ReturnType EcuM_SelectApplicationMode ( AppModeType appMode )` | |
| ***Service ID:*** | 15 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Reentrant | |
| ***Parameters (in):*** | `appMode` | EcuM2081: The application mode taken for next OS start. The type is defined by the operating system. |
| ***Parameters (out):*** | None | |
| ***Return value:*** | E_OK | The new application mode was accepted by EcuM |
| | E_NOT_OK | The new application mode was not accepted by EcuM |
| ***Description:*** | The implementation should store the application mode preferably in a non-initialized area of RAM.<br>The service is intended for implementing AUTOSAR ports. | |
| ***Caveats:*** | -- | |
| ***Configuration:*** | -- | |

#### 8.3.4.2 EcuM_GetApplicationMode

| | | |
|---|---|---|
| ***Service name:*** | EcuM_GetApplicationMode | |
| ***Syntax:*** | `Std_ReturnType EcuM_GetApplicationMode ( AppModeType *appMode )` | |
| ***Service ID:*** | 17 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Reentrant | |
| ***Parameters (in):*** | None | |
| ***Parameters (out):*** | appMode | The currently selected application mode, see also `EcuM_SelectApplicationMode` |
| ***Return value:*** | E_OK | The service always succeeds |
| ***Description:*** | The service is intended for implementing AUTOSAR ports. | |
| ***Caveats:*** | -- | |
| ***Configuration:*** | -- | |

#### 8.3.4.3 EcuM_SelectBootTarget

| Service name: | EcuM_SelectBootTarget | |
|---|---|---|
| Syntax: | `Std_ReturnType EcuM_SelectBootTarget ( EcuM_BootTargetType target )` | |
| Service ID: | 18 | |
| Sync/Async: | Synchronous | |
| Reentrancy: | Reentrant | |
| Parameters (in): | `target` | The selected boot target. |
| Parameters (out): | None | |
| Return value: | E_OK | The new boot target was accepted by EcuM |
| | E_NOT_OK | The new boot target was not accepted by EcuM |
| Description: | **EcuM2247**: The service must store the selected target in a way which is compatible with the boot loader. This may mean format AND location.<br>The service is intended for implementing AUTOSAR ports. | |
| Caveats: | This service may be dependent on the boot loader used. | |
| Configuration: | -- | |

#### 8.3.4.4 EcuM_GetBootTarget

| Service name: | EcuM_GetBootTarget | |
|---|---|---|
| Syntax: | `Std_ReturnType EcuM_GetBootTarget ( EcuM_BootTargetType *target )` | |
| Service ID: | 19 | |
| Sync/Async: | Synchronous | |
| Reentrancy: | Reentrant | |
| Parameters (in): | None | |
| Parameters (out): | target | The currently selected boot target. |
| Return value: | E_OK | The service always succeeds. |
| Description: | see *8.3.4.3 EcuM_SelectBootTarget.*<br>The service is intended for implementing AUTOSAR ports. | |
| Caveats: | -- | |
| Configuration: | -- | |

## 8.4 Scheduled Functions

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

### 8.4.1 EcuM_MainFunction

| | |
|---|---|
| ***Service name:*** | EcuM_MainFunction |
| ***Syntax:*** | `void EcuM_MainFunction(void)` |
| ***Service ID:*** | 24 |
| ***Sync/Async:*** | Synchronous |
| ***Reentrancy:*** | Non-Reentrant |
| ***Timing*** | Variable cyclic |
| ***Description:*** | The purpose of this service is to implement all activities of the ECU State Manager while the OS is up and running.<br>**EcuM2594**: This service must be called on a periodic basis from an adequate BSW task (i.e. a task under control of the BSW scheduler).<br>To determine the period, the system designer should consider the following timings:<br>• 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.<br>• The service will also carry out the wakeup validation protocol (see *7.8 Wakeup Validation Protocol*). The smallest validation timeout typically should limit the period.<br>• 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.<br>**EcuM2656**: The service shall not be called from tasks which may invoke runnable entities. |
| ***Caveats:*** | -- |
| ***Configuration:*** | -- |

### 8.4.2 EcuM_StartupTwo

| | |
|---|---|
| ***Service name:*** | EcuM_StartupTwo |
| ***Syntax:*** | `void EcuM_StartupTwo(void)` |
| ***Service ID:*** | 26 |
| ***Sync/Async:*** | Synchronous |
| ***Reentrancy:*** | Non-Reentrant |
| ***Timing*** | once during startup |
| ***Description:*** | This function implements STARTUP II state. |
| ***Caveats:*** | -- |
| ***Configuration:*** | **EcuM2806**: This function must be called from a task which is started directly as a consequence of StartOS. I.e. either it must be called from an autostart task or it must be called from a task which is explicitely started, e.g. from an OS startup hook. |

## 8.5 Callback Definitions

### 8.5.1 Callbacks for NVRAM Manager

#### 8.5.1.1 EcuM_CB_NfyNvMJobEnd

| | |
|---|---|
| ***Service name:*** | EcuM_CB_NfyNvMJobEnd |

| | |
|---|---|
| ***Syntax:*** | `void EcuM_CB_NfyNvMJobEnd ( uint8 ServiceId, NvM_RequestResultType JobResult )` |
| ***Service ID:*** | 101 |
| ***Sync/Async:*** | Synchronous |
| ***Reentrancy:*** | Reentrant |
| ***Parameters (in):*** | `ServiceId` Unique Service ID of NVRAM manager service. |
| | `JobResult` Covers the job result of the previous processed multi block job. |
| ***Parameters (out):*** | None |
| ***Return value:*** | None |
| ***Description:*** | Used to notify about the end of NVRAM jobs initiated by EcuM<br>The callback must be callable from normal and interrupt execution contexts. |
| ***Caveats:*** | -- |
| ***Configuration:*** | NVRAM manager must be configured to call this callback as a multiple block job end notification. See [14] for details. |

## 8.6 Callout Definitions

### 8.6.1 Generic Callouts

#### 8.6.1.1 EcuM_ErrorHook

| | |
|---|---|
| ***Service name:*** | EcuM_ErrorHook |
| ***Syntax:*** | `void EcuM_ErrorHook(Std_ReturnType reason)` |
| ***Invocation*** | in all states |
| ***Reentrancy:*** | Non-Reentrant |
| ***Class:*** | Mandatory |
| ***Parameters (in):*** | reason Reason for calling the error hook |
| ***Parameters (out):*** | None |
| ***Return value:*** | None |
| ***Description:*** | In unrecoverable error situations, the ECU State Manager will call the error hook. It is up the system integrator to react accordingly (reset, halt, restart, safe state etc. ) |
| ***Caveats:*** | -- |
| ***Configuration:*** | -- |

### 8.6.2 Callouts from STARTUP

#### 8.6.2.1 EcuM_AL_DriverInitOne

| | |
|---|---|
| ***Service name:*** | EcuM_AL_DriverInitOne |
| ***Syntax:*** | `void EcuM_AL_DriverInitOne(void)` |
| ***Invocation*** | in STARTUP I |
| ***Reentrancy:*** | Non-Reentrant |
| ***Class:*** | Mandatory |

| | |
|---|---|
| ***Parameters (in):*** | None |
| ***Parameters (out):*** | None |
| ***Return value:*** | None |
| ***Description:*** | This callout shall provide driver initialization and other hardware-related startup activities in case of a power on reset. |
| ***Caveats:*** | -- |
| ***Configuration:*** | -- |

#### 8.6.2.2 EcuM_AL_DriverInitTwo

| | |
|---|---|
| ***Service name:*** | EcuM_AL_DriverInitTwo |
| ***Syntax:*** | `void EcuM_AL_DriverInitTwo(void)` |
| ***Invocation*** | in STARTUP II |
| ***Reentrancy:*** | Non-Reentrant |
| ***Class:*** | Mandatory |
| ***Parameters (in):*** | None |
| ***Parameters (out):*** | None |
| ***Return value:*** | None |
| ***Description:*** | This callout shall provide driver initialization of drivers which need OS. |
| ***Caveats:*** | -- |
| ***Configuration:*** | -- |

#### 8.6.2.3 EcuM_OnRTEStartup

| | |
|---|---|
| ***Service name:*** | EcuM_OnRTEStartup |
| ***Syntax:*** | `void EcuM_OnRTEStartup(void)` |
| ***Invocation*** | Just before calling RTE_Init |
| ***Reentrancy:*** | Non-Reentrant |
| ***Class:*** | Optional |
| ***Parameters (in):*** | None |
| ***Parameters (out):*** | None |
| ***Return value:*** | None |
| ***Description:*** | -- |
| ***Caveats:*** | -- |
| ***Configuration:*** | -- |

### 8.6.3 Callouts from RUN State

#### 8.6.3.1 EcuM_OnEnterRun

| | |
|---|---|
| ***Service name:*** | EcuM_OnEnterRun |
| ***Syntax:*** | `void EcuM_OnEnterRun(void)` |
| ***Invocation*** | On entry of RUN state. |
| ***Reentrancy:*** | Non-Reentrant |
| ***Class:*** | Optional |
| ***Parameters (in):*** | None |
| ***Parameters (out):*** | None |
| ***Return value:*** | None |
| ***Description:*** | On entry of RUN state is very similar to “just after startup”. This call allows the system designer to notify that RUN state has been reached. |
| ***Caveats:*** | -- |
| ***Configuration:*** | -- |

#### 8.6.3.2 EcuM_OnExitRun

| | |
|---|---|
| ***Service name:*** | EcuM_OnExitRun |
| ***Syntax:*** | `void EcuM_OnExitRun(void)` |
| ***Invocation*** | By EcuM_ReleaseRUN upon detection that the last run request has been released. |
| ***Reentrancy:*** | Non-Reentrant |
| ***Class:*** | Optional |
| ***Parameters (in):*** | None |
| ***Parameters (out):*** | None |
| ***Return value:*** | None |
| ***Description:*** | This call allows the system designer to notify that the APP RUN state is about to be left. |
| ***Caveats:*** | -- |
| ***Configuration:*** | -- |

### 8.6.4 EcuM_OnExitPostRun

| | |
|---|---|
| ***Service name:*** | EcuM_OnExitPostRun |
| ***Syntax:*** | `void EcuM_OnExitPostRun(void)` |
| ***Invocation*** | By EcuM_ReleasePOST_RUN upon detection that the last post run request has been released. |
| ***Reentrancy:*** | Non-Reentrant |
| ***Class:*** | Optional |
| ***Parameters (in):*** | None |
| ***Parameters*** | None |

| | |
|---|---|
| ***(out):*** | |
| ***Return value:*** | None |
| ***Description:*** | This call allows the system designer to notify that the APP POST RUN state is about to be left. |
| ***Caveats:*** | -- |
| ***Configuration:*** | -- |

### 8.6.5 Callouts from SHUTDOWN

#### 8.6.5.1 EcuM_OnPrepShutdown

| | |
|---|---|
| ***Service name:*** | EcuM_OnPrepShutdown |
| ***Syntax:*** | `void EcuM_OnPrepShutdown(void)` |
| ***Invocation*** | On entry of PREP SHUTDOWN |
| ***Reentrancy:*** | Non-Reentrant |
| ***Class:*** | Optional |
| ***Parameters (in):*** | None |
| ***Parameters (out):*** | None |
| ***Return value:*** | None |
| ***Description:*** | This call allows the system designer to notify that the PREP SHUTDOWN state is about to be entered. |
| ***Caveats:*** | -- |
| ***Configuration:*** | -- |

#### 8.6.5.2 EcuM_OnGoSleep

| | |
|---|---|
| ***Service name:*** | EcuM_OnGoSleep |
| ***Syntax:*** | `void EcuM_OnGoSleep(void)` |
| ***Invocation*** | From GO SLEEP |
| ***Reentrancy:*** | Non-Reentrant |
| ***Class:*** | Optional |
| ***Parameters (in):*** | None |
| ***Parameters (out):*** | None |
| ***Return value:*** | None |
| ***Description:*** | This call allows the system designer to notify that the GO SLEEP state is about to be entered. |
| ***Caveats:*** | -- |
| ***Configuration:*** | -- |

#### 8.6.5.3 EcuM_OnGoOffOne

| | |
|---|---|
| ***Service name:*** | EcuM_OnGoOffOne |
| ***Syntax:*** | `void EcuM_OnGoOffOne (void)` |

| | |
|---|---|
| ***Invocation*** | On entry of GO OFF I |
| ***Reentrancy:*** | Non-Reentrant |
| ***Class:*** | Optional |
| ***Parameters (in):*** | None |
| ***Parameters (out):*** | None |
| ***Return value:*** | None |
| ***Description:*** | This call allows the system designer to notify that the GO OFF I state is about to be entered. |
| ***Caveats:*** | -- |
| ***Configuration:*** | -- |

### 8.6.5.4 EcuM_OnGoOffTwo

| | |
|---|---|
| ***Service name:*** | EcuM_OnGoOffTwo |
| ***Syntax:*** | `void EcuM_OnGoOffTwo (void)` |
| ***Invocation*** | On entry of GO OFF II |
| ***Reentrancy:*** | Non-Reentrant |
| ***Class:*** | Optional |
| ***Parameters (in):*** | None |
| ***Parameters (out):*** | None |
| ***Return value:*** | None |
| ***Description:*** | This call allows the system designer to notify that the GO OFF II state is about to be entered. |
| ***Caveats:*** | -- |
| ***Configuration:*** | -- |

### 8.6.5.5 EcuM_PutWakeupSourcesToSleep

| | | |
|---|---|---|
| ***Service name:*** | EcuM_PutWakeupSourcesToSleep | |
| ***Syntax:*** | `void EcuM_PutWakeupSourcesToSleep`<br>`(`<br>`    const EcuM_SleepModeConfigType *sleepMode`<br>`)` | |
| ***Invocation*** | From GOSLEEP II, after interrupts have been disabled and ports have been set to sleep configuration | |
| ***Reentrancy:*** | Non-Reentrant | |
| ***Class:*** | Mandatory | |
| ***Parameters (in):*** | `sleepMode` | The sleep mode, for which the wakeup sources shall be configured. |
| ***Parameters (out):*** | None | |
| ***Return value:*** | None | |
| ***Description:*** | The callout shall invoke a series of wakeup source specific callouts in which the sources are configured accordingly to the selected shutdown target. See chapter *10.6.1 Managing Wakeup Sources* for details. | |

| | |
|---|---|
| | **EcuM2546**: The callout needs to derive the enable parameter for the various `EcuM_PutWKSToSleep_<Name>` callouts from configuration information. This mapping shall be code generated by the configuration tool and must match the mechanism used to generate the `EcuM_WakeupSourceType` type. See *8.2.4 EcuM_WakeupSourceType*. |
| ***Caveats:*** | -- |
| ***Configuration:*** | -- |

### 8.6.5.6 EcuM_GenerateRamHash

| | |
|---|---|
| ***Service name:*** | EcuM_GenerateRamHash |
| ***Syntax:*** | `void EcuM_GenerateRamHash(void)` |
| ***Invocation*** | Just before putting the ECU physically to sleep |
| ***Reentrancy:*** | Non-Reentrant |
| ***Class:*** | Mandatory |
| ***Parameters (in):*** | None |
| ***Parameters (out):*** | None |
| ***Return value:*** | None |
| ***Description:*** | see *8.6.6.1 EcuM_CheckRamHash* |
| ***Caveats:*** | -- |
| ***Configuration:*** | -- |

### 8.6.5.7 EcuM_AL_SwitchOff

| | |
|---|---|
| ***Service name:*** | EcuM_AL_SwitchOff |
| ***Syntax:*** | `void EcuM_AL_SwitchOff(void)` |
| ***Invocation*** | Last activity in SHUTDOWN II |
| ***Reentrancy:*** | Non-Reentrant |
| ***Class:*** | Mandatory |
| ***Parameters (in):*** | None |
| ***Parameters (out):*** | None |
| ***Return value:*** | None |
| ***Description:*** | This callout shall take the code for shutting off the power supply of the ECU. If the ECU cannot unpower itself, a reset may be an adequate reaction. |
| ***Caveats:*** | -- |
| ***Configuration:*** | -- |

## 8.6.6 Callouts from WAKEUP

### 8.6.6.1 EcuM_CheckRamHash

| | |
|---|---|
| ***Service name:*** | EcuM_CheckRamHash |
| ***Syntax:*** | `uint8 EcuM_CheckRamHash(void)` |

| | | |
|---|---|---|
| ***Invocation*** | Early in WAKEUP I | |
| ***Reentrancy:*** | Non-Reentrant | |
| ***Class:*** | Mandatory | |
| ***Parameters (in):*** | None | |
| ***Parameters (out):*** | None | |
| ***Return value:*** | 0 | RAM integrity test failed |
| | else | RAM integrity test passed |
| ***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 check does not need to be exhaustive since this would consume quite some processing time during wakeups. A well designed check will execute quickly and detect RAM integrity defects with a sufficient probability.<br>This specification does not make any assumption about the algorithm chosen for a particular ECU.<br>The areas of RAM which will be checked have to be chosen carefully. It depends on the check algorithm itself and the task structure. Stack contents of the task executing the RAM check e.g. very likely cannot be checked. It is good practice to have the hash generation and checking in the same task and that this task is not preemptible and that there is only little activity between hash generation and hash check.<br>The RAM check itself is provided by the system designer. | |
| ***Caveats:*** | -- | |
| ***Configuration:*** | -- | |

### 8.6.6.2 EcuM_AL_DriverRestart

| | |
|---|---|
| ***Service name:*** | EcuM_EcuM_AL_DriverRestart |
| ***Syntax:*** | `void EcuM_EcuM_AL_DriverRestart(void)` |
| ***Invocation*** | Early in WAKEUP I |
| ***Reentrancy:*** | Non-Reentrant |
| ***Class:*** | Mandatory |
| ***Parameters (in):*** | None |
| ***Parameters (out):*** | None |
| ***Return value:*** | None |
| ***Description:*** | This callout shall provide driver initialization and other hardware-related startup activities in the wakeup case. |
| ***Caveats:*** | -- |
| ***Configuration:*** | -- |

### 8.6.6.3 EcuM_OnWakeupReaction

| | |
|---|---|
| ***Service name:*** | EcuM_OnWakeupReaction |
| ***Syntax:*** | `EcuM_WakeupReactionType EcuM_OnWakeupReaction`<br>`(`<br>`    EcuM_WakeupReactionType wact`<br>`)` |

| | |
|---|---|
| ***Invocation*** | In WAKEUP REACTION after default computation of wakeup reaction. |
| ***Reentrancy:*** | Non-Reentrant |
| ***Class:*** | Optional |
| ***Parameters (in):*** | `wact` The wakeup reaction computed by ECU State Manager |
| ***Parameters (out):*** | None |
| ***Return value:*** | All values The desired wakeup reaction. |
| ***Description:*** | This callout gives the system designer the chance to intercept the automatic boot behavior and to override the wakeup reaction computed from wakeup source. |
| ***Caveats:*** | -- |
| ***Configuration:*** | -- |

### 8.6.7 Callouts from SLEEP State

#### 8.6.7.1 EcuM_SleepActivity

| | |
|---|---|
| ***Service name:*** | EcuM_SleepActivity |
| ***Syntax:*** | `void EcuM_SleepActivity(void)` |
| ***Invocation*** | Periodically in SLEEP state, but only if the CPU is not suspended (i.e. clock is reduced) |
| ***Reentrancy:*** | Non-Reentrant |
| ***Class:*** | Optional |
| ***Parameters (in):*** | None |
| ***Parameters (out):*** | None |
| ***Return value:*** | None |
| ***Description:*** | This callout is invoked periodically in all reduced clock sleep modes.<br>It is explicitely allowed to poll wakeup sources from this callout and to call wakeup notification functions to indicate the end of the sleep state to the ECU State Manager. |
| ***Caveats:*** | -- |
| ***Configuration:*** | The invocation period can be configured by ECUM_SLEEP_ACTIVITY_PERIOD. |

## 8.7 Expected Interfaces

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

### 8.7.1 Mandatory Interfaces

This chapter defines all interfaces which are required to fulfill the core functionality of the module.

| API function | Module | Description |
|---|---|---|
| Mcu_Init | MCU | Initialization of MCU driver |
| Mcu_GetResetReason | MCU | Determining reset reason/wakeup reason of current wakeup cycle |
| Mcu_SetMode | MCU | Change the operating mode of the MCU |
| Mcu_PerformReset | MCU | Perform a reset |
| SchM_Init | SchM | Initialization of module |
| Gpt_Init | GPT | Initialization of GPT driver |
| ComM_Init | ComM | Initialization of Communication Manager |
| ComM_DeInit | ComM | Shutdown of Communication Manager |
| ComM_EcuM_RunModeIndication | ComM | Indicates to Communication Manager that run mode has been entered |
| ComM_EcuM_WakeupIndication | ComM | Indicates to Communication Manager that a passive wakeup of a communication channel has occurred |
| WdgM_Init | WdgM | Initialization of Watchdog Manager |
| WdgM_DeInit | WdgM | Shutdown of Watchdog Manager |
| NvM_Init | NVRAM | Initialization of NVRAM Manager |
| NvM_ReadAll | NVRAM | Start reading of NVRAM contents |
| NvM_WriteAll | NVRAM | Start writing of NVRAM contents |
| NvM_CancelWriteAll | NVRAM | Cancel the active write all job |
| Dem_PreInit | DEM | Pre-Initialization of DEM |
| Dem_Init | DEM | Initialization of DEM |
| Dem_Shutdown | DEM | Shutdown of DEM |
| Dem_ReportErrorStatus | DEM | Reporting of production errors |
| Rte_Start | RTE | Startup of RTE |
| Rte_Stop | RTE | Stopping of RTE |
| Rte_Switch_<p>_<o> | RTE | Indicate a mode index to RTE |
| StartOS | OS | Startup of OS |
| ShutdownOS | OS | Shutdown of OS |
| EnableAllInterrupts | OS | Enable all interrupts |
| DisableAllInterrupts | OS | Disable all interrupts |

**Table 6 - Mandatory interfaces**

### 8.7.2 Optional Interfaces

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

| API function | Module | Description | Configuration parameter (description see chapter 10) |
|---|---|---|---|
| Det_Init | Det | Initialization of module | ECUM_DEV_ERROR_DETECT |
| Det_ReportError | Det | Development error notification | ECUM_DEV_ERROR_DETECT |
| Icu_SetMode | ICU | Set mode of ICU driver | EcuM_DriverInitList<N> |

**Table 7 - Optional Interfaces**

### 8.7.3 Configurable interfaces

There are no configurable interfaces.

## 8.8 API Parameter Checking

If development error detection is enabled for this module, then all services shall test input parameters and running conditions and use the following error codes in an adequate way:

- `ECUM_E_NOT_INITED`
- `ECUM_E_SERVICE_DISABLED`
- `ECUM_E_NULL_POINTER`
- `ECUM_E_INVALID_PAR`

Specific development errors are listed in the functions, where they do apply.

# 9 Sequence Charts

Sequence charts are presented in the flow of the specification text. The following list shows all sequence charts presented in this specification.

-
- *Figure 3 - Startup Sequence (high level* diagram)
-
- *Figure 5 - Init Sequence II (STARTUP* II)
-
- *Figure 7 - RUN State Sequence (high level* diagram)
-
- *Figure 8 - RUN II State* Sequence
-
- *Figure 9 - RUN III State* Sequence
-
- *Figure 11 - Shutdown Sequence (high level* diagram)
-
- *Figure 12 - Deinitialization Sequence I (PREP* SHUTDOWN)
-
- *Figure 13 - Deinitialization Sequence IIa* (GOSLEEP)
-
- *Figure 14 - Deinitialization Sequence IIb (GO OFF* I)
-
- *Figure 15 - Deinitialization Sequence III (GO OFF* II)
-
- *Figure 16 - Wakeup Sequence (high level* diagram)
-
- *Figure 18 - Wakeup Sequence I*
-
- *Figure 19 - Wakeup Validation* Sequence
-
- *Figure 21 - Wakeup Sequence* II

# 10 Configuration specification

## 10.1 Configuration Variants

The ECU State Manager has only one configuration variant.

## 10.2 Type definitions

The following types are only used for configuration parameters.

### 10.2.1 EcuM_LabelType

| *Type:* | String |
|---|---|
| ***Range:*** | -- |
| ***Description:*** | The type may be used to look up named configuration items. It is only used at configuration time and is therefore not part of the API. After configuration, all references to this type shall have been resolved to index numbers into value tables. |

## 10.3 Configurable Parameters

### 10.3.1 Overview

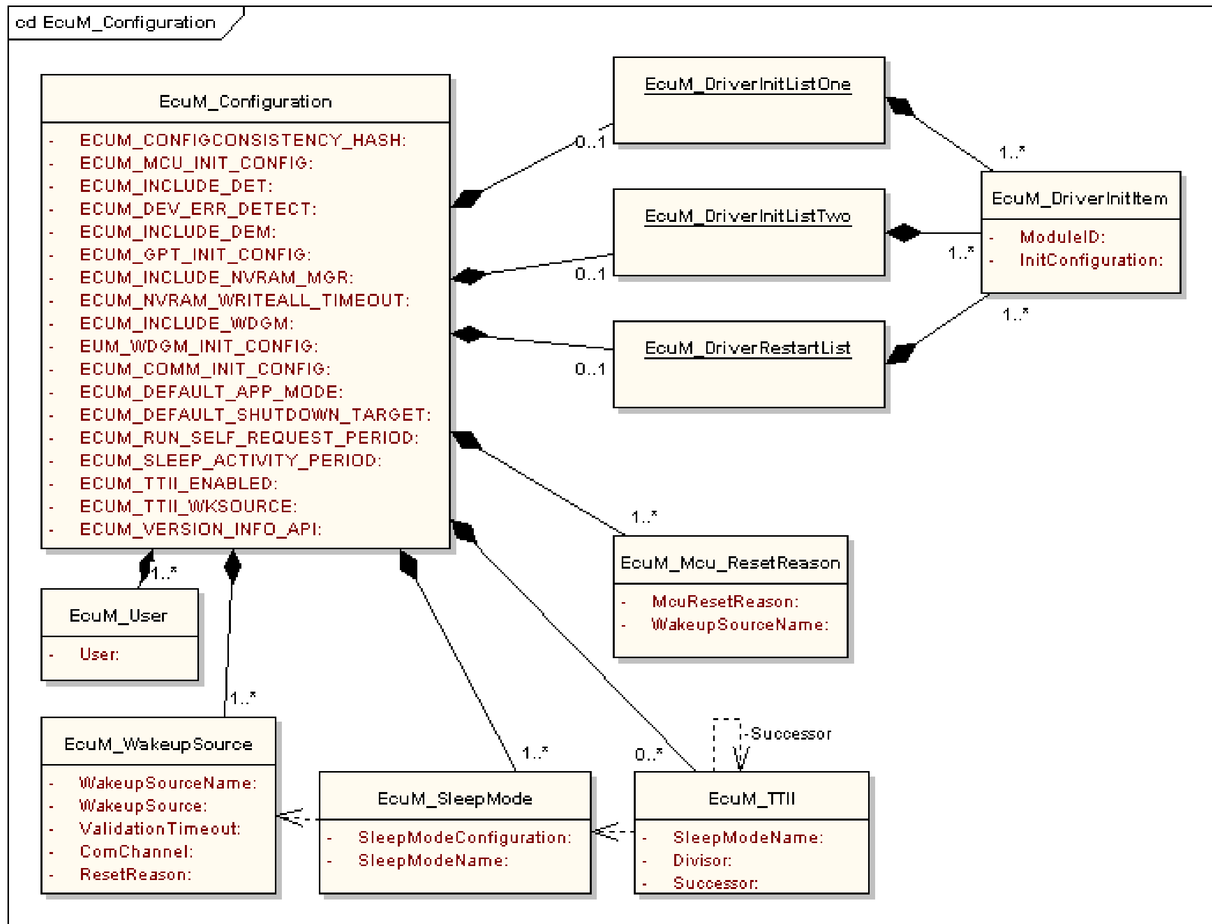


**Figure 30 - Configuration Container Diagram**

### 10.3.2 EcuM_Configuration

| ***SWS Item*** | -- |
|---|---|
| ***Container Name*** | EcuM_Configuration |
| ***Description*** | This container contains the configuration (parameters) of the ECU State Manager. |
| ***Configuration Parameters*** | |

| ***Name*** | `ECUM_CONFIGCONSISTENCY_HASH` |
|---|---|
| ***Description*** | A hash value generated across all pre-compile and link-time parameters of all BSW modules. This hash value is compared against a field in the EcuM_ConfigType and hence allows checking the consistency of the entire configuration.<br>See also *10.5 Checking Configuration Consistency.* |
| ***Type*** | `uint32` |
| ***Unit*** | -- |

| | | | |
|---|---|---|---|
| ***Range*** | -- | | |
| | -- | | |
| ***Configuration Class*** | ***Pre-compile*** | | |
| | ***Link time*** | x | |
| | ***Post Build*** | | |
| ***Scope*** | -- | | |
| ***Dependency*** | none | | |

| | | | |
|---|---|---|---|
| ***Name*** | `ECUM_MCU_INIT_CONFIG` | | |
| ***Description*** | Init configuration for the MCU Driver.<br>**EcuM2804**: The configuration shall reference the startup configuration of the MCU Driver. | | |
| ***Type*** | Reference to `Mcu_ConfigType` | | |
| ***Unit*** | -- | | |
| ***Range*** | -- | | |
| ***Configuration Class*** | ***Pre-compile*** | x | |
| | ***Link time*** | | |
| | ***Post Build*** | | |
| ***Scope*** | -- | | |
| ***Dependency*** | none | | |

| | | | |
|---|---|---|---|
| ***Name*** | `ECUM_GPT_INIT_CONFIG` | | |
| ***Description*** | Init configuration for the General Purpose Timer.<br>**EcuM2805**: The configuration shall reference the startup configuration of the General Purpose Timer. | | |
| ***Type*** | Reference to `Gpt_ConfigType` | | |
| ***Unit*** | -- | | |
| ***Range*** | -- | | |
| ***Configuration Class*** | ***Pre-compile*** | x | |
| | ***Link time*** | | |
| | ***Post Build*** | | |
| ***Scope*** | -- | | |
| ***Dependency*** | none | | |

| | | | |
|---|---|---|---|
| ***Name*** | `ECUM_INCLUDE_DET` | | |
| ***Description*** | If defined, the according BSW module will be initialized by the ECU State Manager. | | |
| ***Type*** | `Boolean` | | |
| ***Unit*** | `--` | | |
| ***Range*** | `STD_ON` | DET is initialized (default) | |
| | `STD_OFF` | DET is not initialized | |
| ***Configuration Class*** | ***Pre-compile*** | x | |
| | ***Link time*** | | |
| | ***Post Build*** | | |
| ***Scope*** | -- | | |
| ***Dependency*** | none | | |

| | |
|---|---|
| ***Name*** | `ECUM_DEV_ERROR_DETECT` |
| ***Description*** | **EcuM2456**: If false, no debug artifacts (e.g. calls to DET) shall remain in the executable object. Initialization of DET, however is controlled by configuration of optional BSW modules.<br>**EcuM2464**: The configuration tool shall report an error if *`ECUM_INCLUDE_DET`* is set to false AND *`ECUM_DEV_ERROR_DETECT`* is set to true. |
| ***Type*** | `Boolean` |

| | | | |
|---|---|---|---|
| ***Unit*** | -- | | |
| ***Range*** | STD_ON | Development errors are reported to DET (default) | |
| | STD_OFF | Development errors are not reported | |
| ***Configuration Class*** | ***Pre-compile*** | x | |
| | ***Link time*** | | |
| | ***Post Build*** | | |
| ***Scope*** | -- | | |
| ***Dependency*** | none | | |

| | | | |
|---|---|---|---|
| ***Name*** | ECUM_INCLUDE_NVRAM_MGR | | |
| ***Description*** | **EcuM2451**: If NVRAM manager is enabled but both flash and EEPROM driver are missing, then an error shall be flagged by the configuration tool. | | |
| ***Type*** | Boolean | | |
| ***Unit*** | -- | | |
| ***Range*** | STD_ON | NVRAM is initialized (default) | |
| | STD_OFF | NVRAM is not initialized | |
| ***Configuration Class*** | ***Pre-compile*** | x | |
| | ***Link time*** | | |
| | ***Post Build*** | | |
| ***Scope*** | -- | | |
| ***Dependency*** | none | | |

| | | | |
|---|---|---|---|
| ***Name*** | ECUM_INCLUDE_DEM | | |
| ***Description*** | **EcuM2452**: If enabled and NVRAM manager is disabled, then an error shall be flagged by the configuration tool. | | |
| ***Type*** | Boolean | | |
| ***Unit*** | -- | | |
| ***Range*** | STD_ON | DEM is initialized (default) | |
| | STD_OFF | DEM is not initialized | |
| ***Configuration Class*** | ***Pre-compile*** | x | |
| | ***Link time*** | | |
| | ***Post Build*** | | |
| ***Scope*** | -- | | |
| ***Dependency*** | none | | |

| | | | |
|---|---|---|---|
| ***Name*** | ECUM_INCLUDE_WDGM | | |
| ***Description*** | This configuration parameter defines whether the watchdog manager is supported by EcuM. This feature is presented for development purpose to compile out the watchdog manager in the early debugging phase. | | |
| ***Type*** | Boolean | | |
| ***Unit*** | -- | | |
| ***Range*** | STD_ON | WdgM is included (default) | |
| | STD_OFF | DEM is not included | |
| ***Configuration Class*** | ***Pre-compile*** | x | |
| | ***Link time*** | | |
| | ***Post Build*** | | |
| ***Scope*** | -- | | |
| ***Dependency*** | none | | |

| | |
|---|---|
| ***Name*** | ECUM_WDGM_INIT_CONFIG |
| ***Description*** | Init configuration for the Watchdog Manager<br>**EcuM2681**: The configuration shall reference the startup configuration |

| | | | |
|---|---|---|---|
| | of the Watchdog Manager. | | |
| ***Type*** | Reference to `WdgM_ConfigType` | | |
| ***Unit*** | -- | | |
| ***Range*** | -- | | |
| ***Configuration Class*** | ***Pre-compile*** | x | |
| | ***Link time*** | | |
| | ***Post Build*** | | |
| ***Scope*** | -- | | |
| ***Dependency*** | none | | |

| | | | |
|---|---|---|---|
| ***Name*** | `ECUM_COMM_INIT_CONFIG` | | |
| ***Description*** | Init configuration for the Communication Manager.<br>**EcuM2786**: The configuration shall reference the startup configuration of the Communication Manager. | | |
| ***Type*** | Reference to `ComM_ConfigType` | | |
| ***Unit*** | -- | | |
| ***Range*** | -- | | |
| ***Configuration Class*** | ***Pre-compile*** | x | |
| | ***Link time*** | | |
| | ***Post Build*** | | |
| ***Scope*** | -- | | |
| ***Dependency*** | none | | |

| | | | |
|---|---|---|---|
| ***Name*** | `ECUM_DEFAULT_APP_MODE` | | |
| ***Description*** | The default application mode loaded when the ECU comes out of reset. | | |
| ***Type*** | Integer (AppModeType) | | |
| ***Unit*** | -- | | |
| ***Range*** | -- | | |
| ***Configuration Class*** | ***Pre-compile*** | | |
| | ***Link time*** | | |
| | ***Post Build*** | x | |
| ***Scope*** | -- | | |
| ***Dependency*** | none | | |

| | | | |
|---|---|---|---|
| ***Name*** | `ECUM_DEFAULT_SHUTDOWN_TARGET` | | |
| ***Description*** | This shutdown target will be used during shutdown process if not overridden by the application. | | |
| ***Type*** | Structure of<br>• `EcuM_StateType`<br>• `uint8`<br>The integer number is the mode qualifier for the default target. It is an index into the *`EcuM_SleepMode`* list. See also parameters of the EcuM_SelectShutdownTarget service. | | |
| ***Unit*** | -- | | |
| ***Range*** | | | |
| ***Configuration Class*** | ***Pre-compile*** | | |
| | ***Link time*** | | |
| | ***Post Build*** | x | |
| ***Scope*** | -- | | |
| ***Dependency*** | none | | |

| | |
|---|---|
| ***Name*** | `ECUM_RUN_SELF_REQUEST_PERIOD` |
| ***Description*** | Period given in milliseconds for which the ECU State Manager will request itself to stay in RUN state when RUN state just has been |

| | | | |
|---|---|---|---|
| | entered. See *7.4.3.2 Entering RUN II* State for details. | | |
| ***Type*** | uint16 | | |
| ***Unit*** | ms | | |
| ***Range*** | -- | | |
| ***Configuration Class*** | ***Pre-compile*** | | |
| | ***Link time*** | | |
| | ***Post Build*** | x | |
| ***Scope*** | -- | | |
| ***Dependency*** | none | | |

| | | | |
|---|---|---|---|
| ***Name*** | ECUM_SLEEP_ACTIVITY_PERIOD | | |
| ***Description*** | Period of the EcuM_SleepActivity callout. The period is given in milliseconds. | | |
| ***Type*** | uint16 | | |
| ***Unit*** | ms | | |
| ***Range*** | -- | | |
| ***Configuration Class*** | ***Pre-compile*** | | |
| | ***Link time*** | | |
| | ***Post Build*** | x | |
| ***Scope*** | -- | | |
| ***Dependency*** | none | | |

| | | | |
|---|---|---|---|
| ***Name*** | ECUM_NVRAM_WRITEALL_TIMEOUT | | |
| ***Description*** | Period given in milliseconds for which the ECU State Manager will wait until it considers a write all job of the NVRAM Manager as failed | | |
| ***Type*** | uint16 | | |
| ***Unit*** | ms | | |
| ***Range*** | -- | | |
| ***Configuration Class*** | ***Pre-compile*** | | |
| | ***Link time*** | | |
| | ***Post Build*** | x | |
| ***Scope*** | -- | | |
| ***Dependency*** | none | | |

| | | | |
|---|---|---|---|
| ***Name*** | ECUM_TTII_ENABLED | | |
| ***Description*** | -- | | |
| ***Type*** | Boolean | | |
| ***Unit*** | -- | | |
| ***Range*** | STD_ON | TTII is enabled. | |
| | STD_OFF | TTII is disabled and all other TTII configuration parameters are ignored. (default) | |
| ***Configuration Class*** | ***Pre-compile*** | x | |
| | ***Link time*** | | |
| | ***Post Build*** | | |
| ***Scope*** | -- | | |
| ***Dependency*** | none | | |

| | |
|---|---|
| ***Name*** | ECUM_TTII_WKSOURCE |
| ***Description*** | This configuration parameter references the first sleep mode to be used by TTII when TTII is activated after a RUN mode.<br><br>**EcuM2785**: Whenever RUN mode is reached, the TTII protocol shall be reset to use the wakeup source referenced by this parameter. |

| | | | |
|---|---|---|---|
| | This configuration parameter is a human readable name for a TTII wakeup source which is only needed by the configuration tool. For imlementation on the ECU, this parameter may be dropped and replaced by a generated list index of EcuM_TTII (see *10.3.10 EcuM_TTII*). | | |
| ***Type*** | `EcuM_LabelType` | | |
| ***Unit*** | -- | | |
| ***Range*** | | | |
| ***Configuration Class*** | ***Pre-compile*** | x | |
| | ***Link time*** | | |
| | ***Post Build*** | | |
| ***Scope*** | -- | | |
| ***Dependency*** | none | | |

| ***Name*** | `ECUM_VERSION_INFO_API` | | |
|---|---|---|---|
| ***Description*** | Switches the version info API on or off | | |
| ***Type*** | `Boolean` | | |
| ***Unit*** | -- | | |
| ***Range*** | `STD_ON` | Version info API is available (default) | |
| | `STD_OFF` | Version info API is not available | |
| ***Configuration Class*** | ***Pre-compile*** | x | |
| | ***Link time*** | | |
| | ***Post Build*** | | |
| ***Scope*** | -- | | |
| ***Dependency*** | none | | |

| ***Included Containers*** | | | |
|---|---|---|---|
| ***Container Name*** | ***Multiplicity*** | ***Scope*** | ***Dependency*** |
| EcuM_DriverInitListOne | 0..1 | ECU | List of module IDs.<br>**EcuM2520**: A configuration tool shall fill the callout `EcuM_AL_DriverInitOne` with initialization calls to the listed drivers in the order in which they occur in the list.<br>The included container has the same structure as `EcuM_DriverInitItem` |
| EcuM_DriverInitListTwo | 0..1 | ECU | List of module IDs.<br>**EcuM2716**: A configuration tool shall fill the callout `EcuM_AL_DriverInitTwo` with initialization calls to the listed drivers in the order in which they occur in the list.<br>The included container has the same structure as `EcuM_DriverInitItem` |
| EcuM_DriverRestartList | 0..1 | ECU | List of module IDs.<br>**EcuM2719**: A configuration tool shall fill the callout `EcuM_AL_DriverRestart` with initialization calls to the listed drivers in the order in which they occur in the list.<br>**EcuM2720**: Entries in this list must |

| | | | |
|---|---|---|---|
| | | | appear in the same order as in the combined list of EcuM_DriverInitListOne and EcuM_DriverInitListTwo. This list may be a real subset though. In all other cases, the generation tool shall report an error. The included container has the same structure as `EcuM_DriverInitItem` |
| EcuM_User | 1..* | ECU | A list of identifiers that are needed to refer to a software component or another appropriate entity in the system which is designated to request the RUN state. Application requestors refer to entities above RTE, system requestors to entities below RTE (e.g. Communication Manager). |
| EcuM_WakeupSource | 1..* | ECU | List of configuration items for the wakeup sources |
| EcuM_SleepMode | 1..* | ECU | List of the configured sleep modes |
| EcuM_TTII | 0..* | ECU | List of sleep modes managed by the Time Triggered Increased Inoperation Protocol (TTII) |

### 10.3.3 EcuM_DriverInitListOne

| ***SWS Item*** | |
|---|---|
| ***Container Name*** | EcuM_DriverInitListOne |
| ***Description*** | This container contains all configuration items of driver init list one |
| ***Configuration Parameters*** | |

| ***Included Containers*** | | | |
|---|---|---|---|
| ***Container Name*** | ***Multiplicity*** | ***Scope*** | ***Dependency*** |
| EcuM_DriverInitItem | 1..n | ECU | |

### 10.3.4 EcuM_DriverInitListTwo

| ***SWS Item*** | |
|---|---|
| ***Container Name*** | EcuM_DriverInitListTwo |
| ***Description*** | This container contains all configuration items of driver init list two |
| ***Configuration Parameters*** | |

| ***Included Containers*** | | | |
|---|---|---|---|
| ***Container Name*** | ***Multiplicity*** | ***Scope*** | ***Dependency*** |
| EcuM_DriverInitItem | 1..n | ECU | |

### 10.3.5 EcuM_DriverRestartList

| SWS Item | |
|---|---|
| Container Name | EcuM_DriverRestartList |
| Description | This container contains all configuration items of driver restart list |
| Configuration Parameters | |

| Included Containers | | | |
|---|---|---|---|
| Container Name | Multiplicity | Scope | Dependency |
| EcuM_DriverInitItem | 1..n | ECU | |

### 10.3.6 EcuM_DriverInitItem

| SWS Item | |
|---|---|
| Container Name | EcuM_DriverInitItem |
| Description | This container describes a structure with the following items as its elements. EcuM_DriverInitItems are concatenated to build up a list. |
| Configuration Parameters | |

| Name | ModuleID | | |
|---|---|---|---|
| Description | An item in an init list is referenced by its module ID. | | |
| Type | String prefix | | |
| Unit | -- | | |
| Range | -- | | |
| Configuration Class | Pre-compile | x | |
| | Link time | | |
| | Post Build | | |
| Scope | -- | | |
| Dependency | none | | |

| Name | InitConfiguration | | |
|---|---|---|---|
| Description | This parameter contains pointer to the init structure of the corresponding BSW module (named Xxx_ConfigType if ModuleID is Xxx). | | |
| Type | Reference to Xxx_ConfigType | | |
| Unit | -- | | |
| Range | -- | | |
| Configuration Class | Pre-compile | x | |
| | Link time | | |
| | Post Build | | |
| Scope | -- | | |
| Dependency | The initialization of MCU and GPT is identical for all reset scenarios. | | |

### 10.3.7 EcuM_WakeupSource

| SWS Item | |
|---|---|

| **Container Name** | EcuM_WakeupSource |
|---|---|
| **Description** | This container describes the parameters for configuring wakeup source. This item can be contained several times and each item shall describe one wakeup source. The different items shall form an array. |
| **Configuration Parameters** | |

| **Name** | WakeupSourceName | |
|---|---|---|
| **Description** | A name to identify this source. | |
| **Type** | String | |
| **Unit** | -- | |
| **Range** | -- | |
| **Configuration Class** | **Pre-compile** | x |
| | **Link time** | |
| | **Post Build** | |
| **Scope** | -- | |
| **Dependency** | none | |

| **Name** | WakeupSource | |
|---|---|---|
| **Description** | The key entry. | |
| **Type** | Integer (enumeration) | |
| **Unit** | -- | |
| **Range** | ECUM_WKSOURCE_POWER | |
| | ECUM_WKSOURCE_RESET | |
| | ECUM_WKSOURCE_INTERNAL_RESET | |
| | ECUM_WKSOURCE_INTERNAL_WDG | |
| | ECUM_WKSOURCE_EXTERNAL_WDG | The external watchdog very likely cannot be determined by the MCU driver. |
| | -1 | The EcuM_WakeupSourceType is generated from this list. Every communication interface which is subject to Communication Management (e.g. CAN, FlexRay, LIN) must be assigned its own entry in the list, i.e. one entry for each CAN bus, LIN bus, etc. |
| **Configuration Class** | **Pre-compile** | x |
| | **Link time** | |
| | **Post Build** | |
| **Scope** | -- | |
| **Dependency** | none | |

| **Name** | ValidationTimeout | |
|---|---|---|
| **Description** | The validation timeout (period for which the ECU State Manager will wait for the validation of a wakeup event) can be defined for each wakeup source independently. The timeout is specified in milliseconds. | |
| **Type** | uint16 | |
| **Unit** | ms | |
| **Range** | 1..* | |
| | 0 | If no validation is needed, then the according timeout must be set to 0. |
| **Configuration Class** | **Pre-compile** | |
| | **Link time** | |
| | **Post Build** | x |

| | | | |
|---|---|---|---|
| **Scope** | -- | | |
| **Dependency** | none | | |

| | | | |
|---|---|---|---|
| **Name** | ComChannel | | |
| **Description** | Allows to look up the COM channel identification from the wakeup source and vice versa. | | |
| **Type** | EcuM_ChannelHandleType | | |
| **Unit** | -- | | |
| **Range** | 0.254 | This wakeup source also is a ComChannel. This configuration parameter describes the channel handle. | |
| | 255 (default) | This wakeup source is not a ComChannel | |
| **Configuration Class** | **Pre-compile** | x | |
| | **Link time** | | |
| | **Post Build** | | |
| **Scope** | -- | | |
| **Dependency** | none | | |

| | | | |
|---|---|---|---|
| **Name** | ResetReason | | |
| **Description** | This parameter describes the mapping of reset reasons detected by the MCU driver into ECU State Manager types | | |
| **Type** | Mcu_ResetType | | |
| **Unit** | -- | | |
| **Range** | MCU_RESET_UNDEFINED | The wakeup source is not mappable to a reset type | |
| | all other | A valid corresponding reset type | |
| **Configuration Class** | **Pre-compile** | x | |
| | **Link time** | | |
| | **Post Build** | | |
| **Scope** | -- | | |
| **Dependency** | none | | |

### 10.3.8 EcuM_User

| | |
|---|---|
| **SWS Item** | |
| **Container Name** | EcuM_User |
| **Description** | The concept of requestors is very similar - if not identical - to the concept of users in the Communication Manager specification. These two parameters should be harmonized during the configuration process. |
| **Configuration Parameters** | |

| | | | |
|---|---|---|---|
| **Name** | User | | |
| **Description** | | | |
| **Type** | EcuM_UserType | | |
| **Unit** | -- | | |
| **Range** | -- | | |
| **Configuration Class** | **Pre-compile** | x | |
| | **Link time** | | |
| | **Post Build** | | |
| **Scope** | -- | | |
| **Dependency** | none | | |

### 10.3.9 EcuM_SleepMode

| SWS Item | |
|---|---|
| Container Name | EcuM_SleepMode |
| Description | The system designer can define sleep modes with EcuM_SleepModeConfigType.<br>The list is considered ordered from 'light' sleep to 'deep' sleep. It can also be seen that modes later in the list should consume less energy.<br>See also *8.3.2.9 EcuM_SelectShutdownTarget* |
| Configuration Parameters | |

| Name | SleepModeConfiguration | |
|---|---|---|
| Description | The parameters of this sleep mode. | |
| Type | EcuM_SleepModeConfigType | |
| Unit | -- | |
| Range | -- | |
| Configuration Class | Pre-compile | x |
| | Link time | |
| | Post Build | |
| Scope | -- | |
| Dependency | none | |

| Name | SleepModeName | |
|---|---|---|
| Description | This item allows to give symbolic names to the different sleep modes. Multiple names can be given to one sleep mode. This definition is application dependent. In combination with the hardware specific *EcuM_SleepMode*, this allows to port SW-Cs across different ECUs. | |
| Type | EcuM_LabelType | |
| Unit | -- | |
| Range | -- | |
| Configuration Class | Pre-compile | x |
| | Link time | |
| | Post Build | |
| Scope | -- | |
| Dependency | none | |

### 10.3.10 EcuM_TTII

| SWS Item | |
|---|---|
| Container Name | EcuM_TTII |
| Description | This container describes a structure and the following configuration items describe its elements. This structures are concatenated to build a list as indicated by *Figure 30 - Configuration Container Diagram*.<br>**EcuM2784**: The list must contain at least on element when ECUM_TTII_ENABLED is set to true. |
| Configuration Parameters | |

| Name | SleepModeName |
|---|---|
| Description | This configuration parameter is a human readable name for a sleep mode which may only be used by the configuration tool. For imlementation on the ECU, this parameter may be dropped and |

| | | |
|---|---|---|
| | replaced by a generated list index of *EcuM_SleepMode* (see *10.3.9 EcuM_SleepMode*).<br>The SleepModeName (or the index, if more suitable for an ECU implementation) is used to reference the EcuM_SleepMode. | |
| **Type** | EcuM_LabelType | |
| **Unit** | -- | |
| **Range** | -- | |
| **Configuration Class** | **Pre-compile** | x |
| | **Link time** | |
| | **Post Build** | |
| **Scope** | -- | |
| **Dependency** | EcuM_SleepMode | |

| | | |
|---|---|---|
| **Name** | Divisor | |
| **Description** | A lookup table to find the divisor preload value. The key is the associated sleep mode. | |
| **Type** | Uint16 | |
| **Unit** | -- | |
| **Range** | 1 | |
| | max(Uint16) | |
| **Configuration Class** | **Pre-compile** | x |
| | **Link time** | |
| | **Post Build** | |
| **Scope** | -- | |
| **Dependency** | none | |

| | | |
|---|---|---|
| **Name** | Successor | |
| **Description** | This item is a reference to the next sleep mode in the TTII protocol. The representation here is human readable but may be replaced by an index for efficient implementation on an ECU. | |
| **Type** | EcuM_LabelType | |
| **Unit** | -- | |
| **Range** | -- | |
| **Configuration Class** | **Pre-compile** | x |
| | **Link time** | |
| | **Post Build** | |
| **Scope** | -- | |
| **Dependency** | none | |

## 10.4 Published Parameters

| ***SWS Item*** | | |
|---|---|---|
| ***Information elements*** | | |
| ***Information element name*** | ***Type / Range*** | ***Information element description*** |
| ECUM_VENDOR_ID | uint16 / -- | Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list |
| ECUM_MODULE_ID | uint8 / -- | Module ID of this module from Module List |
| ECUM_AR_MAJOR_VERSION | uint8 / -- | Major version number of AUTOSAR specification on which the appropriate implementation is based on. |
| ECUM_AR_MINOR_VERSION | uint8 / -- | Minor version number of AUTOSAR specification on which the appropriate implementation is based on. |
| ECUM_AR_PATCH_VERSION | uint8 / -- | Patch level version number of AUTOSAR specification on which the appropriate implementation is based on. |
| ECUM_SW_MAJOR_VERSION | uint8 / -- | Major version number of the vendor specific implementation of the module. The numbering is vendor specific. |
| ECUM_SW_MINOR_VERSION | uint8 / -- | Minor version number of the vendor specific implementation of the module. The numbering is vendor specific. |
| ECUM_SW_PATCH_VERSION | uint8 / -- | Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific. |

**Table 8 - Published parameters**

## 10.5 Checking Configuration Consistency

### 10.5.1 The Necessity for Checking Configuration Consistency in the ECU State Manager

In a AUTOSAR ECU several configuration parameters are set and put into the ECU at different times. Pre-compile parameters are set, put into the generated source code and compiled into object code. When the source code has been compiled, link-time parameters are set, compiled, and linked with the previously configured object code into an image that is put into the ECU. Finally, post-build parameters are set, compiled, linked, and put into the ECU at a different time. All these parameters must match to obtain a stable ECU.

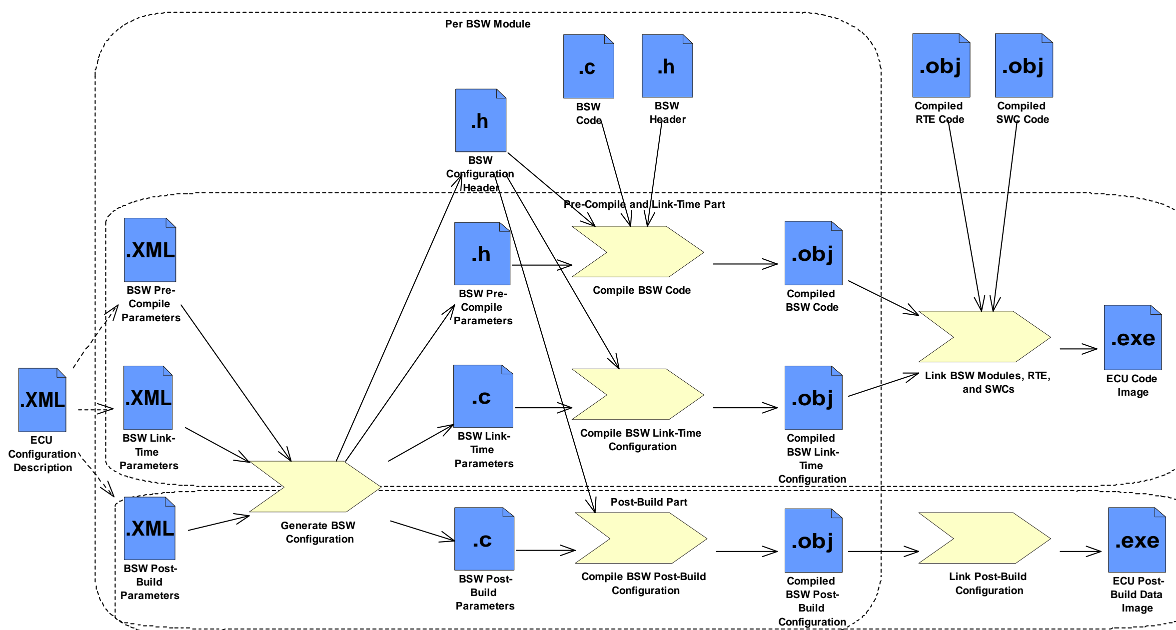


**Figure 31 - BSW Configuration Steps**

Example: The number of watchdogs to be triggered by the Watchdog Manager is set in the pre-compile parameter WDGM_*NUMBER_OF_WATCHDOG_INSTANCES*. For each of these watchdog instances the container *WdgMWatchdogInstance* contains three post-build parameters:

- *WDGM_WATCHDOG_INSTANCE_ID*,
- *WDGM_TRIGGER_SLOW_REFERENCE_CYCLE*, and
- *WDGM_TRIGGER_FAST_REFERENCE_CYCLE*.

The number of *WdgMWatchdogInstance* containers in the post-build data must exactly match the value of *WDGM_NUMBER_OF_WATCHDOG_INSTANCES*. Otherwise, wrong data will be read by the `WdgM_Init` function.

Checking consistency of parameters at configuration time can be done within the configuration tool itself. At compilation time, parameter errors may be detected by the

compiler and at link time, the linker may find additional errors. Unfortunately, finding configuration errors in post-build parameters is very difficult. This can only be achieved at run-time by checking that

- the pre-compile and link-time parameter settings used when compiling the code

are exactly the same as

- the pre-compile and link-time parameter settings used when configuring and compiling the post-build parameters.

This can only be done at run-time.

**EcuM2796**: To avoid multiple checks scattered over the different BSW modules, the ECU State Manager shall check the consistency once before initializing the first BSW module. This also implies that the ECU State Manager must not only check the consistency of its own parameters but of all post-build configurable BSW modules.

**EcuM2797**: The ECU configuration tool shall compute a hash value over all pre-compile and link-time configuration parameters of all BSW modules and put that into the link-time configuration parameter *ECUM_CONFIGCONSISTENCY_HASH.* The hash value is necessary for two reasons. First, the pre-compile and link-time parameters are not accessible anymore at run-time. Second, the check must be very efficient at run-time. Comparing hundreds of parameters would cause an unacceptable delay in the ECU startup process.

**EcuM2798**: The EcuM configuration tool shall put the current value of the configuration parameter *ECUM_CONFIGCONSISTENCY_HASH* into a field in the *EcuM_ConfigType* structure, which contains the root of all post-build configuration parameters. EcuM shall check in *EcuM_Init* that the field in the structure is equal to the value of *ECUM_CONFIGCONSISTENCY_HASH.*

By computing hash values at configuration time and comparing them at run-time the EcuM code becomes very efficient and independent of a certain hash computation algorithm. This allows for the use of complex hash computation algorithms, e.g. cryptographically strong hash functions.

Note that the same hash algorithm can be used to produce the value for the post-build configuration identifier in the *EcuM_ConfigType* structure. Then the hash algorithm is applied to the post-build parameters instead of the pre-compile and link-time parameters.

**EcuM2799**: The used hash computation algorithm shall always produce the same hash value for the same set of configuration data, regardless of the order of configuration parameters in the XML files.

### 10.5.2 Example Hash Computation Algorithm

Note: This chapter is non-normative. It describes one possible way of computing hash values.

A simple CRC over the values of configuration parameters will not serve as a good hash algorithm. It only detects global changes, e.g. one parameter has changed from 1 to 2. But if another parameter changed from 2 to 1, the CRC might stay the same.

Additionally, not only the values of the configuration parameters but also their names must be taken into account in the hash algorithm. One possibility is to build a text file that contains the names of the configuration parameters and containers, separate them from the values using a delimiter, e.g. a colon, and putting each parameter as a line into a text file. For the above Watchdog Manager example only one parameter will be included because only this one is pre-compile configured. The text file would then contain the line:

```
/WdgMConfiguration/WdgM_Trigger/WDGM_NUMBER_OF_WATCHDOG_INSTANCES:2
```

If there are multiple containers of the same type, each container name can be appended with a number, e.g. “_0”, “_1” and so on.

To make the hash value independent of the order in which the parameters are written into the text file, the lines in the file must now be sorted lexicographically.

Finally, a cryptographically strong hash function, e.g. MD5, can be run on the text file to produce the hash value. These hash functions produce completely different hash values for slightly changed input files.

## 10.6 Generated Callouts

Depending on configuration parameters, callouts will be generated which the system designer has to fill with platform dependent code.

### 10.6.1 Managing Wakeup Sources

For each wakeup source described by `EcuM_WakeupSource` one callout is generated

| ***Service name:*** | EcuM_PutWKSToSleep_<Name> | |
|---|---|---|
| ***Syntax:*** | `void EcuM_PutWKSToSleep_<Name>`<br>`(`<br>`uint8 enable,`<br>`uint8 *sleepMode`<br>`)` | |
| ***Invocation*** | From `EcuM_PutWakeupSourcesToSleep` | |
| ***Reentrancy:*** | Reentrant | |
| ***Parameters (in):*** | `enable` | 0 - the wakeup source shall be disabled<br>else - the wakeup source shall be enabled<br>**EcuM2392**: The value of this parameter shall be derived from the configured wakeup sources of the selected shutdown target (see *10.3.9 EcuM_SleepMode)*. |
| | `sleepMode` | The sleep mode for which the wakeup source shall be configured. An index like value which can be dereferenced to a sleep mode (`EcuM_SleepModeConfigType`).<br>See also *8.3.2.9 EcuM_SelectShutdownTarget*. |
| ***Parameters (out):*** | None | |
| ***Return value:*** | None | |
| ***Description:*** | **EcuM2394**: For every configured wakeup source one callout of this type shall be generated. The service name shall be generated from the <Name> of the wakeup source.<br>**EcuM2487**: The callout shall be filled with code which puts the wakeup source to sleep.<br>**EcuM2488**: Depending on the `enable` parameter, the wakeup source shall be enabled or disabled. For an enabled wakeup source the 'wakeup detected' mechanism must be re-armed. | |
| ***Caveats:*** | -- | |
| ***Configuration:*** | -- | |

# 11 Changes to Release 1

No changes, the ECU State Manager is initially released with AUTOSAR release 2.


- AUTOSAR confidential -