

<b>Document Title</b>	Specification of State Management
<b>Document Owner</b>	AUTOSAR
<b>Document Responsibility</b>	AUTOSAR
<b>Document Identification No</b>	908

<b>Document Status</b>	published
<b>Part of AUTOSAR Standard</b>	Adaptive Platform
<b>Part of Standard Release</b>	R21-11

<b>Document Change History</b>			
<b>Date</b>	<b>Release</b>	<b>Changed by</b>	<b>Description</b>
2021-11-25	R21-11	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>• Updated method name in Interface towards Update And Configuration Management</li> <li>• Added new error codes in Interface towards Update And Configuration Management</li> <li>• Fixed error handling in Interface towards Update And Configuration Management</li> <li>• Removed timeout supervision for update session</li> <li>• Removed items regarding LastResetCause in Interface towards Diagnostic Management</li> <li>• Added references from chapter 7 to chapter 9</li> </ul>

2020-11-30	R20-11	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>• Interface towards Update And Configuration Management updated</li> <li>• Interface towards Diagnostic Management updated</li> <li>• Introduced Diagnostic Reset based on Communication Groups</li> <li>• Interface towards Platform Health Management updated</li> <li>• Error reactions for supervised entity failures moved to State Management</li> <li>• Introduced PowerModes based on Communication Groups</li> <li>• RequestState and ReleaseRequest interface removed</li> </ul>
2019-11-28	R19-11	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>• Interface with ExecutionManagement changed to StateClient</li> <li>• RequestState and ReleaseRequest kept deprecated</li> <li>• Changed Document Status from Final to published</li> </ul>
2019-03-29	19-03	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>• Removed components</li> <li>• RequestState and ReleaseRequest are now deprecated</li> <li>• State Managements internal states can now be influenced by "Trigger" and are distributed by "Notifier" fields</li> </ul>
2018-10-31	18-10	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>• Initial release</li> </ul>

## **Disclaimer**

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

## Table of Contents

1	Introduction and functional overview	7
1.1	Interaction with AUTOSAR Runtime for Adaptive	7
2	Acronyms and Abbreviations	8
3	Further applicable specification	10
3.1	Input documents & related standards and norms	10
4	Constraints and assumptions	11
4.1	Known limitations	11
4.2	Applicability to car domains	11
5	Dependencies to other Functional Clusters	12
5.1	Platform dependencies	12
5.1.1	Operating System Interface	12
5.1.2	Execution Manager Interface	12
5.1.3	Platform Health Management	12
5.1.4	Diagnostic Management	12
5.1.5	Update And Configuration Management	12
5.1.6	Network Management	13
5.2	Other dependencies	13
6	Requirements Tracing	14
7	Functional specification	17
7.1	State Management Responsibilities	19
7.1.1	Machine State	20
7.1.1.1	Startup	22
7.1.1.2	Shutdown	22
7.1.1.3	Restart	23
7.1.2	Function Group State	23
7.1.3	State Management Architecture	24
7.2	State Management and Adaptive (Platform) Applications	25
7.2.1	Interaction between the SM and Adaptive Applications	25
7.2.2	Synchronization across multiple Adaptive Applications	27
7.2.2.1	PowerModes for Adaptive (Platform) Applications	29
7.2.2.2	Diagnostic Reset for Adaptive (Platform) Applications	30
7.3	Interaction with Platform Health Management	31
7.4	Interaction with Diagnostic Management	31
7.5	Interaction with Update and Configuration Management	33
7.6	Interaction with Network Management	35
7.7	Interaction with Execution Management	37
7.8	State Management in a virtualized/hierarchical environment	38
7.9	StateManagement lifecycle	39
7.9.1	Startup	39

7.9.2	Shutdown	39
7.9.3	Restart	39
7.10	Configuration	39
8	API specification	40
9	Service Interfaces	41
9.1	Type definitions	41
9.1.1	PowerMode types	41
9.1.2	DiagnosticReset types	41
9.1.3	Data types for Update And Configuration Managemet interaction	42
9.2	Provided Service Interfaces	43
9.2.1	State Management TriggerIn	43
9.2.2	State Management TriggerOut	44
9.2.3	State Management TriggerInOut	45
9.2.4	UpdateRequest	46
9.2.5	Application interaction	48
9.2.5.1	PowerMode	48
9.2.5.2	DiagnosticReset	48
9.3	Required Service Interfaces	50
9.3.1	Network Management	50
9.3.1.1	NetworkManagement NetworkState	50
9.4	Application Errors	51
9.4.1	StateManagement Error Domain	51
A	Interfunctional Cluster Interfaces	51
B	Not applicable requirements	51
C	History of Constraints and Specification Items	51
C.1	Constraint and Specification Item History of this document according to AUTOSAR Release R21-11	52
C.1.1	Added Traceables in R21-11	52
C.1.2	Changed Traceables in R21-11	53
C.1.3	Deleted Traceables in R21-11	53
C.1.4	Added Constraints in R21-11	53
C.1.5	Changed Constraints in R21-11	53
C.1.6	Deleted Constraints in R21-11	53
C.2	Constraint and Specification Item History of this document according to AUTOSAR Release R20-11	54
C.2.1	Added Traceables in R20-11	54
C.2.2	Changed Traceables in R20-11	55
C.2.3	Deleted Traceables in R20-11	55
C.2.4	Added Constraints in R20-11	55
C.2.5	Changed Constraints in R20-11	55
C.2.6	Deleted Constraints in R20-11	55

C.3	Constraint and Specification Item History of this document according to AUTOSAR Release R19-11	56
C.3.1	Added Traceables in 19-11	56
C.3.2	Changed Traceables in 19-11	56
C.3.3	Deleted Traceables in 19-11	56
C.3.4	Added Constraints in 19-11	56
C.3.5	Changed Constraints in 19-11	56
C.3.6	Deleted Constraints in 19-11	56
C.4	Constraint and Specification Item History of this document according to AUTOSAR Release R19-03	57
C.4.1	Added Traceables in 19-03	57
C.4.2	Changed Traceables in 19-03	57
C.4.3	Deleted Traceables in 19-03	57
C.4.4	Added Constraints in 19-03	58
C.4.5	Changed Constraints in 19-03	58
C.4.6	Deleted Constraints in 19-03	58

# 1 Introduction and functional overview

This document is the software specification of the [State Management](#) functional cluster within the [Adaptive Platform Services](#).

[State Management](#) is responsible for determination the state of any of its internal statemachines, based on information received from other [AUTOSAR Adaptive Platform Application](#) or [Adaptive Application](#).

[State Management](#) controls state of (partial networks using provided fields ([NetworkHandle](#)) of [Network Management](#).

[State Management](#) interacts with the [Execution Management](#) to request [Function Groups](#) and the [Machine State](#) to enter specific states that are determined by project requirements. [Function Group States](#) might additionally depend on [Network Managements State](#).

[State Management](#) provides access to its internal state via `ara::com` services. A particular service implements one of standardized service interfaces. The service interfaces have fields for getting current state (field "Notifier" (see section [9.2.2](#)) ) and requesting new state (field "Trigger" (see section [9.2.1](#))). [AUTOSAR Adaptive Platform Applications](#) or [Adaptive Applications](#) can use the fields for reacting on the system state changes or for influencing the system state(when they are configured to have write permissions).

Chapter [7](#) describes how [State Management](#) concepts are realized within the [AUTOSAR Adaptive Platform](#).

## 1.1 Interaction with AUTOSAR Runtime for Adaptive

The set of programming interfaces to the [Adaptive Applications](#) is called AUTOSAR Runtime for Adaptive (ARA). APIs accessed by [State Management](#) using the interfunctional cluster API is described in Appendix [A](#) which is not part of ARA.

The Adaptive AUTOSAR Services are provided via mechanisms provided by the [Communication Management](#) functional cluster [[1](#)] of the [Adaptive Platform Foundation](#)

## 2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant to the [State Management](#) module that are not included in the AUTOSAR glossary[2].

Terms:	Description:
State Management	The element defining modes of operation for <a href="#">AUTOSAR Adaptive Platform</a> . It allows flexible definition of functions which are active on the platform at any given time.
Execution Management [3]	The element of the <a href="#">AUTOSAR Adaptive Platform</a> responsible for the ordered startup and shutdown of the <a href="#">AUTOSAR Adaptive Platform</a> and <a href="#">Adaptive Applications</a> .
Platform Health Management [4]	A <a href="#">Functional Cluster</a> within the <a href="#">Adaptive Platform Foundation</a>
Communication Management [1]	A <a href="#">Functional Cluster</a> within the <a href="#">Adaptive Platform Foundation</a>
Network Management [5]	A <a href="#">Functional Cluster</a> within the <a href="#">Adaptive Platform Services</a> . Part of <a href="#">Communication Management</a> .
Diagnostic Management [6]	A <a href="#">Functional Cluster</a> within the <a href="#">Adaptive Platform Services</a>
Update And Configuration Management [7]	A <a href="#">Functional Cluster</a> within the <a href="#">Adaptive Platform Services</a>
Network Handle	Network Handles are provided by <a href="#">Network Management</a> . A handle represents a set of (partial) networks.
process	A <a href="#">process</a> refers to the OS concept of a running process. <b>Attention:</b> <a href="#">process</a> is <b>not equal</b> to <a href="#">Modelled Process</a> (see below). Hence each <a href="#">Modelled Process</a> has at some time a related (OS) process but a process may not always have a related <a href="#">Modelled Process</a> .
Modelled Process	A <a href="#">Modelled Process</a> is an instance of an <a href="#">Executable</a> to be executed on a <a href="#">Machine</a> and has a 1:1 association with the ARXML/Meta-Model element <a href="#">Modelled Process</a> . This document also uses the term process (without the “modelled” prefix) to refer to the OS concept of a running process.
Function Group	A <a href="#">Function Group</a> is a set of coherent <a href="#">Modelled Processes</a> which need to be controlled consistently. Depending on the state of the <a href="#">Function Group</a> , <a href="#">processes</a> (related to the <a href="#">Modelled Processes</a> ) are started or terminated. <a href="#">Modelled Processes</a> can belong to more than one <a href="#">Function Group State</a> (but at exactly one <a href="#">Function Group</a> ). "MachineFG" is a <a href="#">Function Group</a> with a predefined name, which is mainly used to control <a href="#">Machine</a> lifecycle and <a href="#">processes</a> of platform level <a href="#">Applications</a> . Other <a href="#">Function Groups</a> are sort of general purpose tools used (for example) to control <a href="#">processes</a> of user level <a href="#">Applications</a> .
Function Group State	The element of <a href="#">State Management</a> that characterizes the current status of a set of (functionally coherent) user-level <a href="#">Applications</a> . The set of <a href="#">Function Groups</a> and their <a href="#">Function Group States</a> is machine specific and are configured in the <a href="#">Machine Manifest</a> [8].
Machine State	The state of <a href="#">Function Group</a> "MachineFG" with some predefined states (Startup/Shutdown/Restart).
Execution Manifest	<a href="#">Manifest</a> file to configure execution of an <a href="#">Adaptive Application</a> .



Machine Manifest	<a href="#">Manifest</a> file to configure a <a href="#">Machine</a> . The <a href="#">Machine Manifest</a> holds all configuration information which cannot be assigned to a specific <a href="#">Executable</a> or <a href="#">process</a> .
------------------	--

**Table 2.1: Technical Terms**

The following technical terms used throughout this document are defined in the official [2] AUTOSAR Glossary or [8] TPS Manifest Specification – they are repeated here for tracing purposes.

Term	Description
Adaptive Application	see [2] AUTOSAR Glossary
Application	see [2] AUTOSAR Glossary
AUTOSAR Adaptive Platform	see [2] AUTOSAR Glossary
Adaptive Platform Foundation	see [2] AUTOSAR Glossary
Adaptive Platform Services	see [2] AUTOSAR Glossary
Manifest	see [2] AUTOSAR Glossary
Executable	see [2] AUTOSAR Glossary
Functional Cluster	see [2] AUTOSAR Glossary
Software Cluster	see [2] AUTOSAR Glossary
Diagnostic Address	see [2] AUTOSAR Glossary
Identity and Access Management	see [2] AUTOSAR Glossary
Machine	see [2] AUTOSAR Glossary
Service	see [2] AUTOSAR Glossary
Service Interface	see [2] AUTOSAR Glossary
Service Discovery	see [2] AUTOSAR Glossary

**Table 2.2: Glossary-defined Technical Terms**

## 3 Further applicable specification

### 3.1 Input documents & related standards and norms

The main documents that serve as input for the specification of the [State Management](#) are:

- [1] Specification of Communication Management  
AUTOSAR\_SWS\_CommunicationManagement
- [2] Glossary  
AUTOSAR\_TR\_Glossary
- [3] Specification of Execution Management  
AUTOSAR\_SWS\_ExecutionManagement
- [4] Specification of Platform Health Management  
AUTOSAR\_SWS\_PlatformHealthManagement
- [5] Specification of Network Management  
AUTOSAR\_SWS\_NetworkManagement
- [6] Specification of Diagnostics  
AUTOSAR\_SWS\_Diagnostics
- [7] Specification of Update and Configuration Management  
AUTOSAR\_SWS\_UpdateAndConfigurationManagement
- [8] Specification of Manifest  
AUTOSAR\_TPS\_ManifestSpecification
- [9] Requirements of State Management  
AUTOSAR\_RS\_StateManagement

## 4 Constraints and assumptions

### 4.1 Known limitations

This section lists known limitations of [State Management](#) and their relation to this release of the [AUTOSAR Adaptive Platform](#) with the intent to provide an indication how [State Management](#) within the context of the [AUTOSAR Adaptive Platform](#) will evolve in future releases.

The following functionality is mentioned within this document but is not (fully) specified in this release:

- Section [7.2](#) This document will show the basic principles of the intended functionality of [State Management](#). To enable [State Management](#) to be portable, in future versions of this document standardized fields and values shall be introduced.
- Section [7.4](#) Communication Control for Diagnostic reasons this is not yet discussed with [Diagnostic Management](#).

### 4.2 Applicability to car domains

If a superior [State Management](#) instance to the one from the ECU is available in a hierarchical car context, the [State Management](#) of the ECU shall also evaluate events generated by the superior instance of [State Management](#). Section [7.8](#) will give further details.

## 5 Dependencies to other Functional Clusters

### 5.1 Platform dependencies

#### 5.1.1 Operating System Interface

*State Management* has no direct interface to the Operating System. All OS dependencies are abstracted by the *Execution Management*.

#### 5.1.2 Execution Manager Interface

*State Management* is dependent on *Execution Management* to start and stop processes - as part of defined *Function Groups* or *Machine States*. *State Management* therefore uses the API referenced in Appendix A and defined in [3]. *State Management* additionally uses the StateClient functionality of *Execution Management* to inform *Execution Management* about *State Managements Process State*.

#### 5.1.3 Platform Health Management

*State Management* is dependent on the *Platform Health Management* [4] functional cluster. *Platform Health Management* supervises configured entities and informs *State Management* when any of these entities fails. *State Management* implements the actions needed to recover from such failed supervisions in a project specific way.

#### 5.1.4 Diagnostic Management

*State Management* is dependent on the *Diagnostic Management* [6] functional cluster. *Diagnostic Management* request different reset types for a *Diagnostic Address* at *State Management*. *State Management* implements the actions in a project specific way and prevents the system from shutting down during an active diagnostics session.

#### 5.1.5 Update And Configuration Management

*State Management* is dependent on the *Update and Configuration Management* [7] functional cluster. *Update and Configuration Management* coordinates the update sequence with *State Management* to set a set of *Function Groups*(affected by the update) to dedicated states.

### 5.1.6 Network Management

*State Management* is dependent on the *Network Management* [5] functional cluster. *Network Management* provides multiple *NetworkHandle* fields which represents a set of (partial) networks. *State Management* evaluates the *NetworkCurrentState* field to set *Function Groups* to the corresponding *Function Group State* and set the *NetworkRequestedState* field in dependency of *Function Groups* and their *Function Group State* . Additionally *State Management* shall prevent network from shutting down during an active update or diagnostic session.

## 5.2 Other dependencies

Currently, there are no other library dependencies.

## 6 Requirements Tracing

The following tables reference the requirements specified in [9] and links to the fulfillment of these. Please note that if column “Satisfied by” is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[RS_AP_00114]	C++ interface shall be compatible with C++14.	[SWS_SM_NA]
[RS_AP_00115]	Public namespaces.	[SWS_SM_91004] [SWS_SM_91007] [SWS_SM_91008] [SWS_SM_91009] [SWS_SM_91015] [SWS_SM_91017] [SWS_SM_91020]
[RS_AP_00116]	Header file name.	[SWS_SM_NA]
[RS_AP_00119]	Return values / application errors.	[SWS_SM_91010] [SWS_SM_91012] [SWS_SM_91014] [SWS_SM_91017]
[RS_AP_00120]	Method and Function names.	[SWS_SM_91017]
[RS_AP_00121]	Parameter names.	[SWS_SM_91015] [SWS_SM_91017]
[RS_AP_00122]	Type names.	[SWS_SM_91011] [SWS_SM_91012] [SWS_SM_91013] [SWS_SM_91014] [SWS_SM_91018] [SWS_SM_91019]
[RS_AP_00124]	Variable names.	[SWS_SM_NA]
[RS_AP_00125]	Enumerator and constant names.	[SWS_SM_91010] [SWS_SM_91012] [SWS_SM_91014]
[RS_AP_00127]	Usage of ara::core types.	[SWS_SM_NA]
[RS_AP_00128]	Error reporting.	[SWS_SM_NA]
[RS_AP_00129]	Public types defined by functional clusters shall be designed to allow implementation without dynamic memory allocation.	[SWS_SM_NA]
[RS_AP_00132]	noexcept behavior of API functions	[SWS_SM_NA]
[RS_AP_00133]	noexcept behavior of move and swap operations	[SWS_SM_NA]
[RS_AP_00134]	noexcept behavior of class destructors	[SWS_SM_NA]
[RS_AP_00135]	Avoidance of shared ownership.	[SWS_SM_NA]
[RS_AP_00136]	Usage of string types.	[SWS_SM_NA]
[RS_AP_00137]	Connecting run-time interface with model.	[SWS_SM_NA]
[RS_AP_00138]	Return type of asynchronous function calls.	[SWS_SM_NA]
[RS_AP_00139]	Return type of synchronous function calls.	[SWS_SM_NA]
[RS_AP_00140]	Usage of "final specifier" in ara types.	[SWS_SM_NA]
[RS_AP_00141]	Usage of out parameters.	[SWS_SM_NA]
[RS_AP_00142]	Handling of unsuccessful operations.	[SWS_SM_91010] [SWS_SM_91012] [SWS_SM_91014] [SWS_SM_91017]
[RS_AP_00143]	Use 32-bit integral types by default.	[SWS_SM_NA]
[RS_AP_00144]	Availability of a named constructor.	[SWS_SM_NA]

Requirement	Description	Satisfied by
[RS_AP_00145]	Availability of special member functions.	[SWS_SM_NA]
[RS_AP_00146]	Classes whose construction requires interaction by the ARA framework.	[SWS_SM_NA]
[RS_AP_00147]	Classes which are created by an InstanceSpecifier shall not be copyable, but at most movable.	[SWS_SM_NA]
[RS_AP_00148]	Default arguments are not allowed in virtual functions.	[SWS_SM_NA]
[RS_AP_00149]	Guidance on error handling.	[SWS_SM_91010]
[RS_AP_00150]	Provide only interfaces that are intended to be used by AUTOSAR applications and other Functional Clusters.	[SWS_SM_91001] [SWS_SM_91002] [SWS_SM_91003] [SWS_SM_91004] [SWS_SM_91007] [SWS_SM_91008] [SWS_SM_91009] [SWS_SM_91010] [SWS_SM_91011] [SWS_SM_91012] [SWS_SM_91014] [SWS_SM_91015] [SWS_SM_91016] [SWS_SM_91017] [SWS_SM_91018] [SWS_SM_91019] [SWS_SM_91020]
[RS_AP_00151]	C++ Core Guidelines.	[SWS_SM_NA]
[RS_AP_00152]	Faults inside constructor.	[SWS_SM_NA]
[RS_AP_00153]	Assignment operators should restrict "this" to lvalues	[SWS_SM_NA]
[RS_AP_00154]	Internal namespaces.	[SWS_SM_NA]
[RS_AP_00155]	Avoidance of cluster-specific initialization functions.	[SWS_SM_NA]
[RS_SM_00001]	<i>State Management</i> shall coordinate and control multiple sets of <i>Applications</i> .	[SWS_SM_00001] [SWS_SM_00005] [SWS_SM_00006] [SWS_SM_00400] [SWS_SM_00401] [SWS_SM_91011] [SWS_SM_91012] [SWS_SM_91013] [SWS_SM_91014] [SWS_SM_91015] [SWS_SM_91016] [SWS_SM_91017] [SWS_SM_91020] [SWS_SM_CONSTR_00001]
[RS_SM_00004]	<i>State Management</i> shall provide standardized interfaces.	[SWS_SM_00020] [SWS_SM_00021] [SWS_SM_00202] [SWS_SM_00204] [SWS_SM_00205] [SWS_SM_00206] [SWS_SM_00207] [SWS_SM_00208] [SWS_SM_00209] [SWS_SM_91001] [SWS_SM_91002] [SWS_SM_91003] [SWS_SM_91004] [SWS_SM_91007] [SWS_SM_91008] [SWS_SM_91009] [SWS_SM_91010] [SWS_SM_91011] [SWS_SM_91012] [SWS_SM_91013] [SWS_SM_91014] [SWS_SM_91015] [SWS_SM_91016] [SWS_SM_91017] [SWS_SM_91018] [SWS_SM_91019] [SWS_SM_91020]
[RS_SM_00005]	<i>State Management</i> internal states.	[SWS_SM_00020] [SWS_SM_00021] [SWS_SM_91001] [SWS_SM_91002] [SWS_SM_91003] [SWS_SM_91007] [SWS_SM_91008] [SWS_SM_91009]

Requirement	Description	Satisfied by
[RS_SM_00100]	State Management shall support ECU reset	[SWS_SM_00101] [SWS_SM_00103] [SWS_SM_00104] [SWS_SM_00105] [SWS_SM_00106] [SWS_SM_00107] [SWS_SM_00203] [SWS_SM_91013] [SWS_SM_91014] [SWS_SM_91015]
[RS_SM_00101]	State Management shall support diagnostic reset cause	[SWS_SM_00103] [SWS_SM_00104] [SWS_SM_00105]
[RS_SM_00200]	State Management shall provide an interface between State Management instances.	[SWS_SM_00500] [SWS_SM_00501]
[RS_SM_00300]	State Management shall support variant handling based on calibration data.	[SWS_SM_00005] [SWS_SM_00006]
[RS_SM_00400]	State Management shall establish communication paths dynamically.	[SWS_SM_00300] [SWS_SM_00301] [SWS_SM_00303] [SWS_SM_00304] [SWS_SM_91004]
[RS_SM_00401]	State Management shall control Applications depending on dynamic communication paths .	[SWS_SM_00302]



## 7 Functional specification

Please note that the semantics in the following chapter is not yet fully specified.

[State Management](#) is a functional cluster contained in the [Adaptive Platform Services](#). [State Management](#) is responsible for all aspects of [Operational State Management](#) including handling of incoming events, prioritization of these events/requests setting the corresponding internal States. Incoming events are issued when [AUTOSAR Adaptive Platform](#) or [Adaptive Applications](#) which are configured to have write access permissions change the value of "Trigger" fields provided by [State Management](#). [State Management](#) may consist of one or more state machines, which might be more or less loosely coupled depending on project needs.

Additionally the [State Management](#) takes care of not shutting down the system as long as any diagnostic or update session is active as part of [State Managements](#) internal State. [State Management](#) supervises the shutdown prevention with a project-specific timeout.

In dependency of the current internal States, [State Management](#) might decide to request [Function Groups](#) or [Machine State](#) to enter specific state by using interfaces of [Execution Management](#).

[State Management](#) is responsible for en- and disabling (partial) networks by means of [Network Management](#). [Network Management](#) provides `ara::com` fields (`NetworkHandle`) where each of the fields represents a set of (partial) networks. [State Management](#) can influence these fields in dependency of [Function Groups](#) states and - vice versa - can set [Function Groups](#) to a defined state depending on the value of [Network Managements](#) `NetworkHandle` fields.

[Adaptive Applications](#) and [AUTOSAR Adaptive Platform Applications](#) can register to the events of the "Notifier" fields provided by [State Management](#). They can change their internal behavior based on the value provided in the fields. [Adaptive Applications](#) and [AUTOSAR Adaptive Platform Applications](#) can influence the internal States of [State Management](#) by writing to the "Trigger" fields provided by [State Management](#).

This chapter describes the functional behavior of [State Management](#) and the relation to other [AUTOSAR Adaptive Platform Applications](#) [State Management](#) interacts with.

- Section [7.1](#) covers the core [State Management](#) run-time responsibilities including the start of [Applications](#).
- Section [7.2](#) describes how [Adaptive Applications](#) and [AUTOSAR Adaptive Platform Applications](#) could be influenced in their behavior based on provided "Notifier" fields of [State Management](#) and how they can influence the internal states of [State Management](#) by using provided "Trigger" fields.
- Section [7.4](#) covers several topics related to [Diagnostic Management](#) including execution of different reset types

- Section 7.5 describes how [Update and Configuration Management](#) interacts with [State Management](#)
- Section 7.6 documents support provided by [Network Management](#) to de-/activate (partial) networks in dependency of [Function Group States](#) and vice versa.
- Section 7.7 describes how [Execution Management](#) is used to change [Function Group State](#) or [Machine State](#).
- Section 7.8 provides an introduction to how [State Management](#) will work within a virtualized/hierarchical environment.

## 7.1 State Management Responsibilities

*State Management* is the functional cluster which is responsible for determining the current internal States, and for initiating *Function Group* and *Machine State* transitions by requesting them from *Execution Management*.

*State Management* is the central point where any operation event is received that might have an influence to the internal States of *State Management*. The *State Management* is responsible to evaluate these events and decide based on

- Event type (defined in project specific implementation based on project specific requirements).
- Event priority (defined in project specific implementation based on project specific requirements).
- Application identifier (Application identifier is not supported in this release. It is under discussion with FT-SEC if such an identifier could be provided by *Identity and Access Management*).

If an *State Management's* internal State change is triggered then *Execution Management* may be requested to set *Function Groups* or *Machine State* into new *Function Group State*.

The state change request for *Function Groups* can be issued by several *AUTOSAR Adaptive Platform Applications*:

- *Platform Health Management* to trigger error recovery, e.g. to activate fall-back Functionality.
- *Diagnostic Management*, to switch the system into different diagnostic states and to issue resets of the system.
- *Update and Configuration Management* to switch the system into states where software or configuration can be updated and updates can be verified.
- *Network Management* to coordinate required functionality and network state. This is no active request by *Network Management*. *Network Management* provides several sets of *NetworkHandle* fields, where *State Management* registers to and reacts on changes of these fields issued by *Network Management*.

The final decision if any effect is performed is taken by *State Management's* internal logic based on project-specific requirements.

*Adaptive Applications* may provide their own property or event via an *ara.com* interface, where the *State Management* is subscribing to, to trigger *State Management* internal events. Since *State Management* functionality is critical, access from other *Adaptive Applications* must be secured, e.g. by *Identity and Access Management*.

- *State Management* shall be monitored and supervised by *Platform Health Management*.

- [State Management](#) provides `ara::com` fields as interface to provide information about its current internal States

[State Management](#) is responsible for handling the following states:

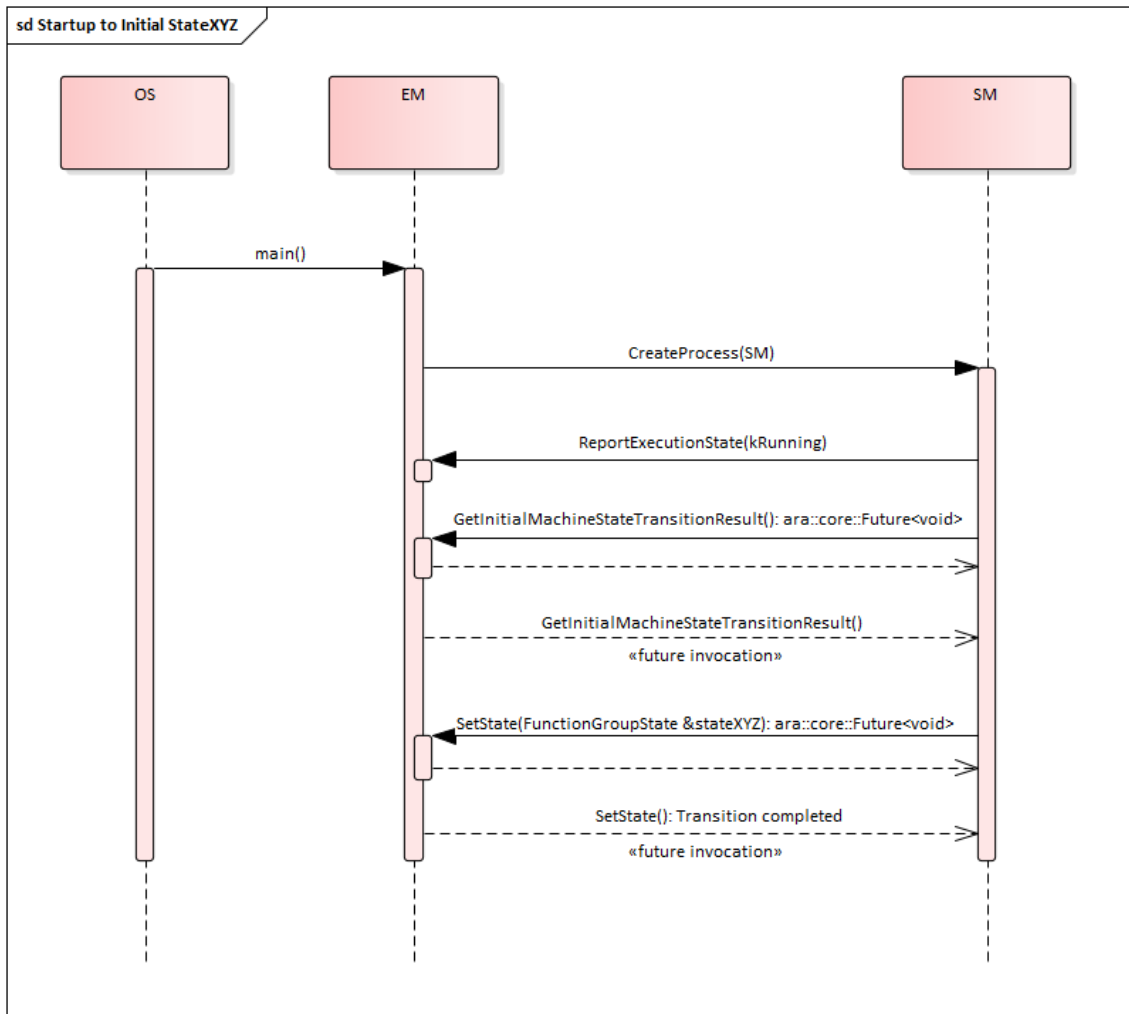
- Machine State see [7.1.1](#)
- Function Group State see [7.1.2](#)

### 7.1.1 Machine State

A [Machine State](#) is a specific type of [Function Group State](#) (see [7.1.2](#)). [Machine States](#) and all other [Function Group States](#) are determined and requested by the [State Management](#) functional cluster, see [7.1.3](#). The set of active States is significantly influenced by vehicle-wide events and modes which are evaluated into [State Managements](#) internal States.

The [Function Group States](#), including the [Machine State](#), define the current set of running [Modelled Processes](#). Each [Application](#) can declare in its [Execution Manifests](#) in which [Function Group States](#) its [Modelled Processes](#) have to be running.

The start-up sequence from initial state `Startup` to the point where [State Management](#), SM, requests the initial running machine state `Driving` is illustrated in [Figure 7.1](#) as an example `Driving` [Function Group State](#) is no mandatory [Function Group State](#).



**Figure 7.1: Start-up Sequence – from Startup to initial running state Driving**

An arbitrary state change sequence to machine state `StateXYZ` is illustrated in Figure 7.2. Here, on receipt of the state change request, `Execution Management` terminates running `Modelled Processes` and then starts `Modelled Processes` active in the new state before confirming the state change to `State Management`.

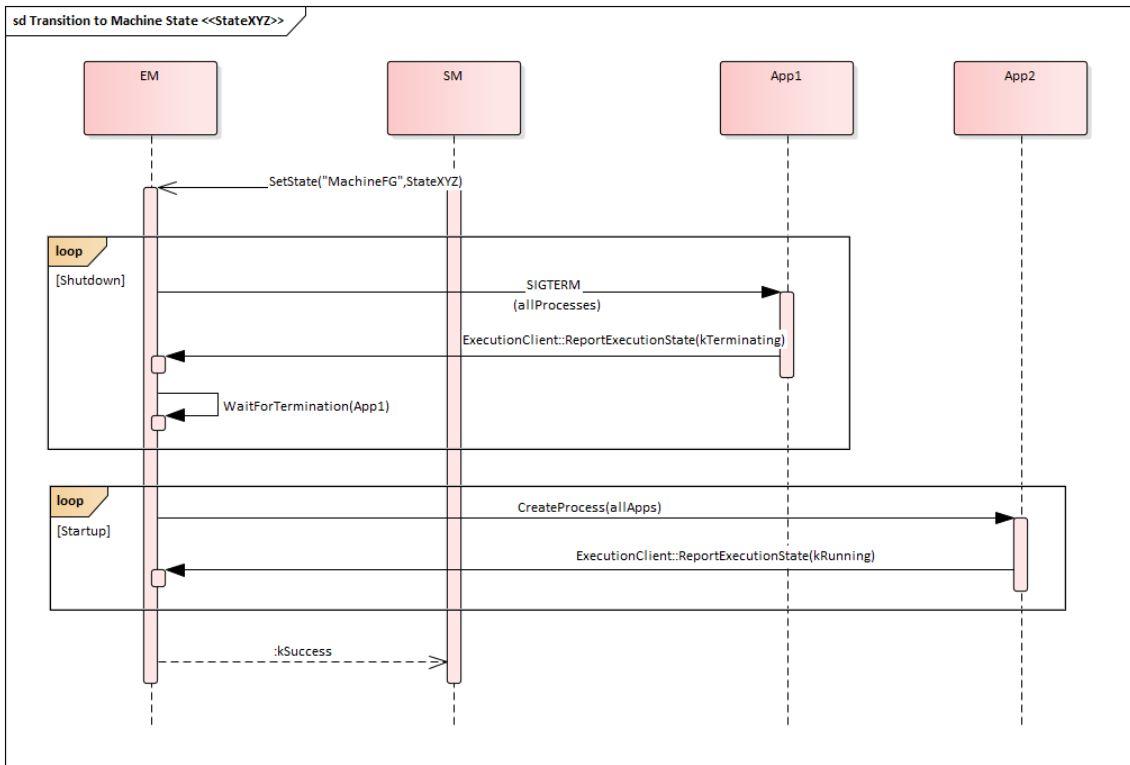


Figure 7.2: State Change Sequence – Transition to machine state StateXYZ

### 7.1.1.1 Startup

Execution Management will be controlled by State Management and therefore it should not execute any Function Group State changes on its own. This creates some expectations towards system configuration. The configuration shall be done in this way that State Management will run in every Machine State (this includes Startup, Shutdown and Restart). Above expectation is needed in order to ensure that there is always a software entity that can introduce changes in the current state of the Machine. If (for example) system integrator doesn't configure State Management to be started in Startup Machine State, then Machine will never be able transit to any other state and will be stuck forever in it. This also applies to any other Machine State state that doesn't have State Management configured.

### 7.1.1.2 Shutdown

As mentioned in 7.1.1.1 AUTOSAR assumes that State Management will be configured to run in Shutdown. State transition is not a trivial system change and it can fail for a number of reasons. When ever this happens you may want State Management to be still alive, so you can report an error and wait for further instructions. Please note that the very purpose of this state is to shutdown Machine (this includes State Management) in a clean manner. Unfortunately this means that at some point State

`Management` will no longer be available and it will not be able to report errors anymore. Those errors will be handled in a implementation specific way.

### 7.1.1.3 Restart

As mentioned in 7.1.1.1 AUTOSAR assumes that `State Management` will be configured to run in `Restart`. The reasons for doing so are the same as for 7.1.1.2.

## 7.1.2 Function Group State

If more than one group of functionally coherent `Applications` is installed on the same machine, the `Machine State` mechanism is not flexible enough to control these functional clusters individually, in particular if they have to be started and terminated with interleaving lifecycles. Many different `Machine States` would be required in this case to cover all possible combinations of active functional clusters.

To support this use case, additional `Function Groups` and `Function Group States` can be configured. Other use cases where starting and terminating individual groups of `Modelled Processes` might be necessary including diagnostic and error recovery.

In general, `Machine States` are used to control machine lifecycle (startup/shutdown/restart) and `Modelled Processes` of platform level `Applications` while other `Function Group States` individually control `Modelled Processes` which belong to groups of functionally coherent user level `Applications`.

**[SWS\_SM\_00001]{DRAFT} Available Function Group (states)** [`State Management` shall obtain available `Function Groups` and their potential states from the `Machine Manifest` to set-up the `Function Group` specific state management.] (*RS\_SM\_00001*)

`Modelled Processes` reference in their `Execution Manifest` the states in which they want to be executed. A state can be any `Function Group State`, including a `Machine State`. For details see [8], especially "Mode-dependent Startup Configuration" chapter and "Function Groups" chapter.

The arbitrary state change sequence as shown in Figure 7.2 applies to state changes of any `Function Group` - just replace "MachineState" by the name of the `Function Group`. On receipt of the state change request, `Execution Management` terminates not longer needed `Modelled Processes` and then starts `Modelled Processes` active in the new `Function Group State` before confirming the state change to `State Management`.

From the point of view of `Execution Management`, `Function Groups` are independent entities that doesn't influence each other. However from the point of view of `State Management` this may not always be the true. Let's consider a simple use

case of **Machine** shutdown. From the point of view of **Execution Management State Management** (at some point in time) will request a **Machine State** transition to **Shutdown** state. One of the **Modelled Processes** configured to run in that particular state, will initiate OS / HW shutdown and the **Machine** will power off. However from the point of view of **State Management** you will need to assess, if it's valid to request a **Machine State** transition to **Shutdown** state. Even if the assessment was positive and the **Machine** can be powered off, project specific requirements may mandate to switch all available **Function Groups** to **Off** state before we start power off sequence. For this reason we are considering existence of dependencies between **Function Groups**. Please note that currently those dependencies are implementation specific and configurable by integrator (i.e. all **Function Groups** are independent unless integrator change this).

The system might contain calibration data for variant handling. This might include that some of the **Function Groups** configured in the **Machine Manifest** are not intended to be executed on this system. therefore **State Management** has to evaluate calibration data to gather information about **Function Groups** not configured for the system variant

**[SWS\_SM\_00005]{DRAFT} Function Group Calibration Support** [**State Management** shall receive information about deactivated **Function Groups** from calibration data.](**RS\_SM\_00001**, **RS\_SM\_00300**)

The storage and reception of calibration data is implementation specific.

**[SWS\_SM\_00006]{DRAFT} Function Group Calibration Support** [**State Management** shall decline the request of **Adaptive Applications** and **AUTOSAR Adaptive Platform Applications** to change the **Function Group State** of a **Function Group** which is not configured to run in this variant.](**RS\_SM\_00001**, **RS\_SM\_00300**)

### 7.1.3 State Management Architecture

**State Management** is the functional cluster which is responsible for determining the current set of active **Function Group States**, including the **Machine State**, and for initiating State transitions by requesting them from **Execution Management**. **Execution Management** performs the State transitions and controls the actual set of running **Modelled Processes**, depending on the current States.

**State Management** is the central point where new **Function Group States** can be requested and where the requests are arbitrated, including coordination of contradicting requests from different sources. Additional data and events might need to be considered for arbitration.

**State Management** functionality is highly project specific, and AUTOSAR decided against specifying functionality like the Classic Platforms BswM for the Adaptive Platform. It is planned to only specify set of basic service interfaces, and to encapsulate the actual arbitration logic into project specific code (e.g. a library), which can



be plugged into the [State Management](#) framework and has standardized interfaces between framework and arbitration logic, so the code can be reused on different platforms.

The arbitration logic code might be individually developed or (partly) generated, based on standardized configuration parameters.

An overview of the interaction of [State Management](#), [AUTOSAR Adaptive Platform Applications](#) and [Adaptive Applications](#) is shown in Figure 7.3.

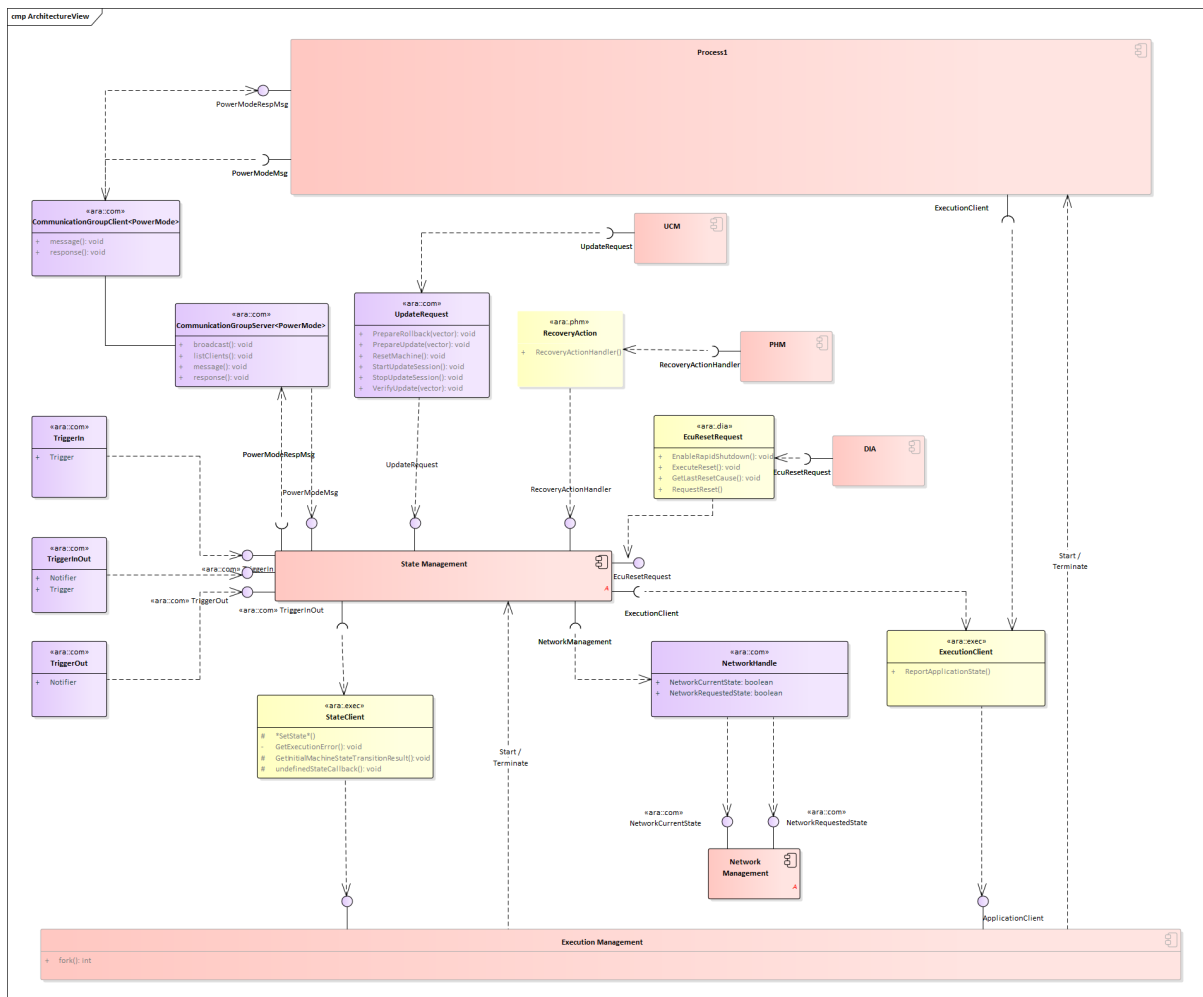


Figure 7.3: State Management Architecture

## 7.2 State Management and Adaptive (Platform) Applications

### 7.2.1 Interaction between the SM and Adaptive Applications

Some [Adaptive Applications](#), including [AUTOSAR Adaptive Platform Applications](#), might have the need to interact with [State Management](#). Therefore [State Management](#) provides a service interface `TriggerOut` with a `Notifier`

(see section 9.2.2) field, where each [Adaptive Application](#) can subscribe to, thus it is informed whenever a [State Management's](#) internal State changes. When an [Adaptive Application](#) recognizes the change it can carry out the appropriate action.

In the opposite way each [Adaptive Application](#) can influence the behavior of [State Management](#) by writing to the [Trigger](#) fields provided (as part of the service interface [TriggerIn](#)) by [State Management](#). Therefore the [Adaptive Application](#) has to be configured in a way that write access to [State Management's](#) fields is granted.

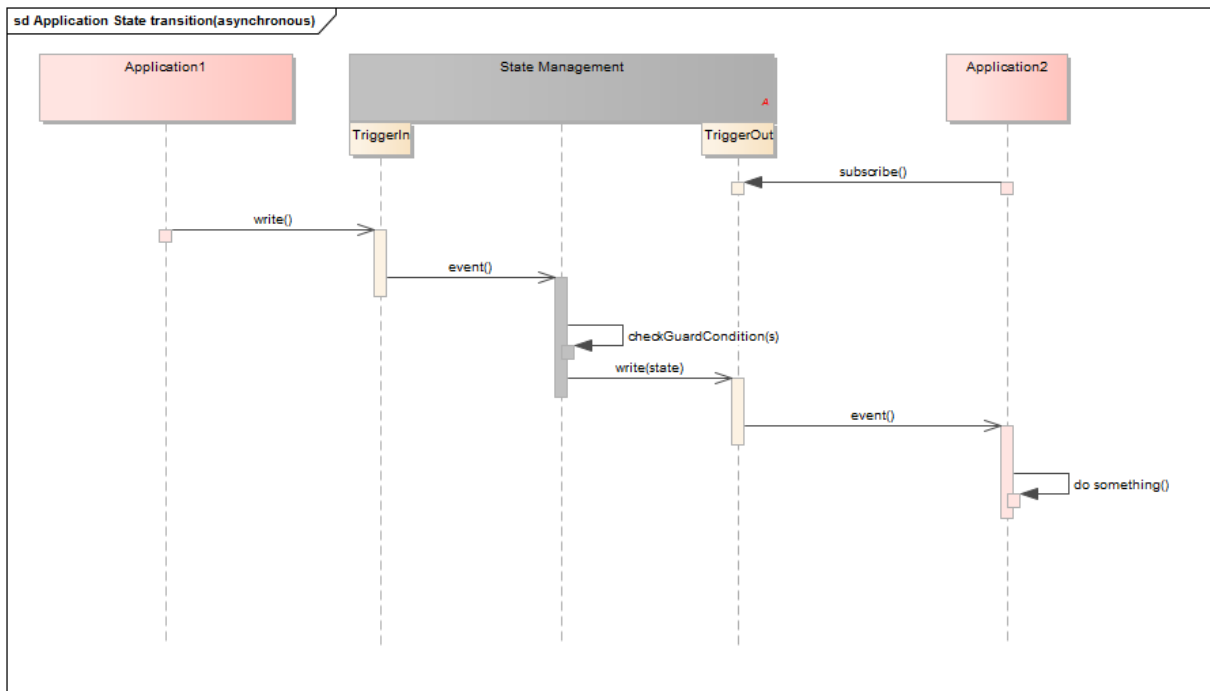
[State Management](#) provides a third service interface ([TriggerInOut](#)), where both fields are available: [Trigger](#) and [Notifier](#). This combined field is provided with the intention that whenever the [Trigger](#) field changes the [Notifier](#) field changes as well after [State Management](#) has carried out its operation issued by the [Trigger](#) change.

**[SWS\_SM\_00020] InternalState Propagation** [[State Management](#) shall support implementation of multiple instances of [TriggerOut](#) with a [Notifier](#) field which reflect [State Management's](#) internal states thus Application can get [State Management's](#) states.] ([RS\\_SM\\_00004](#), [RS\\_SM\\_00005](#))

**[SWS\_SM\_00021] InternalState Influence** [[State Management](#) shall support implementation of multiple instances of [TriggerIn](#) with a [Trigger](#) field which affect [State Management's](#) internal states thus Application can influence [State Management's](#) states.] ([RS\\_SM\\_00004](#), [RS\\_SM\\_00005](#))

Please note that the types (and therefore the content) of the provided fields are project-specific.

An overview of the interaction of [State Management](#) and [Adaptive Applications](#) for a non-synchronized behavior is shown in Figure 7.4.



**Figure 7.4: Non-Synchronized Application State handling**

## 7.2.2 Synchronization across multiple Adaptive Applications

Some scenarios in [AUTOSAR Adaptive Platform](#) might require a more sophisticated handling, where a change in [State Managements](#) internal state could only be finally carried out, when related [Modelled Processes](#) have entered a dedicated 'State', which is triggered by [State Management](#).

These triggers will be probably dedicated to a different set of Processes, depending on the functionality to be achieved. [State Management](#) sees currently two different use-cases:

- addressing all running [Modelled Processes](#) in a machine for PowerModes
- addressing running [Modelled Processes](#) for diagnostic reset reasons.

To have the possibility and flexibility to address different groups of [Modelled Processes](#) a new communication pattern called [CommunicationGroups](#) (see [SWS-CommunicationManagement \[1\]](#)) was introduced.

This pattern defines a kind of compound service with a proxy and a skeleton for the server as well as for the clients.

With this approach a server can:

- broadcast a message to all clients in the group
- send a message to a dedicated client in the group

- can get a list of all clients in the group
- receive the replies from all clients in the group

Conclusively a client can

- receive messages from the server
- send a reply to the server

Please note that it is essential, that a client replies to each server request, independently if the request could be fulfilled by the client or not.

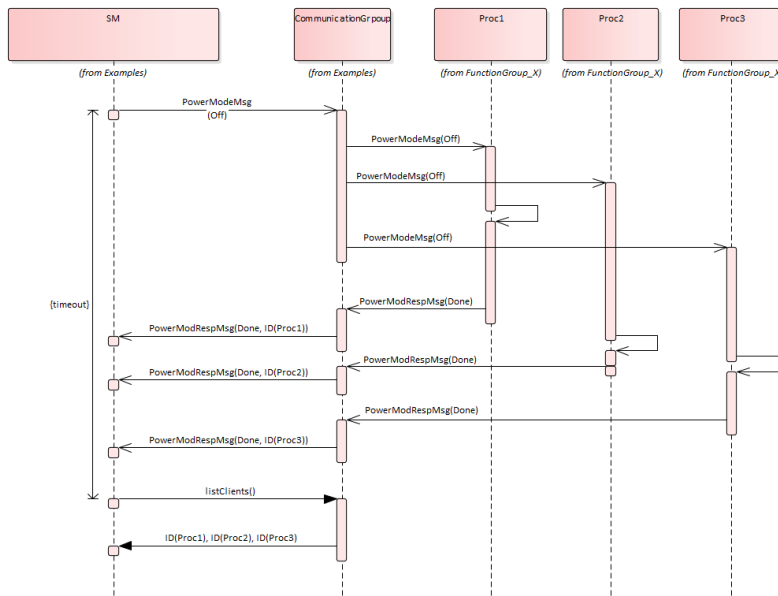
To have a unique understanding of the messages and replies these will be defined as a template and the tooling will generate corresponding proxies and skeletons.(for details see SWS-CommunicationManagement)

So now [State Management](#) as a server of (multiple) CommunicationGroups can send a message to all the clients in a group and can check if

- all clients answered the request
- all clients sent the expected answer

If any of the clients did not answer or did not reply with the expected answer [State Management](#) can retry to achieve the requested state by addressing the misbehaving client directly. When the client still does not answer(or does not answer with expected reply) [State Management](#) can do further project-specific actions. Due to the asynchronous nature of CommunicationGroups it is necessary that [State Management](#) supervises the reception of the answers from all clients with a project-specific timeout.

An overview of the interaction of [State Management](#) and [Adaptive Applications](#) for a synchronized behavior is shown in Figure 7.5.



**Figure 7.5: PowerModes as example of Synchronized Application State handling**

### 7.2.2.1 PowerModes for Adaptive (Platform) Applications

The PowerModes are intended to influence the internal behavior of all Processes in the system. Currently, there are three modes supported, but there might be more modes introduced in future releases of this document.

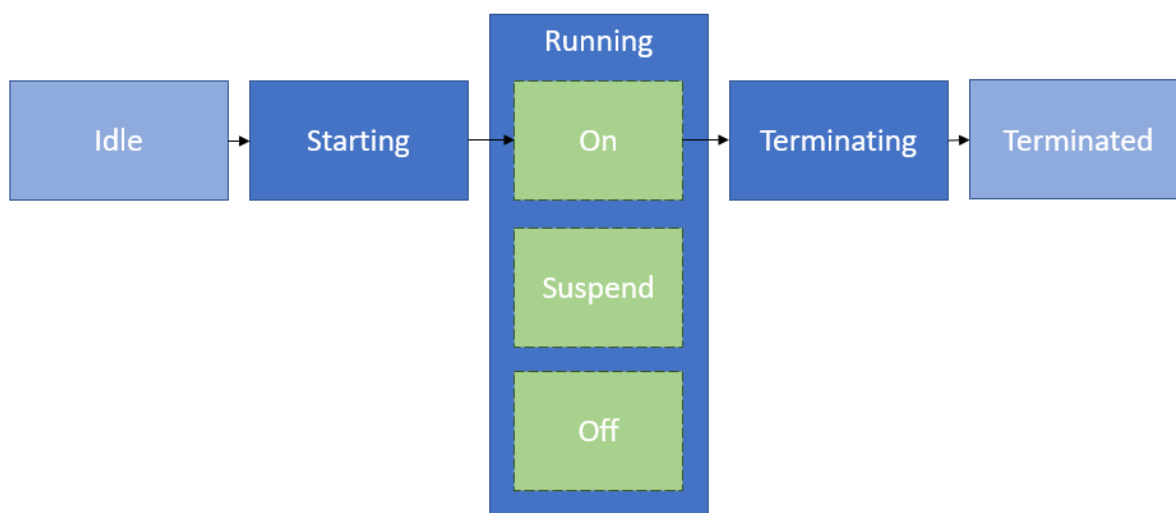
The modes are defined as follows:

- "On" : A *Modelled Process* that receives this *PowerMode* behaves normally as it has been spawned by ExecutionManagement. It is used to "undo" the other *PowerMode* requests. *Modelled Processes* that are just spawned should behave like an "On" is requested as *PowerMode*.
- "Suspend" : This *PowerMode* is intended to be used as a signal to the *Modelled Processes* that the system is suspended( e.g. to RAM or to disc). The implementation of the necessary actions(e.g. setting drivers to a proprietary mode, ...) will be project-specific and might depend on the environment(e.g. used OS).
- "Off" : A *Modelled Process* that receives this *PowerMode* behaves like it receives a SIGTERM from *Execution Management*, beside exiting.

This *PowerMode* is used to realize the so called "late-wakeup", where a new wakeup reason is found during a proceeding shutdown(e.g. short-time low voltage). When the new wakeup reason is found an "On" request will be sent to the *Modelled Processes*, thus they can immediately continue with their "normal" work without the need to be spawned again(e.g. from the filesystem). A *Modelled Process* which has just received the "Off" *PowerMode* (and carried out the necessary actions) and receives a

SIGTERM from [Execution Management](#) afterward, can perform its shutdown much faster because it has already done all the necessary steps to be prepared for exiting.

[Modelled Processes](#) that support the PowerModes are expected to behave like they would have received an "On" request when they are entering "Running" state when being spawned by [Execution Management](#) to keep compatibility with [Modelled Processes](#) which do not support the PowerModes.



**Figure 7.6: PowerModes for Adaptive (Platform) Applications**

Please note that [Modelled Processes](#) that support either "Off" or "Suspend" or both of these PowerModes support the "On" [PowerMode](#), too.

The service interface for the [PowerMode](#), the defined messages and replies can be found in [9.2.5.1 Service Interface](#) and [9.1.1 Type definition](#).

### 7.2.2.2 Diagnostic Reset for Adaptive (Platform) Applications

The Diagnostic Reset Service is provided for Diagnostic Reset functionality of [Diagnostic Management](#). The rationale behind this is to change the behavior of [Modelled Processes](#) without the need to terminate and restart them. This service is intended to influence [Modelled Processes](#) that are addressed by [Diagnostic Address](#). If all [Modelled Processes](#) or only a subset is affected depends on the system design. Therefore it is recommended to limit access to the service by IAM.

The reaction of the Adaptive (Platform) Applications to the request itself is project-specific.

Details for the complete interaction of [Diagnostic Management](#) and [State Management](#) can be found in [7.4 Interaction with Diagnostic Management](#).

The service interface for the Diagnostic Reset, the defined messages, and replies can be found in [9.2.5.2 Service Interface](#) and [9.1.2 Type definition](#).

Please note that this interface just provides means to the developer of [State Management](#) to realize the project-specific needs for Diagnostic Reset use cases.

### 7.3 Interaction with Platform Health Management

[Platform Health Management](#) is responsible for monitoring supervised entities via local supervision(s) and checking the status of health channels. Failures in local supervision(s) will be accumulated in a global supervision. The scope of a global supervision is a single [Function Group](#) (or a part of it). For details see SWS-PlatformHealthManagement[4]. As soon as a global supervision enters the stopped state or a health channel contains information that is relevant for [State Management](#), [Platform Health Management](#) will notify [State Management](#) via C++ API provided by Platform Health Manager. C++ interface is provided as a class with virtual functions, which have to be implemented by [State Management](#).

When [State Management](#) receives notification from [Platform Health Management](#) it can evaluate the information from the notification and initiate the project-specific actions to recover from the failure (e.g. request [Execution Management](#) to switch a [Function Group](#) to another [Function Group State](#), request [Execution Management](#) for a restart of the Machine, ...).

Note: [Platform Health Management](#) monitors the return of the RecoverHandler() with a configurable timeout. If after a configurable amount of retries the [State Management](#) will still not regularly return from the RecoveryHandler() [Platform Health Management](#) will do its own countermeasures by wrongly triggering or stop triggering the serviced watchdog.

### 7.4 Interaction with Diagnostic Management

[Diagnostic Management](#) is responsible for diagnosing, configuring and resetting [Diagnostic Addresses](#). The relation between a [Diagnostic Addresses](#) and a Software Cluster is project specific. The interface between [Diagnostic Management](#) and [State Management](#) is provided by [Diagnostic Management](#) as C++ API. The interface is provided as a class with virtual functions, which have to be implemented by [State Management](#).

[Diagnostic Management](#) provides the `ara::diag::EcuResetRequest` interface to forward ECU Reset service requests to [State Management](#). [State Management](#) processes the request and executes the reset of the [Diagnostic Address](#) related entity.

From [Diagnostic Management](#) point of view several different reset types have to be carried out to fulfill functionality of [Diagnostic Management](#). Because the interpretation of the reset types (defined in ISO 14229-1)

- `hardReset`

- keyOffOnReset
- softReset
- customReset

is done differently by each OEM, parts of the reset functionality have to be delegated by *State Management* to *Adaptive Applications* and AUTOSAR Adaptive Platform Applications.

A "keyOffOnReset" may be translated by *State Managements* internal logic to stop and start the *Function Group* which relate to the requested *Diagnostic Addresses*.

A "softReset" may be translated by *State Managements* internal logic to request *Modelled Processes* (within the *Function Groups* which relate to the requested *Diagnostic Address*) to perform internal functionality without the need to terminate and start them again. Therefore *State Management* provides a service interface in the scope of a *CommunicationGroup*. All *Modelled Processes* which should support this feature have to use the *ara::com* methods and fields generated from the message and reply message definition provided in 9.1.2

**[SWS\_SM\_00101]{DRAFT} Diagnostic Reset** [*State Management* shall implement means to receive reset requests for *Diagnostic Addresses* from *Diagnostic Management*. *State Management* shall carry out the project specific actions for the specific reset type.](*RS\_SM\_00100*)

This functionality is project specific. So therefore the correct mapping has to be done by the project specific code.

*State Management* is the central point in the system, where a reset for the *Machine* could be requested. So *State Management* has to keep track of reset causes and has to reset the persistent reset cause when it is newly spawned.

**[SWS\_SM\_00103]{DRAFT} Diagnostic Reset Last Cause** [*State Management* shall provide functionality to persist reset type before *Machine* reset is carried out.](*RS\_SM\_00100, RS\_SM\_00101*)

**[SWS\_SM\_00104]{DRAFT} Diagnostic Reset Last Cause Retrieval** [*State Management* shall read out the last persisted reset cause when *State Management* is spawned. This reset cause has to be provided via C++ interface towards *Diagnostic Management*.](*RS\_SM\_00100, RS\_SM\_00101*)

**[SWS\_SM\_00105]{DRAFT} Diagnostic Reset Last Cause Reset** [*State Management* shall reset the last persisted reset cause immediately after *State Management* has read out the current value.](*RS\_SM\_00100, RS\_SM\_00101*)

When *State Management* does not see any reason(project specific) to keep the machine alive any longer it will normally not shutdown the machine immediately, but will keep it alive for a configurable amount of time. Under some conditions it is needed that this waitingtime is reduced as much as possible (e.g. end of line diagnostics). This has to be supported by *State Management* too.



**[SWS\_SM\_00106]{DRAFT} Enabling of rapid shutdown** [State Management shall implement means to reduce the waitingtime to shutdown the machine as much as possible] (*RS\_SM\_00100*)

There might be reasons that *Diagnostic Management* needs to withdraw a previously enabled rapid shutdown. This usecase has to be supported by *State Management* too.

**[SWS\_SM\_00107]{DRAFT} Disabling of rapid shutdown** [State Management shall implement means to set the waitingtime to shutdown the machine to the configured value] (*RS\_SM\_00100*)

## 7.5 Interaction with Update and Configuration Management

*Update and Configuration Management* is responsible for installing, removing or updating *Software Clusters* as smallest updatable entity. To enable *Update and Configuration Management* to fulfill its functionality *State Management* offers a service interface (see 9.2.4) to be used by *Update and Configuration Management*.

Please note that system integrator has to limit usage of this interface to *Update and Configuration Management* by configuring *Identity and Access Management*.

In a first step *Update and Configuration Management* will ask *State Management* if it is allowed to perform an update. The decision will depend on current state of the machine (or whole vehicle) and has to be done in a project specific way.

**[SWS\_SM\_00203]{DRAFT} Start update session** [State Management shall provide the service interface *UpdateRequest* to *Update and Configuration Management* with the method call *RequestUpdateSession* to check if an update can be performed.] (*RS\_SM\_00100*)

As soon as *State Management* allows updating, it is necessary that *State Management* denies any further request for a new update session. To assure a higher consistency in the *AUTOSAR Adaptive Platform*, multiple update sessions at a time shall be not allowed.

**[SWS\_SM\_00209]{DRAFT} Preventing multiple update sessions** [*RequestUpdateSession* shall return *kNotAllowedMultipleUpdateSessions* in case the method *RequestUpdateSession* is called during an already active Update Session] (*RS\_SM\_00004*)

As soon as *State Management* allows updating, it is necessary that *State Management* prevents system from shutting down.

However AUTOSAR fully recognizes that there could be valid reasons to restart/shutdown machine even during an active update session (e.g. low voltage, high temperature,...). For that reasons AUTOSAR does not prevent *State Management* from

restarting/shutting down machine, but advises that such a decision should be carefully evaluated before being executed. Please note that AUTOSAR also recognizes that projects could have an arbitrary timeout restriction on the duration of the update session. This could be done for practical reasons and is allowed from the perspective of the AUTOSAR.

Additionally *State Management* has to persist the information about an ongoing update session, thus, after a machine restart (independently if restart was expected or not), *Update and Configuration Management* can continue to update. To continue the update in a consistent way it will be needed that only a few *Function Groups* will be set to a meaningful *Function Group State* (project specific). At least *Update and Configuration Management* has to be in a running state.

**[SWS\_SM\_00204]{DRAFT} Persist session status** [*State Management* shall persist information about ongoing update session, thus it can be read out after any kind of *Machine* reset.](*RS\_SM\_00004*)

In some cases it is needed that *Update and Configuration Management* issues a reset of the *Machine* (expected reset), e.g. when *Functional Clusters* like *State Management*, *Platform Health Management* or *Execution Management* are affected by the update. This has to be supported by *State Management*. At least this might be simply implemented by requesting *Machine State* restart from *Execution Management*.

**[SWS\_SM\_00202]{DRAFT} Reset Execution** [*State Management* shall implement the service interface *UpdateRequest* to *Update and Configuration Management* with the method call *ResetMachine* to request a *Machine* reset.](*RS\_SM\_00004*)

*Update and Configuration Management* has to inform *State Management* when no more operations for the update have to be done, thus *State Management* can clear now the information about an ongoing update and can continue its regular job. Please note, that all *State Management* activities after the *StopUpdateSession* is requested are fully project specific, like setting the impacted *Function Groups* into a meaningful *Function Group State*.

**[SWS\_SM\_00205]{DRAFT} Stop update session** [*State Management* shall provide the service interface *UpdateRequest* to *Update and Configuration Management* with the method call *StopUpdateSession* thus it can inform *State Management* that the update session is finished.](*RS\_SM\_00004*)

During the update there will be up to three different steps, depending if a *Software Cluster* is installed, removed or updated. If and when the steps are done depends additionally on the success or fail of the previous steps. To support *Update and Configuration Management* to request these steps *State Management* provides three different methods as part of the service interface *UpdateRequest*.

**[SWS\_SM\_00206]{DRAFT} prepare update** [*State Management* shall provide the service interface *UpdateRequest* to *Update and Configuration Management*

with the method call `PrepareUpdate` thus it can request `State Management` to perform a preparation of the given `Function Groups` to be updated.](*RS\_SM\_00004*)

**[SWS\_SM\_00207]{DRAFT} prepare verify** [`State Management` shall provide the service interface `UpdateRequest` to `Update and Configuration Management` with the method call `VerifyUpdate` thus it can request `State Management` to perform a verification of the given `Function Groups`.](*RS\_SM\_00004*)

**[SWS\_SM\_00208]{DRAFT} prepare rollback** [`State Management` shall provide the service interface `UpdateRequest` to `Update and Configuration Management` with the method call `PrepareRollback` thus it can request `State Management` to perform a preparation of the given `Function Groups` to be rolled back.](*RS\_SM\_00004*)

For updating a `Software Cluster Update` and `Configuration Management` will call the method `PrepareUpdate` (as part of the service interface `UpdateRequest`) in a first step. `State Management` will at least set all the `Function Groups`, given as parameter, to `Off` state. In next step `Update and Configuration Management` will perform the real update (e.g. exchange executable, change manifests,...). As following step `Update and Configuration Management` uses the `VerifyUpdate` to request `State Management` to perform a verification of the update. Therefore `State Management` will at least set all the `Function Groups`, given as parameter, to `Verify` state. These request will be reported to `Update and Configuration Management` as failed when any of the `Function Groups` could not be set to the requested `Function Group State`. A failure will also be reported when one of these functions is called, before `State Management` granted the right to update.

When any of these steps fails, `Update and Configuration Management` can decide to revert previous changes. Therefore `Update and Configuration Management` uses `PrepareRollback` function, where `State Management` will at least set all the `Function Groups`, given as parameter, to `Off` state.

When a `Software Cluster` is removed by `Update and Configuration Management`, `VerifyUpdate` will never be called by `Update and Configuration Management`. Contrary to that `PrepareUpdate` will never be called, when a new `Software Cluster` is installed into the `Machine`.

For more detail about the update process see sequence diagrams and descriptions in [7].

## 7.6 Interaction with Network Management

To be portable between different ECUs the `Adaptive Applications` should not have the need to know which networks are needed to fulfill its functionality, because on different ECUs the networks could be configured differently. To control the availability of

networks for several [Adaptive Applications State Management](#) interacts with [Network Management](#) via a service interface.

[Network Management](#) provides multiple instances of [NetworkHandles](#), where each represents a set of (partial) networks.

The [NetworkHandles](#) are defined in the [Machine Manifest](#) and are there assigned to a [Function Group State](#).

An overview of the interaction of [State Management](#), [Network Management](#) and [Adaptive Applications](#) is shown in [Figure 7.8](#).

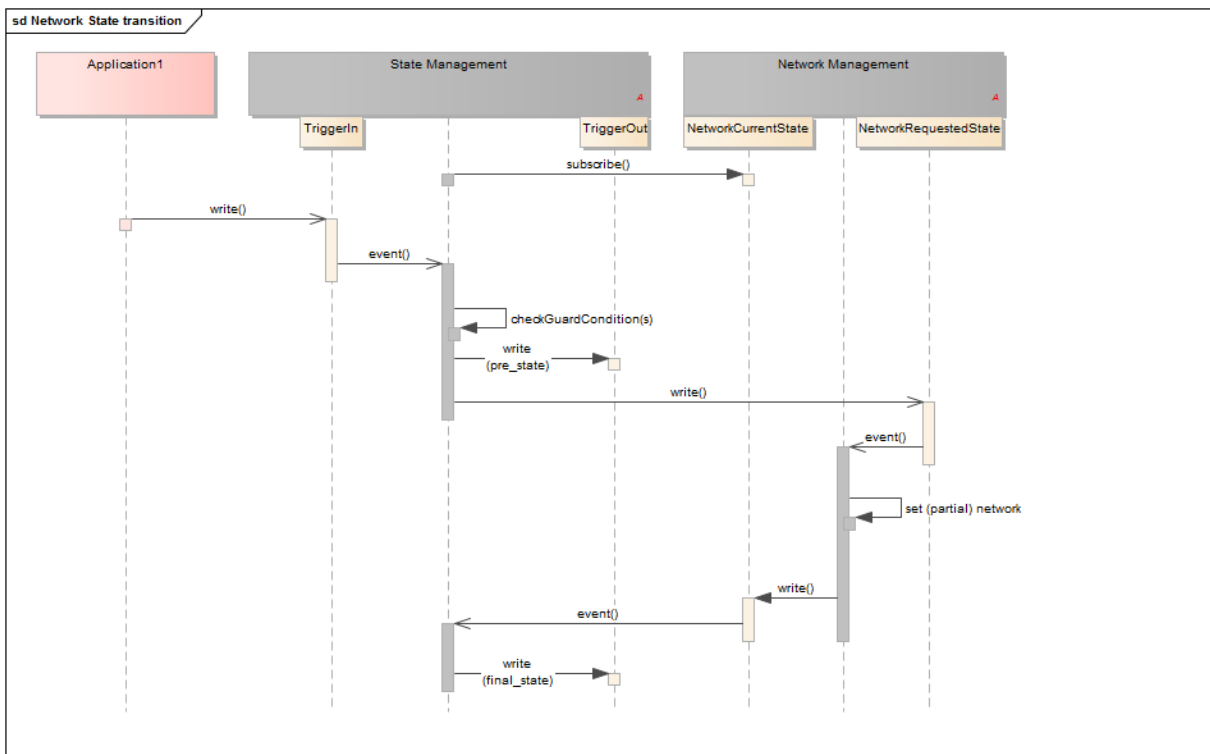


Figure 7.7: Switching Network State by "Trigger"

**[SWS\_SM\_00300]{DRAFT} NetworkHandle Configuration** [[State Management](#) shall receive information about [NetworkHandles](#) and their associated [Function Group States](#) from [Machine Manifest](#).] ([RS\\_SM\\_00400](#))

Whenever (partial) networks are activated or deactivated from outside request and this set of (partial) networks is represented by a [NetworkHandle](#) in [Machine Manifest](#) [Network Management](#) will change the value of the corresponding [NetworkHandle](#). [State Management](#) is notified about the change, because it has registered to all available [NetworkHandle](#) fields. When [State Management](#) recognizes a change in a field's value it sets the corresponding [Function Group](#) in the [Function Group State](#) where the [NetworkHandle](#) is configured for in the [Machine Manifest](#).

**[SWS\_SM\_00301]{DRAFT} NetworkHandle Registration** [[State Management](#) shall register for all [NetworkHandles](#) provided by [Network Managements](#) which are available from [Machine Manifest](#).] ([RS\\_SM\\_00400](#))

**[SWS\_SM\_00302]{DRAFT} NetworkHandle to FunctionGroupState** [State Management shall set Function Groups to the corresponding Function Group State which is configured in the Machine Manifest for the NetworkHandle when it recognizes a change in NetworkHandle value.](RS\_SM\_00401)

Vice versa State Managements shall change the value of the NetworkHandle when a Function Group has to change its Function Group State and an association between this Function Group State and the Network handle is available in Machine Manifest. Network Management will recognize this change and will change the state of the (partial) networks accordingly to the NetworkHandle.

**[SWS\_SM\_00303]{DRAFT} FunctionGroupState to NetworkHandle** [State Management shall change the value of NetworkHandle when Function Groups changes its Function Group State and a NetworkHandle is associated to this Function Group State in the Machine Manifest.](RS\_SM\_00400)

It might be needed that a Function Group stays longer in its Function Group State when the causing (partial) network set has been switched off or a (partial) network is longer available than the causing Function Group has been switched to Function Group State 'Off'. This is called 'afterrun'. The corresponding timeout-value has to be configured in Machine Manifest

**[SWS\_SM\_00304]{DRAFT} Network Afterrun** [State Management shall support means to support 'afterrun' to switch off related Function Groups or (partial) networks. The timeout value for this 'afterrun' has to be read from e.g. Machine Manifest.](RS\_SM\_00400)

## 7.7 Interaction with Execution Management

Execution Management is used to execute the Function Group State changes. The decision to change the state of Machine State or the Function Group State of Function Groups might come from inside of State Management based on State Managements States (or other project specific requirements) or might be requested at State Management from an external Adaptive Application.

An overview of the interaction of State Management, Execution Management and Adaptive Applications is shown in Figure 7.8.

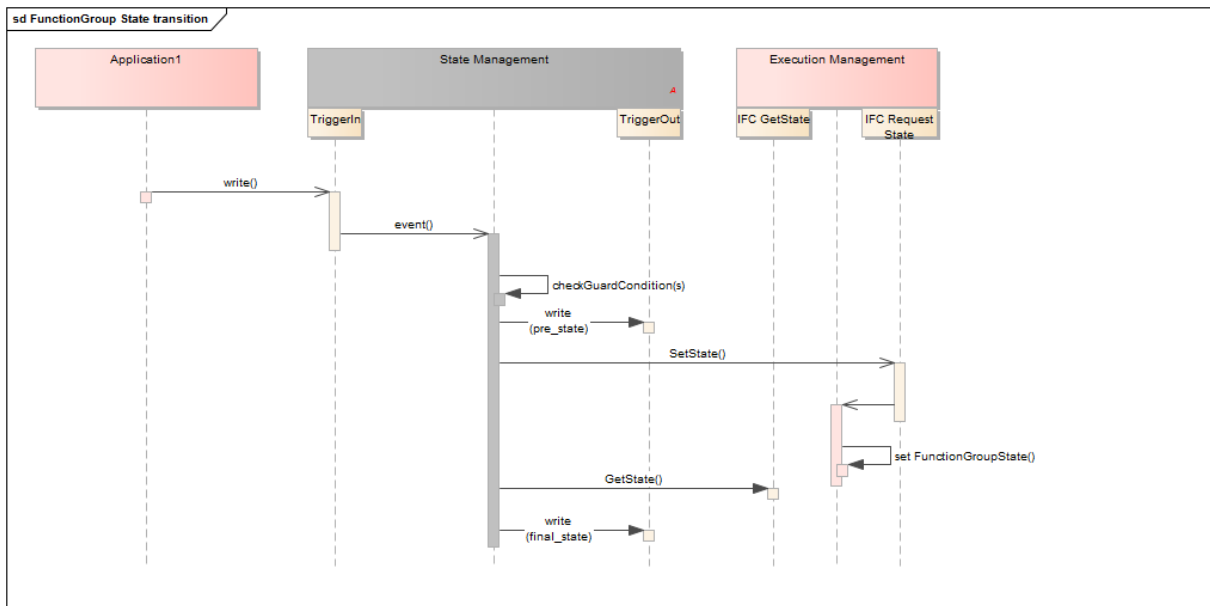


Figure 7.8: Switching FunctionGroup State by "Trigger"

**[SWS\_SM\_00400]{DRAFT} Execution Management** [State Management shall use API of Execution Management to change the state of Machine State or Function Group State of Function Groups.](RS\_SM\_00001)

Execution Management might not be able to carry out the requested Function Group State change due to several reasons (e.g. corrupted binary). Execution Management returns the result of the request.

**[SWS\_SM\_00401]{DRAFT} Execution Management Results** [State Management shall evaluate the results of request to Execution Management. Based on the results State Management may do project-specific actions](RS\_SM\_00001)

## 7.8 State Management in a virtualized/hierarchical environment

On an ECU several machines might run in a virtualized environment. Each of the virtual machines might contain an AUTOSAR Adaptive platform. So therefore each of the virtual machines contain State Management. To have coordinated control over the several virtual machines there has to be virtual machine which supervises the whole ECU state. This is not only valid for a virtualized environment, but for a hierarchical environment, too.

**[SWS\_SM\_00500]{DRAFT} Virtualized/hierarchical State Management** [State Management shall be able to register to the "Trigger" fields of a supervising State Management instance to receive information about the whole ECU state.](RS\_SM\_00200)

**[SWS\_SM\_00501]{DRAFT} Virtualized/hierarchical State Management internal State** [State Management shall implement means to calculate its internal States

based on information from a supervising [State Management](#) instance.]([RS\\_SM\\_00200](#))

## 7.9 StateManagement lifecycle

### 7.9.1 Startup

State management lifecycle fully depends on machine state. Details can be found in [7.1.1.1](#)

### 7.9.2 Shutdown

State management lifecycle fully depends on machine state. Details can be found in [7.1.1.2](#)

### 7.9.3 Restart

State management lifecycle fully depends on machine state. Details can be found in [7.1.1.3](#)

## 7.10 Configuration

State Management should be configured to run in every Machine State (this includes Startup, Shutdown and Restart) other than Off. This expectation is needed to ensure that there is always a software entity that can introduce changes in the current state of the Machine. If (for example) the system integrator does not configure State Management to be started in Startup Machine State, then Machine will never be able transit to any other state and will be stuck forever in it.

**[SWS\_SM\_CONSTR\_00001]{DRAFT} Existence of State Management** [At least one [Modelled Process](#) with `Process.functionClusterAffinity` with the value `STATE_MANAGEMENT` shall be configured to run in each MachineFG state except Off, whenever one such [Modelled Process](#) is configured to run in MachineFG state Startup.]([RS\\_SM\\_00001](#))

## 8 API specification

[State Management](#) does not provide any API. All functional interfaces will be found in Chapter [9](#) Service Interfaces.



## 9 Service Interfaces

### 9.1 Type definitions

#### 9.1.1 PowerMode types

[SWS\_SM\_91011]{DRAFT} [

<b>Name</b>	PowerModeMsg	
<b>Kind</b>	STRING	
<b>Derived from</b>	-	
<b>Description</b>	Message to all running Processes in the system to indicate a request for a PowerMode switch	
<b>Range / Symbol</b>	<b>Limit</b>	<b>Description</b>
On	'On'	normal operation.
Off	'Off'	persist data preparation for shutdown.
Suspend	'Suspend'	prepare for suspend2ram.

]([RS\\_SM\\_00004](#), [RS\\_SM\\_00001](#), [RS\\_AP\\_00150](#), [RS\\_AP\\_00122](#))

[SWS\_SM\_91012]{DRAFT} [

<b>Name</b>	PowerModeRespMsg	
<b>Kind</b>	VALUE	
<b>Derived from</b>	-	
<b>Description</b>	Reply message from Process, which received PowerModeMessage from State Management	
<b>Range / Symbol</b>	<b>Limit</b>	<b>Description</b>
kDone	0	requested mode sucessfully reached.
kFailed	1	requested mode not reached.
kBusy	2	cant process requested mode e.g. important things are ongoing.
kNotSupported	3	requested mode not supported.

]([RS\\_SM\\_00004](#), [RS\\_SM\\_00001](#), [RS\\_AP\\_00150](#), [RS\\_AP\\_00122](#), [RS\\_AP\\_00142](#), [RS\\_AP\\_00119](#), [RS\\_AP\\_00125](#))

#### 9.1.2 DiagnosticReset types

[SWS\_SM\_91013]{DRAFT} [

<b>Name</b>	DiagnosticResetMsg	
<b>Kind</b>	STRING	
<b>Derived from</b>	-	





<b>Description</b>	Message to all Processes(in a SoftwareCluster) to indicate a request to perform Diagnostic SoftReset	
<b>Range / Symbol</b>	<b>Limit</b>	<b>Description</b>
SoftReset	'SoftReset'	normal operation.

](RS\_SM\_00001, RS\_SM\_00004, RS\_SM\_00100, RS\_AP\_00122)

[SWS\_SM\_91014]{DRAFT} [

<b>Name</b>	DiagnosticResetRespMsg	
<b>Kind</b>	VALUE	
<b>Derived from</b>	-	
<b>Description</b>	Reply message from Process, which received DiagnosticResetMessage from State Management	
<b>Range / Symbol</b>	<b>Limit</b>	<b>Description</b>
kDone	0	reset performed sucessfully.
kFailed	1	reset not sucessfully performed.
kBusy	2	can't perform reset(e.g. important things are ongoing).
kNotSupported	3	reset not supported.

](RS\_SM\_00001, RS\_SM\_00004, RS\_SM\_00100, RS\_AP\_00150, RS\_AP\_00122, RS\_AP\_00142, RS\_AP\_00119, RS\_AP\_00125)

### 9.1.3 Data types for Update And Configuration Managemet interaction

[SWS\_SM\_91018]{DRAFT} [

<b>Name</b>	FunctionGroupListType
<b>Kind</b>	VECTOR
<b>Subelements</b>	<a href="#">FunctionGroupNameType</a>
<b>Derived from</b>	-
<b>Description</b>	A list of FunctionGroups.

](RS\_SM\_00004, RS\_AP\_00150, RS\_AP\_00122)

[SWS\_SM\_91019]{DRAFT} [

<b>Name</b>	FunctionGroupNameType
<b>Kind</b>	STRING
<b>Derived from</b>	-
<b>Description</b>	full qualified FunctionGroup shortName.

](RS\_SM\_00004, RS\_AP\_00150, RS\_AP\_00122)

## 9.2 Provided Service Interfaces

### 9.2.1 State Management TriggerIn

Port

[SWS\_SM\_91001]{DRAFT} [

<b>Name</b>	TriggerIn_{State}		
<b>Kind</b>	ProvidedPort	<b>Interface</b>	TriggerIn
<b>Description</b>	To be used by Adaptive (Platform) Applications to trigger State Management to change its internal state.		
<b>Variation</b>			

]([RS\\_SM\\_00004](#), [RS\\_SM\\_00005](#), [RS\\_AP\\_00150](#))

Service Interface

[SWS\_SM\_91007] [

<b>Name</b>	TriggerIn_{StateGroup}
<b>NameSpace</b>	ara::sm
<b>Field</b>	Trigger
<b>Description</b>	Value to be evaluated by State Management in a projectspecific way.
<b>Type</b>	project_specific
<b>HasGetter</b>	false
<b>HasNotifier</b>	false
<b>HasSetter</b>	true

]([RS\\_SM\\_00004](#), [RS\\_SM\\_00005](#), [RS\\_AP\\_00150](#), [RS\\_AP\\_00115](#))

## 9.2.2 State Management TriggerOut

Port

[SWS\_SM\_91002]{DRAFT} [

<b>Name</b>	TriggerOut_{State}		
<b>Kind</b>	ProvidedPort	<b>Interface</b>	<a href="#">TriggerOut</a>
<b>Description</b>	To be used by Adaptive (Platform) Applications to be informed when State Management has changed its internal state.		
<b>Variation</b>			

]([RS\\_SM\\_00004](#), [RS\\_SM\\_00005](#), [RS\\_AP\\_00150](#))

Service Interface

[SWS\_SM\_91008] [

<b>Name</b>	TriggerOut_{StateGroup}
<b>NameSpace</b>	ara::sm

<b>Field</b>	Notifier
<b>Description</b>	To be set by State Management in a projectspecific way to inform Adaptive (Platform) Applications about changes within State Management
<b>Type</b>	project_specific
<b>HasGetter</b>	true
<b>HasNotifier</b>	true
<b>HasSetter</b>	false

]([RS\\_SM\\_00004](#), [RS\\_SM\\_00005](#), [RS\\_AP\\_00150](#), [RS\\_AP\\_00115](#))

### 9.2.3 State Management TriggerInOut

Port

[SWS\_SM\_91003]{DRAFT} [

<b>Name</b>	TriggerInOut_{State}		
<b>Kind</b>	ProvidedPort	<b>Interface</b>	<a href="#">TriggerInOut</a>
<b>Description</b>	To be used by Adaptive (Platform) Applications to trigger State Management to change its internal state and to get information when it is carried out.		
<b>Variation</b>			

]([RS\\_SM\\_00004](#), [RS\\_SM\\_00005](#), [RS\\_AP\\_00150](#))

Service Interface

[SWS\_SM\_91009]{DRAFT} [

<b>Name</b>	TriggerInOut_{StateGroup}
<b>NameSpace</b>	ara::sm

<b>Field</b>	Trigger
<b>Description</b>	Value to be evaluated by State Management in a projectspecific way.
<b>Type</b>	project_specific
<b>HasGetter</b>	false
<b>HasNotifier</b>	false
<b>HasSetter</b>	true

<b>Field</b>	Notifier
<b>Description</b>	To be set by State Management in a projectspecific way to inform Adaptive (Platform) Applications about changes within State Management
<b>Type</b>	project_specific
<b>HasGetter</b>	true
<b>HasNotifier</b>	true
<b>HasSetter</b>	false

]([RS\\_SM\\_00004](#), [RS\\_SM\\_00005](#), [RS\\_AP\\_00150](#), [RS\\_AP\\_00115](#))

## 9.2.4 UpdateRequest

The `UpdateRequest` interface is intended to be used by `Update and Configuration Management` to interact with `State Management` to perform updates (including installation and removal) of `Software Clusters`.

Port

[SWS\_SM\_91016]{DRAFT} [

<b>Name</b>	UpdateRequest		
<b>Kind</b>	ProvidedPort	<b>Interface</b>	<a href="#">UpdateRequest</a>
<b>Description</b>	To be used by Update And Configuration Management to request State Management to perform steps for updating SoftwareClusters.		
<b>Variation</b>			

]([RS\\_SM\\_00001](#), [RS\\_SM\\_00004](#), [RS\\_AP\\_00150](#))

Service Interface

[SWS\_SM\_91017]{DRAFT} [

<b>Name</b>	UpdateRequest		
<b>NameSpace</b>	ara::sm		

<b>Method</b>	ResetMachine		
<b>Description</b>	Requests a reset of the machine. Before the reset is performed all information within the machine shall be persisted. Request will be rejected when RequestUpdateSession was not called successfully before.		
<b>FireAndForget</b>	false		
<b>Application Errors</b>	<a href="#">kRejected</a>	Requested operation was rejected due to State Managements/machines internal state.	

<b>Method</b>	StopUpdateSession		
<b>Description</b>	Has to be called by Update And Configuration Management once the update is finished to let State Management know that the update is done and the Machine is in a stable state. Request will be rejected when RequestUpdateSession was not called successfully before.		
<b>FireAndForget</b>	false		
<b>Application Errors</b>	<a href="#">kRejected</a>	Requested operation was rejected due to State Managements/machines internal state.	

<b>Method</b>	RequestUpdateSession		
<b>Description</b>	Has to be called by Update And Configuration Management once it has to start interaction with State Management. State Management might decline this request when machine is not in a state to be updated.		
<b>FireAndForget</b>	false		
<b>Application Errors</b>	<a href="#">kRejected</a>	Requested operation was rejected due to State Managements/machines internal state.	
<b>Application Errors</b>	<a href="#">kNotAllowedMultipleUpdateSessions</a>	Request for new session was rejected as only single active (update) session is allowed.	

<b>Method</b>	PrepareUpdate	
<b>Description</b>	Has to be called by Update And Configuration Management after State Management allowed to update. State Management will decline this request when RequestUpdateSession was not called before successfully.	
<b>FireAndForget</b>	false	
<b>Parameter</b>	functionGroupList	
	<b>Description</b>	The list of FunctionGroups within the SoftwareCluster to be prepared to be updated.
	<b>Type</b>	functionGroupListType
	<b>Variation</b>	
	<b>Direction</b>	IN
<b>Application Errors</b>	kRejected	Requested operation was rejected due to State Managements/machines internal state.
<b>Application Errors</b>	kPrepare- Failed	Preparation step of update failed.

<b>Method</b>	VerifyUpdate	
<b>Description</b>	Has to be called by Update And Configuration Management after State Management allowed to update and the update preparation has been done. State Management will decline this request when Prepare Update was not called before successfully.	
<b>FireAndForget</b>	false	
<b>Parameter</b>	functionGroupList	
	<b>Description</b>	The list of FunctionGroups within the SoftwareCluster to be verified.
	<b>Type</b>	functionGroupListType
	<b>Variation</b>	
	<b>Direction</b>	IN
<b>Application Errors</b>	kRejected	Requested operation was rejected due to State Managements/machines internal state.
<b>Application Errors</b>	kVerify- Failed	Verification step of update failed.

<b>Method</b>	PrepareRollback	
<b>Description</b>	Has to be called by Update And Configuration Management after State Management allowed to update.	
<b>FireAndForget</b>	false	
<b>Parameter</b>	functionGroupList	
	<b>Description</b>	The list of FunctionGroups within the SoftwareCluster to be prepared to roll back.
	<b>Type</b>	functionGroupListType
	<b>Variation</b>	
	<b>Direction</b>	IN
<b>Application Errors</b>	kRejected	Requested operation was rejected due to State Managements/machines internal state.
<b>Application Errors</b>	kRollback- Failed	Rollback step of update failed.

|(RS\_SM\_00001, RS\_SM\_00004, RS\_AP\_00150, RS\_AP\_00115, RS\_AP\_00120, RS\_AP\_00142, RS\_AP\_00119, RS\_AP\_00121)

## 9.2.5 Application interaction

Application interface is intended to be used by every [Adaptive Application](#) to enable StateManagement to achieve a synchronized behavior of all applications

### 9.2.5.1 PowerMode

Service Interface

[SWS\_SM\_91020]{DRAFT} [

<b>Name</b>	PowerMode
<b>NameSpace</b>	ara::sm

<b>Method</b>	message	
<b>Description</b>	sends PowerModeMsg defined in 9.1 Type definition to all Processes to request a PowerMode.	
<b>Parameter</b>	msg	
	<b>Description</b>	Message to all running Processes in the system to indicate a request to enter this state.
	<b>Type</b>	<a href="#">PowerModeMsg</a>
	<b>Variation</b>	
<b>Direction</b>	OUT	

<b>Method</b>	event	
<b>Description</b>	All Processes which got a PowerMode request sends this as answer to State Management	
<b>Parameter</b>	respMsg	
	<b>Description</b>	ResponseMessage from a Processes which received PowerMode request from State Management.
	<b>Type</b>	<a href="#">PowerModeRespMsg</a>
	<b>Variation</b>	
<b>Direction</b>	OUT	

]([RS\\_SM\\_00001](#), [RS\\_SM\\_00004](#), [RS\\_AP\\_00150](#), [RS\\_AP\\_00115](#))

### 9.2.5.2 DiagnosticReset

Service Interface

[SWS\_SM\_91015]{DRAFT} [

<b>Name</b>	DiagnosticReset
<b>NameSpace</b>	ara::sm



<b>Method</b>	message	
<b>Description</b>	sends DiagnosticResetMsg defined in 9.1 Type definition to all Processes in a SoftwareCluster.	
<b>Parameter</b>	msg	
	<b>Description</b>	Message to all running Processes in the SoftwareCluster to indicate a request to perform softReset.
	<b>Type</b>	<a href="#">DiagnosticResetMsg</a>
	<b>Variation</b>	
	<b>Direction</b>	OUT

<b>Method</b>	event	
<b>Description</b>	All Processes which got a DiagnosticReset request sends this as answer to State Management	
<b>Parameter</b>	respMsg	
	<b>Description</b>	ResponseMessage from a Processes which received DiagnosticReset request from State Management.
	<b>Type</b>	<a href="#">DiagnosticResetRespMsg</a>
	<b>Variation</b>	
	<b>Direction</b>	OUT

|(RS\_SM\_00001, RS\_SM\_00004, RS\_SM\_00100, RS\_AP\_00150, RS\_AP\_00115, RS\_AP\_00121)

## 9.3 Required Service Interfaces

### 9.3.1 Network Management

#### 9.3.1.1 NetworkManagement NetworkState

Port

[SWS\_SM\_91004]{DRAFT} [

<b>Name</b>	NetworkState_{NetworkHandle}		
<b>Kind</b>	RequiredPort	<b>Interface</b>	NetworkState
<b>Description</b>	Provides information about network status per NetworkHandle. Intended to be only used by State Management!		
<b>Variation</b>	<b>FOR</b> NetworkHandle : MODEL.filterType("NetworkHandle");		

](RS\_SM\_00004, RS\_SM\_00400, RS\_AP\_00150, RS\_AP\_00115)

## 9.4 Application Errors

This chapter lists all errors of [State Management](#)

### 9.4.1 StateManagement Error Domain

[SWS\_SM\_91010]{DRAFT} [

Name	Code	Description
kNotAllowedMultipleUpdateSessions	9	Request for new session was rejected as only single active (update) session is allowed.
kPrepareFailed	7	Preparation step of update failed.
kRejected	5	Requested operation was rejected due to State Managements/ machines internal state.
kRollbackFailed	8	Rollback step of update failed.
kVerifyFailed	6	Verification step of update failed.

]([RS\\_SM\\_00004](#), [RS\\_AP\\_00150](#), [RS\\_AP\\_00125](#), [RS\\_AP\\_00142](#), [RS\\_AP\\_00119](#), [RS\\_AP\\_00149](#))

## A Interfunctional Cluster Interfaces

No IFC-Interfaces are provided by [State Management](#).

## B Not applicable requirements

[SWS\_SM\_NA]{DRAFT} **Not applicable requirements** [These requirements are not applicable as they are not within the scope of this release.]([RS\\_AP\\_00132](#), [RS\\_AP\\_00134](#), [RS\\_AP\\_00133](#), [RS\\_AP\\_00153](#), [RS\\_AP\\_00144](#), [RS\\_AP\\_00152](#), [RS\\_AP\\_00145](#), [RS\\_AP\\_00146](#), [RS\\_AP\\_00147](#), [RS\\_AP\\_00127](#), [RS\\_AP\\_00143](#), [RS\\_AP\\_00129](#), [RS\\_AP\\_00135](#), [RS\\_AP\\_00136](#), [RS\\_AP\\_00137](#), [RS\\_AP\\_00140](#), [RS\\_AP\\_00148](#), [RS\\_AP\\_00155](#), [RS\\_AP\\_00128](#), [RS\\_AP\\_00114](#), [RS\\_AP\\_00151](#), [RS\\_AP\\_00154](#), [RS\\_AP\\_00116](#), [RS\\_AP\\_00124](#), [RS\\_AP\\_00141](#), [RS\\_AP\\_00138](#), [RS\\_AP\\_00139](#))

## C History of Constraints and Specification Items

Please note that the lists in this chapter also include constraints and specification items that have been removed from the specification in a later version. These constraints and specification items do not appear as hyperlinks in the document.

## C.1 Constraint and Specification Item History of this document according to AUTOSAR Release R21-11

### C.1.1 Added Traceables in R21-11

Number	Heading
[SWS_SM_00001]	Available Function Group (states)
[SWS_SM_00005]	Function Group Calibration Support
[SWS_SM_00006]	Function Group Calibration Support
[SWS_SM_00020]	InternalState Propagation
[SWS_SM_00021]	InternalState Influence
[SWS_SM_00101]	Diagnostic Reset
[SWS_SM_00103]	Diagnostic Reset Last Cause
[SWS_SM_00104]	Diagnostic Reset Last Cause Retrieval
[SWS_SM_00105]	Diagnostic Reset Last Cause Reset
[SWS_SM_00106]	Enabling of rapid shutdown
[SWS_SM_00107]	Disabling of rapid shutdown
[SWS_SM_00202]	Reset Execution
[SWS_SM_00203]	Start update session
[SWS_SM_00204]	Persist session status
[SWS_SM_00205]	Stop update session
[SWS_SM_00206]	prepare update
[SWS_SM_00207]	prepare verify
[SWS_SM_00208]	prepare rollback
[SWS_SM_00209]	Preventing multiple update sessions
[SWS_SM_00300]	NetworkHandle Configuration
[SWS_SM_00301]	NetworkHandle Registration
[SWS_SM_00302]	NetworkHandle to FunctionGroupState
[SWS_SM_00303]	FunctionGroupState to NetworkHandle
[SWS_SM_00304]	Network Afterrun
[SWS_SM_00400]	Execution Management
[SWS_SM_00401]	Execution Management Results
[SWS_SM_00500]	Virtualized/hierarchical State Management
[SWS_SM_00501]	Virtualized/hierarchical State Management internal State
[SWS_SM_91001]	
[SWS_SM_91002]	
[SWS_SM_91003]	
[SWS_SM_91004]	
[SWS_SM_91007]	



△

Number	Heading
[SWS_SM_91008]	
[SWS_SM_91009]	
[SWS_SM_91010]	
[SWS_SM_91011]	
[SWS_SM_91012]	
[SWS_SM_91013]	
[SWS_SM_91014]	
[SWS_SM_91015]	
[SWS_SM_91016]	
[SWS_SM_91017]	
[SWS_SM_91018]	
[SWS_SM_91019]	
[SWS_SM_91020]	
[SWS_SM_- CONSTR_00001]	Existence of State Management
[SWS_SM_NA]	Not applicable requirements

**Table C.1: Added Traceables in R21-11**

### C.1.2 Changed Traceables in R21-11

none

### C.1.3 Deleted Traceables in R21-11

none

### C.1.4 Added Constraints in R21-11

none

### C.1.5 Changed Constraints in R21-11

none

### C.1.6 Deleted Constraints in R21-11

none

## C.2 Constraint and Specification Item History of this document according to AUTOSAR Release R20-11

### C.2.1 Added Traceables in R20-11

Number	Heading
[SWS_SM_00001]	Available Function Group (states)
[SWS_SM_00005]	Function Group Calibration Support
[SWS_SM_00006]	Function Group Calibration Support
[SWS_SM_00020]	InternalState Propagation
[SWS_SM_00021]	InternalState Influence
[SWS_SM_00100]	Prevent Shutdown due to Diagnostic Session
[SWS_SM_00101]	Diagnostic Reset
[SWS_SM_00103]	Diagnostic Reset Last Cause
[SWS_SM_00104]	Diagnostic Reset Last Cause Retrieval
[SWS_SM_00105]	Diagnostic Reset Last Cause Reset
[SWS_SM_00200]	Prevent Shutdown during to Update Session
[SWS_SM_00201]	Supervision of Shutdown Prevention
[SWS_SM_00202]	Reset Execution
[SWS_SM_00203]	Start update session
[SWS_SM_00204]	Persist session status
[SWS_SM_00205]	Stop update session
[SWS_SM_00206]	prepare update
[SWS_SM_00207]	prepare verify
[SWS_SM_00208]	prepare rollback
[SWS_SM_00300]	NetworkHandle Configuration
[SWS_SM_00301]	NetworkHandle Registration
[SWS_SM_00302]	NetworkHandle to FunctionGroupState
[SWS_SM_00303]	FunctionGroupState to NetworkHandle
[SWS_SM_00304]	Network Afterrun
[SWS_SM_00400]	Execution Management
[SWS_SM_00401]	Execution Management Results
[SWS_SM_00402]	Function Group State Change Results
[SWS_SM_00500]	Virtualized/hierarchical State Management
[SWS_SM_00501]	Virtualized/hierarchical State Management internal State
[SWS_SM_91001]	
[SWS_SM_91002]	
[SWS_SM_91003]	
[SWS_SM_91004]	



△

Number	Heading
[SWS_SM_91007]	
[SWS_SM_91008]	
[SWS_SM_91009]	
[SWS_SM_91010]	
[SWS_SM_91011]	
[SWS_SM_91012]	
[SWS_SM_91013]	
[SWS_SM_91014]	
[SWS_SM_91015]	
[SWS_SM_91016]	
[SWS_SM_91017]	
[SWS_SM_91018]	
[SWS_SM_91019]	
[SWS_SM_91020]	

**Table C.2: Added Traceables in R20-11**

### **C.2.2 Changed Traceables in R20-11**

none

### **C.2.3 Deleted Traceables in R20-11**

none

### **C.2.4 Added Constraints in R20-11**

none

### **C.2.5 Changed Constraints in R20-11**

none

### **C.2.6 Deleted Constraints in R20-11**

none

### C.3 Constraint and Specification Item History of this document according to AUTOSAR Release R19-11

#### C.3.1 Added Traceables in 19-11

none

#### C.3.2 Changed Traceables in 19-11

Number	Heading
[SWS_SM_00500]	Virtualized/hierarchical State Management
[SWS_SM_00501]	Virtualized/hierarchical State Management internal State

**Table C.3: Changed Traceables in 19-11**

#### C.3.3 Deleted Traceables in 19-11

none

#### C.3.4 Added Constraints in 19-11

none

#### C.3.5 Changed Constraints in 19-11

none

#### C.3.6 Deleted Constraints in 19-11

none



## C.4 Constraint and Specification Item History of this document according to AUTOSAR Release R19-03

### C.4.1 Added Traceables in 19-03

Number	Heading
[SWS_SM_00020]	InternalState Propagation
[SWS_SM_00021]	InternalState Influence
[SWS_SM_00202]	Reset Execution

**Table C.4: Added Traceables in 19-03**

### C.4.2 Changed Traceables in 19-03

Number	Heading
[SWS_SM_00002]	Function Group State Change Request
[SWS_SM_00003]	Function Group State Retrieval
[SWS_SM_00004]	Function Group State Change Request Result
[SWS_SM_00006]	Function Group Calibration Support
[SWS_SM_00200]	Prevent Shutdown during to Update Session
[SWS_SM_00201]	Supervision of Shutdown Prevention
[SWS_SM_00302]	NetworkHandle to FunctionGroupState
[SWS_SM_00401]	Execution Management Results
[SWS_SM_00402]	Function Group State Change Results
[SWS_SM_00500]	Virtualized/hierarchical State Management
[SWS_SM_00501]	Virtualized/hierarchical State Management internal State

**Table C.5: Changed Traceables in 19-03**

### C.4.3 Deleted Traceables in 19-03

Number	Heading
[SWS_SM_00010]	Component (states)
[SWS_SM_00011]	Component (states) Handling
[SWS_SM_00012]	Component (states) Registration
[SWS_SM_00013]	Component (states) Configuration
[SWS_SM_00014]	Component (states) Enforcement
[SWS_SM_00015]	Component (states) Transitions





Number	Heading
[SWS_SM_00102]	Component States for Reset

**Table C.6: Deleted Traceables in 19-03****C.4.4 Added Constraints in 19-03**

none

**C.4.5 Changed Constraints in 19-03**

none

**C.4.6 Deleted Constraints in 19-03**

none