

<b>Document Title</b>	Explanation of Adaptive Platform Software Architecture
<b>Document Owner</b>	AUTOSAR
<b>Document Responsibility</b>	AUTOSAR
<b>Document Identification No</b>	982

<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>• Applied a more fine-grained description schema for functional clusters and interfaces in the Building Block View.</li> <li>• Removed functional cluster RESTful Communication</li> <li>• Added functional cluster Adaptive Intrusion Detection System Manager</li> <li>• Added section for clarification of diagnostic deployment options</li> </ul>
2020-11-30	R20-11	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	6
1.1	Objectives	6
1.2	Scope	6
1.3	Document Structure	7
2	Definition of Terms and Acronyms	8
2.1	Acronyms and Abbreviations	8
2.2	Definition of Terms	8
3	Related Documentation	9
4	Overview and Goals	10
4.1	Requirements Overview	10
4.2	Quality Goals	12
4.3	Stakeholders	12
5	Architecture Constraints	13
5.1	Internal Interfaces	13
5.2	Distributed Work	14
6	Quality Requirements	15
6.1	Quality Attributes	15
6.1.1	AUTOSAR Adaptive Platform Standard	15
6.1.2	AUTOSAR Adaptive Platform Stack	18
6.1.3	AUTOSAR Adaptive Application	18
6.2	Quality Scenarios	19
7	System Scope and Context	20
7.1	Adaptive Application	20
7.2	Dependencies	20
7.2.1	Crypto Provider	20
7.2.2	Operating System	21
7.2.3	Watchdog	21
7.3	External Systems	21
7.3.1	AUTOSAR Adaptive Application	21
7.3.2	AUTOSAR Classic Platform	22
7.3.3	Third-party Platform	22
7.3.4	Diagnostic Client	22
7.3.5	Backend	22
8	Solution Strategy	23
8.1	Architectural Approach	23
8.2	Decomposition Strategy	23
8.3	UML Profile	24
8.4	Technology	25

8.4.1	Implementation Language	25
8.4.2	Parallel Processing	26
8.5	Design Principles	26
8.5.1	Leveraging existing standards	26
8.5.2	SOLID principles	26
8.5.3	Acyclic Dependencies Principle	27
8.6	Deployment	27
8.7	Verification and Validation	28
9	Building Block View	29
9.1	Overview	29
9.1.1	Description pattern	29
9.2	Runtime	30
9.2.1	Execution Management	30
9.2.2	State Management	37
9.2.3	Log and Trace	43
9.2.4	Core	47
9.2.5	Operating System Interface	49
9.3	Communication	51
9.3.1	Communication Management	51
9.3.2	Network Management	57
9.3.3	Time Synchronization	59
9.4	Storage	66
9.4.1	Persistency	66
9.5	Security	74
9.5.1	Cryptography	75
9.5.2	Identity and Access Management	107
9.5.3	Adaptive Intrusion Detection System Manager	110
9.6	Safety	112
9.6.1	Platform Health Management	113
9.7	Configuration	117
9.7.1	Update and Configuration Management	118
9.7.2	Registry	127
9.8	Diagnostics	129
9.8.1	Diagnostic Management	129
10	Runtime View	150
10.1	Overview	150
10.2	AUTOSAR Runtime for Adaptive Applications Lifecycle	150
10.2.1	Machine Startup	150
10.2.2	Machine Shutdown	151
10.2.3	Function Group State Transition	152
10.2.4	Failure Recovery	153
10.3	Communication	154
10.4	Update and Configuration Management	155
10.4.1	Update of an Adaptive Application	155

11	Deployment View	157
11.1	Vehicle Software Deployment	157
12	Cross-cutting Concepts	158
12.1	Overview of Platform Entities	158
12.2	Function Group	159
12.3	Function Group State	159
12.4	Software Cluster	159
12.5	Machine	162
12.6	Manifest	162
12.7	Application Design	163
12.8	Execution Manifest	164
12.9	Service Instance Manifest	164
12.10	Machine Manifest	165
12.11	Diagnostics deployment	165
12.12	Error Handling	167
12.13	Trusted Platform	167
12.14	Secure Communication	168
13	Risks and Technical Debt	169
13.1	Risks	169
13.1.1	Risk Assessment	169
13.1.2	Risk List	170
13.2	Technical Debt	170

# 1 Introduction

This explanatory document provides an overview of the software architecture of the AUTOSAR Adaptive Platform standard.

## 1.1 Objectives

This document is an architecture description of the AUTOSAR Adaptive Platform in accordance to [1, ISO/IEC 42010] and has the following main objectives:

- Identify the **stakeholders** of the AUTOSAR Adaptive Platform and their **concerns**.
- Identify the **system scope** and provide **overview information** of the AUTOSAR Adaptive Platform.
- Provide definitions for all used **architecture viewpoints** and a **mapping of all stakeholder concerns to those viewpoints**.
- Provide an **architecture view** and its **architecture models** for each architecture viewpoint used in this architecture description.
- Provide **correspondence rules** and **correspondences** among the contents of this architecture description.
- Provide an **architecture rationale** (explanation, justification, reasoning for decisions made) on a high level. A more in-depth documentation of decisions is provided in [2, EXP\_SWArchitecturalDecisions].
- Provide a **record of known inconsistencies** among the architecture description.

Please note that the AUTOSAR Adaptive Platform standard is defined by means of requirements and software specification documents. Those documents deliberately lack specifications of dependencies and interfaces between the building blocks of the AUTOSAR Adaptive Platform to provide more degrees of freedom for stack vendors in their solution design. This document describes the original architectural design of the AUTOSAR Adaptive Platform including details how the building blocks should interact with each other. It is an example how an implementation of the standard should work internally. However, a stack vendor is free to choose another design as long as it complies with the binding AUTOSAR Adaptive Platform standard.

## 1.2 Scope

This explanatory document applies to the AUTOSAR Adaptive Platform. It is recommended to get an overview of the AUTOSAR Adaptive Platform for all members of the working groups, stack vendors, and application developers.

## 1.3 Document Structure

This document is organized as follows. Section 4 provides an overview of the main requirements for the AUTOSAR Adaptive Platform, the top quality goals of its architecture, and a list of stakeholders that are affected by it. Section 5 lists requirements that constrain design and implementation decisions or decisions about the development process.

Section 6 is the base for discovering trade-offs and sensitivity points in the architecture of the AUTOSAR Adaptive Platform by introducing a quality attribute tree followed by the most important quality scenarios. The system context in which the AUTOSAR Adaptive Platform is intended to be used is outlined in section 7. Section 8 summarizes the fundamental decisions and solution strategies, that shape the architecture of the AUTOSAR Adaptive Platform such as technology decisions or architectural patterns to be used.

Sections 9 to 11 explain the software architecture from different view points. First, section 9 explains the decomposition of the AUTOSAR Adaptive Platform into Functional Clusters and their interdependencies. Then, section 10 demonstrates how the main use cases are realized using the Functional Clusters in the AUTOSAR Adaptive Platform. Section 11 shows different scenarios how applications based on the AUTOSAR Adaptive Platform may be deployed.

Section 12 provides an overview of concepts and patterns used by the AUTOSAR Adaptive Platform. Section 13 lists and rates risks associated with the architecture of the AUTOSAR Adaptive Platform and technical debt.

## 2 Definition of Terms and Acronyms

### 2.1 Acronyms and Abbreviations

Abbreviation / Acronym	Description
DoIP	Diagnostics over Internet Protocol
POSIX	Portable Operating System Interface
SecOC	AUTOSAR Secure Onboard Communication
TLS	Transport Layer Security
UML	Unified Modeling Language

### 2.2 Definition of Terms

This section lists terms that are specific to this document. A list of general terms for AUTOSAR is provided in the [3, glossary].

Term	Description
Functional Cluster	A logical group of functionality within the AUTOSAR Adaptive Platform. <code>Functional Clusters</code> are the second level of abstraction in the building block view (cf. Chapter 9). They are also subject of the individual specification documents that make up the AUTOSAR Adaptive Platform standard.
Function Group	A set of modeled <code>Processes</code> . See Section 12.2 for details.
Thread	The smallest sequence of instructions that can be managed independently by a scheduler. Multiple <code>Threads</code> can be executed concurrently within one <code>Process</code> sharing resources such as memory.
Watchdog	An external component that supervises execution of the AUTOSAR Adaptive Platform. See Section 7.2.3 for details.



### 3 Related Documentation

This document provides a high-level overview of the AUTOSAR Adaptive Platform architecture. It is closely related to general requirements for AUTOSAR Adaptive Platform specified in [4, RS\_Main] and [5, RS\_General], and the architectural decisions documented in [2, EXP\_SWArchitecturalDecisions].

The individual building blocks of the architecture (`Functional Clusters`) are specified in separate documents. Each `Functional Cluster` defines one or more requirements specification(s) (RS document), one or more software specification(s) (SWS document) and one or more explanatory document(s) (EXP document). Please refer to these documents for any details on the AUTOSAR Adaptive Platform standard.

## 4 Overview and Goals

In conventional automotive systems ECUs are used to replace or augment electro-mechanical systems. Those resource constrained, deeply-embedded ECUs typically perform basic control functions by creating electrical output signals (e.g. for actors) based on input signals (e.g. from sensors) and information from other ECUs connected to the vehicle network. Much of the control software is specifically designed and implemented for the target vehicle and does not change significantly during vehicle lifetime. The AUTOSAR Classic Platform standard addresses the needs of these deeply-embedded systems.

Recent and future vehicle functions, such as highly automated driving, will introduce complex and computing resource demanding software that shall fulfill strict safety, integrity and security requirements. Such software performs for example, environment perception and behavior planning, and interacts with external backend and infrastructure systems. The software in the vehicle regularly needs to be updated during the life-cycle of the vehicle, due to evolving external systems, improved or added functionality, or security problems. The AUTOSAR Classic Platform standard cannot fulfill the needs of such systems. Therefore, AUTOSAR specifies a second software platform, the AUTOSAR Adaptive Platform. It provides high-performance computing and communication mechanisms as well as a flexible software configuration, for example, to support software update over-the-air. Features that are specifically defined for the AUTOSAR Classic Platform, such as access to electrical signals and automotive specific bus systems, can be integrated into the AUTOSAR Adaptive Platform but is not in the focus of standardization.

### 4.1 Requirements Overview

This section provides an overview of the basic requirements for the AUTOSAR Adaptive Platform that impact its architecture. The corresponding requirement identifiers are provided in square brackets. Please refer to [4, RS\_Main] and [5, RS\_General] for any details, rationale or intended use-cases of these requirements.

#### **Support of state-of-the-art Technology**

The AUTOSAR Adaptive Platform aims to support resource-intensive (memory, cpu) applications on state-of-the-art hardware. Therefore, the AUTOSAR Adaptive Platform shall support high performance computing platforms [RS\_Main\_00002] as well as virtualized environments [RS\_Main\_00511]. The AUTOSAR Adaptive Platform shall be able to run multiple applications in parallel [RS\_Main\_00049], each with concurrent application internal control flows [RS\_Main\_00050].

## Software Update and Configuration

The AUTOSAR Adaptive Platform shall support a flexible (configuration) data and software update. Hereby, AUTOSAR Adaptive Platform shall support up- and download of such update packages [RS\_Main\_00650] and change of communication and application software at runtime [RS\_Main\_00503].

AUTOSAR shall provide a unified way to describe software systems deployed to Adaptive and / or Classic platforms [RS\_Main\_00161]. That kind of description shall also support the deployment and reallocation of AUTOSAR Application Software [RS\_Main\_00150], and shall provide means to describe interfaces of the entire system [RS\_Main\_00160].

## Security

The AUTOSAR Adaptive Platform shall support the development of secure systems [RS\_Main\_00514] with secure access to ECU data and services [RS\_Main\_00170], and secure onboard communication [RS\_Main\_00510].

## Safety

The AUTOSAR Adaptive Platform shall support the development of safety related systems [RS\_Main\_00010] that are reliable [RS\_Main\_00011] and highly available [RS\_Main\_00012]. The AUTOSAR Adaptive Platform specifications shall be analyzable and support methods to demonstrate the achievement of safety related properties accordingly [RS\_Main\_00350].

## Reuse and Interoperability

The AUTOSAR Adaptive Platform shall support standardized interoperability with non-AUTOSAR software [RS\_Main\_00190] as well as (source code) portability for AUTOSAR Adaptive Applications across different implementations of the platform [RS\_AP\_00111]. Hereby, the AUTOSAR Adaptive Platform shall provide means to describe a component model for application software [RS\_Main\_00080], and support bindings for different programming languages [RS\_Main\_00513].

## Communication

The AUTOSAR Adaptive Platform shall support standardized automotive communication protocols [RS\_Main\_00280] for intra ECU communication [RS\_Main\_01001] with different network topologies [RS\_Main\_00230].

## Diagnostics

The AUTOSAR Adaptive Platform shall provide diagnostics means during runtime for production and services purposes [RS\_Main\_00260].

## 4.2 Quality Goals

This section will list the top quality goals for the architecture whose fulfillment is of highest importance to the major stakeholders in a future version of this document. Please refer to the currently un-prioritized list of Quality Attributes in Section 6.1.

## 4.3 Stakeholders

This section lists the stakeholders of the AUTOSAR Adaptive Platform architecture and their main expectations.

Role	Expectation
Project Leader	Overview of technical risks and technical debt in the AUTOSAR Adaptive Platform.
Working Group Architecture	Concise yet thorough documentation of the goals and driving forces of the AUTOSAR Adaptive Platform. Documentation of the original architectural design of the AUTOSAR Adaptive Platform standard. Documentation of identified technical risks and technical debt in the AUTOSAR Adaptive Platform.
Working Group	Consolidated overview of the AUTOSAR Adaptive Platform architecture. Realization of use-cases that span multiple <code>Functional Clusters</code> . Usage of interfaces within the AUTOSAR Adaptive Platform. Guidelines and patterns for <code>Functional Cluster</code> and interface design.
Stack Developer	Consolidated overview of the original architectural design of the AUTOSAR Adaptive Platform. Realization of use-cases that span multiple <code>Functional Clusters</code> . Usage of interfaces within the AUTOSAR Adaptive Platform.
Application Developer	Overview of the building blocks of the AUTOSAR Adaptive Platform and their purpose and provided functionality. Explanation of the concepts used in the AUTOSAR Adaptive Platform.
Integrator	Overview and explanation of the original architectural design of the AUTOSAR Adaptive Platform. Overview of extension points of the AUTOSAR Adaptive Platform.

**Table 4.1: Stakeholder table with roles and expectations**

## 5 Architecture Constraints

AUTOSAR is a worldwide development partnership of vehicle manufacturers, suppliers, service providers and companies from the automotive electronics, semiconductor and software industry. AUTOSAR standardizes the AUTOSAR Adaptive Platform automotive middleware. The AUTOSAR Adaptive Platform is not a concrete implementation. The AUTOSAR Adaptive Platform standard leaves a certain degree of freedom to implementers of the standard, as most standards do. On the one hand, more freedom enables competition among the different implementations and a broader choice of properties for users of the AUTOSAR Adaptive Platform. On the other hand, a more strict standardization makes the different implementations compatible and exchangeable (within the standardized scope). Naturally, those attributes are in conflict. It is usually a choice of the standardization organization to evaluate the attributes and define the desired level of strictness.

The AUTOSAR Classic Platform is rather strict in that sense by specifying a detailed layered software architecture imposing many constraints on its implementations. The AUTOSAR Adaptive Platform launched with a less strict approach. That less strict approach puts constraints on the AUTOSAR Adaptive Platform architecture as outlined below.

### 5.1 Internal Interfaces

An important architectural constraint is that only interfaces that are intended to be used by applications or interfaces that are used to extend the functionality of the AUTOSAR Adaptive Platform shall be standardized. Internal interfaces between the building blocks of the AUTOSAR Adaptive Platform shall not be standardized. This approach leaves a lot of freedom to design and optimize the internals of an AUTOSAR Adaptive Platform stack. However, it also imposes constraints on how the AUTOSAR Adaptive Platform architecture can be defined and described in this document. Also, this means that it might not be possible to use different functional clusters from different AUTOSAR Adaptive Platform stack vendors.

First, the existence of internal interfaces and their usage by other building blocks is in most cases a recommendation and reflects the original design approach of the authors of the standard. The same applies to any interactions described in this document that make use of such internal interfaces.

Second, some quality attributes may be hard to ensure in general by the architecture of the standard. Additional measures like security or safety considerations lack well-defined inputs such as data flows or specifications of interdependencies. Consequently, a more thorough design phase is required when an AUTOSAR Adaptive Platform stack is implemented.

## 5.2 Distributed Work

Standardization of the AUTOSAR Adaptive Platform is a worldwide distributed effort. The individual building blocks are specified by dedicated working groups in separate documents to be able to scale in that distributed setup. This impacts the way the AUTOSAR Adaptive Platform architecture is described in this document.

First, this document shows interfaces on an architectural level only. This document does not specify details of interfaces such as individual operations. This keeps redundancies and thus dependencies between this document and the documents actually specifying the individual building blocks manageable. Another consequence is that the interactions shown in this document are not based on actual operations specified in the interfaces but rather on an architectural level as well.

Second, this document aims to provide guidance for the working groups in specifying the individual building blocks by defining patterns and concepts to solve common problems. This guidance should help to build a uniform and consistent standard from ground up.

## 6 Quality Requirements

Quality requirements define the expectations of AUTOSAR Adaptive Platform stakeholders for the quality and the attributes of the AUTOSAR Adaptive Platform standard that indicate whether the quality factors are satisfied or not. Section 6.1 starts by listing the quality attributes that, in the end, are used to assess whether the AUTOSAR Adaptive Platform and its software architecture satisfies the expected quality factors or not. Section 6.2 then provides quality scenarios that operationalize quality attributes and turn them into measurable quantities by describing the reaction of the system to a stimulus in a certain situation.

### 6.1 Quality Attributes

The AUTOSAR Adaptive Platform has many stakeholders with different concerns. Thus, this document uses the following three quality attribute categories that correspond to the three main stakeholder groups in order to make the requirements and their impact on the architecture more comprehensible:

- **AUTOSAR Adaptive Platform Standard:** Quality requirements for the AUTOSAR standard itself. These requirements may directly affect the architecture of the AUTOSAR Adaptive Platform.
- **AUTOSAR Adaptive Platform Stack:** Quality requirements for an implementation of the AUTOSAR standard as an AUTOSAR stack. These requirements may indirectly affect the architecture of the AUTOSAR Adaptive Platform.
- **AUTOSAR Adaptive Application:** Quality requirements for an application based on an AUTOSAR stack. These requirements may transitively affect the architecture of the AUTOSAR Adaptive Platform.

The quality attributes are organized according to the Architecture Tradeoff Analysis Method (ATAM) [6] as a tree, one for each of the quality attribute categories. The leaves of those trees are the individual quality attributes.

#### 6.1.1 AUTOSAR Adaptive Platform Standard

- Functional suitability
  - The software architecture shall reflect the project objectives (POs) and be the consistent source for all specifications (here: completeness with respect to the PO; see also usability below).
  - The standard shall not contain elements that are not traceable to POs, change requests (CRs), or concepts.

- The standard shall contain at least one element derived from each PO, CR, or concept.
- Performance efficiency
  - The specification shall allow for a run-time efficient implementation. Run-time efficiency refers to all resource consumption, CPU, RAM, ROM.
- Compatibility
  - The standard shall retain older versions of its elements in the face of change.
  - The standard shall be interoperable with pre-existing standards, especially the AUTOSAR Classic Platform. Pre-existing standards means network protocols and the like.
  - The standard shall adopt new versions of pre-existing standards only after an impact analysis.
- Usability
  - The use of the standard shall be as easy as possible for suppliers and application developers. Easy means: not much material and resources required.
  - The holistic approach shall not be broken (avoid different approaches in one standard).
  - The standard shall contain application sample code for all its elements.
  - The standard shall contain documentation of the use cases for its elements.
  - The standard shall document the semantics of its elements.
  - The standard shall document its decisions, consequences, and implementation restrictions (both for stack & apps) including their rationale.
  - The standards elements shall be easy to use and hard to misuse.
  - The standard shall stick to pre-existing standards, as far as no functional requirements are compromised.
  - The standard shall be as stable as possible.
  - AUTOSAR standards shall not change disruptive but rather evolve evolutionary (for example, backward-compatibility can be a help).
  - The software architecture shall reflect the PO and be the consistent source for all specifications (here: consistency; see also functional suitability above).
- Reliability
  - The standard shall classify its elements with respect to safety relevance (that is, functional clusters shall be marked if they participate in safety critical operations of the platform).



- The standard shall specify control flow restrictions between its elements in order to achieve freedom from interference.
- The standard shall contain use case driven argumentation for safety scenarios that can be used to build a safety case. (This should help the stack implementers in getting a certification, if they follow the standard.)
- Security
  - The standard shall specify data flow restrictions between its elements, and between applications.
  - The standard shall classify its elements with respect to security sensitivity (that is, functional clusters shall be marked if they handle sensitive data.)
  - The standard shall contain use case driven argumentation for security scenarios that can be used to build a security case. (This should help the stack implementers in getting a certification, if they follow the standard.)
- Maintainability
  - It shall be possible in an efficient way to maintain AUTOSAR Adaptive Platform without preventing the introduction of new technologies (efficient in terms of effort on the modification of the standard).
  - The impact set of a change shall be available.
  - The standard shall be structured in a way that minimizes change impact.
  - It shall be possible to drop/deprecate elements of the standard.
  - It shall be easy to add new features/needs without breaking the maintainability or the need to redesign the software architecture. Easy means quick, with low effort, local changes only and no heavy side effects.
  - The maturity of parts of the standard shall be visible.
  - The process shall enforce an architectural impact analysis in a very early stage of the change process.
  - The process shall enforce minimizing changes, that is not adding similar functionality multiple times.
- Portability
  - Applications shall be portable between different stack implementations and different machines.
  - It shall be possible to scale the software architecture to the given project needs.

### 6.1.2 AUTOSAR Adaptive Platform Stack

- Compatibility
  - An AUTOSAR Adaptive Platform stack implementation shall be capable to offer multiple versions of the same service.
  - An instance of an AUTOSAR Adaptive Platform stack implementation shall be able to co-exist with other instances on different machines, within the same vehicle.
- Usability
  - An AUTOSAR Adaptive Platform stack implementation shall explicitly document restrictions on the application development that go beyond the standard.
- Maintainability
  - An AUTOSAR Adaptive Platform stack implementation shall be traceable to the contents of the standard.
  - An AUTOSAR Adaptive Platform stack implementation shall support multiple versions of the same service.
- Portability
  - An AUTOSAR Adaptive Platform stack shall be portable to a different custom hardware.
  - An AUTOSAR Adaptive Platform stack shall provide mechanisms to replace parts.

### 6.1.3 AUTOSAR Adaptive Application

- Usability
  - *No Goal:* An application developer shall be able to supply custom implementation for pre-defined platform functionality.
- Maintainability
  - An application shall explicitly state which parts of the standard it uses.
- Portability
  - An application entirely based on AUTOSAR Adaptive Platform (i.e. without custom extensions) shall be portable to another AUTOSAR Adaptive Platform stack of the same version without modifications to the application source code itself (source code compatibility).

## 6.2 Quality Scenarios

There are currently no quality scenarios defined for the AUTOSAR Adaptive Platform.

## 7 System Scope and Context

This chapter provides an overview of the AUTOSAR Adaptive Platform system context by separating the AUTOSAR Adaptive Platform and its communication partners (e.g., external systems). Considering Figure 7.1, there are three categories of communication partners for the AUTOSAR Adaptive Platform.

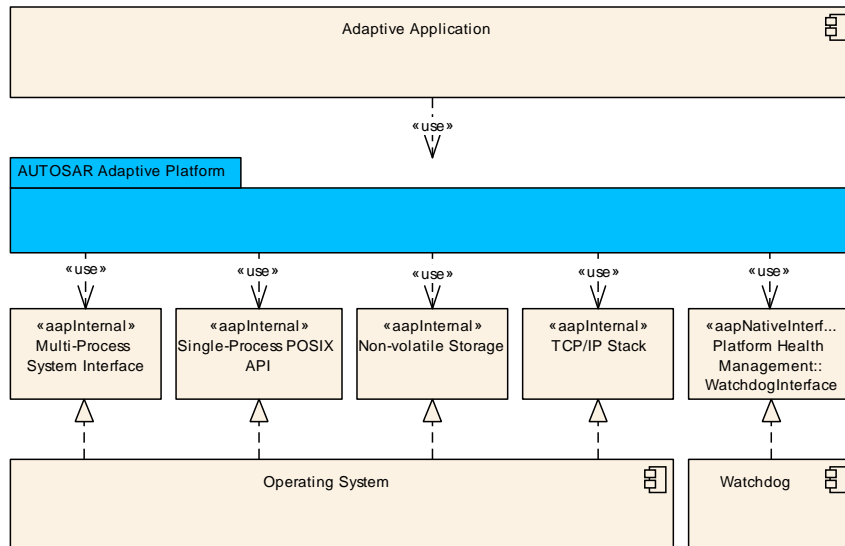


Figure 7.1: Overview of AUTOSAR Adaptive Platform and its context

The AUTOSAR Adaptive Platform is conceptually a middleware. AUTOSAR Adaptive Platform provides services to Adaptive Applications (cf. Section 7.1) beyond those available from the underlying operating system, drivers, and extensions (cf. Section 7.2). Section 7.3 describes the third category that are external systems communicating with (an Adaptive Application via) AUTOSAR Adaptive Platform.

### 7.1 Adaptive Application

Adaptive Applications are built on the functionality provided by the AUTOSAR Adaptive Platform. They directly use the various interfaces provided by the individual building blocks of AUTOSAR Adaptive Platform described in more detail in chapter 9.

### 7.2 Dependencies

#### 7.2.1 Crypto Provider

Crypto Provider is a component that provides implementations of cryptographic routines and hash functions to the AUTOSAR Adaptive Platform.

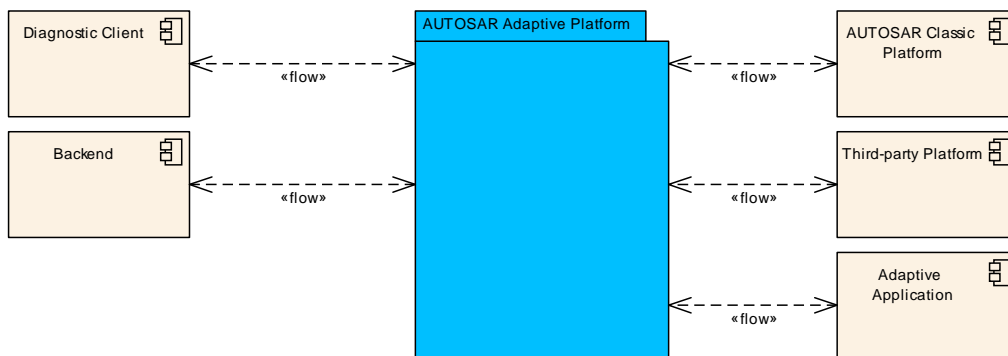
## 7.2.2 Operating System

The `Operating System` is the main component that AUTOSAR Adaptive Platform uses to provide its services. The `Operating System` controls processes and threads, and provides inter-process communication facilities. The `Operating System` also provides access to network interfaces, protocols like `TCP/IP`, and access to non-volatile storage.

## 7.2.3 Watchdog

The `Watchdog` is a component to control the hardware watchdog of the machine an AUTOSAR Adaptive Platform runs on.

## 7.3 External Systems



**Figure 7.2: Overview of the AUTOSAR Adaptive Platform and external systems**

The AUTOSAR Adaptive Platform supports applications that are operated in heterogeneous environments. This section lists the external systems that AUTOSAR Adaptive Platform is intended to interface.

### 7.3.1 AUTOSAR Adaptive Application

There may be many `Adaptive Applications` deployed in a vehicle on different `Machines`. An `Adaptive Application` that does not run on the current instance of the AUTOSAR Adaptive Platform is therefore an external system to the AUTOSAR Adaptive Platform. Such `Adaptive Applications` may exchange data such as sensor or status information. During a software update of the entire vehicle, the update of the individual AUTOSAR Adaptive Platforms could be coordinated by a central `Adaptive Application` that makes use of the `UCM Master` addon to `UCM`.

### 7.3.2 AUTOSAR Classic Platform

The AUTOSAR Classic Platform is the main platform for deeply-embedded applications such as sensor/actor systems. Adaptive Applications may require access for example to sensor data provided by an AUTOSAR Classic Platform ECU and vice versa.

### 7.3.3 Third-party Platform

Besides the both platforms (AUTOSAR Adaptive Platform and AUTOSAR Classic Platform) provided by AUTOSAR, there may be ECUs in a vehicle and other systems that are built on different platforms that need to communicate with an Adaptive Application via AUTOSAR Adaptive Platform.

### 7.3.4 Diagnostic Client

A Diagnostic Client uses the diagnostic services provided by the AUTOSAR Adaptive Platform.

### 7.3.5 Backend

A Backend system provides Software Packages for download and controls the update process via Update and Configuration Management.

## 8 Solution Strategy

The AUTOSAR Adaptive Platform is a standard for an automotive middleware. It is not a concrete implementation. The AUTOSAR Adaptive Platform standard leaves a certain degree of freedom to its implementers by defining requirements and software specifications that need to be fulfilled without specifying how.

### 8.1 Architectural Approach

To support the complex applications, while allowing maximum flexibility and scalability in processing distribution and compute resource allocations, AUTOSAR Adaptive Platform follows the concept of a service-oriented architecture (SOA). In a service-oriented architecture a system consists of a set of services, in which one may use another in turn, and applications that use one or more of the services depending on its needs. Often service-oriented architectures exhibit system-of-system characteristics, which AUTOSAR Adaptive Platform also has. A service, for instance, may reside on a local ECU that an application also runs, or it can be on a remote ECU, which is also running another instance of AP. The application code is the same in both cases - the communication infrastructure will take care of the difference providing transparent communication. Another way to look at this architecture is that of distributed computing, communicating over some form of message passing. At large, all these represent the same concept. This message passing, communication-based architecture can also benefit from the rise of fast and high-bandwidth communication such as Ethernet.

### 8.2 Decomposition Strategy

The building blocks of the AUTOSAR Adaptive Platform architecture are refined step-by-step in this document according to the model depicted in figure 8.1. The top-level categories are chosen to give an overview from a users perspective what kind of functionality the AUTOSAR Adaptive Platform provides. A category contains one or more `Functional Clusters`. The `Functional Clusters` of the AUTOSAR Adaptive Platform are defined to group a specific coherent technical functionality. `Functional Clusters` themselves specify a set of interfaces and components to provide and realize that technical functionality. The building block view also contains information of the `Functional Clusters` interdependencies based on interfaces from other `Functional Clusters` they use. However, note that these interdependencies are recommendations rather than strict specifications because they would constrain implementations.

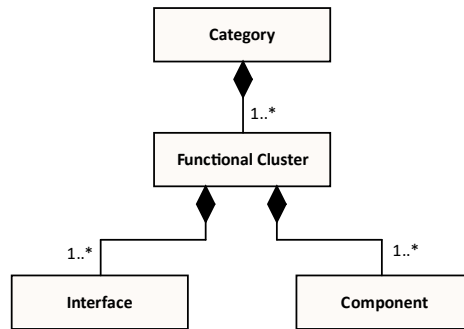


Figure 8.1: Type model of building blocks

### 8.3 UML Profile

The UML diagrams presented in this document use a UML profile to provide a more precise semantics of the elements and relationships. Table 8.1 provides an overview of the stereotypes in that profile and their semantics. The names of all stereotypes that are specific to the AUTOSAR architecture are prefixed with *aap* (short for AUTOSAR Architecture Profile) to make them easily distinguishable from standard UML stereotypes and keywords.

Stereotype	Metaclass	Semantics
<code>aapInternal</code>	Interface	Internal interfaces shall be used only by Functional Clusters within the platform. Internal interfaces shall be realized by components that are part of the respective stack implementation (i.e., another Functional Cluster, additional middleware, drivers, or the operating system).
<code>aapPlatformExtension</code>	Interface	Platform extension interfaces shall be used only by Functional Clusters within the platform. Platform extension interfaces shall be realized either by third-party components (including application-level components) or components are part of the respective stack implementation (i.e., another Functional Cluster, additional middleware, drivers, or the operating system).
<code>aapAPI</code>	Interface	An interface of the public API of the platform. Such interfaces may be used by Adaptive Applications and other Functional Clusters within the platform.
<code>aapNativeInterface</code>	Interface	An interface defined in the respective programming language of the stack implementation.



Stereotype	Metaclass	Semantics
aapPortInterface	Interface	An aapPortInterface relates to an element in the Manifest using the Instance-Specifier pattern. Such interfaces are either defined in the respective programming language of the stack implementation or they are generated in that language by the stack tooling. The property Generated specifies if they are generated.
aapAraComServiceInterface	Interface	An ara::com service interface defined and configured in the Manifest. The tooling of the stack implementation will generate stub and skeleton implementations of these interfaces in the respective programming language.
aapFunctionalCluster	Component	A functional cluster of the AUTOSAR Adaptive Platform.
aapServiceMethod	Operation	A method specified as part of an ara::com service interface.
aapCallbackMethod	Operation	A method that acts as a callback.
aapServiceField	Attribute	A field specified as part of an ara::com service interface.
aapServiceEvent	Attribute	An event specified as part of an ara::com service interface.
aapProvidedPort	Usage, Realization	Denotes that the underlying relationship is configured in the Manifest using a provided port.
aapRequiredPort	Usage, Realization	Denotes that the underlying relationship is configured in the Manifest using a required port.

**Table 8.1: Overview of Stereotypes**

## 8.4 Technology

### 8.4.1 Implementation Language

C++ is the programming language of choice for the AUTOSAR Adaptive Platform and Adaptive Applications. C++ was chosen due to its safer programming model (compared to C) and availability of certified compilers that produce highly optimized machine code. Such properties are especially important for safety- and performance-critical, real-time applications (such as typical Adaptive Applications) where C++ has become more and more popular in the software industry and academics.

## 8.4.2 Parallel Processing

Although the design for AUTOSAR Adaptive Platform as a service-oriented architecture inherently leverages parallel processing, the advancement of (heterogeneous) many-core processors offers additional opportunities. The AUTOSAR Adaptive Platform is designed to scale its functionality and performance as (heterogeneous) many-core technologies advance. Hardware and platform interface specifications are one part of that equation. However, advancements in operating system and hypervisor technologies as well as development tools (for example automatic parallelization) are also crucial and are to be fulfilled by AUTOSAR Adaptive Platform providers, the software industry, and academics.

## 8.5 Design Principles

The architecture of the AUTOSAR Adaptive Platform is based on several design principles that are outlined below.

### 8.5.1 Leveraging existing standards

AUTOSAR Adaptive Platform aims to leverage existing standards and specifications wherever possible. For example, AUTOSAR Adaptive Platform is built on the existing and open C++ standard (cf. Section 8.4.1) to facilitate a faster development of the AUTOSAR Adaptive Platform itself and benefiting from the eco-systems of such standards. It is, therefore, a critical focus in developing the AUTOSAR Adaptive Platform specification not to casually introduce a new replacement functionality that an existing standard already offers. For instance, no new interfaces are casually introduced just because an existing standard provides the functionality required but the interface is superficially hard to understand.

### 8.5.2 SOLID principles

The SOLID principles [7] are a central part of the design principles of AUTOSAR. While these five principles are all valid, only the Single-responsibility Principle, the Interface Segregation Principle and the Dependency Inversion Principle are relevant on the abstraction level of this document. Therefore, they are elaborated in the following.

#### 8.5.2.1 Single-responsibility Principle

The single-responsibility principle (SRP, SWEBOK3) [7] states that a component or class should be responsible for a single part of the overall functionality provided by the software. That responsibility should be encapsulated by the component or class. The

services provided by the component or class (via its interface(s)) should be closely aligned with its responsibility.

The single-responsibility principle minimizes the reasons (i.e. a change to the single responsibility) that require a change to its interface. Thus, it minimizes impact on clients of such an interface and leads to a more maintainable architecture (or code).

### 8.5.2.2 Interface Segregation Principle

The interface segregation principle (ISP) [7], [8] states that clients should not be forced to depend on methods that they don't use. As a consequence of the interface segregation principle, interfaces should be split up to reflect different roles of clients.

Similar to the single-responsibility principle, the segregation of interfaces reduce the impact of a change to an interface to the clients and suppliers of an segregated interface.

### 8.5.2.3 Dependency Inversion Principle

The dependency inversion principle (DIP) [7], [8] states that high-level building blocks should not depend on low-level building blocks. Both should depend on abstractions (e.g. interfaces). Furthermore, the dependency inversion principle states that abstractions (e.g. interfaces) should not depend on details. Details (e.g. a concrete implementation) should depend on abstractions.

The dependency inversion principle results in a decoupling of the implementations of building blocks. This is important to scale implementation efforts (cf. Section 5.2) and to perform proper integration tests.

### 8.5.3 Acyclic Dependencies Principle

The acyclic dependencies principle (ADP) [7], [8] states that dependencies between building blocks should form a directed acyclic graph.

The acyclic dependencies principle helps to identify participating building blocks and reason about error propagation and freedom from interference. In general, it also reduces the extend of building blocks to consider during activities such as test, build and deployment.

## 8.6 Deployment

The AUTOSAR Adaptive Platform supports the incremental deployment of applications, where resources and communications are managed dynamically to reduce the

effort for software development and integration, enabling short iteration cycles. Incremental deployment also supports explorative software development phases. For product delivery, the AUTOSAR Adaptive Platform allows the system integrator to carefully limit dynamic behavior to reduce the risk of unwanted or adverse effects allowing safety qualification. Dynamic behavior of an application will be limited by constraints stated in the `Execution Manifest` (cf. Section 12.8), for example, dynamic allocation of resources and communication paths are only possible in defined ways, within configured ranges. Implementations of an AUTOSAR Adaptive Platform may further remove dynamic capabilities from the software configuration for production use. Examples of reduced behavioral dynamics might be:

- Pre-determination of the service discovery process
- Restriction of dynamic memory allocation to the startup phase only
- Fair scheduling policy in addition to priority-based scheduling
- Fixed allocation of processes to CPU cores
- Access to pre-existing files in the file-system only
- Constraints for AUTOSAR Adaptive Platform API usage by applications
- Execution of authenticated code only

## 8.7 Verification and Validation

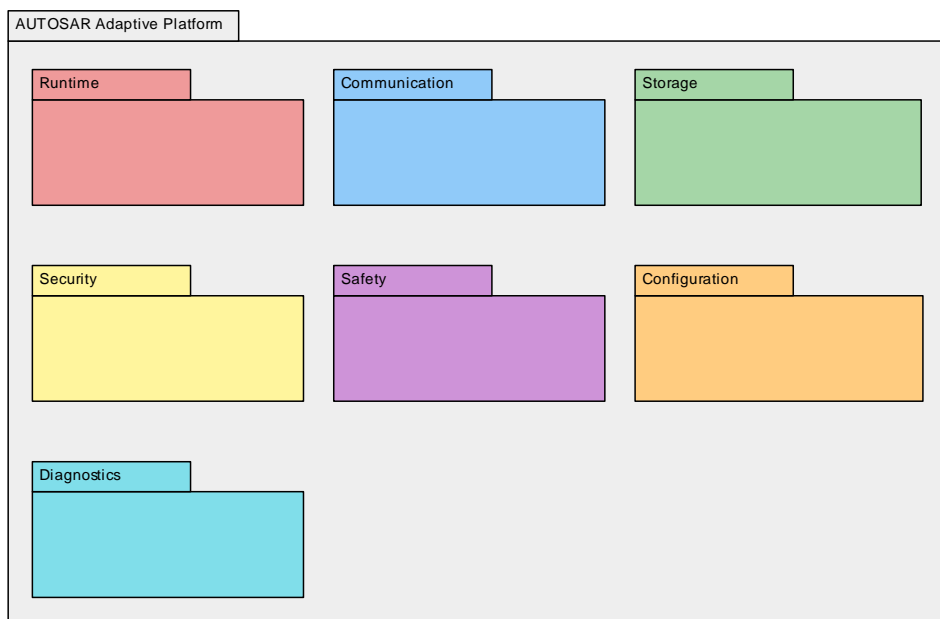
The AUTOSAR Adaptive Platform standard uses a dedicated implementation of the standard (AUTOSAR Adaptive Platform Demonstrator) to validate the requirements and to verify the (still abstract) software design imposed by the individual software specifications.

## 9 Building Block View

This chapter provides an overview of the static structure of the AUTOSAR Adaptive Platform by describing the high-level building blocks and their inter-dependencies. Please note that the use of interfaces between `FunctionalClusters` in the AUTOSAR Adaptive Platform is currently not standardized. Some aspects, for example, access management, are also not yet fully incorporated and standardized in all `FunctionalClusters`.

### 9.1 Overview

Figure 9.1 provides an overview of the different categories of building blocks available in the AUTOSAR Adaptive Platform. The categories are explained in more detail in the subsequent sections.



**Figure 9.1: Overview of the AUTOSAR Adaptive Platform and its building block categories**

#### 9.1.1 Description pattern

The description of all building blocks (`FunctionalClusters`) in this section uses the same pattern. Each `FunctionalCluster` is described in a separate section of the document. Such a section starts with an overview of the responsibilities of the `FunctionalCluster` followed by a sub-section called "Defined interfaces". The sub-section "Defined interfaces" lists all architectural interfaces specified in the namespace of the `FunctionalCluster`. Each interface is detailed in a separate table.

The subsection "Provided interfaces" then lists all interfaces provided by the `FunctionalCluster` (i.e. it is fully implemented by the `FunctionalCluster`) to other `FunctionalClusters`.

The last subsection "Required interfaces" lists all interfaces required by the `FunctionalCluster` from other `FunctionalClusters` and external components like the operating system.

## 9.2 Runtime

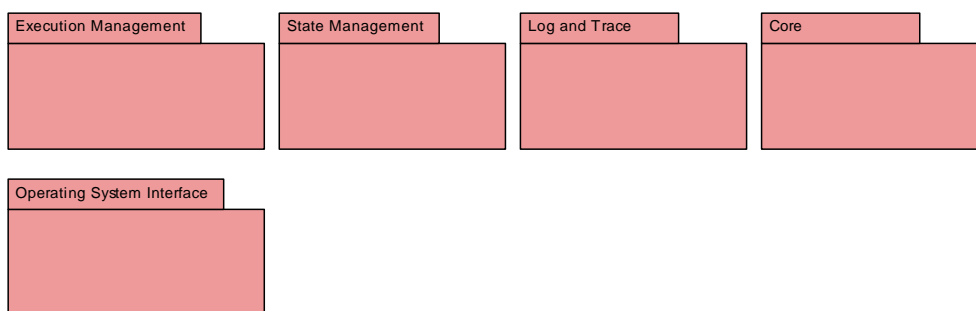


Figure 9.2: Overview of Runtime and its building blocks

### 9.2.1 Execution Management

<b>Name:</b>	Execution Management
<b>Short name:</b>	exec
<b>Category:</b>	Runtime
<b>Daemon-based:</b>	Yes
<b>Responsibilities:</b>	<p>Execution Management is responsible to control Processes of the AUTOSAR Adaptive Platform and Adaptive Applications. That is, it starts, configures, and stops Processes as configured in Function Group States using interfaces of the Operating System. The Operating System is responsible for runtime scheduling of those Process. The configuration of Processes that Execution Management performs includes limiting their resource consumption (CPU time, memory) using Resource Groups provided by the Operating System.</p> <p>Execution Management is the entry point of AUTOSAR Adaptive Platform and is started by the Operating System during system boot. Execution Management then controls the startup and shutdown of the AUTOSAR Adaptive Platform (see Section 10.2.1 for details). Execution Management</p>



△

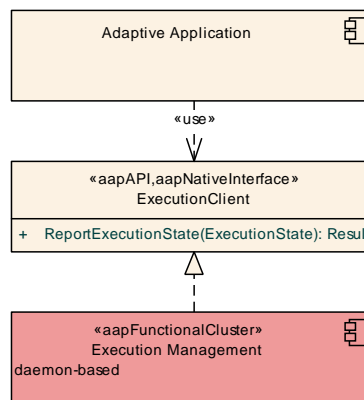
	<p style="text-align: center;">△</p> <p>optionally supports authenticated startup where it maintains the chain of trust when starting from a Trust Anchor. During authenticated startup Execution Management validates the authenticity and integrity of Processes and shall prevent their execution if violations are detected. Through these mechanisms, a trusted platform can be established (cf. Section 12.13).</p>
--	---

### 9.2.1.1 Defined interfaces

The interfaces of Execution Management are categorized into interfaces for state reporting (see Section 9.2.1.1.1), interfaces for the deterministic execution (see Section 9.2.1.1.2), and interfaces for State Management (see Section 9.2.1.1.3).

#### 9.2.1.1.1 Interfaces for state reporting

All processes started by Execution Management (i.e. all processes of the AUTOSAR Adaptive Platform and all processes of Adaptive Applications) shall report their execution state back to Execution Management via the ExecutionClient interface (cf. Figure 9.3).

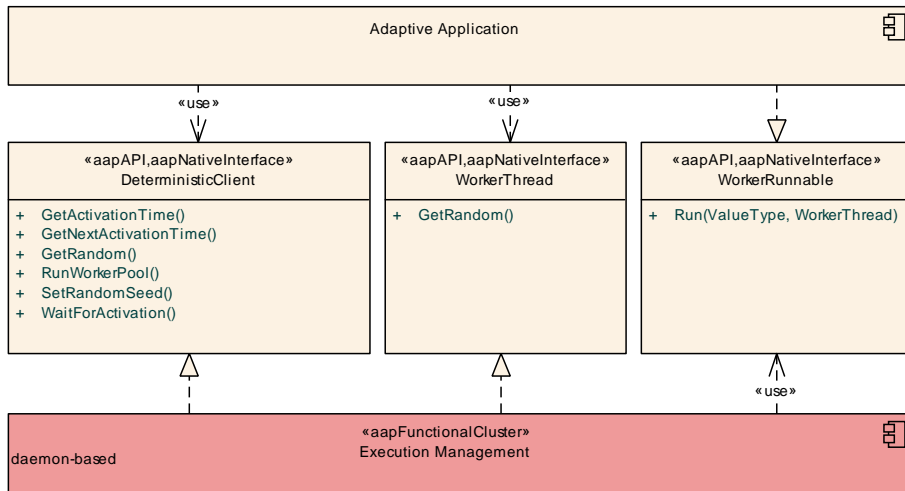


**Figure 9.3: Interfaces for state reporting**

<b>Name:</b>	ExecutionClient	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides functionality for a Process to report its execution state to Execution Management.	
<b>Operations:</b>	ReportExecutionState	Report the internal state of a Process to Execution Management.

**9.2.1.1.2 Interfaces for deterministic execution**

The [DeterministicClient](#) API (cf. Figure 9.4) provides operations to perform deterministic execution of tasks.



**Figure 9.4: Interfaces for deterministic execution**

<b>Name:</b>	DeterministicClient	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides the functionality for an application to run a cyclic deterministic execution. Each modelled <code>Process</code> which needs support for cyclic deterministic execution has to instantiate this interface.	
<b>Operations:</b>	GetActivationTime	Get the timestamp of the activation point.
	GetNextActivationTime	Get the timestamp of the next activation point.
	GetRandom	Returns a deterministic sequence of random numbers.
	RunWorkerPool	Runs tasks in a deterministic worker pool.
	SetRandomSeed	Seed the random number generator used for redundantly executed deterministic clients.
	WaitForActivation	Blocks and returns with a process control value when the next activation is triggered by the runtime.

<b>Name:</b>	WorkerRunnable	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	





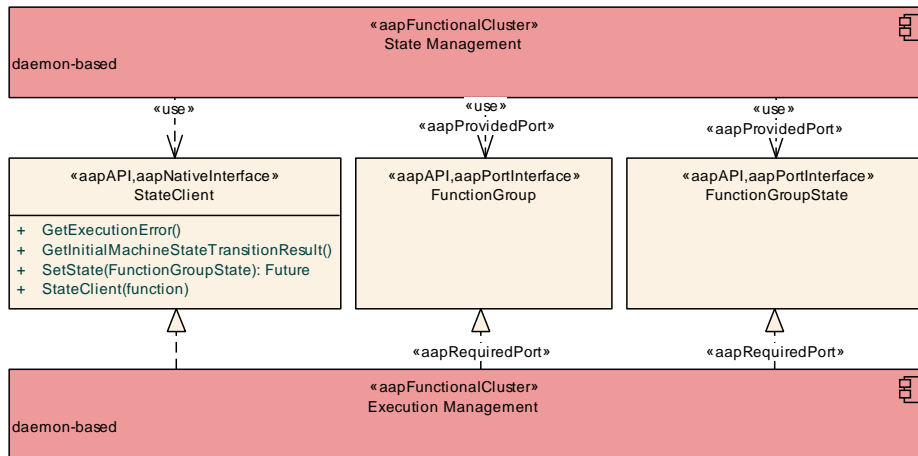


<b>Description:</b>	This interface is used to implement worker runnables for the <a href="#">DeterministicClient</a> .	
<b>Operations:</b>	Run	Runs the deterministic client worker runnable.

<b>Name:</b>	WorkerThread	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface is used to implement worker threads for the <a href="#">DeterministicClient</a> .	
<b>Operations:</b>	GetRandom	Returns a deterministic pseudo-random number which is unique for each container element in the worker pool.

### 9.2.1.1.3 Interfaces for State Management

The [StateClient](#) API (cf. Figure 9.5) provides operations to control [FunctionGroups](#) and their [FunctionGroupStates](#).



**Figure 9.5: Interfaces for State Management**

<b>Name:</b>	StateClient	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides the functionality to request <a href="#">FunctionGroupState</a> transitions and to perform error detection and error handling.	



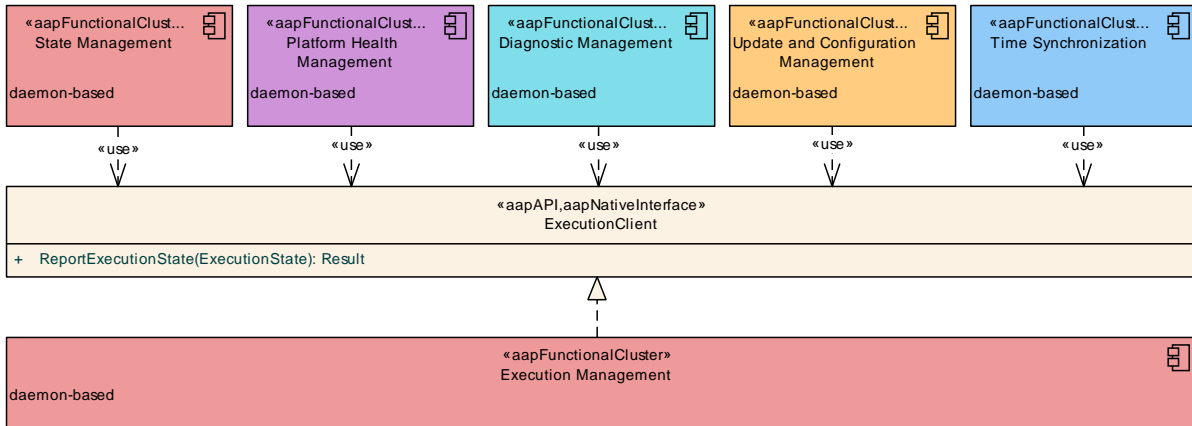


<b>Operations:</b>	GetExecutionError	Returns the execution error which changed the given <a href="#">FunctionGroup</a> to an undefined <a href="#">FunctionGroupState</a> .
	GetInitialMachineStateTransitionResult	Retrieve the result of <code>Machine State</code> initial transition to <code>Startup</code> state.
	SetState	Request a <a href="#">FunctionGroupState</a> transition for a single <a href="#">FunctionGroup</a> .
	StateClient	Constructor of this interface. It requires a callback to be invoked if a <a href="#">FunctionGroup</a> changes its state unexpectedly to an undefined <a href="#">FunctionGroupState</a> , i.e. without previous request by <code>SetState()</code> . The affected <a href="#">FunctionGroup</a> is provided as an argument to the callback.

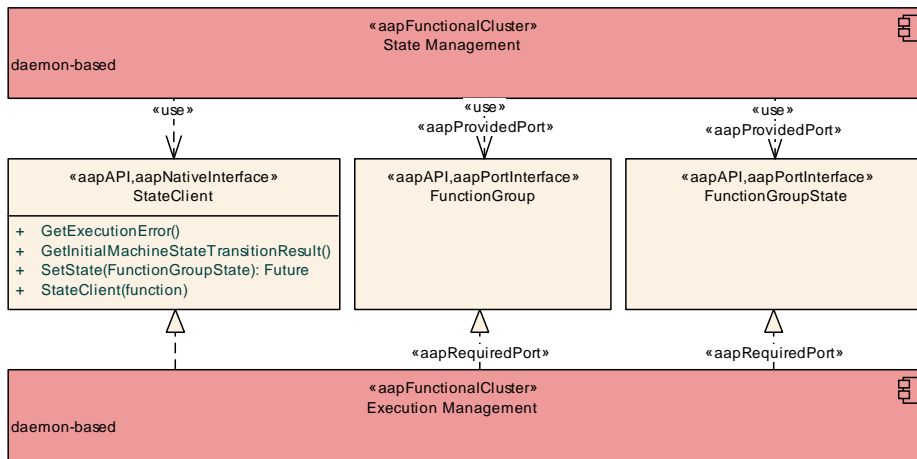
<b>Name:</b>	FunctionGroupState
<b>Technology:</b>	Port interface
<b>Generated:</b>	No
<b>Meta-model interface type:</b>	ModeDeclaration
<b>Usage:</b>	Public API
<b>Description:</b>	Represents a <code>Function Group State</code> defined in the Manifest.

<b>Name:</b>	FunctionGroup
<b>Technology:</b>	Port interface
<b>Generated:</b>	No
<b>Meta-model interface type:</b>	ModeDeclarationGroup
<b>Usage:</b>	Public API
<b>Description:</b>	Represents a <code>Function Group</code> defined in the Manifest.

**9.2.1.2 Provided interfaces**



**Figure 9.6: Users of the ExecutionClient interface**

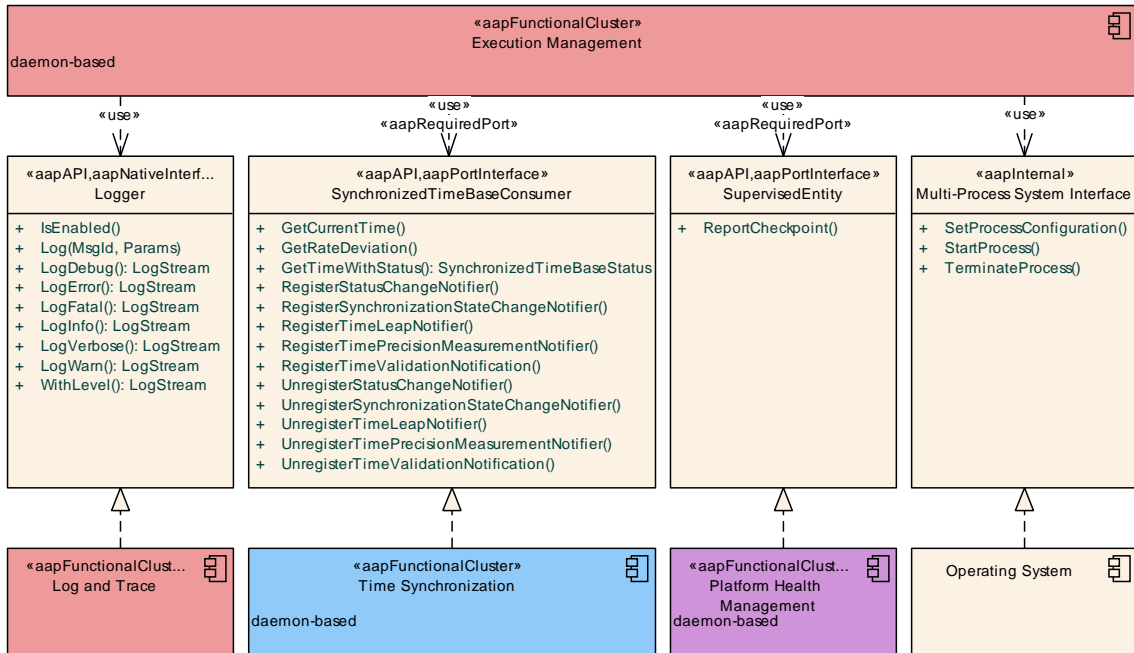


**Figure 9.7: Users of the State Management interfaces**

<b>Interface</b>	<b>Requiring functional clusters</b>
ExecutionClient	Diagnostic Management (see Section 9.8.1)
	Platform Health Management (see Section 9.6.1)
	State Management (see Section 9.2.2)
	Time Synchronization (see Section 9.3.3)
	Update and Configuration Management (see Section 9.7.1)
FunctionGroup	State Management (see Section 9.2.2)
FunctionGroupState	State Management (see Section 9.2.2)
StateClient	State Management (see Section 9.2.2)

**Table 9.1: Interfaces provided by Execution Management to other Functional Clusters**

**9.2.1.3 Required interfaces**



**Figure 9.8: Interfaces required by Execution Management**

Interface	Purpose
n/a	Execution Management should use this interface to read/write persistent data.
KeyValueStorage	Execution Management should use this interface to read/write persistent data.
Logger	Execution Management shall use this interface to log standardized messages.
Manifest Accessor	Execution Management shall use this interface to read the configuration of its <b>DeterministicClient</b> and information about Function Groups and Processes from the Manifests.
Multi-Process System Interface	Execution Management shall use this interface to start, configure and control os-level processes.
SupervisedEntity	Execution Management shall use this interface to enable supervision of its process(es) by Platform Health Management.
SynchronizedTimeBaseConsumer	The <b>DeterministicClient</b> implementation in Execution Management should use this interface to synchronize execution of <b>DeterministicClients</b> .
WorkerRunnable	Execution Management shall use this interface to execute <b>WorkerRunnables</b> using its <b>DeterministicClient</b> implementation.

**Table 9.2: Interfaces required by Execution Management**

## 9.2.2 State Management

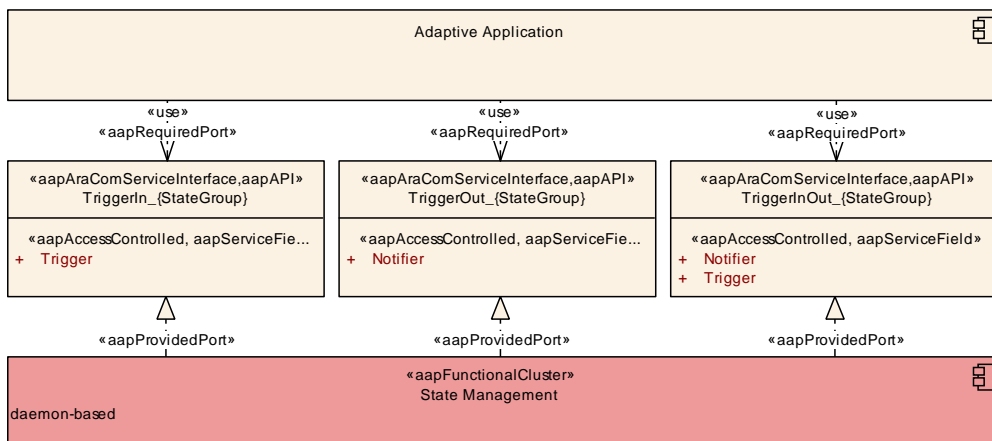
<b>Name:</b>	State Management
<b>Short name:</b>	sm
<b>Category:</b>	Runtime
<b>Daemon-based:</b>	Yes
<b>Responsibilities:</b>	<p>State Management determines the desired target state of the AUTOSAR Runtime for Adaptive Applications based on various application-specific inputs. That target state is the set of active Function Group States of all Function Groups running on the AUTOSAR Runtime for Adaptive Applications. State Management delegates to Execution Management to switch the individual Function Groups to the respective Function Group States.</p> <p>State Management is a unique component in the AUTOSAR Adaptive Platform because it is not part of an AUTOSAR Adaptive Platform stack. The logic of State Management currently needs to be implemented as application-specific code and then configured and integrated with an AUTOSAR Adaptive Platform stack.</p>

### 9.2.2.1 Defined interfaces

The interfaces of State Management are categorized into interfaces for triggering state changes (see Section 9.2.2.1.1), interfaces for diagnostic reset (see Section 9.2.2.1.2), interfaces for requesting a Power Mode (see Section 9.2.2.1.3), and interfaces for interaction with Update and Configuration Management (see Section 9.2.2.1.4).

#### 9.2.2.1.1 Interfaces for triggering state changes

State Management provides several interface blueprints to get and set its internal state (cf. Figure 9.9).



**Figure 9.9: Interfaces for triggering state changes**

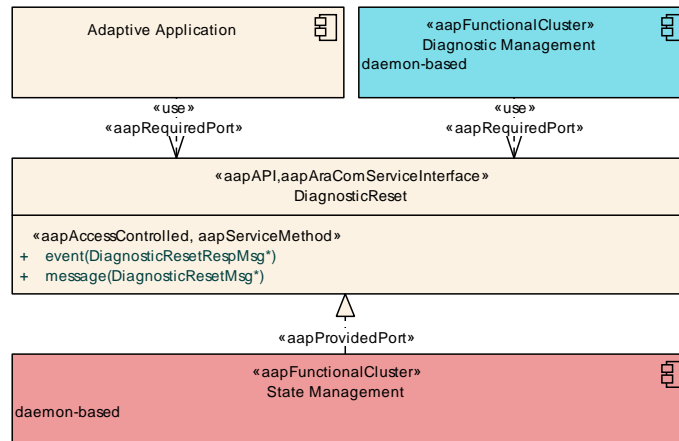
<b>Name:</b>	TriggerIn_{StateGroup}	
<b>Technology:</b>	ara::com service interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface is to be used by Adaptive Applications to trigger State Management to change its internal state.	
<b>Fields:</b>	Trigger	A value to be evaluated by State Management in a project-specific way.

<b>Name:</b>	TriggerOut_{StateGroup}	
<b>Technology:</b>	ara::com service interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface is to be used by Adaptive Applications to be informed when State Management has changed its internal state.	
<b>Fields:</b>	Notifier	To be set by State Management in a project-specific way to inform Adaptive Applications about changes within State Management.

<b>Name:</b>	TriggerInOut_{StateGroup}	
<b>Technology:</b>	ara::com service interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface is to be used by Adaptive Applications to trigger State Management to change its internal state and to get information when it is carried out.	
<b>Fields:</b>	Notifier	To be set by State Management in a project-specific way to inform Adaptive Applications about changes within State Management.
	Trigger	A value to be evaluated by State Management in a project-specific way.

### 9.2.2.1.2 Interfaces for requesting a diagnostic reset

The [DiagnosticReset](#) interface propagates a diagnostic reset request ([DiagnosticReset::message\(\)](#)) to all affected `Processes`. These `Processes` then shall answer the diagnostic reset request by calling [DiagnosticReset::event\(\)](#).

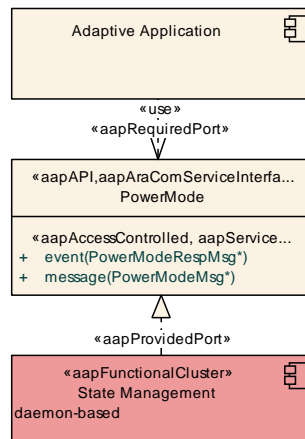


**Figure 9.10: Interface for handling a diagnostic reset**

<b>Name:</b>	DiagnosticReset	
<b>Technology:</b>	ara::com service interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides functionality to handle diagnostic reset requests.	
<b>Operations:</b>	event	All Processes which got a diagnostic reset request shall call this method to provide an answer to State Management.
	message	Sends a diagnostic reset message to all affected Processes.

### 9.2.2.1.3 Interfaces for requesting a Power Mode

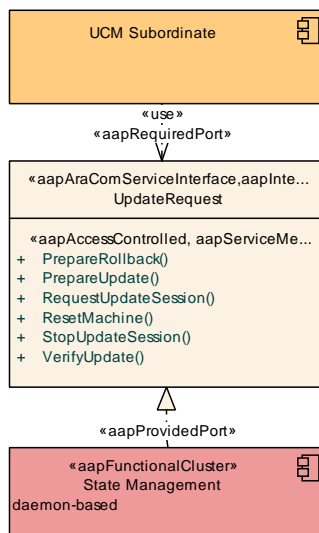
The **PowerMode** interface propagates a diagnostic Power Mode request (**PowerMode::message()**) to all running Processes. These Processes then shall answer the Power Mode request by calling **PowerMode::event()**.



**Figure 9.11: Interface for handling a Power Mode request**

<b>Name:</b>	PowerMode	
<b>Technology:</b>	ara::com service interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides functionality to handle <code>Power Mode</code> requests.	
<b>Operations:</b>	event	All Processes which have received a <code>Power Mode</code> request shall call this method to provide an answer to <code>State Management</code> .
	message	Sends a <code>Power Mode</code> request to all running Processes.

### 9.2.2.1.4 Interfaces for interaction with Update and Configuration Management



**Figure 9.12: Interface for software update handling**

<b>Name:</b>	UpdateRequest	
<b>Technology:</b>	ara::com service interface	
<b>Usage:</b>	Internal	
<b>Description:</b>	This interface is intended to be used by <code>Update and Configuration Management</code> to interact with <code>State Management</code> to perform updates (including installation and removal) of <code>Software Clusters</code> .	
<b>Operations:</b>	PrepareRollback	Prepares the affected <code>Function Groups</code> for a rollback.
	PrepareUpdate	Prepares the affected <code>Function Groups</code> for an update.

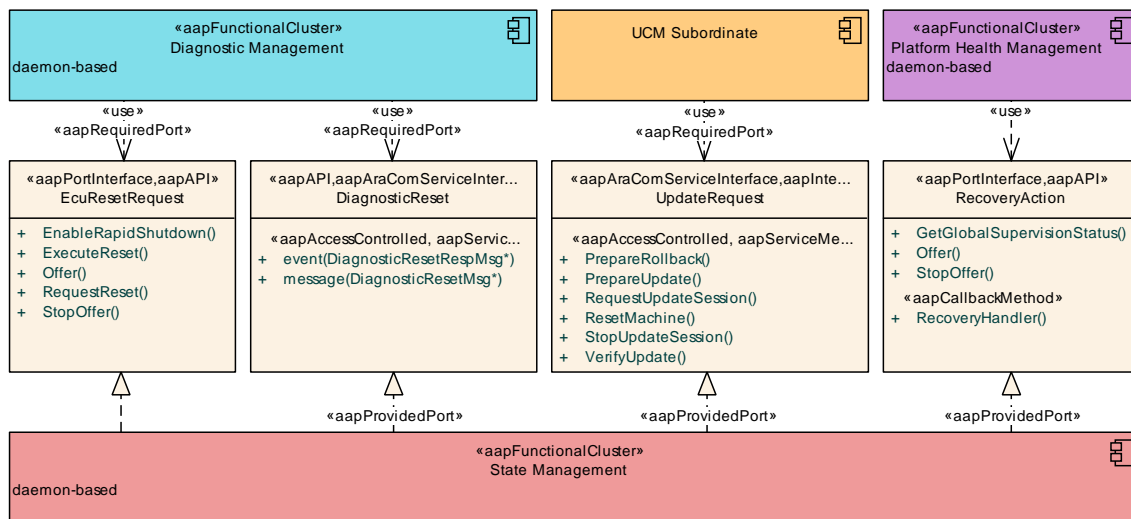






	RequestUpdateSession	Requests an update session. State Management might decline this request when the Machine is not in a state to be updated.
	ResetMachine	Requests an orderly reset of the Machine. Before the reset is performed all information within the Machine shall be persisted.
	StopUpdateSession	Ends an update session.
	VerifyUpdate	Verifies the affected Function Groups after an update.

### 9.2.2.2 Provided interfaces

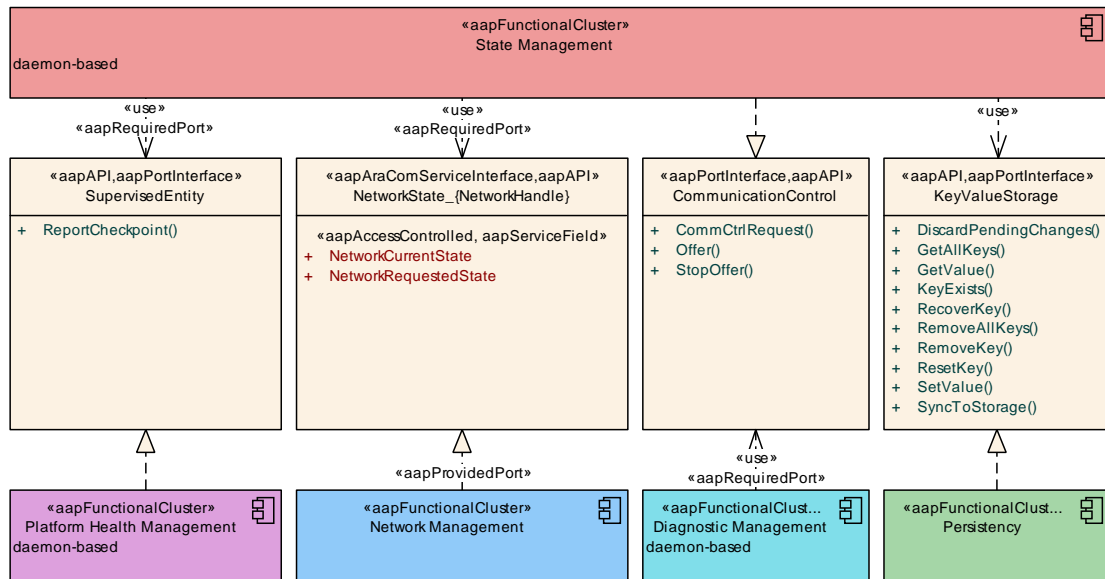


**Figure 9.13: Users of the State Management interfaces**

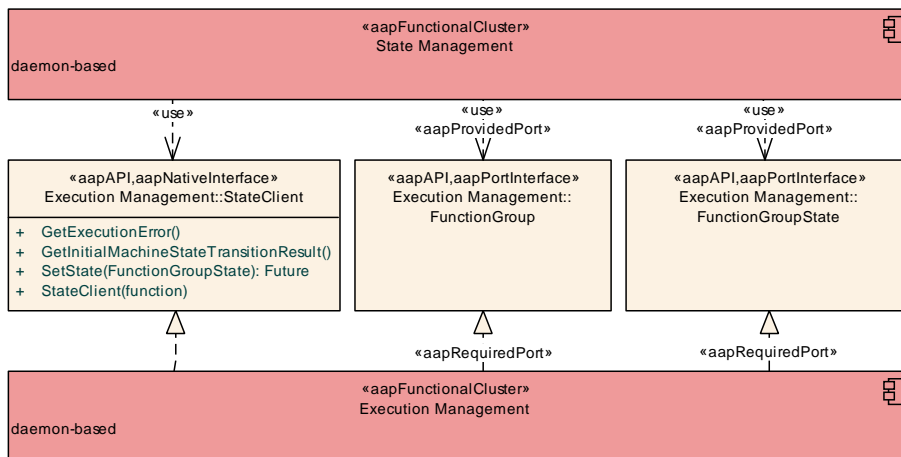
<b>Interface</b>	<b>Requiring functional clusters</b>
CommunicationControl	Diagnostic Management (see Section 9.8.1)
DiagnosticReset	Diagnostic Management (see Section 9.8.1)
EcuResetRequest	Diagnostic Management (see Section 9.8.1)
RecoveryAction	Platform Health Management (see Section 9.6.1)
UpdateRequest	Update and Configuration Management (see Section 9.7.1)

**Table 9.3: Interfaces provided by State Management to other Functional Clusters**

**9.2.2.3 Required interfaces**



**Figure 9.14: Interfaces required by State Management**



**Figure 9.15: Interfaces of Execution Management required by State Management**

Interface	Purpose
n/a	This interface should be used to persist information (e.g. update session).
ExecutionClient	This interface shall be used to report the state of the StateManagement process(es).
FunctionGroup	This interface shall be used to request FunctionGroupState transitions and to check their status.
FunctionGroupState	This interface shall be used to request FunctionGroupState transitions and to check their status.





<a href="#">KeyValueStorage</a>	This interface should be used to persist information (e.g. update session).
<a href="#">Logger</a>	State Management shall use this interface to log standardized messages.
<a href="#">NetworkState_{NetworkHandle}</a>	This interface shall be used to retrieve information about the network status of a NetworkHandle.
<a href="#">StateClient</a>	This interface shall be used to request FunctionGroupState transitions.
<a href="#">SupervisedEntity</a>	State Management shall use this interface to enable supervision of its process(es) by Platform Health Management.

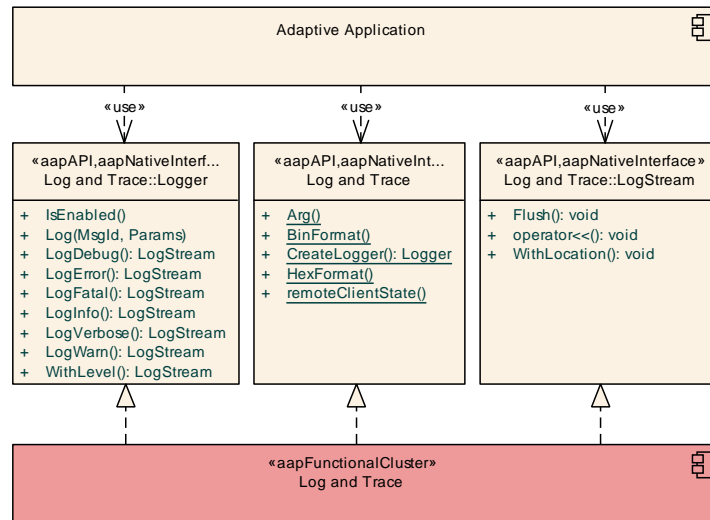
**Table 9.4: Interfaces required by State Management**

### 9.2.3 Log and Trace

<b>Name:</b>	Log and Trace
<b>Short name:</b>	log
<b>Category:</b>	Runtime
<b>Daemon-based:</b>	No
<b>Responsibilities:</b>	Log and Trace provides functionality to build and log messages of different severity. An Adaptive Application can be configured to forward log messages to various sinks, for example to a network, a serial bus, the console, and to non-volatile storage.

#### 9.2.3.1 Defined interfaces

The entry point to the Log and Trace framework is the [CreateLogger\(\)](#) operation that constructs a new [Logger](#) context. Afterwards, new messages can be constructed using the [LogStream](#) that is returned by the operations in [Logger](#), for example [LogInfo\(\)](#).



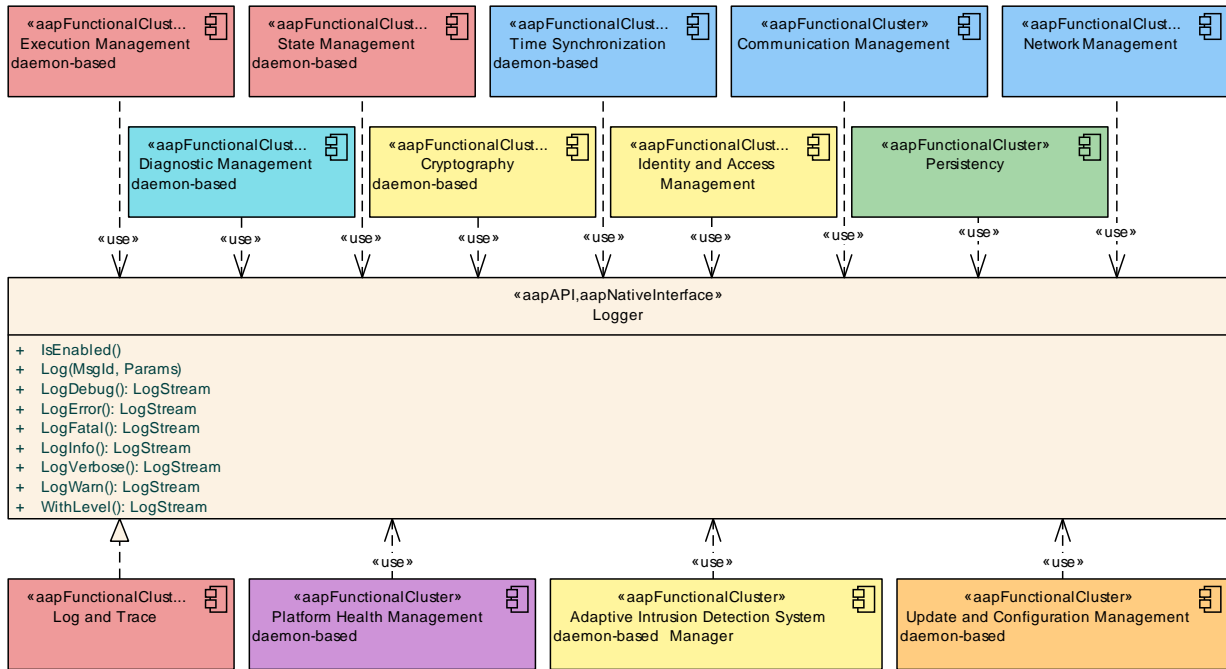
**Figure 9.16: Interfaces of Log and Trace**

<b>Name:</b>	Logger	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface represents a logger context. The logging framework defines contexts which can be seen as logger instances within one application process or process scope. A context will be automatically registered against the Logging back-end during creation phase, as well as automatically deregistered during process shutdown phase.	
<b>Operations:</b>	IsEnabled	Check if the provided log level is enabled in the current configured log level.
	Log	Logs a message modeled in the Manifest.
	LogDebug	Creates a <a href="#">LogStream</a> object with <code>Debug</code> severity.
	LogError	Creates a <a href="#">LogStream</a> object with <code>Error</code> severity.
	LogFatal	Creates a <a href="#">LogStream</a> object with <code>Fatal</code> severity.
	LogInfo	Creates a <a href="#">LogStream</a> object with <code>Info</code> severity.
	LogVerbose	Creates a <a href="#">LogStream</a> object with <code>Verbose</code> severity.
	LogWarn	Creates a <a href="#">LogStream</a> object with <code>Warn</code> severity.
	WithLevel	Creates a <a href="#">LogStream</a> object with the provided log level.

<b>Name:</b>	n/a	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides access to the logging framework and utility operations to control the format of value printed to the log output.	
<b>Operations:</b>	Arg	Create a wrapper object. The wrapper object holds a value and an optional name and unit of the value.
	BinFormat	Conversion of an integer into a binary value. Negatives are represented in 2's complement. The number of represented digits depends on the overloaded parameter type length.
	CreateLogger	Creates a <a href="#">Logger</a> object, holding the context which is registered in the logging framework.
	HexFormat	Conversion of an integer into a hexadecimal value. Negatives are represented in 2's complement. The number of represented digits depends on the overloaded parameter type length.
	remoteClientState	Fetches the connection state from the <a href="#">DLT</a> back-end of a possibly available remote client.

<b>Name:</b>	LogStream	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides functionality to construct a single log message by appending data using stream operators.	
<b>Operations:</b>	Flush	Sends out the current log buffer and initiates a new message stream. Calling this operation is only necessary if the <a href="#">LogStream</a> is intended to be reused within the same scope.
	WithLocation	Add source file location into the message.
	operator«	Writes a value into the log message. Several overloads exist to control the output format.

**9.2.3.2 Provided interfaces**

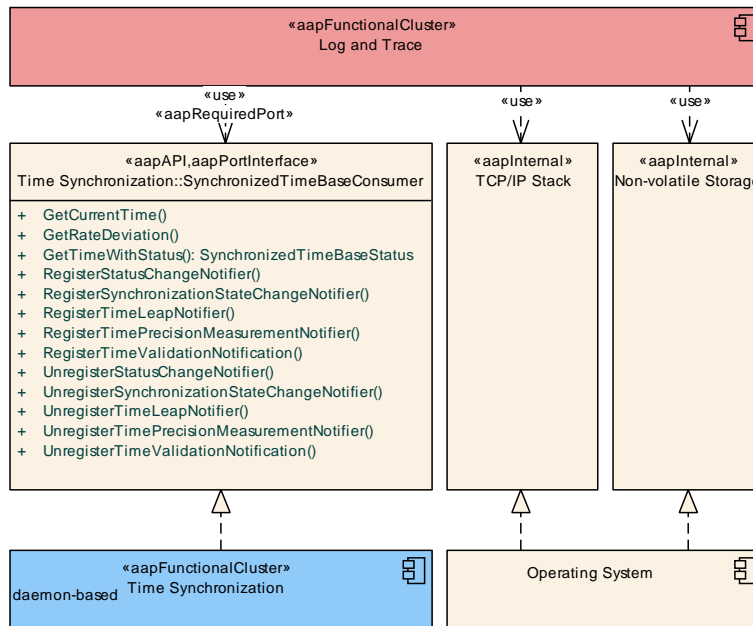


**Figure 9.17: Users of the Log and Trace interfaces**

<b>Interface</b>	<b>Requiring functional clusters</b>
Logger	Adaptive Intrusion Detection System Manager (see Section 9.5.3)
	Communication Management (see Section 9.3.1)
	Cryptography (see Section 9.5.1)
	Diagnostic Management (see Section 9.8.1)
	Execution Management (see Section 9.2.1)
	Identity and Access Management (see Section 9.5.2)
	Network Management (see Section 9.3.2)
	Persistency (see Section 9.4.1)
	Platform Health Management (see Section 9.6.1)
	State Management (see Section 9.2.2)
	Time Synchronization (see Section 9.3.3)
Update and Configuration Management (see Section 9.7.1)	

**Table 9.5: Interfaces provided by Log and Trace to other Functional Clusters**

**9.2.3.3 Required interfaces**



**Figure 9.18: Interfaces required by LogAndTrace**

<i>Interface</i>	<i>Purpose</i>
<a href="#">Manifest Accessor</a>	Log and Trace shall use this interface to read information about modeled messages from the Manifests.
Non-volatile Storage	Log and Trace should use this interface to write log messages to a non-volatile storage, e.g., a file in a filesystem.
<a href="#">SynchronizedTimeBaseConsumer</a>	Log and Trace shall use this interface to determine the timestamps that are associated with log messages.
TCP/IP Stack	Log and Trace shall use this interface to write log messages to an IP-based network stream.

**Table 9.6: Interfaces required by Log and Trace**

**9.2.4 Core**

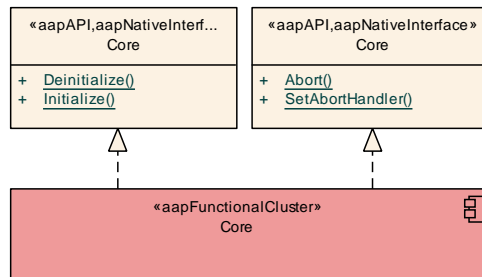
<b>Name:</b>	Core
<b>Short name:</b>	core
<b>Category:</b>	Runtime
<b>Daemon-based:</b>	No





<b>Responsibilities:</b>	Core provides functionality for initialization and de-initialization of the AUTOSAR Runtime for Adaptive Applications as well as termination of Processes.
--------------------------	--

### 9.2.4.1 Defined interfaces



**Figure 9.19: Interfaces of Core**

#### 9.2.4.1.1 Interfaces for initialization and de-initialization

The AUTOSAR Adaptive Platform for Applications needs to be initialized by an application before it is used (using [Initialize\(\)](#)) and de-initialized after it is no longer used (using [Deinitialize\(\)](#)).

<b>Name:</b>	n/a	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides global initialization and shutdown functions that initialize respectively deinitialize data structures and threads of the AUTOSAR Runtime for Adaptive Applications.	
<b>Operations:</b>	Deinitialize	Destroy all data structures and threads of the AUTOSAR Adaptive Runtime for Applications. After this call, no interaction with the AUTOSAR Adaptive Runtime for Applications is possible.
	Initialize	Initializes data structures and threads of the AUTOSAR Adaptive Runtime for Applications. Prior to this call, no interaction with the AUTOSAR Adaptive Runtime for Applications is possible.



### 9.2.4.1.2 Interfaces for process termination

The AUTOSAR Adaptive Platform for Applications provides an explicit abnormal termination facility using [Abort\(\)](#).

<b>Name:</b>	n/a	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides operation for abnormal termination of processes.	
<b>Operations:</b>	Abort	Abort the current process. This function will never return to its caller.
	SetAbortHandler	Set a custom global abort handler function and return the previously installed one.

### 9.2.4.2 Provided interfaces

`Core` currently provides no interfaces to other `Functional Clusters`.

### 9.2.4.3 Required interfaces

`Core` currently requires no interfaces.

## 9.2.5 Operating System Interface

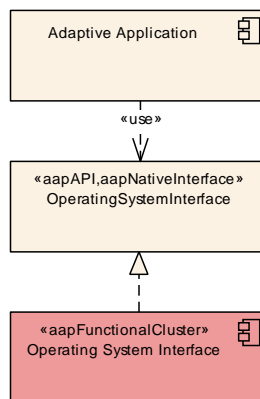
<b>Name:</b>	Operating System Interface
<b>Short name:</b>	n/a
<b>Category:</b>	Runtime
<b>Daemon-based:</b>	No
<b>Responsibilities:</b>	The <code>Operating System Interface</code> provides functionality for implementing multi-threaded real-time embedded applications and corresponds to the [9, POSIX PSE51 profile]. That profile provides support to create <code>Threads</code> that may be executed in parallel on modern multi-core processors and to control their properties such as stack memory or their scheduling. In addition, primitives for shared resource access are provided such as <code>Semaphores</code> or memory locking. Asynchronous (real-time) signals and message passing enable inter-process communication. High resolution timers and clocks are provided to control real-time behavior precisely. Some input/output functions are provided as well but no file system APIs.





	<p>△</p> <p>POSIX PSE51 and the Operating System Interface do not provide any means to execute and control Processes. Processes (of the AUTOSAR Adaptive Platform) are entirely controlled by Execution Management via interfaces that are implementation specific.</p> <p>Note that a typical AUTOSAR Adaptive Platform stack will not provide an actual implementation of the Operating System Interface because all functionality is already provided by standard libraries of the programming language (e.g. Standard C++ Library).</p>
--	---

### 9.2.5.1 Defined interfaces



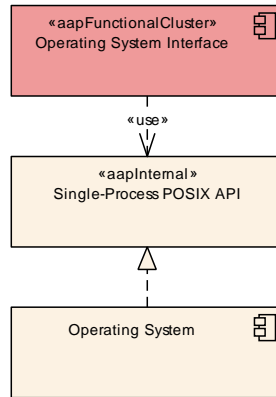
**Figure 9.20: Interfaces defined by Operating System Interface**

<b>Name:</b>	OperatingSystemInterface
<b>Technology:</b>	Native interface
<b>Usage:</b>	Public API
<b>Description:</b>	This interface represents the [9, POSIX PSE51 profile] API. The API is not detailed in this document.

### 9.2.5.2 Provided interfaces

Operating System Interface currently provides no interfaces to other Functional Clusters.

### 9.2.5.3 Required interfaces

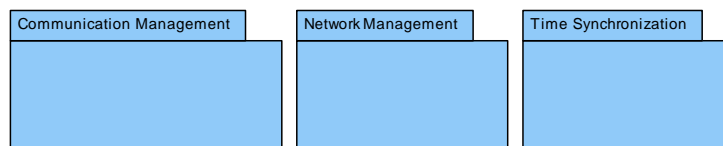


**Figure 9.21: Interfaces required by Operating System Interface**

<b>Interface</b>	<b>Purpose</b>
Single-Process POSIX API	Operating System Interface uses this interface to provide the functionality of the [9, POSIX PSE51 profile], e.g. Threads. Usually the whole [9, POSIX PSE51 profile] API will already be provided by the underlying operating system. In this case, the Operating System Interface will not have an implementation in the Adaptive Platform stack.

**Table 9.7: Interfaces required by Operating System Interface**

## 9.3 Communication



**Figure 9.22: Overview of Communication and its building blocks**

### 9.3.1 Communication Management

<b>Name:</b>	Communication Management
<b>Short name:</b>	com
<b>Category:</b>	Communication



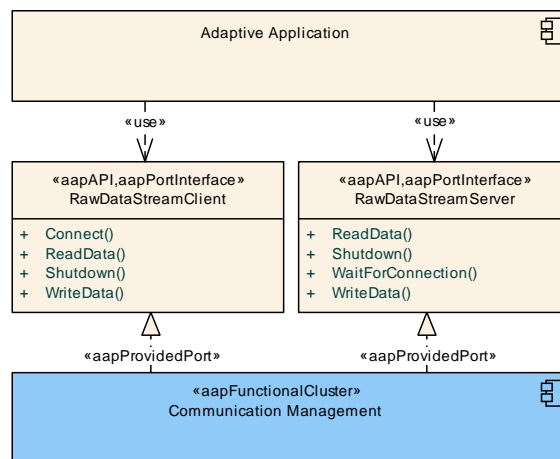


<b>Daemon-based:</b>	No
<b>Responsibilities:</b>	Communication Management is responsible for all levels of service-oriented and raw communication between applications in a distributed real-time embedded environment. That is, intra-process communication, inter-process communication and inter-machine communication. The latter is also possible with AUTOSAR Classic Platforms and third-party platforms. Communication paths can be established at design-, start-up-, and run-time. Communication Management consists of a generic part that handles brokering and configuration as well as (potentially generated) skeletons for service providers and respective proxies for service consumers.

### 9.3.1.1 Defined interfaces

The interfaces of Communication Management are categorized into interfaces for raw data streams (see Section 9.3.1.1.1), interfaces for SecOC (see Section 9.3.1.1.2), and interfaces freshness value management (see Section 9.3.1.1.3). Please note that a implementation of Communication Management will generate additional interfaces for each modeled Service, e.g. a Proxy and a Skeleton interface. However, these generated interfaces are not covered in this document.

#### 9.3.1.1.1 Interfaces for raw data streams



**Figure 9.23: Interfaces for raw data streams**

<b>Name:</b>	RawDataStreamClient
<b>Technology:</b>	Port interface
<b>Generated:</b>	No
<b>Meta-model interface type:</b>	RawDataStreamClientInterface

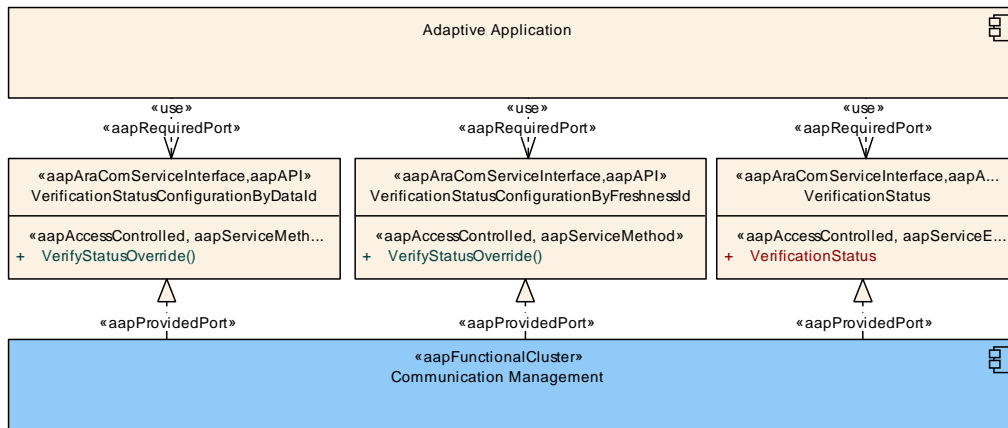




<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides functionality for a client reading and writing binary data streams over a network connection.	
<b>Operations:</b>	Connect	Sets up a socket connection for the raw data stream defined by the instance, and establish a connection to the TCP server.
	ReadData	Requests to read a number of bytes of data from the socket connection for the raw data stream defined by the instance.
	Shutdown	Closes the socket connection for the raw data stream defined by the instance. Both, the receiving and the sending part of the socket connection shall be shut down.
	WriteData	Requests to write of a number of bytes to the socket connection for the raw data stream defined by the instance.

<b>Name:</b>	RawDataStreamServer	
<b>Technology:</b>	Port interface	
<b>Generated:</b>	No	
<b>Meta-model interface type:</b>	RawDataStreamServerInterface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides functionality for a server reading and writing binary data streams over a network connection.	
<b>Operations:</b>	ReadData	Requests to read a number of bytes of data from the socket connection for the raw data stream defined by the instance.
	Shutdown	Closes the socket connection for the raw data stream defined by the instance. Both the receiving and the sending part of the socket connection shall be shut down.
	WaitForConnection	Initializes the socket and prepare the <a href="#">RawDataStreamServer</a> instance for incoming connections.
	WriteData	Requests to write of a number of bytes to the socket connection for the raw data stream defined by the instance.

**9.3.1.1.2 Interfaces for SecOC**



**Figure 9.24: Interfaces for SecOC**

<b>Name:</b>	VerificationStatus	
<b>Technology:</b>	ara::com service interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides an event to informed about the verification status of messages.	
<b>Events:</b>	VerificationStatus	This event is fired for each verification and holds the verification status.

<b>Name:</b>	VerificationStatusConfigurationByDataId	
<b>Technology:</b>	ara::com service interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides functionality to control the verification status of messages.	
<b>Operations:</b>	VerifyStatusOverride	This service method provides the ability to force to accept or to drop a message with or without performing the verification of authenticator or independent of the authenticator verification result, and to force a specific result allowing additional fault handling in the application.

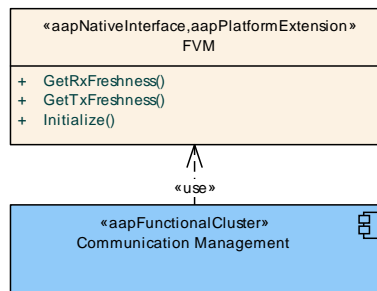
<b>Name:</b>	VerificationStatusConfigurationByFreshnessId	
<b>Technology:</b>	ara::com service interface	





<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides functionality to control the verification status of messages.	
<b>Operations:</b>	VerifyStatusOverride	This service method provides the ability to force to accept or to drop a message with or without performing the verification of authenticator or independent of the authenticator verification result, and to force a specific result allowing additional fault handling in the application.

### 9.3.1.1.3 Interfaces for freshness value management



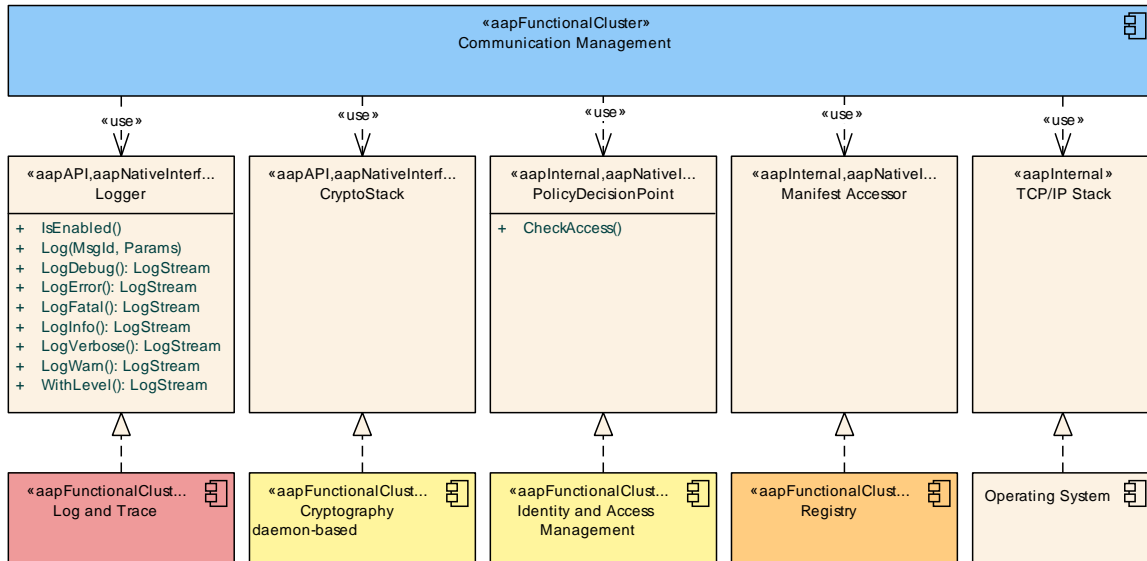
**Figure 9.25: Interfaces for freshness value management**

<b>Name:</b>	FVM	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Platform extension	
<b>Description:</b>	This interface provides functionality for freshness value management.	
<b>Operations:</b>	GetRxFreshness	Obtain the current freshness value for received messages.
	GetTxFreshness	Obtain the current freshness value for transmitted messages.
	Initialize	Initializes freshness value manager plugin implementation.

### 9.3.1.2 Provided interfaces

CommunicationManagement currently provides no interfaces to other Functional Clusters.

**9.3.1.3 Required interfaces**



**Figure 9.26: Interfaces required by Communication Management**

<b>Interface</b>	<b>Purpose</b>
CryptoStack	Communication Management shall use this interface to establish encrypted connections and generate / verify checksums (MAC).
EventReporter	This interface should be used to report security events.
FVM	Communication Management shall use this interface to get freshness values.
Logger	Communication Management shall use this interface to log standardized messages.
Manifest Accessor	Communication Management shall use this interface to read service information from the Manifests.
OperatingSystemInterface	Communication Management should use this interface to create and control Threads used by the implementation.
PolicyDecisionPoint	Communication Management shall use this interface to check access to ara::com service methods, fields, and events.
TCP/IP Stack	Communication Management shall use this interface to establish and control TCP/IP-based network connections.

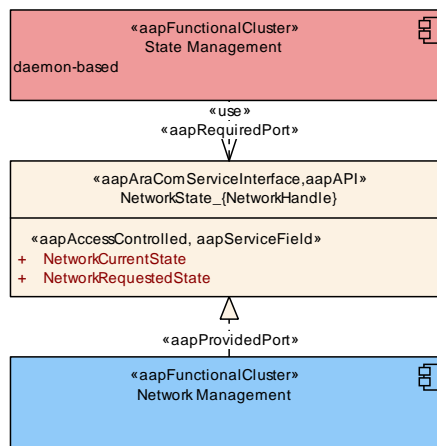
**Table 9.8: Interfaces required by Communication Management**



### 9.3.2 Network Management

<b>Name:</b>	Network Management
<b>Short name:</b>	nm
<b>Category:</b>	Communication
<b>Daemon-based:</b>	No
<b>Responsibilities:</b>	Network Management provides functionality to request and query the network states for logical network handles, Such network handles can be mapped to physical or partial networks.

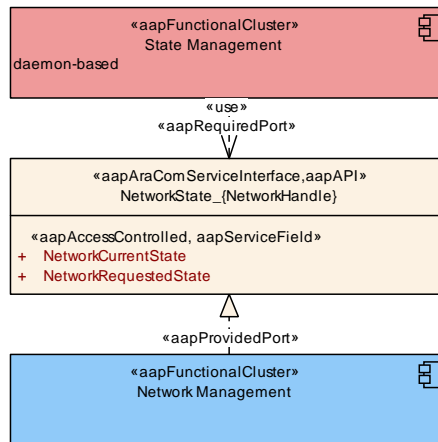
#### 9.3.2.1 Defined interfaces



**Figure 9.27: Interfaces for time base providers**

<b>Name:</b>	NetworkState_{NetworkHandle}	
<b>Technology:</b>	ara::com service interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides information about network status per NetworkHandle. This interface is intended to be used by StateManagement only.	
<b>Fields:</b>	NetworkCurrentState	Indicates if a PNC / VLAN / Physical Network is currently active or not.
	NetworkRequestedState	Request a PNC / VLAN / Physical Network to get active or to release it.

**9.3.2.2 Provided interfaces**

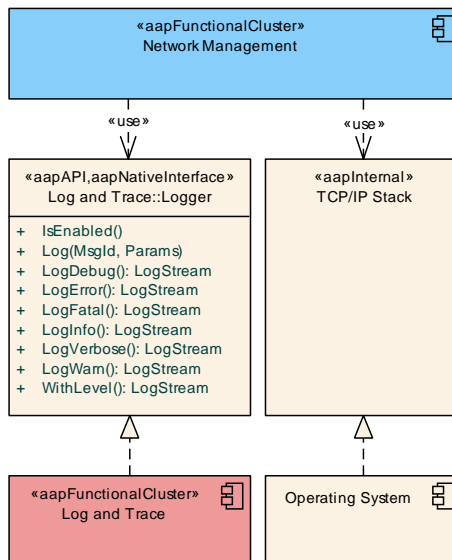


**Figure 9.28: Users of Network Management interfaces**

<i>Interface</i>	<i>Requiring functional clusters</i>
NetworkState_{NetworkHandle}	State Management (see Section 9.2.2)

**Table 9.9: Interfaces provided by Network Management to other Functional Clusters**

**9.3.2.3 Required interfaces**



**Figure 9.29: Interfaces required by Network Management**

<b>Interface</b>	<b>Purpose</b>
<a href="#">Logger</a>	Network Management shall use this interface to log standardized messages.
<a href="#">Manifest Accessor</a>	Network Management shall use this interface to read information about NmNetworkHandles from the Manifests.
TCP/IP Stack	Network Management should use this interface to set and to determine the status of IP-based networks.

**Table 9.10: Interfaces required by Network Management**

### 9.3.3 Time Synchronization

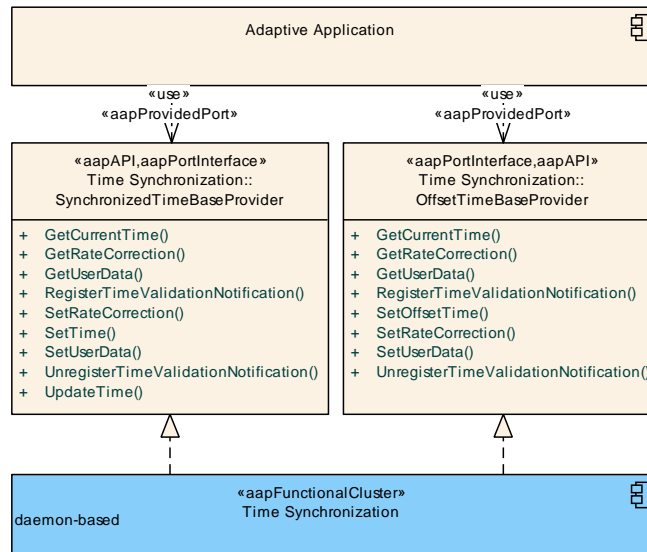
<b>Name:</b>	Time Synchronization
<b>Short name:</b>	tsync
<b>Category:</b>	Communication
<b>Daemon-based:</b>	Yes
<b>Responsibilities:</b>	Time Synchronization provides synchronized time information in distributed applications. Synchronized time information between different applications and/or Machines is of paramount importance when the correlation of different events across a distributed system is needed, either to be able to track such events in time or to trigger them at an accurate point in time.

#### 9.3.3.1 Defined interfaces

The interfaces of Time Synchronization are categorized into interfaces for providing time information (see Section 9.3.3.1.1) and interfaces for consuming time information (see Section 9.3.3.1.2).

##### 9.3.3.1.1 Interfaces for time base providers

Time Synchronization defines the [SynchronizedTimeBaseProvider](#) and [OffsetTimeBaseProvider](#) interfaces to provide time information for a synchronized time base.



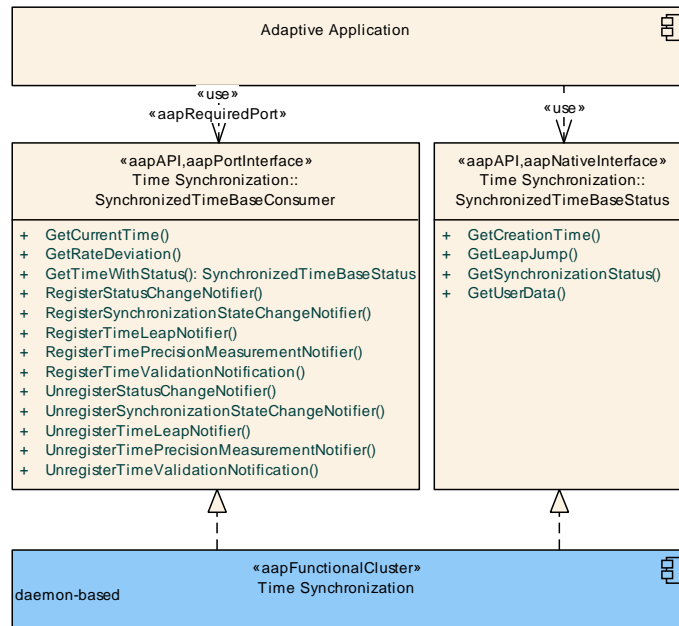
**Figure 9.30: Interfaces for time base providers**

<b>Name:</b>	SynchronizedTimeBaseProvider	
<b>Technology:</b>	Port interface	
<b>Generated:</b>	No	
<b>Meta-model interface type:</b>	SynchronizedTimeBaseProviderInterface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Provides access to a synchronized time base. It allows to get the current time point, the rate deviation, the current status and the received user data.	
<b>Operations:</b>	GetCurrentTime	Obtain the current time (regardless of the current sync status).
	GetRateCorrection	Obtain the current rate deviation of the clock.
	GetUserData	Get the user defined data of the time base.
	RegisterTimeValidationNotification	Used by time provider applications to receive time sync parameters. A maximum of one notifier can be registered. Every further registration overwrites the current registration.
	SetRateCorrection	Set the rate correction that will be applied to time values.
	SetTime	Set a new time value for the clock. Setting a new time also triggers transmission on the bus.
	SetUserData	Set the user data of the time base.
	UnregisterTimeValidationNotification	Used by time provider applications to receive time sync parameters.
	UpdateTime	Set a new time value for the clock. The clock value is only updated locally, transmission on the bus will happen in the next cycle.

<b>Name:</b>	OffsetTimeBaseProvider	
<b>Technology:</b>	Port interface	
<b>Generated:</b>	No	
<b>Meta-model interface type:</b>	SynchronizedTimeBaseProviderInterface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Provides access to the offset time base. It allows to get the current time point, the rate deviation, the current status and the received user data.	
<b>Operations:</b>	GetCurrentTime	Get the current time (regardless of the current sync status).
	GetRateCorrection	Obtain the current rate deviation of the clock.
	GetUserData	Get the user defined data of the time base.
	RegisterTimeValidationNotification	Used by time provider applications to receive time sync parameters. A maximum of one notifier can be registered. Every further registration overwrites the current registration.
	SetOffsetTime	Set a new offset time value for the clock. Setting a new time also triggers transmission on the bus.
	SetRateCorrection	Set the rate correction that will be applied to time values.
	SetUserData	Set the user data of the time base.
	UnregisterTimeValidationNotification	Used by time provider applications to receive time sync parameters.

### 9.3.3.1.2 Interfaces for time base consumers

Time Synchronization defines the [SynchronizedTimeBaseConsumer](#) interface to retrieve time information for a synchronized time base. [SynchronizedTimeBaseStatus](#) is used to determine the status of a synchronized time base.



**Figure 9.31: Interfaces for time base consumers**

<b>Name:</b>	SynchronizedTimeBaseConsumer	
<b>Technology:</b>	Port interface	
<b>Generated:</b>	No	
<b>Meta-model interface type:</b>	SynchronizedTimeBaseConsumerInterface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Provides access to the synchronized time base. It allows to get the current time point, the rate deviation, the current status and the received user data.	
<b>Operations:</b>	GetCurrentTime	Obtain the current time (regardless of the current sync status).
	GetRateDeviation	Obtain the current rate deviation of the clock.
	GetTimeWithStatus	Obtain a snapshot of the current state of the clock. This includes status flags, clock configuration and the actual time value of the created status object.
	RegisterStatusChangeNotifier	Register a notifier function which is called if a status flag is changed (i.e. synchronization state, time leap or userdata). A maximum of one notifier can be registered. Every further registration overwrites the current registration.
	RegisterSynchronizationStateChangeNotifier	Register a notifier function which is called if a synchronization state is changed. A maximum of one notifier can be registered. Every further registration overwrites the current registration.

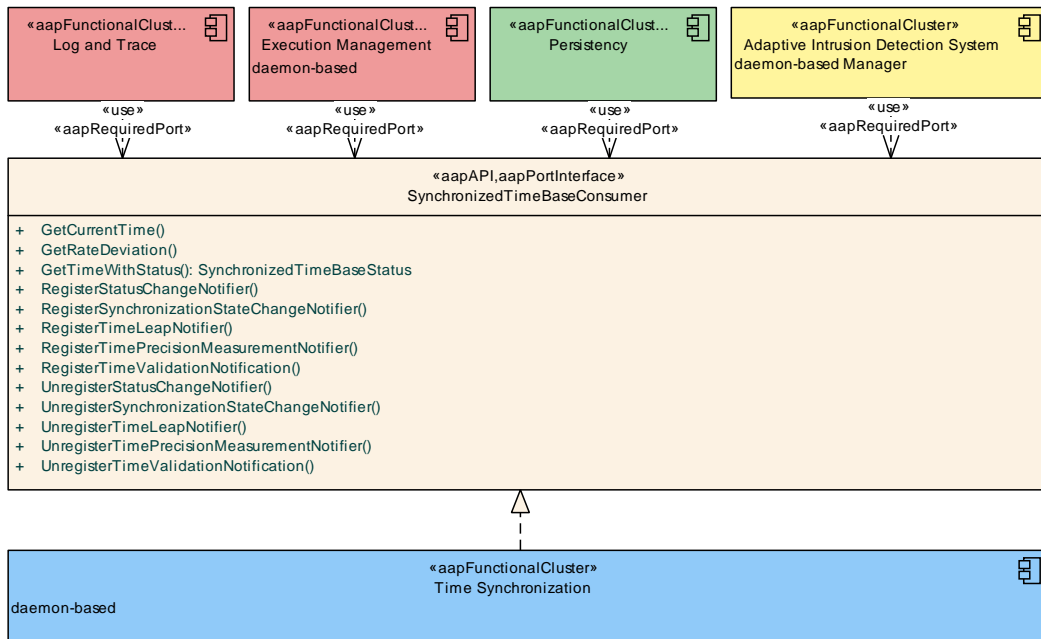




	RegisterTimeLeapNotifier	Register a notifier function which is called if a time leap happened. A maximum of one notifier can be registered. Every further registration overwrites the current registration.
	RegisterTimePrecisionMeasurementNotifier	Register a notifier function which is called if a new time precision snapshot is available. A maximum of one notifier can be registered. Every further registration overwrites the current registration. <i>Time Synchronization</i> will not do any queuing. If needed it has to be done within the notifier.
	RegisterTimeValidationNotification	Used by time consumer applications to receive time sync parameters. A maximum of one notifier can be registered. Every further registration overwrites the current registration.
	UnregisterStatusChangeNotifier	Unregister a notifier function which is called if a status flag is changed (i.e. synchronization state, time leap or userdata).
	UnregisterSynchronizationStateChangeNotifier	Unregister a notifier function which is called if a synchronization state is changed.
	UnregisterTimeLeapNotifier	Unregister a notifier function which is called if a time leap happened.
	UnregisterTimePrecisionMeasurementNotifier	Unregister a notifier function which is called if a new time precision snapshot is available.
	UnregisterTimeValidationNotification	Unregister a notifier function for receiving time sync parameters.

<b>Name:</b>	SynchronizedTimeBaseStatus	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Represents a snapshot of a time point including its states.	
<b>Operations:</b>	GetCreationTime	Obtain the creation time of this object.
	GetLeapJump	Determine the direction of a leap jump. Only the jump until the previous object creation is included.
	GetSynchronizationStatus	Returns the synchronization state when the object was created.
	GetUserData	Returns the user defined data of the time base.

**9.3.3.2 Provided interfaces**



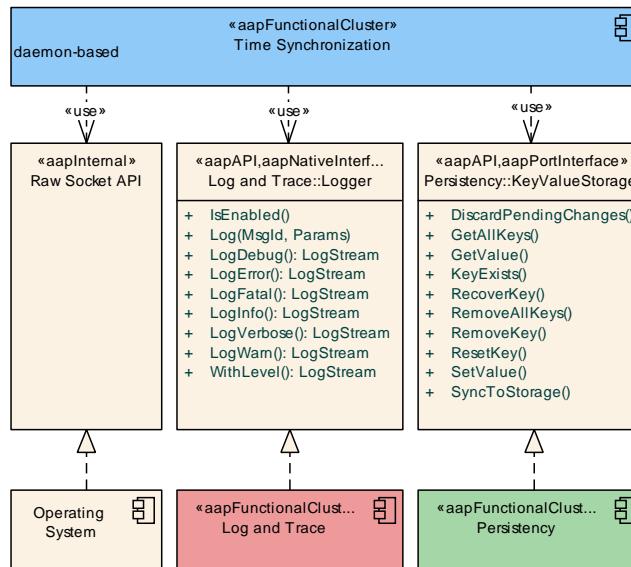
**Figure 9.32: Users of Time Synchronization interfaces**

<b>Interface</b>	<b>Requiring functional clusters</b>
SynchronizedTimeBaseConsumer	Adaptive Intrusion Detection System Manager (see Section 9.5.3)
	Execution Management (see Section 9.2.1)
	Log and Trace (see Section 9.2.3)
	Persistency (see Section 9.4.1)

**Table 9.11: Interfaces provided by Time Synchronization to other Functional Clusters**



**9.3.3.3 Required interfaces**



**Figure 9.33: Interfaces required by Time Synchronization**

<b>Interface</b>	<b>Purpose</b>
n/a	Time Synchronization should use this interface to persist the last received timestamp to enable a faster startup.
ExecutionClient	Time Synchronization shall use this interface to report the state of its daemon process.
KeyValueStorage	Time Synchronization should use this interface to persist the last received timestamp to enable a faster startup.
Logger	Time Synchronization shall use this interface to log standardized messages.
Manifest Accessor	Time Synchronization shall use this interface to read information about TimeBaseResources as well as their providers and consumers from the Manifests.
Raw Socket API	Time Synchronization should use this interface to send and receive raw ethernet packets as required by the time synchronization protocol.
SupervisedEntity	Time Synchronization should use this interface to enable supervision of its daemon process by Platform Health Management

**Table 9.12: Interfaces required by Time Synchronization**

## 9.4 Storage



**Figure 9.34: Overview of Storage and its building blocks**

### 9.4.1 Persistency

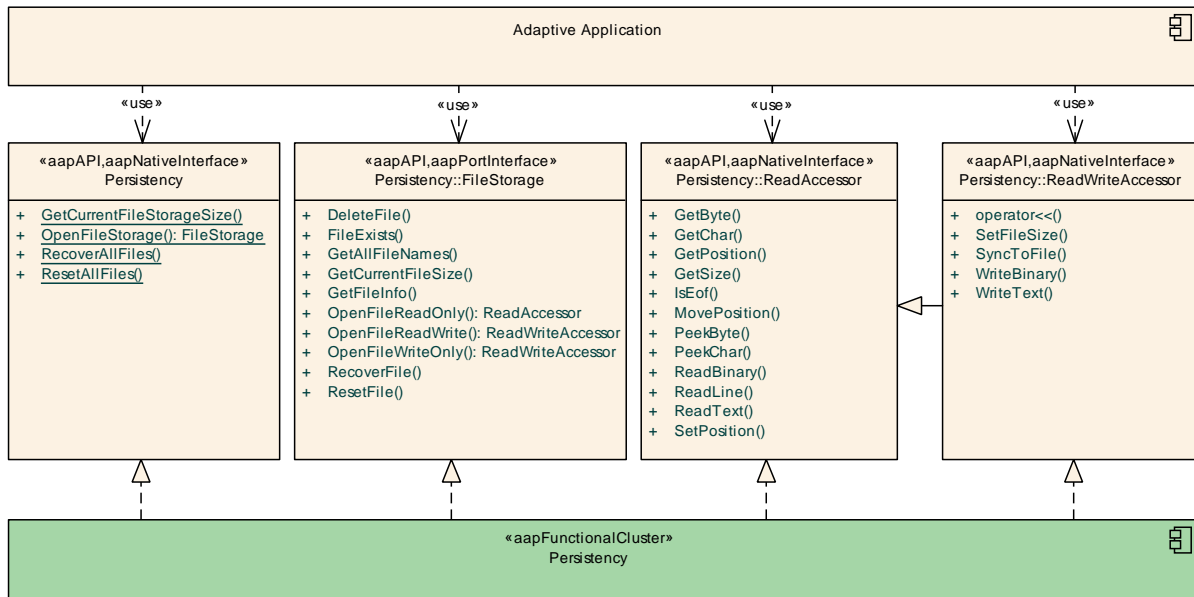
<b>Name:</b>	Persistency
<b>Short name:</b>	per
<b>Category:</b>	Storage
<b>Daemon-based:</b>	No
<b>Responsibilities:</b>	<p>Persistency provides functionality to store and retrieve information to/from non-volatile storage of a Machine.</p> <p>Persistent data is always private to one Process and is persisted across boot and ignition cycles. There is no mechanism available to share data between different Processes using Persistency to prevent a second path of data exchange besides Communication Management. However, Persistency supports concurrent access from multiple threads of the same application running in the context of the same Process.</p> <p>Persistency offers integrity of the stored data and provides error detection and correction schemes. Persistency also offers confidentiality of the stored data using encryption.</p> <p>Persistency provides statistics, for example, the number of used resources.</p>

#### 9.4.1.1 Defined interfaces

The interfaces of Persistency are categorized into interfaces for file access (see Section 9.4.1.1.1), interfaces for a key-value-based data access (see Section 9.4.1.1.2) and interfaces for general management of persistent data (see Section 9.4.1.1.3).

##### 9.4.1.1.1 Interfaces for file storage

Persistency provides read and write access to plain files by means of a FileStorage (cf. Figure 9.35). A FileStorage has to be opened using OpenFileStorage. A FileStorage then provides access to several files using their name.



**Figure 9.35: Interfaces for file storage**

<b>Name:</b>	n/a	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides functions to open and manage <a href="#">FileStorages</a> .	
<b>Operations:</b>	GetCurrentFileStorageSize	Returns the space in bytes currently occupied by a <a href="#">FileStorage</a> .
	OpenFileStorage	Opens a <a href="#">FileStorage</a> .
	RecoverAllFiles	Recovers a <a href="#">FileStorage</a> including all files.
	ResetAllFiles	Resets a <a href="#">FileStorage</a> including all files.

<b>Name:</b>	FileStorage	
<b>Technology:</b>	Port interface	
<b>Generated:</b>	No	
<b>Meta-model interface type:</b>	PersistencyFileStorageInterface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides functions to open and manage files.	
<b>Operations:</b>	DeleteFile	Deletes a file from this <a href="#">FileStorage</a> .
	FileExists	Checks if a file exists in this <a href="#">FileStorage</a> .
	GetAllFileNames	Returns a list of all currently available files of this <a href="#">FileStorage</a> .





	GetCurrentFileSize	Returns the space in bytes currently occupied by the content of a file of this <a href="#">FileStorage</a> .
	GetFileInfo	Returns additional information on a file of this <a href="#">FileStorage</a> .
	OpenFileReadOnly	Opens a file of this <a href="#">FileStorage</a> for reading.
	OpenFileReadWrite	Opens a file of this <a href="#">FileStorage</a> for reading and writing.
	OpenFileWriteOnly	Opens a file of this <a href="#">FileStorage</a> for writing.
	RecoverFile	Recovers a file of this <a href="#">FileStorage</a> .
	ResetFile	Resets a file of this <a href="#">FileStorage</a> to its initial content.

<b>Name:</b>	ReadAccessor	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides functions to read text and binary data from a file.	
<b>Operations:</b>	GetByte	Returns the byte at the current position of the file, advancing the current position.
	GetChar	Returns the character at the current position of the file, advancing the current position.
	GetPosition	Returns the current position relative to the beginning of the file.
	GetSize	Returns the current size of a file in bytes.
	IsEof	Checks if the current position is at end of file.
	MovePosition	Moves the current position in the file relative to the origin.
	PeekByte	Returns the byte at the current position of the file.
	PeekChar	Returns the character at the current position of the file.
	ReadBinary	Reads all remaining bytes into a Vector of Byte, starting from the current position.
	ReadLine	Reads a complete line of characters into a String, advancing the current position accordingly.
	ReadText	Reads all remaining characters into a String, starting from the current position.
	SetPosition	Sets the current position relative to the beginning of the file.

<b>Name:</b>	ReadWriteAccessor	
<b>Extended interface:</b>	<a href="#">ReadAccessor</a>	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides functions to read and write text and binary data from / to a file.	
<b>Operations:</b>	SetFileSize	Reduces the size of the file to 'size', effectively removing the current content of the file beyond this size.
	SyncToFile	Triggers flushing of the current file content to the physical storage.
	WriteBinary	Writes binary data to the file.
	WriteText	Writes a string to the file.
	operator«	Writes a string to the file.

### 9.4.1.1.2 Interfaces for key-value storage

Persistency provides read and write access to data structured as key-value pairs by means of the [KeyValueStorage](#) API (cf. Figure 9.36). A [KeyValueStorage](#) has to be created by calling [OpenKeyValueStorage](#). A [KeyValueStorage](#) then provides access to data stored for individual keys using the [GetValue](#) and [SetValue](#) operations.

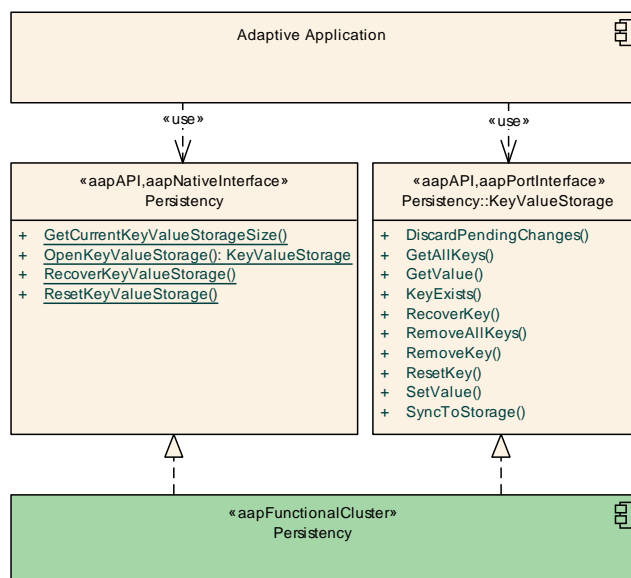


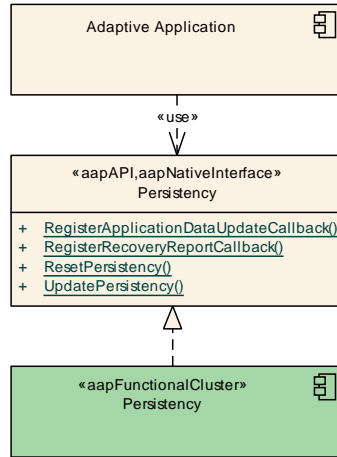
Figure 9.36: Interfaces for key-value-based data storage

<b>Name:</b>	n/a	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides functions to open and manage <a href="#">KeyValueStorages</a> .	
<b>Operations:</b>	GetCurrentKeyValueStorage-Size	Returns the space in bytes currently occupied by a <a href="#">KeyValueStorage</a> .
	OpenKeyValueStorage	Opens a <a href="#">KeyValueStorage</a> .
	RecoverKeyValueStorage	Recovers a <a href="#">KeyValueStorage</a> .
	ResetKeyValueStorage	Resets a <a href="#">KeyValueStorage</a> to the initial state.

<b>Name:</b>	KeyValueStorage	
<b>Technology:</b>	Port interface	
<b>Generated:</b>	No	
<b>Meta-model interface type:</b>	PersistencyKeyValueStorageInterface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides functions to access values associated with keys.	
<b>Operations:</b>	DiscardPendingChanges	Discards changed key-value pairs of the <a href="#">KeyValueStorage</a> and re-reads them from the physical storage.
	GetAllKeys	Returns a list of all currently available keys of this <a href="#">KeyValueStorage</a> .
	GetValue	Returns the value assigned to a key of this <a href="#">KeyValueStorage</a> .
	KeyExists	Checks if a key exists in this <a href="#">KeyValueStorage</a> .
	RecoverKey	Recovers a single key of this <a href="#">KeyValueStorage</a> .
	RemoveAllKeys	Removes all keys and associated values from this <a href="#">KeyValueStorage</a> .
	RemoveKey	Removes a key and the associated value from this <a href="#">KeyValueStorage</a> .
	ResetKey	Resets a key of this <a href="#">KeyValueStorage</a> to its initial value.
	SetValue	Stores a key in this <a href="#">KeyValueStorage</a> .
	SyncToStorage	Triggers flushing of changed key-value pairs of the <a href="#">KeyValueStorage</a> to the physical storage.

**9.4.1.1.3 Interfaces for general persistency handling**

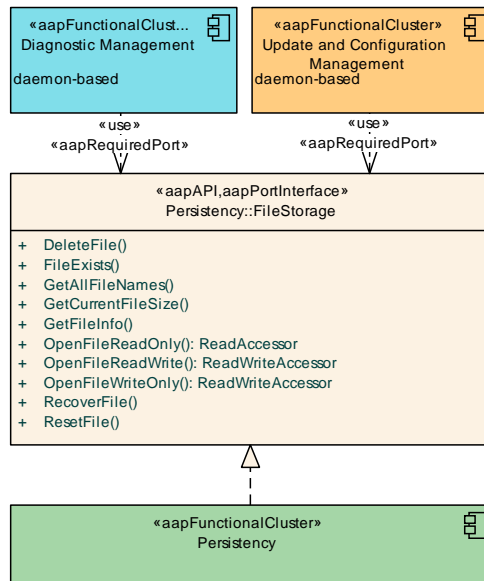
Persistency provides operations for handling and recovery of persistent data of a Process (cf. Figure 9.37).



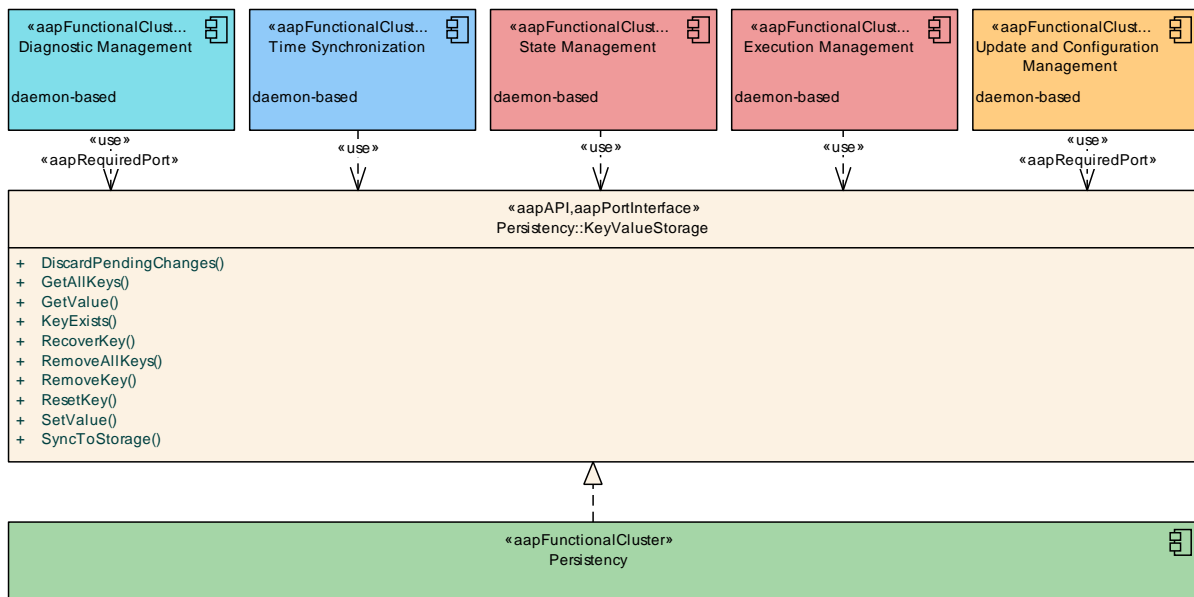
**Figure 9.37: Interfaces for general persistency handling**

<b>Name:</b>	n/a	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides operations manage persistent data.	
<b>Operations:</b>	RegisterApplicationDataUpdateCallback	Registers an application data update callback with Persistency.
	RegisterRecoveryReportCallback	Register a recovery reporting callback with Persistency.
	ResetPersistency	Resets all FileStorages and KeyValueStorages by entirely removing their content.
	UpdatePersistency	Updates all FileStorages and KeyValueStorages after a new manifest was installed.

**9.4.1.2 Provided interfaces**



**Figure 9.38: Users of the FileStorage interfaces**



**Figure 9.39: Users of the KeyValueStorage interfaces**

<b>Interface</b>	<b>Requiring functional clusters</b>
n/a	Diagnostic Management (see Section 9.8.1)
n/a	Update and Configuration Management (see Section 9.7.1)
n/a	Diagnostic Management (see Section 9.8.1)
n/a	Execution Management (see Section 9.2.1)



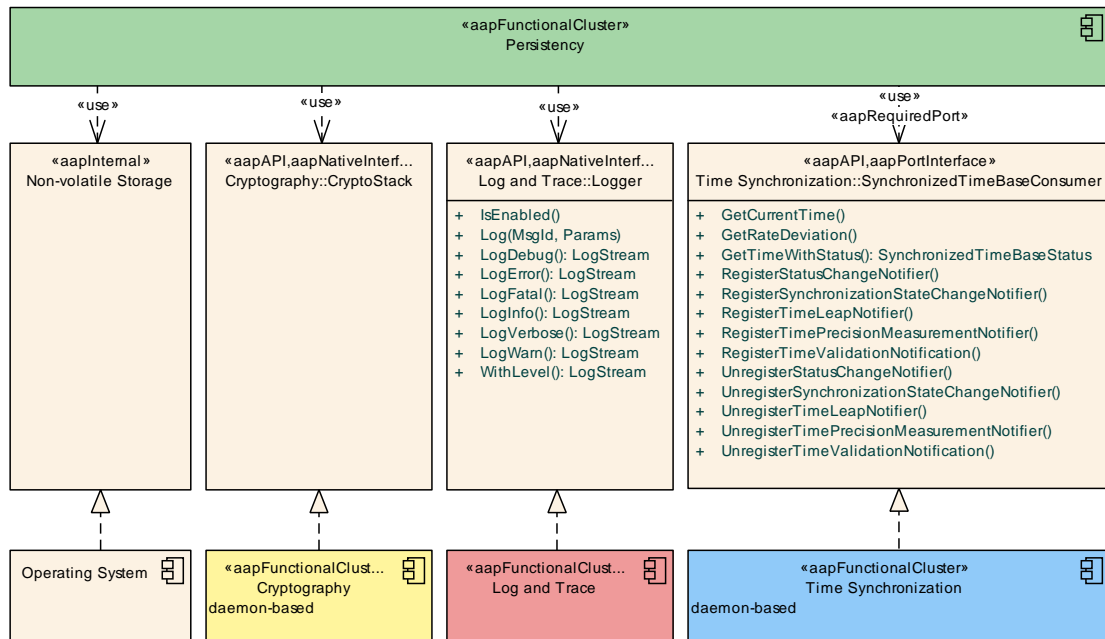


△

	State Management (see Section <a href="#">9.2.2</a> )
	Time Synchronization (see Section <a href="#">9.3.3</a> )
	Update and Configuration Management (see Section <a href="#">9.7.1</a> )
n/a	Diagnostic Management (see Section <a href="#">9.8.1</a> )
	Update and Configuration Management (see Section <a href="#">9.7.1</a> )
FileStorage	Diagnostic Management (see Section <a href="#">9.8.1</a> )
	Update and Configuration Management (see Section <a href="#">9.7.1</a> )
KeyValueStorage	Diagnostic Management (see Section <a href="#">9.8.1</a> )
	Execution Management (see Section <a href="#">9.2.1</a> )
	State Management (see Section <a href="#">9.2.2</a> )
	Time Synchronization (see Section <a href="#">9.3.3</a> )
	Update and Configuration Management (see Section <a href="#">9.7.1</a> )
ReadAccessor	Diagnostic Management (see Section <a href="#">9.8.1</a> )
	Update and Configuration Management (see Section <a href="#">9.7.1</a> )
ReadWriteAccessor	Diagnostic Management (see Section <a href="#">9.8.1</a> )
	Update and Configuration Management (see Section <a href="#">9.7.1</a> )

**Table 9.13: Interfaces provided by Persistency to other Functional Clusters**

**9.4.1.3 Required interfaces**

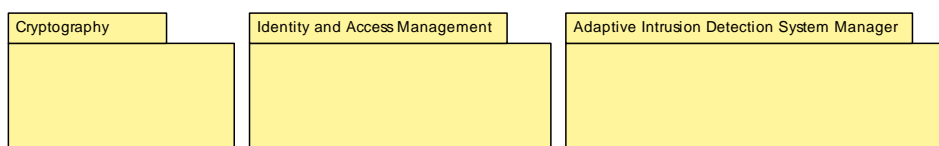


**Figure 9.40: Interfaces required by Persistency**

Interface	Purpose
CryptoStack	Persistency uses this interface to ensure confidentiality and integrity of the persisted data.
Logger	Persistency shall use this interface to log standardized messages.
Manifest Accessor	Persistency shall use this interface to read its configuration information from the Manifests.
Non-volatile Storage	Persistency uses this interface to access the non-volatile storage provided by the underlying operating system, for example, a file system.
SynchronizedTimeBaseConsumer	Persistency should use this interface to determine timestamps included in the meta-information of files, e.g., modification timestamp.

**Table 9.14: Interfaces required by Persistency**

**9.5 Security**



**Figure 9.41: Overview of Security and its building blocks**

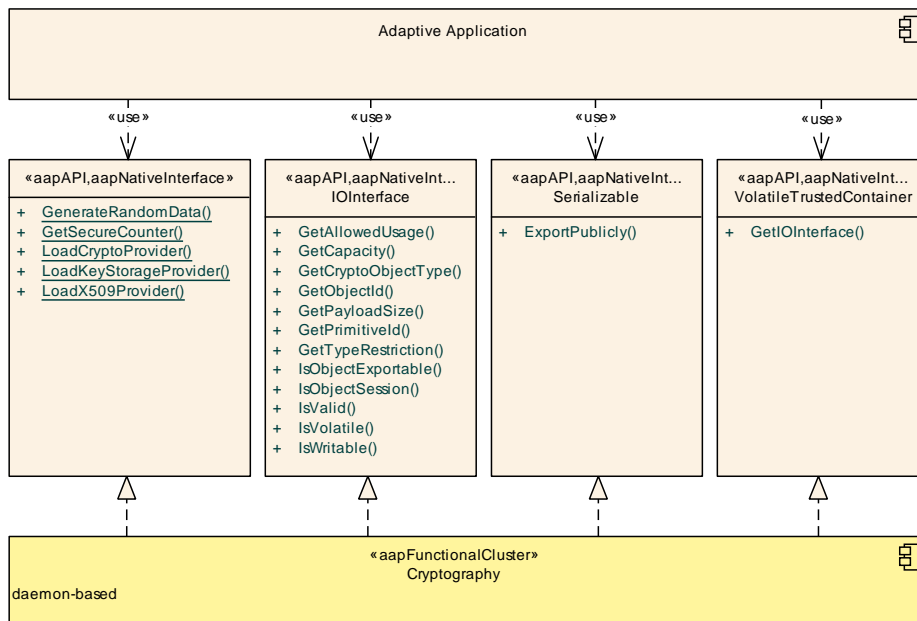
### 9.5.1 Cryptography

<b>Name:</b>	Cryptography
<b>Short name:</b>	crypto
<b>Category:</b>	Security
<b>Daemon-based:</b>	Yes
<b>Responsibilities:</b>	<p><code>Cryptography</code> provides various cryptographic routines to ensure confidentiality of data, to ensure integrity of data (e.g., using hashes), and auxiliary functions for example key management and random number generation. <code>Cryptography</code> is designed to support encapsulation of security-sensitive operations and decisions in a separate component, such as a Hardware Security Module (HSM). Additional protection of keys and key usage can be provided by constraining keys to particular usages (e.g., decrypt-only), or limiting the availability of keys to individual applications as reported by Identity and Access Management.</p> <p>Depending on application support, <code>Cryptography</code> can also be used to protect session keys and intermediate secrets when processing cryptographic protocols such as TLS and SecOC.</p>

#### 9.5.1.1 Defined interfaces

##### 9.5.1.1.1 Common interfaces

The main entry point for using the `Cryptography` API are the factory functions `LoadCryptoProvider()` for using cryptographic routines, `LoadKeyStorageProvider()` for access to the key store, and `LoadX509Provider()` for X.509 certificate handling.



**Figure 9.42: Common Interfaces of Cryptography**

<b>Name:</b>	n/a	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides the main entry points for using the <code>Cryptography</code> API.	
<b>Operations:</b>	<code>GenerateRandomData</code>	Generate random data of the requested size.
	<code>GetSecureCounter</code>	Get current value of 128 bit <code>SecureCounter</code> supported by the <code>Crypto Stack</code> .
	<code>LoadCryptoProvider</code>	Factory that creates or returns existing single instance of a specific <code>CryptoProvider</code> .
	<code>LoadKeyStorageProvider</code>	Factory that creates or returns an existing single instance of the <code>KeyStorageProvider</code> .
	<code>LoadX509Provider</code>	Factory that creates or returns an existing single instance of the <code>X509Provider</code> .

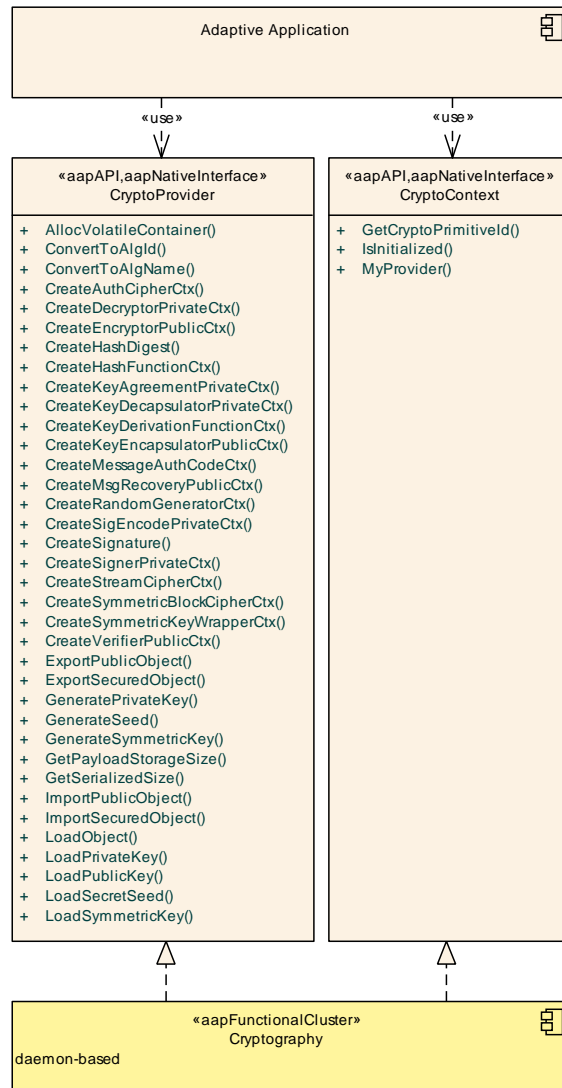
<b>Name:</b>	IOInterface	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Interface for saving and loading of security objects.	
<b>Operations:</b>	<code>GetAllowedUsage</code>	Return actual allowed key/seed usage flags defined by the key slot prototype for this "Actor" and current content of the container.
	<code>GetCapacity</code>	Return capacity of the underlying resource.
	<code>GetCryptoObjectType</code>	Return the type of the object referenced by this <code>IOInterface</code> .
	<code>GetObjectId</code>	Return an ID of an object stored to this <code>IOInterface</code> .
	<code>GetPayloadSize</code>	Return size of an object payload stored in the underlying buffer.
	<code>GetPrimitiveId</code>	Get vendor specific ID of the primitive.
	<code>GetTypeRestriction</code>	Return content type restriction.
	<code>IsObjectExportable</code>	Return the <code>exportable</code> attribute of an object stored to the container.
	<code>IsObjectSession</code>	Return the <code>session</code> (or <code>temporary</code> ) attribute of an object as set.
	<code>IsValid</code>	Get whether the underlying <code>KeySlot</code> is valid.
	<code>IsVolatile</code>	Return volatility of the underlying buffer.
	<code>IsWritable</code>	Get whether the underlying <code>KeySlot</code> is writable.

<b>Name:</b>	Serializable	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Serializable object interface.	
<b>Operations:</b>	ExportPublicly	Serialize itself publicly.

<b>Name:</b>	VolatileTrustedContainer	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface is used for buffering <a href="#">Cryptography API</a> objects in RAM.	
<b>Operations:</b>	GetIOInterface	Retrieve the <a href="#">IOInterface</a> used for importing/exporting objects into this container.

### 9.5.1.1.2 General cryptography interfaces

The [CryptoProvider](#) interface provides access to various cryptographic routines. Each of those routines is managed by specializations of the [CryptoContext](#) interface.



**Figure 9.43: General Cryptography Interfaces**

<b>Name:</b>	CryptoProvider	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This is a "factory" interface of all supported crypto primitives and a "trusted environment" for internal communications between them.	
<b>Operations:</b>	AllocVolatileContainer	Allocate a VolativeTrustedContainer according to directly specified capacity.
	ConvertToAlgId	Convert a common name of crypto algorithm to a correspondent vendor specific binary algorithm ID.
	ConvertToAlgName	Convert a vendor specific binary algorithm ID to a correspondent common name of the crypto algorithm.





CreateAuthCipherCtx	Create a symmetric authenticated cipher context.
CreateDecryptorPrivateCtx	Create a decryption private key context.
CreateEncryptorPublicCtx	Create an encryption public key context.
CreateHashDigest	Construct signature object from directly provided components of a hash digest.
CreateHashFunctionCtx	Create a hash function context.
CreateKeyAgreementPrivateCtx	Create a key-agreement private key context.
CreateKeyDecapsulatorPrivateCtx	Create a key-decapsulator private key context of a key encapsulation mechanism.
CreateKeyDerivationFunctionCtx	Create a key derivation function context.
CreateKeyEncapsulatorPublicCtx	Create a key-encapsulator public key context of a key encapsulation mechanism.
CreateMessageAuthCodeCtx	Create a symmetric message authentication code context.
CreateMsgRecoveryPublicCtx	Create a message recovery public key context.
CreateRandomGeneratorCtx	Create a random number generator context.
CreateSigEncodePrivateCtx	Create a signature encoding private key context.
CreateSignature	Construct a signature object from directly provided components of a digital signature/MAC or authenticated encryption (AE/AEAD).
CreateSignerPrivateCtx	Create a signature private key context.
CreateStreamCipherCtx	Create a symmetric stream cipher context.
CreateSymmetricBlockCipherCtx	Create a symmetric block cipher context.
CreateSymmetricKeyWrapperCtx	Create a symmetric key-wrap algorithm context.
CreateVerifierPublicCtx	Create a signature verification public key context.
ExportPublicObject	Export publicly an object.
ExportSecuredObject	Export a crypto object in a secure manner.
GeneratePrivateKey	Allocate a new private key context of correspondent type and generates the key value randomly.
GenerateSeed	Generate a random Secret Seed object of requested algorithm.
GenerateSymmetricKey	Allocate a new symmetric key object and fill it by a new randomly generated value.



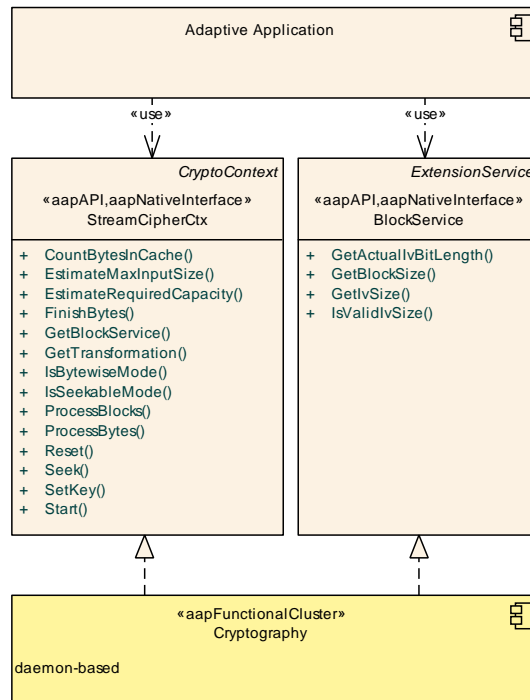


	GetPayloadStorageSize	Return minimally required capacity of a key slot for saving of the object's payload.
	GetSerializedSize	Return required buffer size for serialization of an object in specific format.
	ImportPublicObject	Import publicly serialized object to a storage location.
	ImportSecuredObject	Import securely serialized object to the persistent or volatile storage.
	LoadObject	Load any crypto object from the <a href="#">IOInterface</a> provided.
	LoadPrivateKey	Load a <a href="#">PrivateKey</a> from the <a href="#">IOInterface</a> provided.
	LoadPublicKey	Load a <a href="#">PublicKey</a> from the <a href="#">IOInterface</a> provided.
	LoadSecretSeed	Load a <a href="#">SecretSeed</a> from the <a href="#">IOInterface</a> provided.
	LoadSymmetricKey	Load a <a href="#">SymmetricKey</a> from the <a href="#">IOInterface</a> provided.

<b>Name:</b>	CryptoContext	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	A common interface of a mutable cryptographic context, i.e. that is not bound to a single crypto object.	
<b>Operations:</b>	GetCryptoPrimitiveld	Returns a <a href="#">CryptoPrimitiveld</a> instance containing instance identification.
	IsInitialized	Check if the crypto context is already initialized and ready to use.
	MyProvider	Get a reference to the <a href="#">CryptoProvider</a> of this context.



**9.5.1.1.3 Cryptography context interfaces**



**Figure 9.44: BlockService and CryptoContext Interfaces**

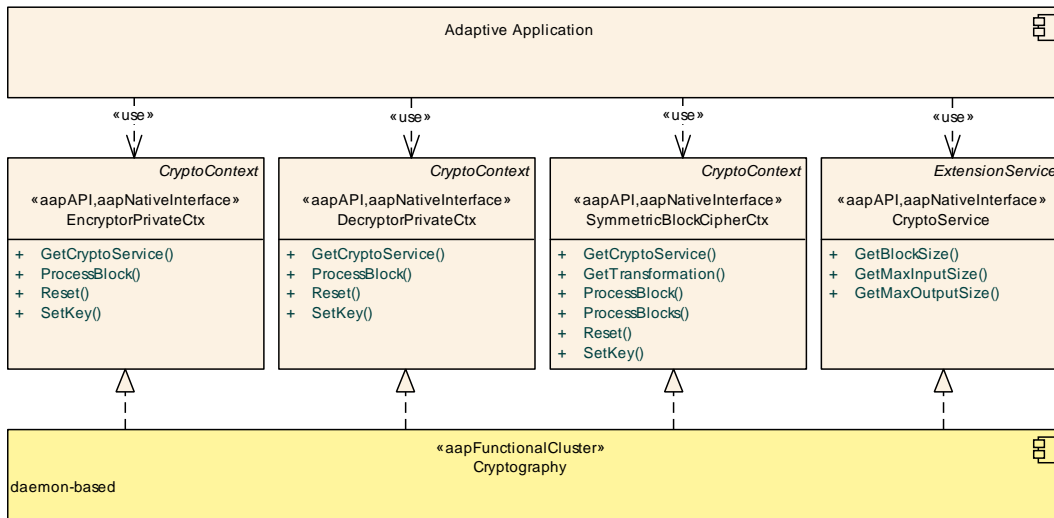
<b>Name:</b>	BlockService	
<b>Extended interface:</b>	<a href="#">ExtensionService</a>	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Extension meta-information service for block cipher contexts.	
<b>Operations:</b>	GetActualIvBitLength	Get the actual bit-length of an initialization vector loaded to the context.
	GetBlockSize	Get the block (or internal buffer) size of the base algorithm.
	GetIvSize	Get default expected size of the initialization vector or nonce.
	IsValidIvSize	Verify validity of specific initialization vector length.

<b>Name:</b>	StreamCipherCtx	
<b>Extended interface:</b>	<a href="#">CryptoContext</a>	





<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Generalized stream cipher context interface covering all modes of operation.	
<b>Operations:</b>	CountBytesInCache	Count number of bytes now kept in the context cache.
	EstimateMaxInputSize	Estimate maximal number of input bytes that may be processed for filling of an output buffer without overflow.
	EstimateRequiredCapacity	Estimate minimal required capacity of the output buffer, which is enough for saving a result of input data processing.
	FinishBytes	Process the final part of message (that may be not aligned to the block-size boundary).
	GetBlockService	Get the <a href="#">BlockService</a> instance.
	GetTransformation	Get the kind of transformation configured for this context: Encrypt or Decrypt.
	IsBytewiseMode	Check the operation mode for the byte-wise property.
	IsSeekableMode	Check if the seek operation is supported in the current mode.
	ProcessBlocks	Process initial parts of message aligned to the block-size boundary.
	ProcessBytes	Process a non-final part of message (that is not aligned to the block-size boundary).
	Reset	Clear the crypto context.
	Seek	Set the position of the next byte within the stream of the encryption/decryption gamma.
	SetKey	Set (deploy) a key to the stream cipher algorithm context.
	Start	Initialize the context for a new data stream processing or generation (depending from the primitive).



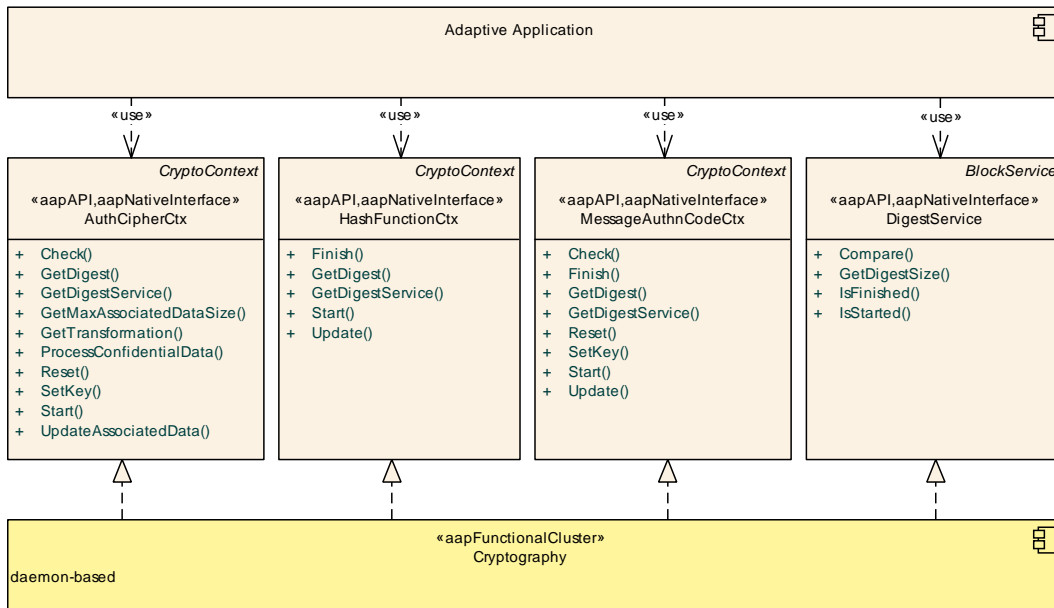
**Figure 9.45: CryptoService and CryptoContext Interfaces**

<b>Name:</b>	CryptoService	
<b>Extended interface:</b>	<a href="#">ExtensionService</a>	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Extension meta-information service for cryptographic contexts.	
<b>Operations:</b>	GetBlockSize	Get block (or internal buffer) size of the base algorithm.
	GetMaxInputSize	Get maximum expected size of the input data block.
	GetMaxOutputSize	Get maximum possible size of the output data block.

<b>Name:</b>	EncryptorPrivateCtx	
<b>Extended interface:</b>	<a href="#">CryptoContext</a>	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Asymmetric decryption private key context interface.	
<b>Operations:</b>	GetCryptoService	Get the <a href="#">CryptoService</a> instance.
	ProcessBlock	Encrypt an input block according to the encryptor configuration.
	Reset	Clear the crypto context.
	SetKey	Set (deploy) a key to the decryptor private algorithm context.

<b>Name:</b>	DecryptorPrivateCtx	
<b>Extended interface:</b>	<a href="#">CryptoContext</a>	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Asymmetric decryption private key context interface.	
<b>Operations:</b>	GetCryptoService	Get the <a href="#">CryptoService</a> instance.
	ProcessBlock	Decrypt an input block according to the decryptor configuration.
	Reset	Clear the crypto context.
	SetKey	Set (deploy) a key to the decryptor private algorithm context.

<b>Name:</b>	SymmetricBlockCipherCtx	
<b>Extended interface:</b>	<a href="#">CryptoContext</a>	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Interface of a symmetric block cipher context with padding.	
<b>Operations:</b>	GetCryptoService	Get the <a href="#">CryptoService</a> instance.
	GetTransformation	Get the kind of transformation configured for this context: Encrypt or Decrypt.
	ProcessBlock	Process (encrypt / decrypt) an input block according to the configuration.
	ProcessBlocks	Process (encrypt / decrypt) input blocks according to the configuration.
	Reset	Clear the crypto context.
	SetKey	Set (deploy) a key to the symmetric algorithm context.



**Figure 9.46: DigestService and CryptoContext Interfaces**

<b>Name:</b>	DigestService	
<b>Extended interface:</b>	<a href="#">BlockService</a>	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Extension meta-information service for digest producing contexts.	
<b>Operations:</b>	Compare	Compare the calculated digest against an expected value.
	GetDigestSize	Get the output digest size.
	IsFinished	Check current status of the stream processing: finished or not.
	IsStarted	Check current status of the stream processing: started or not.

<b>Name:</b>	AuthCipherCtx	
<b>Extended interface:</b>	<a href="#">CryptoContext</a>	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Generalized authenticated cipher context interface.	
<b>Operations:</b>	Check	Check the calculated digest against an expected signature object.





	GetDigest	Retrieve the calculated digest.
	GetDigestService	Get the <a href="#">DigestService</a> instance.
	GetMaxAssociatedDataSize	Get maximal supported size of associated public data.
	GetTransformation	Get the kind of transformation configured for this context: Encrypt or Decrypt.
	ProcessConfidentialData	Process confidential data and return the result.
	Reset	Clear the crypto context.
	SetKey	Set (deploy) a key to the authenticated cipher symmetric algorithm context.
	Start	Initialize the context for a new data processing or generation (depending from the primitive).
	UpdateAssociatedData	Update the digest calculation by the specified data.

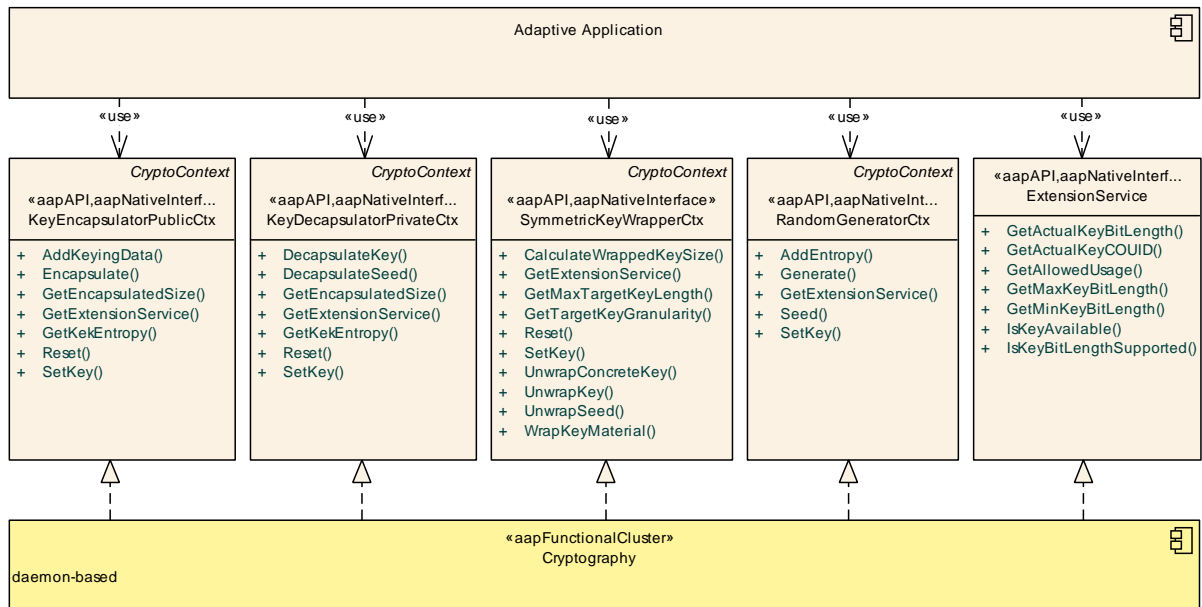
<b>Name:</b>	HashFunctionCtx	
<b>Extended interface:</b>	<a href="#">CryptoContext</a>	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Hash function interface.	
<b>Operations:</b>	Finish	Finish the digest calculation and optionally produce the "signature" object.
	GetDigest	Get requested part of calculated digest.
	GetDigestService	Get the <a href="#">DigestService</a> instance.
	Start	Initialize the context for a new data stream processing or generation (depending on the primitive).
	Update	Update the digest calculation context by a new part of the message.

<b>Name:</b>	MessageAuthnCodeCtx	
<b>Extended interface:</b>	<a href="#">CryptoContext</a>	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	





<b>Description:</b>	Keyed message authentication code context interface definition (MAC/HMAC).	
<b>Operations:</b>	Check	Check the calculated digest against an expected "signature" object.
	Finish	Finish the digest calculation and optionally produce the "signature" object.
	GetDigest	Get requested part of calculated digest to existing memory buffer.
	GetDigestService	Get the <a href="#">DigestService</a> instance.
	Reset	Clear the crypto context.
	SetKey	Set (deploy) a key to the message authn code algorithm context.
	Start	Initialize the context for a new data stream processing or generation (depending from the primitive).
	Update	Update the digest calculation context by a new part of the message.



**Figure 9.47: ExtensionService and CryptoContext Interfaces (1 of 2)**

<b>Name:</b>	ExtensionService	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Basic meta-information service for all contexts.	
<b>Operations:</b>	GetActualKeyBitLength	Get actual bit-length of a key loaded to the context.





	GetActualKeyCOUID	Get the CryptoObjectUid of the key deployed to the context this extension service is attached to.
	GetAllowedUsage	Get allowed usages of this context (according to the key object attributes loaded to this context).
	GetMaxKeyBitLength	Get maximum supported key length in bits.
	GetMinKeyBitLength	Get minimal supported key length in bits.
	IsKeyAvailable	Check if a key has been set to this context.
	IsKeyBitLengthSupported	Verify if the specific key length is supported by the context.

<b>Name:</b>	KeyEncapsulatorPublicCtx	
<b>Extended interface:</b>	<a href="#">CryptoContext</a>	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Asymmetric key encapsulation mechanism public key context interface.	
<b>Operations:</b>	AddKeyingData	Add the content to be encapsulated (payload) according to RFC 5990 ("keying data").
	Encapsulate	Encapsulate the last set keying-data.
	GetEncapsulatedSize	Get fixed size of the encapsulated data block.
	GetExtensionService	Get the <a href="#">ExtensionService</a> instance.
	GetKekEntropy	Get entropy (bit-length) of the key encryption key (KEK) material.
	Reset	Clear the crypto context.
	SetKey	Set (deploy) a key to the key encapsulator public algorithm context.

<b>Name:</b>	KeyDecapsulatorPrivateCtx	
<b>Extended interface:</b>	<a href="#">CryptoContext</a>	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Asymmetric key encapsulation mechanism private key context interface.	



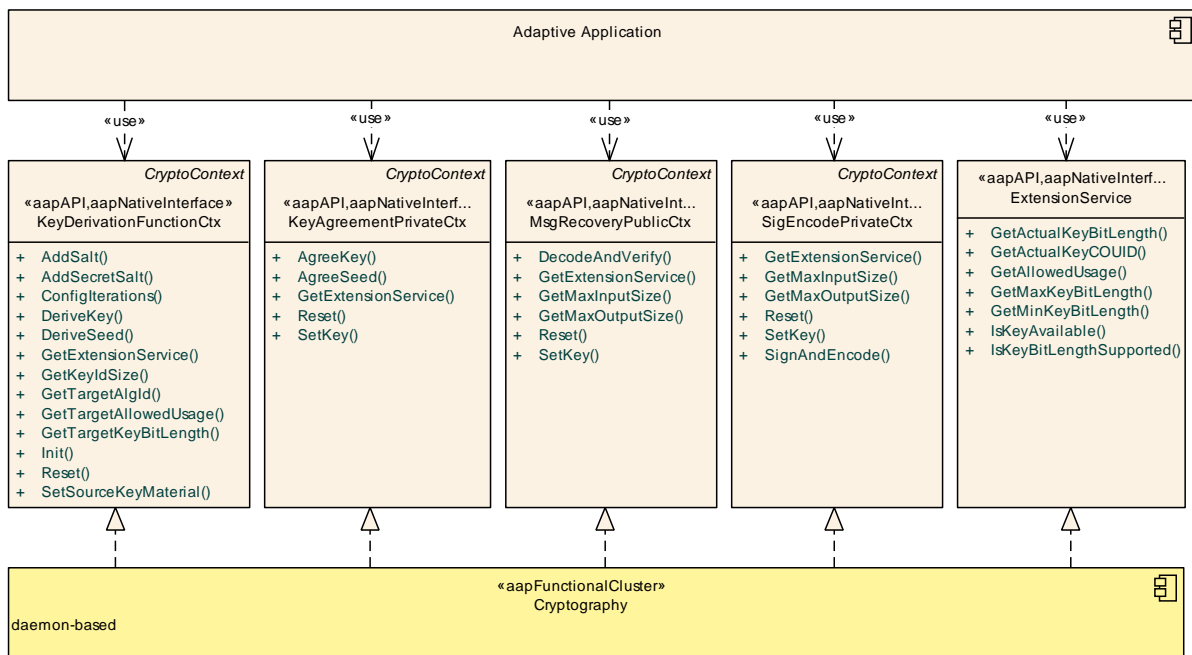




<b>Operations:</b>	DecapsulateKey	Decapsulate the keying data to be used for subsequent processing (e.g. secure communication).
	DecapsulateSeed	Decapsulate key material.
	GetEncapsulatedSize	Get fixed size of the encapsulated data block.
	GetExtensionService	Get the <a href="#">ExtensionService</a> instance.
	GetKekEntropy	Get entropy (bit-length) of the key encryption key (KEK) material.
	Reset	Clear the crypto context.
	SetKey	Set (deploy) a key to the key decapsulator private algorithm context.

<b>Name:</b>	SymmetricKeyWrapperCtx	
<b>Extended interface:</b>	<a href="#">CryptoContext</a>	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Context of a symmetric key wrap algorithm (for AES it should be compatible with RFC3394 or RFC5649).	
<b>Operations:</b>	CalculateWrappedKeySize	Calculate size of the wrapped key in bytes from original key length in bits.
	GetExtensionService	Get the <a href="#">ExtensionService</a> instance.
	GetMaxTargetKeyLength	Get maximum length of the target key supported by the implementation.
	GetTargetKeyGranularity	Get expected granularity of the target key (block size).
	Reset	Clear the crypto context.
	SetKey	Set (deploy) a key to the symmetric key wrapper algorithm context.
	UnwrapConcreteKey	Execute the "key unwrap" operation for the provided BLOB and produce a key object of the expected type.
	UnwrapKey	Execute the "key unwrap" operation for the provided BLOB and produce a key object.
	UnwrapSeed	Execute the "key unwrap" operation for the provided BLOB and produce a <a href="#">SecretSeed</a> object.
WrapKeyMaterial	Execute the "key wrap" operation for the provided key material.	

<b>Name:</b>	RandomGeneratorCtx	
<b>Extended interface:</b>	<a href="#">CryptoContext</a>	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Interface of a random number generator context.	
<b>Operations:</b>	AddEntropy	Update the internal state of the RNG by mixing it with the provided additional entropy.
	Generate	Return an allocated buffer with a generated random sequence of the requested size.
	GetExtensionService	Get the <a href="#">ExtensionService</a> instance.
	Seed	Set the internal state of the RNG using the provided seed.
	SetKey	Set the internal state of the RNG using the provided seed.



**Figure 9.48: ExtensionService and CryptoContext Interfaces (2 of 2)**

<b>Name:</b>	KeyDerivationFunctionCtx	
<b>Extended interface:</b>	<a href="#">CryptoContext</a>	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Key derivation function interface.	



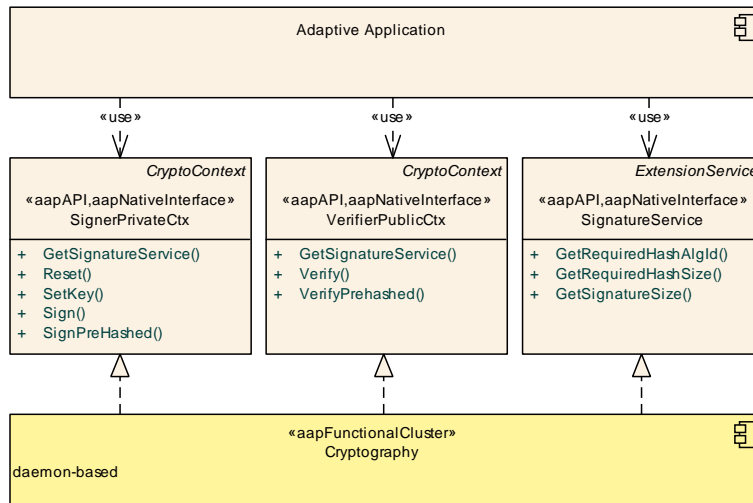


<b>Operations:</b>	AddSalt	Add a salt value stored in a non-secret memory region.
	AddSecretSalt	Add a secret salt value stored in a <a href="#">SecretSeed</a> object.
	ConfigIterations	Configure the number of iterations that will be applied by default.
	DeriveKey	Derive a symmetric key from the provided key material and provided context configuration.
	DeriveSeed	Derive key material (secret seed) from the provided "master" key material and the provided context configuration.
	GetExtensionService	Get the <a href="#">ExtensionService</a> instance.
	GetKeyIdSize	Get the fixed size of the target key ID required by diversification algorithm.
	GetTargetAlgId	Get the symmetric algorithm ID of target key.
	GetTargetAllowedUsage	Get allowed key usage of target key.
	GetTargetKeyBitLength	Get the bit-length of target (diversified) keys.
	Init	Initialize this context by setting at least the target key ID.
	Reset	Clear the crypto context.
	SetSourceKeyMaterial	Set (deploy) key-material to the key derivation algorithm context.

<b>Name:</b>	KeyAgreementPrivateCtx	
<b>Extended interface:</b>	<a href="#">CryptoContext</a>	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Key agreement private key context interface (Diffie Hellman or conceptually similar).	
<b>Operations:</b>	AgreeKey	Produce a common symmetric key via execution of the key-agreement algorithm between this private key.
	AgreeSeed	Produce a common secret seed via execution of the key-agreement algorithm between this private key.
	GetExtensionService	Get the <a href="#">ExtensionService</a> instance.
	Reset	Clear the crypto context.
	SetKey	Set (deploy) a key to the key agreement private algorithm context.

<b>Name:</b>	MsgRecoveryPublicCtx	
<b>Extended interface:</b>	<a href="#">CryptoContext</a>	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	A public key context for asymmetric recovery of a short message and its signature verification (RSA-like).	
<b>Operations:</b>	DecodeAndVerify	Process (encrypt / decrypt) an input block according to the cryptor configuration.
	GetExtensionService	Get the <a href="#">ExtensionService</a> instance.
	GetMaxInputSize	Get maximum expected size of the input data block.
	GetMaxOutputSize	Get maximum possible size of the output data block.
	Reset	Clear the crypto context.
	SetKey	Set (deploy) a key to the msg recovery public algorithm context.

<b>Name:</b>	SigEncodePrivateCtx	
<b>Extended interface:</b>	<a href="#">CryptoContext</a>	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	A private key context for asymmetric signature calculation and short message encoding (RSA-like).	
<b>Operations:</b>	GetExtensionService	Get the <a href="#">ExtensionService</a> instance.
	GetMaxInputSize	Get maximum expected size of the input data block.
	GetMaxOutputSize	Get maximum possible size of the output data block.
	Reset	Clear the crypto context.
	SetKey	Set (deploy) a key to the sig encode private algorithm context.
	SignAndEncode	Process (encrypt / decrypt) an input block according to the cryptor configuration.



**Figure 9.49: SignatureService and CryptoContext Interfaces**

<b>Name:</b>	SignatureService	
<b>Extended interface:</b>	<a href="#">ExtensionService</a>	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Extension meta-information service for signature contexts.	
<b>Operations:</b>	GetRequiredHashAlgId	Get an ID of hash algorithm required by current signature algorithm.
	GetRequiredHashSize	Get the hash size required by current signature algorithm.
	GetSignatureSize	Get size of the signature value produced and required by the current algorithm.

<b>Name:</b>	SignerPrivateCtx	
<b>Extended interface:</b>	<a href="#">CryptoContext</a>	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Signature private key context interface.	
<b>Operations:</b>	GetSignatureService	Get the <a href="#">SignatureService</a> instance.
	Reset	Clear the crypto context.
	SetKey	Set (deploy) a key to the signer private algorithm context.
	Sign	Sign a directly provided hash or message value.

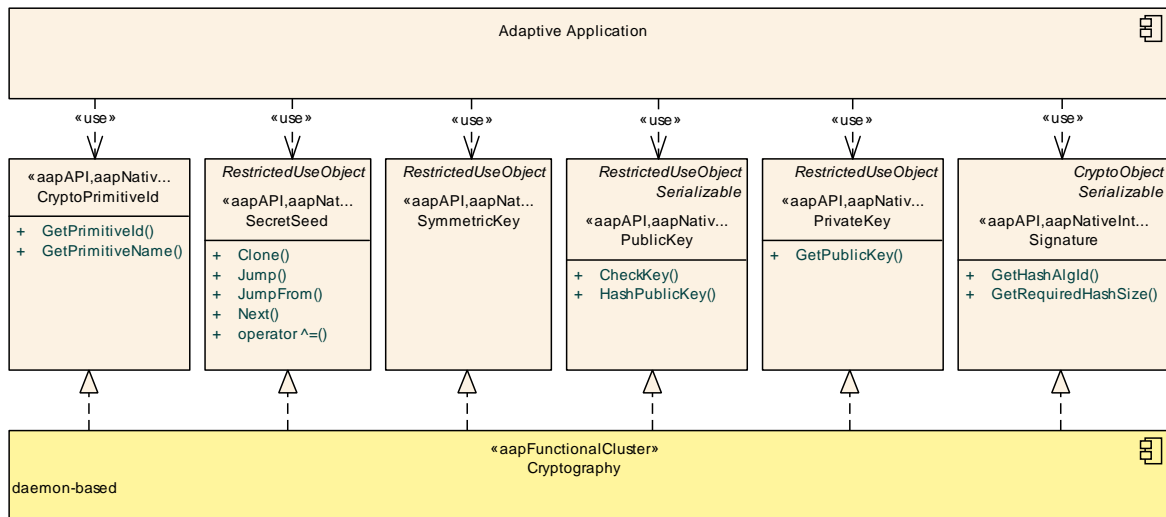




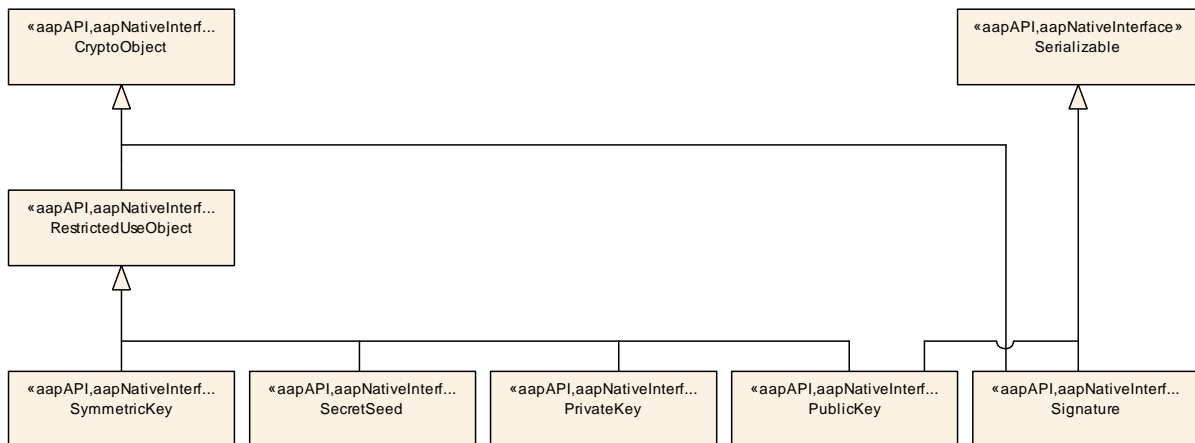
	SignPreHashed	Sign a provided digest value stored in the hash-function context.
--	---------------	---

<b>Name:</b>	VerifierPublicCtx	
<b>Extended interface:</b>	CryptoContext	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Signature verification public key context interface.	
<b>Operations:</b>	GetSignatureService	Get the <a href="#">SignatureService</a> instance.
	Verify	Verify signature BLOB by a directly provided hash or message value.
	VerifyPrehashed	Verify a signature by a digest value stored in the hash-function context.

### 9.5.1.1.4 Cryptographic object interfaces



**Figure 9.50: Cryptographic Object Interfaces**



**Figure 9.51: Taxonomy of Cryptographic Object Interfaces**

<b>Name:</b>	CryptoObject	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	A common interface for all cryptographic objects recognizable by the <a href="#">CryptoProvider</a> .	
<b>Operations:</b>	GetCryptoPrimitiveld	Return the <a href="#">CryptoPrimitiveld</a> of this object.
	GetObjectld	Return the object's COIdentifier, which includes the object's type and UID.
	GetPayloadSize	Return actual size of the object's payload.
	HasDependence	Return the COIdentifier of the object that this object depends on.
	IsExportable	Get the <code>exportability</code> attribute of the crypto object.
	IsSession	Return the <code>session</code> (or <code>temporary</code> ) attribute of the object.
	Save	Save itself to provided <a href="#">IOInterface</a>

<b>Name:</b>	RestrictedUseObject	
<b>Extended interface:</b>	<a href="#">CryptoObject</a>	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	A common interface for all objects supporting the usage restriction.	
<b>Operations:</b>	GetAllowedUsage	Get allowed usages of this object.

<b>Name:</b>	CryptoPrimitiveld	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Common interface for identification of all CryptoPrimitives and their keys & parameters.	
<b>Operations:</b>	GetPrimitiveld	Get vendor specific ID of the primitive.
	GetPrimitiveName	Get a unified name of the primitive.

<b>Name:</b>	SecretSeed	
<b>Extended interface:</b>	<a href="#">RestrictedUseObject</a>	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Secret seed object contains a raw bit sequence of specific length (without any filtering of allowed/disallowed values)!	
<b>Operations:</b>	Clone	Clone this object to new session object.
	Jump	Set value of this seed object as a "jump" from it's current state to specified number of steps, according to "counting" expression defined by a cryptographic algorithm associated with this object.
	JumpFrom	Set value of this seed object as a "jump" from an initial state to specified number of steps, according to "counting" expression defined by a cryptographic algorithm associated with this object.
	Next	Set next value of the secret seed according to "counting" expression defined by a cryptographic algorithm associated with this object.
	operator ^=	XOR value of this seed object with another one and save result to this object. If seed sizes in this object and in the source argument are different then only correspondent number of leading bytes in this seed object shall be updated.

<b>Name:</b>	SymmetricKey	
<b>Extended interface:</b>	<a href="#">RestrictedUseObject</a>	
<b>Technology:</b>	Native interface	







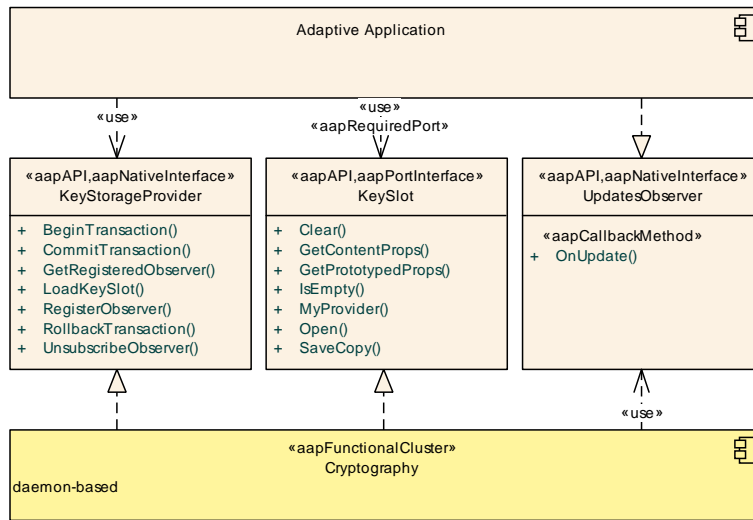
<b>Usage:</b>	Public API
<b>Description:</b>	Symmetric Key interface.

<b>Name:</b>	PublicKey	
<b>Extended interface:</b>	<a href="#">RestrictedUseObject</a> <a href="#">Serializable</a>	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	General asymmetric public key interface.	
<b>Operations:</b>	CheckKey	Check the key for its correctness.
	HashPublicKey	Calculate hash of the public key value.

<b>Name:</b>	PrivateKey	
<b>Extended interface:</b>	<a href="#">RestrictedUseObject</a>	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Generalized asymmetric private key interface.	
<b>Operations:</b>	GetPublicKey	Get the public key correspondent to this private key.

<b>Name:</b>	Signature	
<b>Extended interface:</b>	<a href="#">CryptoObject</a> <a href="#">Serializable</a>	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface is applicable for keeping the Digital Signature, Hash Digest, (Hash-based) Message Authentication Code (MAC/HMAC).	
<b>Operations:</b>	GetHashAlgId	Get an ID of hash algorithm used for this signature object production.
	GetRequiredHashSize	Get the hash size required by current signature algorithm.

**9.5.1.1.5 Cryptographic key handling interfaces**



**Figure 9.52: Cryptographic Key Handling Interfaces**

<b>Name:</b>	KeyStorageProvider	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Key Storage Provider interface.	
<b>Operations:</b>	BeginTransaction	Begin new transaction for key slots update.
	CommitTransaction	Commit changes of the transaction to storage.
	GetRegisteredObserver	Get the currently registered <a href="#">UpdatesObserver</a> for key slots.
	LoadKeySlot	Load a key slot.
	RegisterObserver	Register an <a href="#">UpdatesObserver</a> for key slots.
	RollbackTransaction	Rollback all changes executed during the transaction in storage.
	UnsubscribeObserver	Unregister an <a href="#">UpdatesObserver</a> from key slots.

<b>Name:</b>	KeySlot
<b>Technology:</b>	Port interface
<b>Generated:</b>	No
<b>Meta-model interface type:</b>	CryptoKeySlot
<b>Usage:</b>	Public API

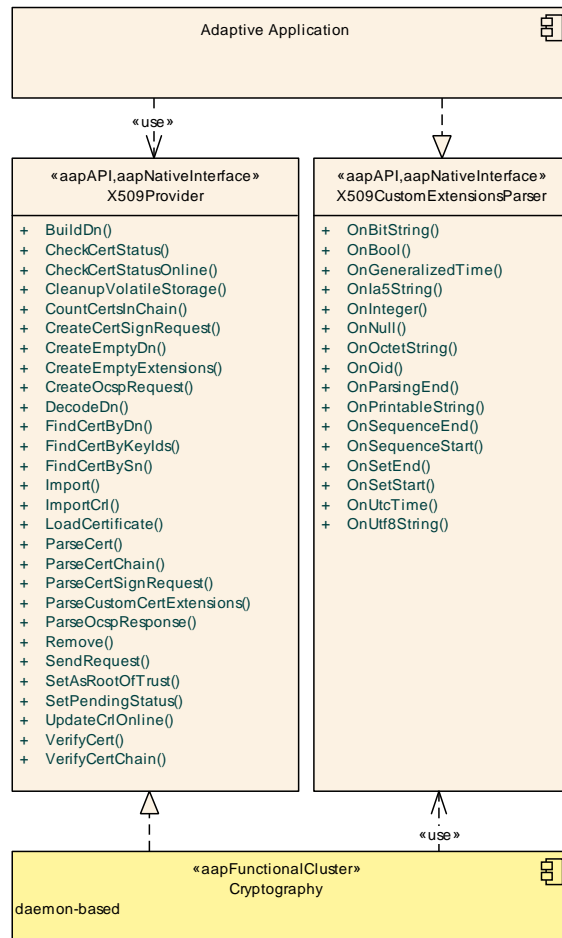




<b>Description:</b>	Key slot interface enables access to a physical key-slot.	
<b>Operations:</b>	Clear	Clear the content of this key slot.
	GetContentProps	Get an actual properties of a content in the key slot.
	GetPrototypedProps	Get the prototyped properties of the key slot.
	IsEmpty	Check the slot for emptiness.
	MyProvider	Retrieve the instance of the <a href="#">CryptoProvider</a> that owns this <a href="#">KeySlot</a> .
	Open	Open this key slot and return an <a href="#">IOInterface</a> to its content.
	SaveCopy	Save the content of a provided source <a href="#">IOInterface</a> to this key slot.

<b>Name:</b>	UpdatesObserver	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Interface for observing updates on key slots.	
<b>Operations:</b>	OnUpdate	This method is called if the content of the specified slots was changed.

**9.5.1.1.6 X.509 certificate handling interfaces**



**Figure 9.53: X.509 Certificate Handling Interfaces**

<b>Name:</b>	X509Provider	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	X.509 Provider interface supporting two internal storage types: volatile (or session) and persistent.	
<b>Operations:</b>	BuildDn	Create completed X.500 Distinguished Name structure from the provided string representation.
	CheckCertStatus	Check certificate status by directly provided OCSP response.
	CheckCertStatusOnline	Check certificate status via On-line Certificate Status Protocol (OCSP).
	CleanupVolatileStorage	Cleanup the volatile certificates storage.





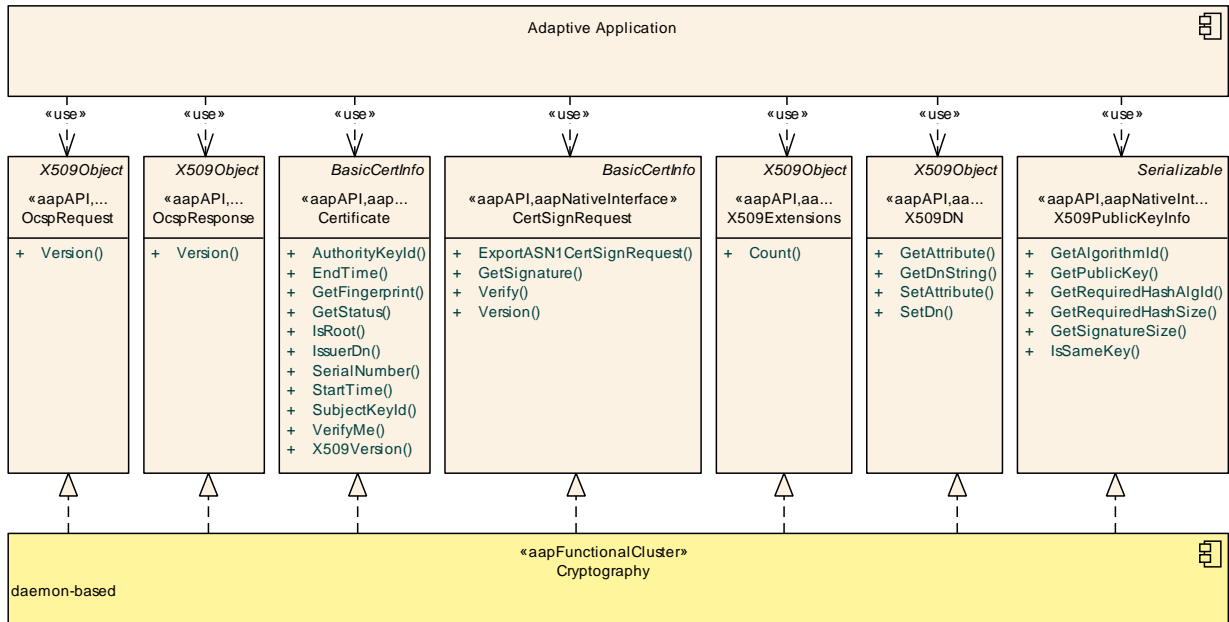
CountCertsInChain	Count number of certificates in a serialized certificate chain represented by a single BLOB.
CreateCertSignRequest	Create certification request for a private key loaded to the context.
CreateEmptyDn	Create an empty X.500 Distinguished Name (DN) structure.
CreateEmptyExtensions	Create an empty X.509 Extensions structure.
CreateOcsRequest	Create OCSP request for specified certificate(s).
DecodeDn	Decode X.500 Distinguished Name structure from the provided serialized format.
FindCertByDn	Find a certificate by the subject and issuer Distinguished Names (DN).
FindCertByKeyIds	Find a certificate by its SKID & AKID.
FindCertBySn	Find a certificate by its serial number and issue DN.
Import	Import the certificate to volatile or persistent storage.
ImportCrl	Import Certificate Revocation List (CRL) or Delta CRL from a memory BLOB.
LoadCertificate	Load a certificate from the persistent certificate storage.
ParseCert	Parse a serialized representation of the certificate and create its instance.
ParseCertChain	Parse a serialized representation of the certificate chain and create their instances.
ParseCertSignRequest	Parse a certificate signing request (CSR) provided by the user.
ParseCustomCertExtensions	Parse the custom X.509 extensions.
ParseOcsResponse	Parse serialized OCSP response and create correspondent interface instance.
Remove	Remove specified certificate from the storage (volatile or persistent) and destroy it.
SendRequest	Send prepared certificate request to CA and save it to volatile or persistent storage.
SetAsRootOfTrust	Set specified CA certificate as a "root of trust".
SetPendingStatus	Set the "pending" status associated to the CSR that means that the CSR already sent to CA.
UpdateCrlOnline	Get Certificate Revocation List (CRL) or Delta CRL via on-line connection.
VerifyCert	Verify status of the provided certificate by locally stored CA certificates and CRLs only.



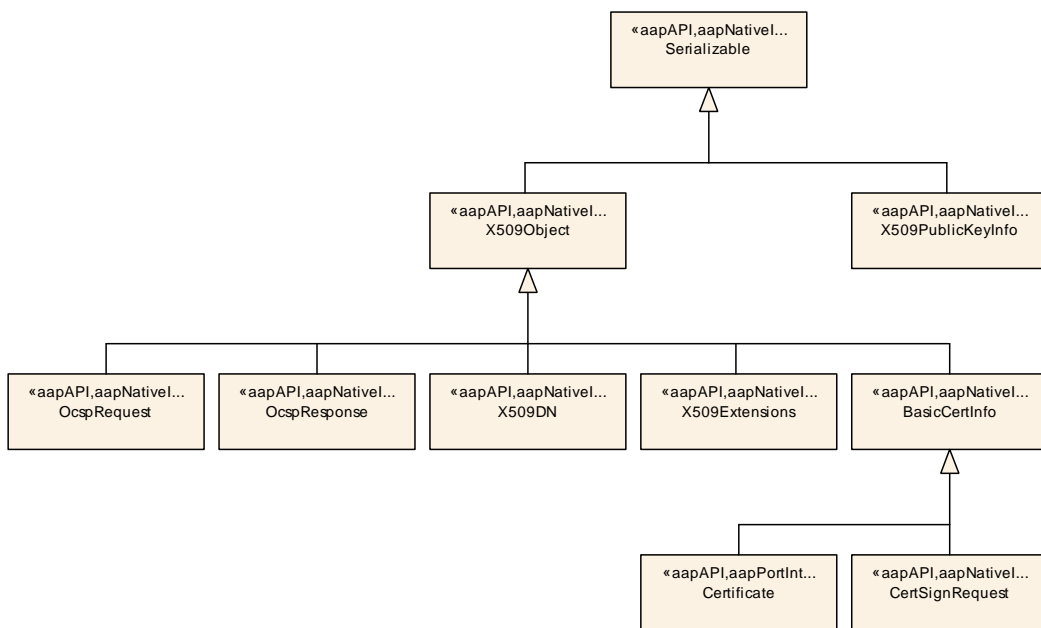


	VerifyCertChain	Verify status of the provided certification chain by locally stored CA certificates and CRLs only.
--	-----------------	--

<b>Name:</b>	X509CustomExtensionsParser	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	X.509 custom extensions parser. This callback interface is to be implemented by an application.	
<b>Operations:</b>	OnBitString	Called when a bit string is encountered.
	OnBool	Called when a boolean is encountered.
	OnGeneralizedTime	Called when a generalized time is encountered.
	OnIa5String	Called when an IA5 string is encountered.
	OnInteger	Called when an integer is encountered.
	OnNull	Called when a NULL is encountered.
	OnOctetString	Called when an octet string is encountered.
	OnOid	Called when an oid is encountered.
	OnParsingEnd	Called when the parsing is completed.
	OnPrintableString	Called when a printable string is encountered.
	OnSequenceEnd	Called when a sequence ends.
	OnSequenceStart	Called when a sequence starts.
	OnSetEnd	Called when a set ends.
	OnSetStart	Called when a set starts.
OnUtcTime	Called when a UTC time is encountered.	
OnUtf8String	Called when an UTF8 string is encountered.	



**Figure 9.54: X.509 Certificate Object Interfaces**



**Figure 9.55: Taxonomy of X.509 Certificate Object Interfaces**

<b>Name:</b>	OcsRequest
<b>Extended interface:</b>	X509Object
<b>Technology:</b>	Native interface
<b>Usage:</b>	Public API
<b>Description:</b>	On-line Certificate Status Protocol Request.





<b>Operations:</b>	Version	Get version of the OCSP request format.
--------------------	---------	---

<b>Name:</b>	Ocsponse	
<b>Extended interface:</b>	X509Object	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	On-line Certificate Status Protocol Response.	
<b>Operations:</b>	Version	Get version of the OCSP response format.

<b>Name:</b>	Certificate	
<b>Extended interface:</b>	BasicCertInfo	
<b>Technology:</b>	Port interface	
<b>Generated:</b>	No	
<b>Meta-model interface type:</b>	CryptoCertificate	
<b>Usage:</b>	Public API	
<b>Description:</b>	X.509 Certificate interface.	
<b>Operations:</b>	AuthorityKeyId	Get the DER encoded <code>AuthorityKeyIdentifier</code> of this certificate.
	EndTime	Get the <code>NotAfter</code> of the certificate.
	GetFingerprint	Calculate a fingerprint from the whole certificate.
	GetStatus	Return last verification status of the certificate.
	IsRoot	Check whether this certificate belongs to a root CA.
	IssuerDn	Get the issuer certificate DN.
	SerialNumber	Get the serial number of this certificate.
	StartTime	Get the <code>NotBefore</code> of the certificate.
	SubjectKeyId	Get the DER encoded <code>SubjectKeyIdentifier</code> of this certificate.
	VerifyMe	Verify signature of the certificate.
	X509Version	Get the X.509 version of this certificate object.



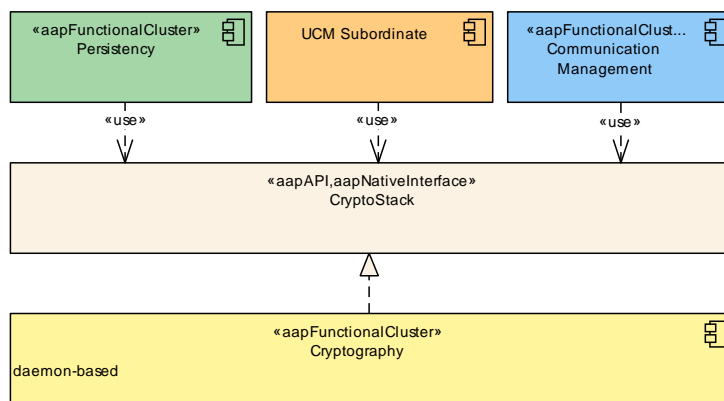
<b>Name:</b>	CertSignRequest	
<b>Extended interface:</b>	BasicCertInfo	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Certificate Signing Request (CSR) object interface.	
<b>Operations:</b>	ExportASN1CertSignRequest	Export this certificate signing request in DER encoded ASN1 format.
	GetSignature	Return signature object of the request.
	Verify	Verifies self-signed signature of the certificate request.
	Version	Return format version of the certificate request.

<b>Name:</b>	X509Extensions	
<b>Extended interface:</b>	X509Object	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Interface of X.509 Extensions.	
<b>Operations:</b>	Count	Count number of elements in the sequence.

<b>Name:</b>	X509DN	
<b>Extended interface:</b>	X509Object	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Interface of X.509 Distinguished Name (DN).	
<b>Operations:</b>	GetAttribute	Get a DN attribute.
	GetDnString	Get the whole Distinguished Name (DN) as a single string.
	SetAttribute	Set a DN attribute.
	SetDn	Set whole Distinguished Name (DN) from a single string.

<b>Name:</b>	X509PublicKeyInfo	
<b>Extended interface:</b>	<a href="#">Serializable</a>	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	X.509 Public Key Information interface.	
<b>Operations:</b>	GetAlgorithmId	Get the CryptoPrimitiveld instance of this class.
	GetPublicKey	Get public key object of the subject.
	GetRequiredHashAlgId	Get an ID of hash algorithm required by current signature algorithm.
	GetRequiredHashSize	Get the hash size required by current signature algorithm.
	GetSignatureSize	Get size of the signature value produced and required by the current algorithm.
	IsSameKey	Verify the sameness of the provided and kept public keys.

### 9.5.1.2 Provided interfaces

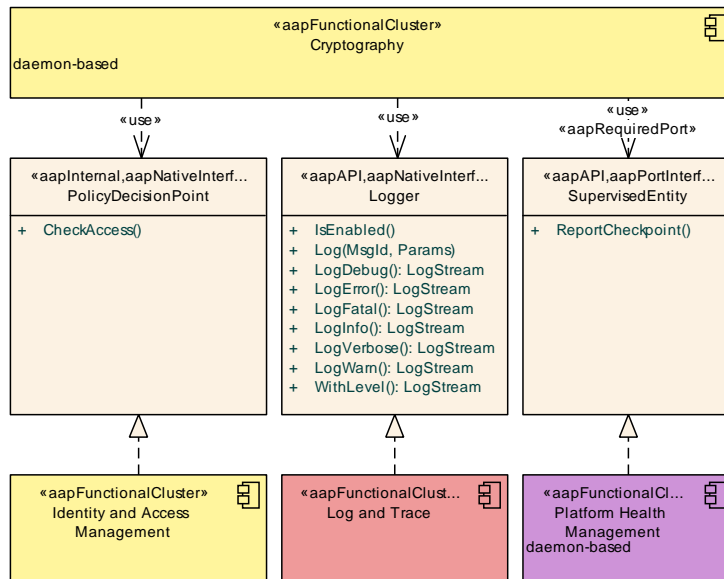


**Figure 9.56: Users of the Cryptography interfaces**

<b>Interface</b>	<b>Requiring functional clusters</b>
CryptoStack	Communication Management (see Section 9.3.1)
	Persistence (see Section 9.4.1)
	Update and Configuration Management (see Section 9.7.1)

**Table 9.15: Interfaces provided by Cryptography to other Functional Clusters**

**9.5.1.3 Required interfaces**



**Figure 9.57: Interfaces required by Cryptography**

Interface	Purpose
<a href="#">EventReporter</a>	This interface should be used to e.g., report attempts to change root certificates.
<a href="#">Logger</a>	Cryptography shall use this interface to log standardized messages.
<a href="#">Manifest Accessor</a>	Cryptography shall use this interface to read its configuration information from the Manifests.
<a href="#">PolicyDecisionPoint</a>	Cryptography shall use this interface to check access to certificates.
<a href="#">SupervisedEntity</a>	Cryptography should use this interface for supervision of its daemon process(es).
<a href="#">UpdatesObserver</a>	Cryptography uses this interface to notify when a key has been updated.
<a href="#">X509CustomExtensionsParser</a>	Cryptography uses this interface for propagating parser events.

**Table 9.16: Interfaces required by Cryptography**

**9.5.2 Identity and Access Management**

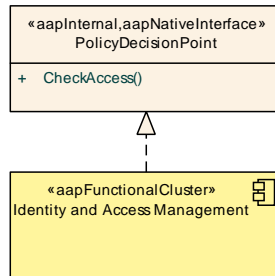
<b>Name:</b>	Identity and Access Management
<b>Short name:</b>	iam





<b>Category:</b>	Security
<b>Daemon-based:</b>	No
<b>Responsibilities:</b>	Identity and Access Management checks access to resources of the AUTOSAR Adaptive Platform, for example, on Service Interfaces and Functional Clusters. Identity and Access Management hereby introduces access control for Adaptive Applications and protection against privilege escalation in case of attacks. In addition, Identity and Access Management enables integrators to verify access on resources requested by Adaptive Applications in advance during deployment.

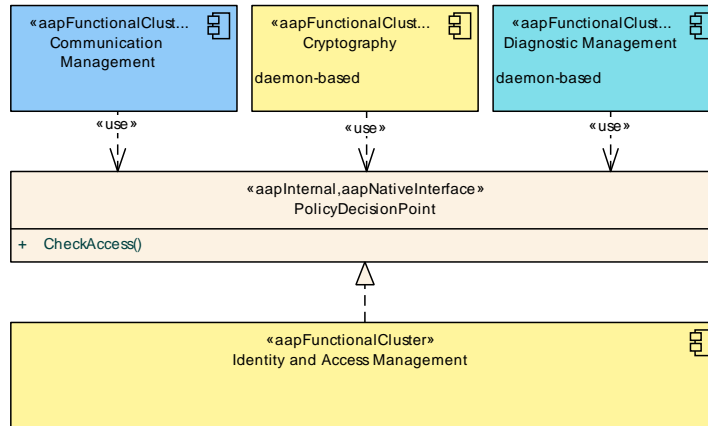
### 9.5.2.1 Defined interfaces



**Figure 9.58: Interfaces of Identity and Access Management**

<b>Name:</b>	PolicyDecisionPoint	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Internal	
<b>Description:</b>	This interface serves Policy Enforcement Points with authorization decisions.	
<b>Operations:</b>	CheckAccess	Evaluates an access request against the authorization policies ( <i>Grant</i> ) before issuing an access decision.

**9.5.2.2 Provided interfaces**

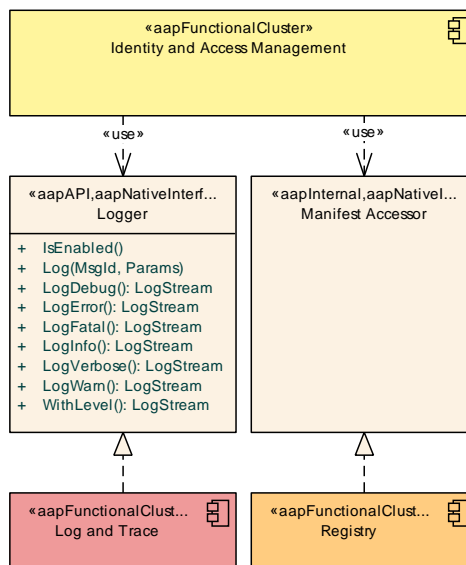


**Figure 9.59: Users of the Identity and Access Management interfaces**

<i>Interface</i>	<i>Requiring functional clusters</i>
PolicyDecisionPoint	Communication Management (see Section 9.3.1)
	Cryptography (see Section 9.5.1)
	Diagnostic Management (see Section 9.8.1)

**Table 9.17: Interfaces provided by Identity and Access Management to other Functional Clusters**

**9.5.2.3 Required interfaces**



**Figure 9.60: Interfaces required by Identity and Access Management**

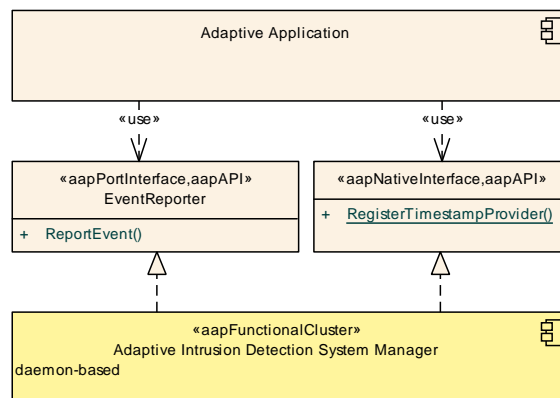
<b>Interface</b>	<b>Purpose</b>
EventReporter	This interface should be used to e.g., report denied access.
Logger	Identity and Access Management shall use this interface to log standardized messages.
Manifest Accessor	Identity and Access Management should use this interface to access Grant information modeled in the Manifests.

**Table 9.18: Interfaces required by Identity and Access Management**

### 9.5.3 Adaptive Intrusion Detection System Manager

<b>Name:</b>	Adaptive Intrusion Detection System Manager
<b>Short name:</b>	idsm
<b>Category:</b>	Security
<b>Daemon-based:</b>	Yes
<b>Responsibilities:</b>	Adaptive Intrusion Detection System Manager provides functionality to report security events.

#### 9.5.3.1 Defined interfaces



**Figure 9.61: Interfaces of Adaptive Intrusion Detection System Manager**

<b>Name:</b>	EventReporter
<b>Technology:</b>	Port interface
<b>Generated:</b>	No
<b>Meta-model interface type:</b>	SecurityEventDefinition





<b>Usage:</b>	Public API	
<b>Description:</b>	This interface is used to report security events to the Adaptive Intrusion Detection System Manager.	
<b>Operations:</b>	ReportEvent	Create a new security event at the Adaptive Intrusion Detection System Manager.

<b>Name:</b>	n/a	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Provides functionality to register a timestamp provider with the Adaptive Intrusion Detection System Manager.	
<b>Operations:</b>	RegisterTimestampProvider	Register a callback for providing timestamps to the Adaptive Intrusion Detection System Manager.

### 9.5.3.2 Provided interfaces

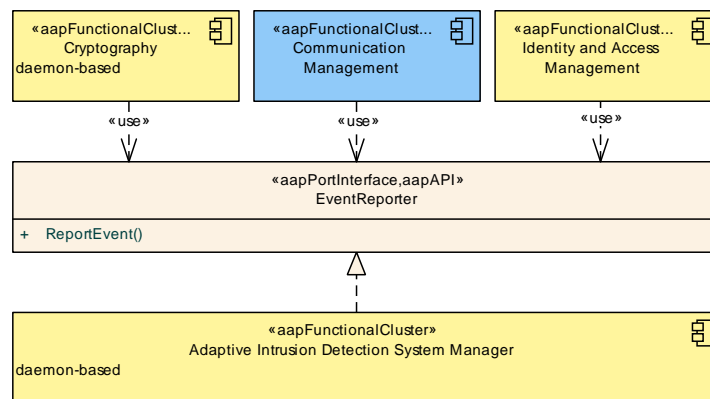
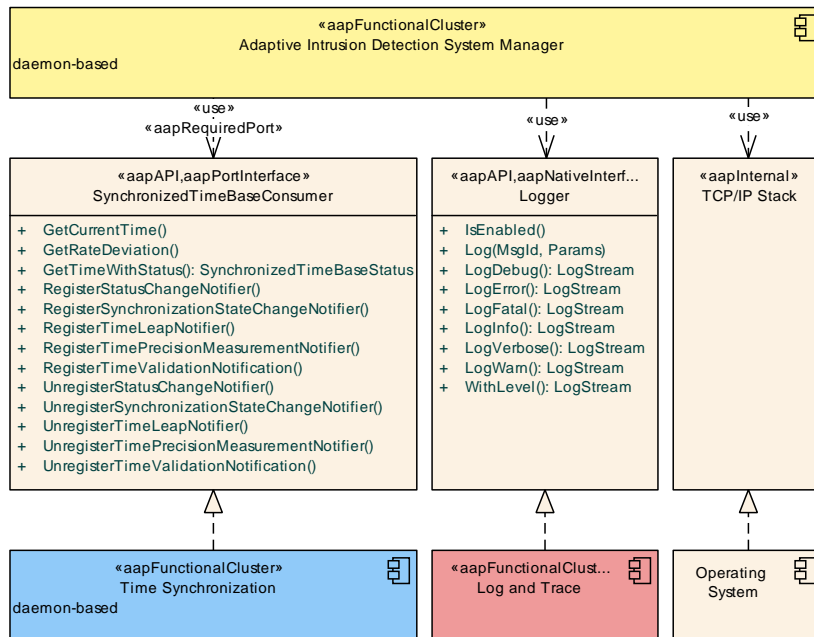


Figure 9.62: Users of the Adaptive Intrusion Detection System Manager interfaces

Interface	Requiring functional clusters
EventReporter	Communication Management (see Section 9.3.1)
	Cryptography (see Section 9.5.1)
	Identity and Access Management (see Section 9.5.2)

Table 9.19: Interfaces provided by Adaptive Intrusion Detection System Manager to other Functional Clusters

**9.5.3.3 Required interfaces**

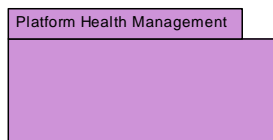


**Figure 9.63: Interfaces required by Adaptive Intrusion Detection System Manager**

Interface	Purpose
Logger	Adaptive Intrusion Detection System Manager shall use this interface to log standardized messages.
SynchronizedTimeBaseConsumer	Adaptive Intrusion Detection System Manager shall use this interface to determine timestamps of security events.
TCP/IP Stack	Adaptive Intrusion Detection System Manager shall use this interface to propagate qualified security events via the IDS protocol.

**Table 9.20: Interfaces required by Adaptive Intrusion Detection System Manager**

**9.6 Safety**



**Figure 9.64: Overview of Safety and its building blocks**



## 9.6.1 Platform Health Management

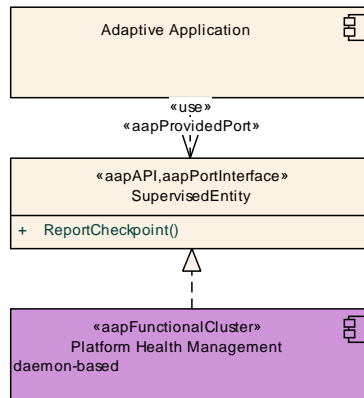
<b>Name:</b>	Platform Health Management
<b>Short name:</b>	phm
<b>Category:</b>	Safety
<b>Daemon-based:</b>	Yes
<b>Responsibilities:</b>	<p>Platform Health Management performs (aliveness, logical, and deadline) supervision of Processes in safety-critical setups and reports failures to State Management. Platform Health Management also controls the Watchdog that in turn supervises the Platform Health Management.</p> <p>An Alive Supervision checks that a supervised entity is not running too frequently and not too rarely. A Deadline Supervision checks that steps in a supervised entity are executed within the configured minimum and maximum time. A Logical Supervision checks that the control flow during execution matches the designed control flow. All types of supervision can be used independently and are performed based on reporting of Checkpoints by the supervised entity.</p> <p>State Management and Execution Management are the fundamental Functional Clusters of the AUTOSAR Adaptive Platform and need to run and work properly in any case. Therefore, Platform Health Management shall always supervise the corresponding Processes for State Management and Execution Management. Supervision failures in these Processes shall be recovered by a reset of the Machine because the normal way of error recovery (via State Management and Execution Management) is no longer reliable.</p>

### 9.6.1.1 Defined interfaces

The interfaces of Platform Health Management are categorized into interfaces for supervision (see Section 9.6.1.1.1), interfaces for performing recovery actions (see Section 9.6.1.1.2), and interfaces for hardware watchdog handling (see Section 9.6.1.1.3).

#### 9.6.1.1.1 Interfaces for supervision

Processes that are supervised by Platform Health Management shall report via the [SupervisedEntity](#) interface when they have reached a certain checkpoint in their control flow (see Figure 9.65). Platform Health Management independently monitors that all checkpoints configured in the Manifest have been reached in time and in the expected order (depending on the type of supervision).

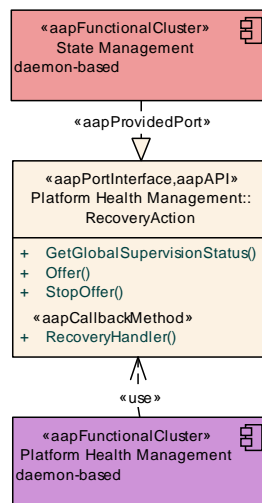


**Figure 9.65: Interfaces for supervision**

<b>Name:</b>	SupervisedEntity	
<b>Technology:</b>	Port interface	
<b>Generated:</b>	No	
<b>Meta-model interface type:</b>	PhmSupervisedEntityInterface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides functions to report checkpoints to Platform Health Management.	
<b>Operations:</b>	ReportCheckpoint	Reports an occurrence of a checkpoint.

**9.6.1.1.2 Interfaces for recovery**

Platform Health Management defines the [RecoveryAction](#) API to trigger a recovery action in case a supervision failed (see [Figure 9.66](#)).



**Figure 9.66: Interfaces for recovery**

<b>Name:</b>	RecoveryAction	
<b>Technology:</b>	Port interface	
<b>Generated:</b>	No	
<b>Meta-model interface type:</b>	PhmRecoveryActionInterface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides functions to control triggering of recovery actions, to determine the status of the supervision and a callback to perform recovery.	
<b>Operations:</b>	GetGlobalSupervisionStatus	Returns the status of global supervision that the supervised entity belongs to.
	Offer	Enables potential invocations of the callback <a href="#">RecoveryAction::RecoveryHandler</a>
	RecoveryHandler	Callback to be invoked by Platform Health Management upon a supervision failure. The handler invocation needs to be enabled before using <a href="#">RecoveryAction::Offer</a> .
	StopOffer	Disables potential invocations of the callback <a href="#">RecoveryAction::RecoveryHandler</a>

### 9.6.1.1.3 Interfaces for watchdog handling

Platform Health Management defines the [WatchdogInterface](#) extension API to interact with the hardware watchdog (see Figure 9.67).

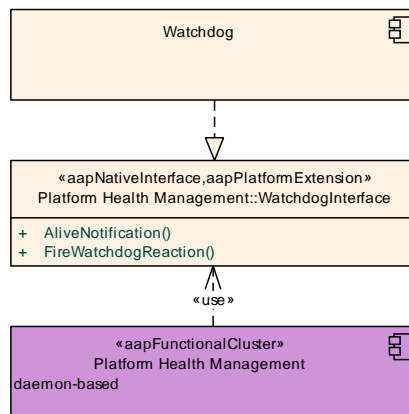


Figure 9.67: Interfaces for watchdog handling

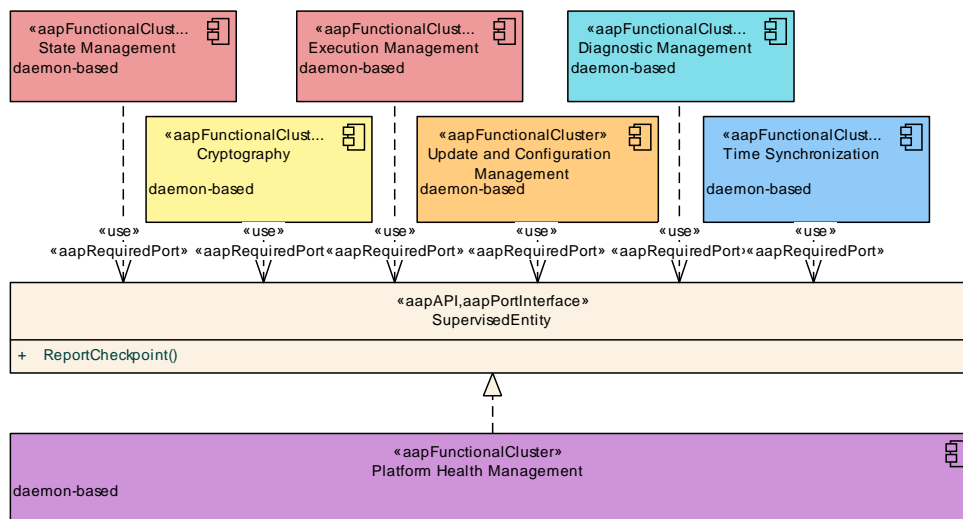
<b>Name:</b>	WatchdogInterface
<b>Technology:</b>	Native interface
<b>Usage:</b>	Platform extension





<b>Description:</b>	This interface provides functions to control the hardware watchdog.	
<b>Operations:</b>	AliveNotification	Called cyclically by Platform Health Management in configurable cycle time. Note: This time might differ from the cycle time of triggering the "real" hardware watchdog. If Platform Health Management does not report aliveness in configured time, WatchdogInterface shall initiate watchdog reaction.
	FireWatchdogReaction	Initiates an error reaction of the hardware watchdog.

### 9.6.1.2 Provided interfaces

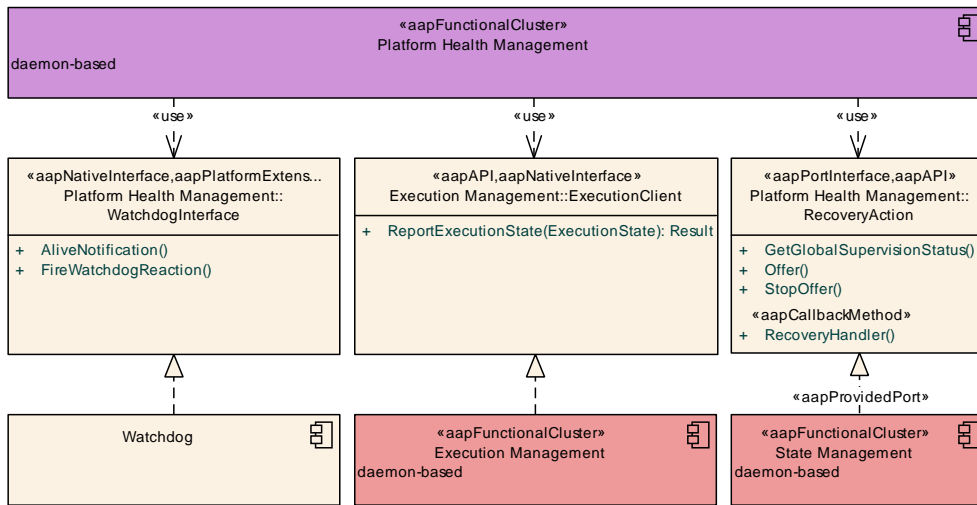


**Figure 9.68: Users of the SupervisedEntity interface**

Interface	Requiring functional clusters
SupervisedEntity	Cryptography (see Section 9.5.1)
	Diagnostic Management (see Section 9.8.1)
	Execution Management (see Section 9.2.1)
	State Management (see Section 9.2.2)
	Time Synchronization (see Section 9.3.3)
	Update and Configuration Management (see Section 9.7.1)

**Table 9.21: Interfaces provided by Platform Health Management to other Functional Clusters**

**9.6.1.3 Required interfaces**

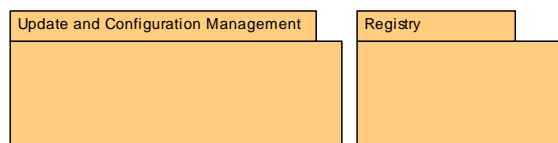


**Figure 9.69: Interfaces required by Platform Health Management**

Interface	Purpose
<a href="#">ExecutionClient</a>	Platform Health Management uses this interface to report the state of its daemon process to Execution Management.
<a href="#">Logger</a>	Platform Health Management shall use this interface to log standardized messages.
<a href="#">Manifest Accessor</a>	Platform Health Management shall use this interface to read information about <a href="#">SupervisedEntities</a> from the Manifests.
<a href="#">RecoveryAction</a>	Platform Health Management uses this interface to trigger failure recovery.
<a href="#">WatchdogInterface</a>	Platform Health Management uses this interface to control the hardware watchdog.

**Table 9.22: Interfaces required by Platform Health Management**

**9.7 Configuration**



**Figure 9.70: Overview of Configuration and its building blocks**

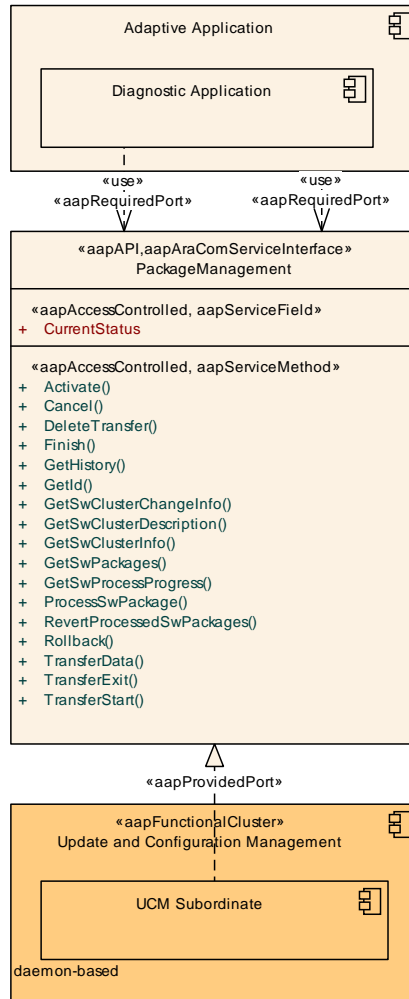
## 9.7.1 Update and Configuration Management

<b>Name:</b>	Update and Configuration Management
<b>Short name:</b>	ucm
<b>Category:</b>	Configuration
<b>Daemon-based:</b>	Yes
<b>Responsibilities:</b>	<p>Update and Configuration Management is responsible for updating, installing, removing and keeping a record of the software on an AUTOSAR Adaptive Platform in a safe and secure way. Hereby, Update and Configuration Management enables to update the software and its configuration flexibly through over-the-air updates (OTA).</p> <p>Update and Configuration Management is separated into two main components UCM Master and UCM Subordinate. UCM Subordinate controls the update process on the local Adaptive Platform. UCM Master controls an update of the software in the entire vehicle.</p>

### 9.7.1.1 Defined interfaces

The interfaces of Update and Configuration Management are categorized into interfaces for UCM Subordinate (see Section 9.7.1.1.1), interfaces for UCM Master (see Section 9.7.1.1.2), and interfaces for the D-PDU API (see Section 9.7.1.1.3).

**9.7.1.1.1 UCM Subordinate**



**Figure 9.71: Interfaces of UCM Subordinate**

<b>Name:</b>	PackageManagement	
<b>Technology:</b>	ara::com service interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides functionality for managing and transferring Software Packages to an UCM.	
<b>Operations:</b>	Activate	Activates the processed components.
	Cancel	Aborts an ongoing processing of a Software Package.
	DeleteTransfer	Delete a transferred Software Package.





	Finish	Finishes the processing for the current set of processed <code>Software Packages</code> . It does a cleanup of all data of the processing including the sources of the <code>Software Packages</code> .
	GetHistory	Retrieve all actions that have been performed by UCM.
	GetId	Get the UCM Instance Identifier.
	GetSwClusterChangeInfo	Get a list of pending changes to the set of <code>Software Clusters</code> on the Adaptive Platform. The returned list includes all <code>Software Clusters</code> that are to be added, updated or removed. The list of changes is extended in the course of processing <code>Software Packages</code> .
	GetSwClusterDescription	Get the general information of the <code>Software Clusters</code> present in the platform.
	GetSwClusterInfo	Get a list of <code>Software Clusters</code> that are in state <code>kPresent</code> .
	GetSwPackages	Get the <code>Software Packages</code> that available in UCM.
	GetSwProcessProgress	Get the progress of the currently processed <code>Software Package</code> .
	ProcessSwPackage	Process a previously transferred <code>Software Package</code> .
	RevertProcessedSwPackages	Revert the changes done by processing (by calling <code>ProcessSwPackage()</code> ) of one or several <code>Software Packages</code> .
	Rollback	Rollback the system to the state before the packages were processed.
	TransferData	Block-wise transfer of a <code>Software Package</code> to UCM.
	TransferExit	Finish the transfer of a <code>Software Package</code> to UCM.
	TransferStart	Start the transfer of a <code>Software Package</code> .
<b>Fields:</b>	CurrentStatus	The current status of the UCM.



9.7.1.1.2 UCM Master

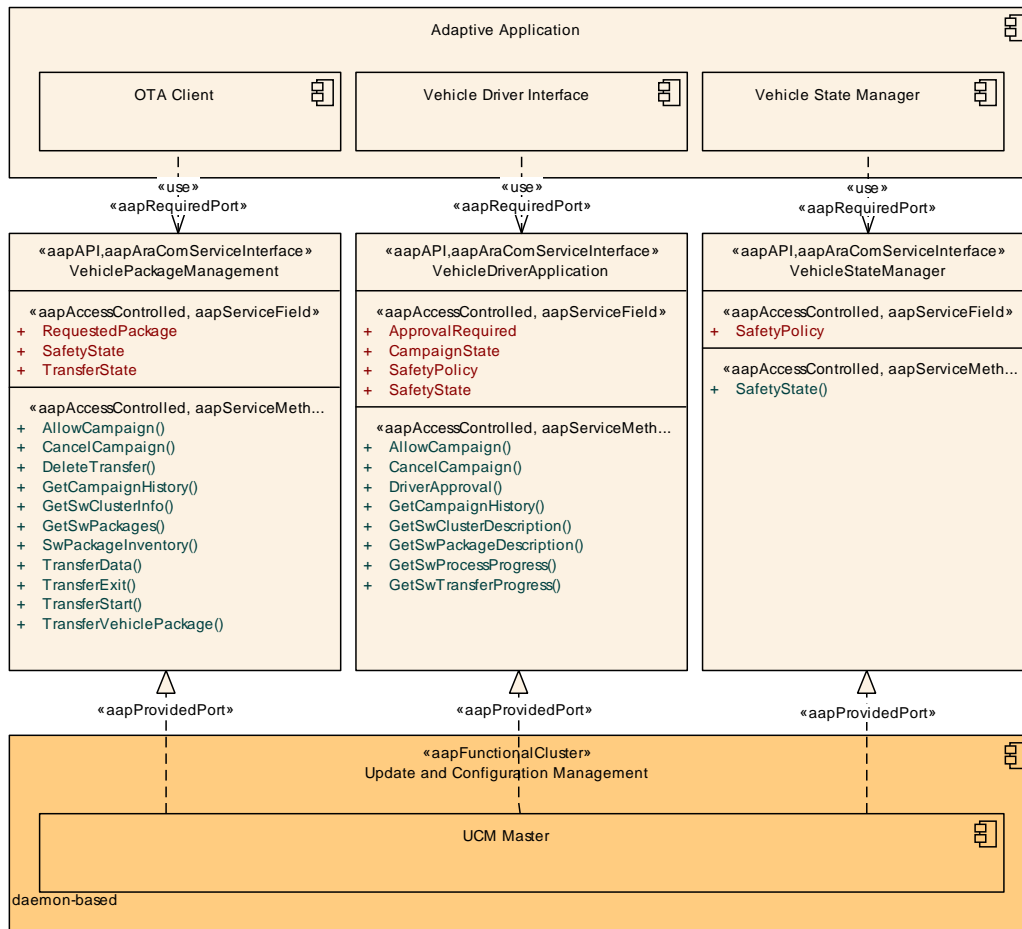


Figure 9.72: Interfaces of UCM Master

<b>Name:</b>	VehiclePackageManagement	
<b>Technology:</b>	ara::com service interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides functionality for managing and transferring Vehicle Packages and Software Packages to UCM Master.	
<b>Operations:</b>	AllowCampaign	Allows a new campaign to start.
	CancelCampaign	Aborts an ongoing campaign processing of a Vehicle Package.
	DeleteTransfer	Delete a transferred Software Package or Vehicle Package.
	GetCampaignHistory	Retrieve all actions that have been performed by UCM Master.
	GetSwClusterInfo	Get a list of SoftwareClusters that are in state kPresent.





	GetSwPackages	Get the Software Packages that are part of current campaign handled by UCM Master.
	SwPackageInventory	Performs an inventory of all Software Packages.
	TransferData	Block-wise transfer of a Software Package or Vehicle Package to UCM Master.
	TransferExit	Finish the transfer of a Software Package or Vehicle Package to UCM Master.
	TransferStart	Start the transfer of a Software Package to UCM Master.
	TransferVehiclePackage	Start the transfer of a Vehicle Package to UCM Master.
<b>Fields:</b>	RequestedPackage	Software Package to be transferred to UCM Master.
	SafetyState	Vehicle state computed by the Vehicle State Manager Adaptive Application.
	TransferState	The current status of a campaign from an OTA Client perspective.

<b>Name:</b>	VehicleDriverApplication	
<b>Technology:</b>	ara::com service interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides functionality to interact with the vehicle driver, for example to approve updates.	
<b>Operations:</b>	AllowCampaign	Allow a new campaign to start.
	CancelCampaign	Aborts an ongoing campaign processing of a Vehicle Package.
	DriverApproval	Inform UCM Master of the driver's notification resolution (approve or reject).
	GetCampaignHistory	Retrieve all actions that have been performed by UCM Master.
	GetSwClusterDescription	Get the general information of the Software Clusters present in the Adaptive Platform.
	GetSwPackageDescription	Get the general information of the Software Packages that are part of current campaign handled by UCM Master.
	GetSwProcessProgress	Get the progress of the current package processing.
	GetSwTransferProgress	Get the progress of the current package transfer.

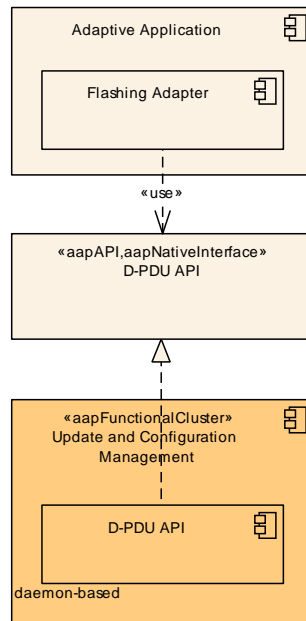




<b>Fields:</b>	ApprovalRequired	Flag to inform an Adaptive Application if approval from a driver is required at current state based on the Vehicle Package Manifest.
	CampaignState	The current status of campaign.
	SafetyPolicy	Safety policy from the Vehicle Package to be computed by the Vehicle State Manager Adaptive Application.
	SafetyState	Vehicle state computed by the Vehicle State Manager Adaptive Application.

<b>Name:</b>	VehicleStateManager	
<b>Technology:</b>	ara::com service interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides functionality for a Vehicle State Manager Adaptive Application to inform UCM Master about the safety state and policy of the vehicle.	
<b>Operations:</b>	SafetyState	Called by the Vehicle State Manager Adaptive Application when safety state is changed.
<b>Fields:</b>	SafetyPolicy	Safety policy from the Vehicle Package to be computed by the Vehicle State Manager Adaptive Application.

**9.7.1.1.3 D-PDU API**



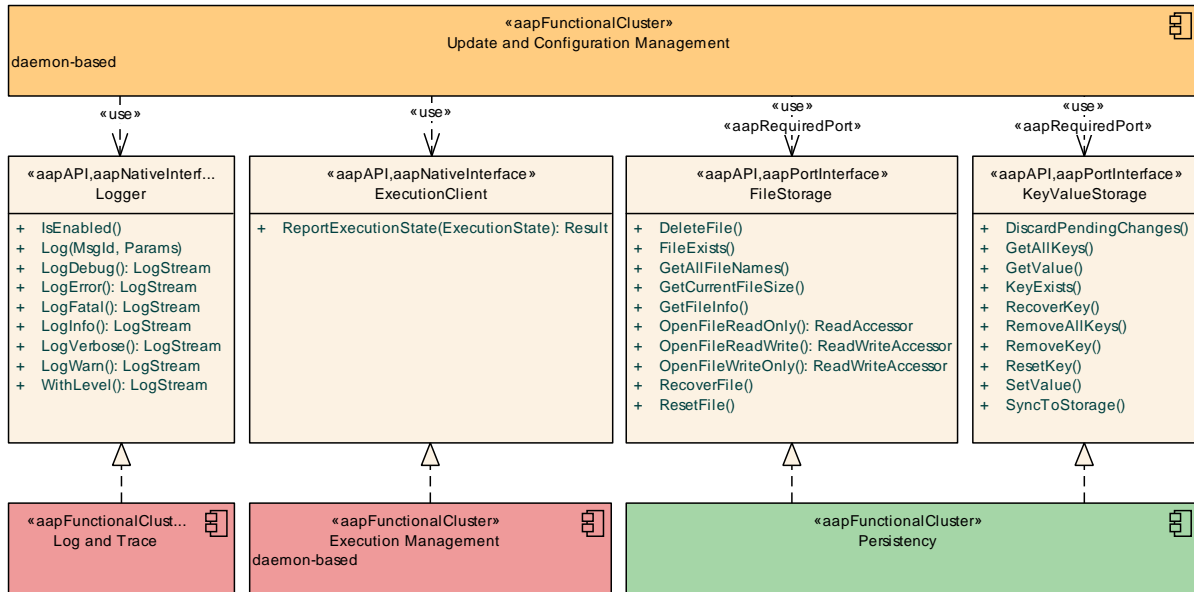
**Figure 9.73: Interfaces of UCM Master**

<b>Name:</b>	D-PDU API
<b>Technology:</b>	Native interface
<b>Usage:</b>	Public API
<b>Description:</b>	This interface represents the Diagnostic Protocol Data Unit Application Programming Interface. It is not detailed in this document.

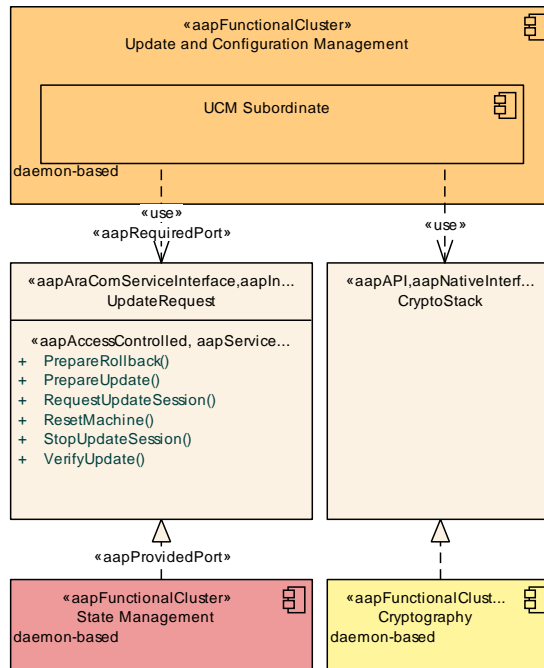
**9.7.1.2 Provided interfaces**

Update and Configuration Management does not provide any interfaces to other Functional Clusters.

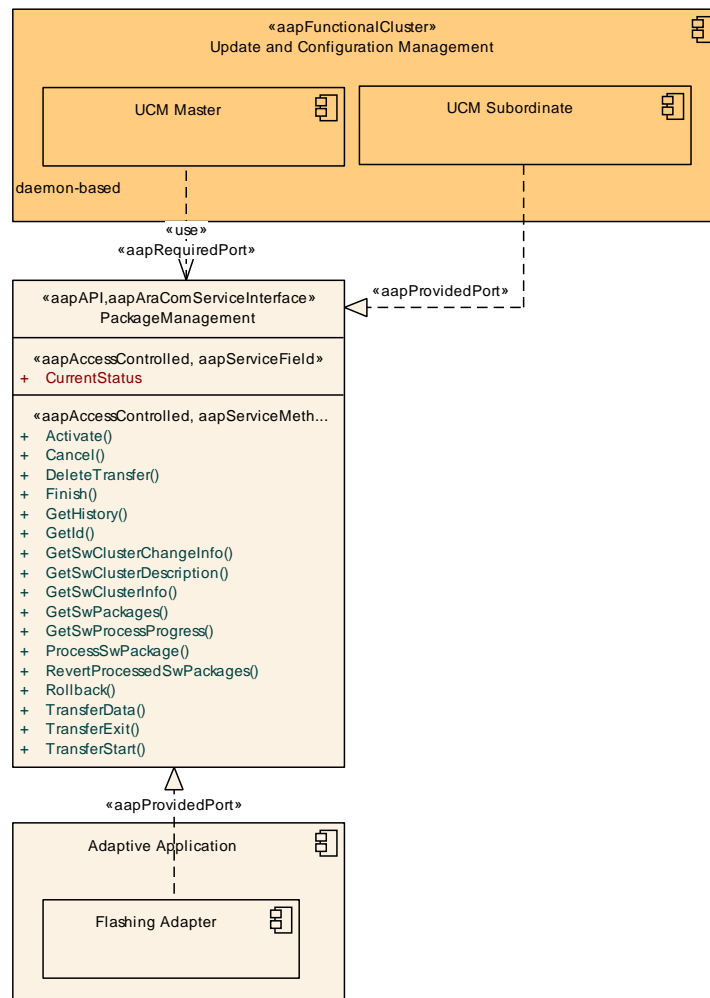
**9.7.1.3 Required interfaces**



**Figure 9.74: Interfaces required by Update and Configuration Management**



**Figure 9.75: Interfaces required by UCM Subordinate**



**Figure 9.76: Interfaces required by UCM Master**

<b>Interface</b>	<b>Purpose</b>
<a href="#">ExecutionClient</a>	This interface shall be used by the daemon process(es) inside Update and Configuration Management to report their execution state to Execution Management.
<a href="#">FileStorage</a>	This interface should be used to store files, for example downloaded packages.
<a href="#">KeyValueStorage</a>	This interface should be used to store internal state of Update and Configuration Management.
<a href="#">Logger</a>	Update and Configuration Management shall use this interface to log standardized messages.
<a href="#">Manifest Accessor</a>	Update and Configuration Management shall use this interface to read information about its configuration from the Manifests.
<a href="#">ReadAccessor</a>	This interface should be used to store files, for example downloaded packages.





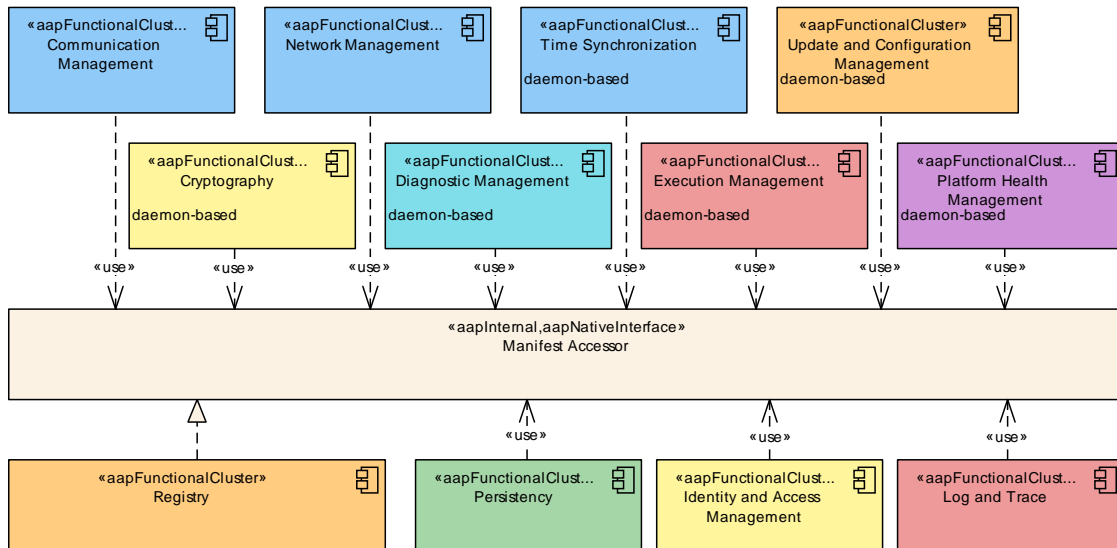
<a href="#">ReadWriteAccessor</a>	This interface should be used to store files, for example downloaded packages.
<a href="#">SupervisedEntity</a>	This interface should be used to supervise the daemon process(es) of Update and Configuration Management.

**Table 9.23: Interfaces required by Update and Configuration Management**

### 9.7.2 Registry

<b>Name:</b>	Registry
<b>Short name:</b>	n/a
<b>Category:</b>	Configuration
<b>Daemon-based:</b>	No
<b>Responsibilities:</b>	The Registry is an internal component of the AUTOSAR Adaptive Platform that provides access the information stored in Manifests. It is not intended to be used by Adaptive Applications directly.

#### 9.7.2.1 Defined interfaces



**Figure 9.77: Interfaces of Registry**

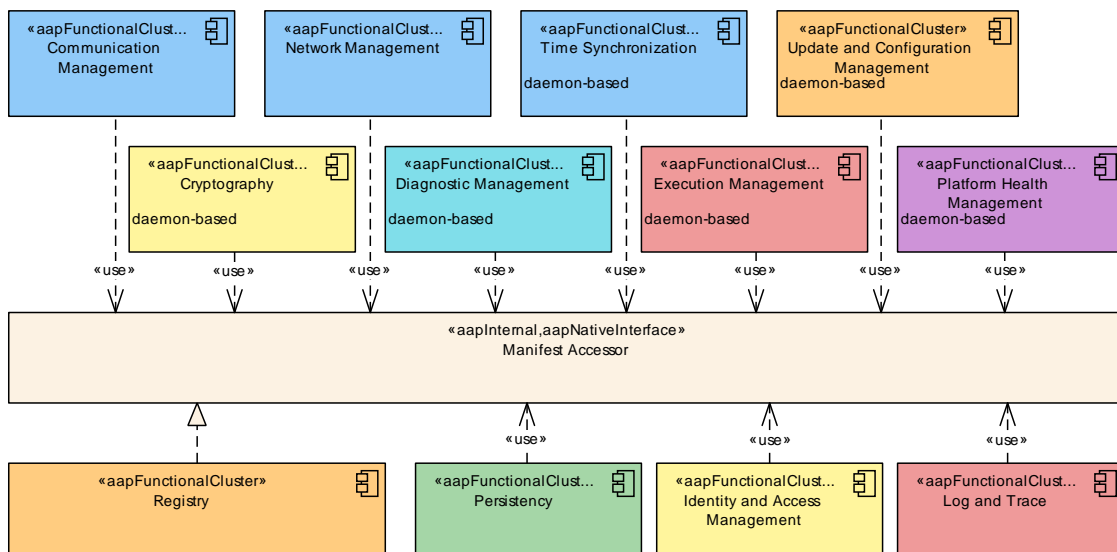
<b>Name:</b>	Manifest Accessor
<b>Technology:</b>	Native interface





<b>Usage:</b>	Internal
<b>Description:</b>	This interface provides functionality to read information that was modeled in the Manifest(s). This interface is not detailed in this document.

### 9.7.2.2 Provided interfaces



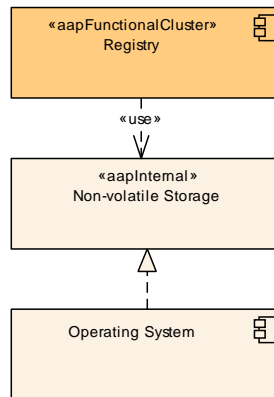
**Figure 9.78: Users of the Registry interfaces**

<b>Interface</b>	<b>Requiring functional clusters</b>
Manifest Accessor	Communication Management (see Section 9.3.1)
	Cryptography (see Section 9.5.1)
	Diagnostic Management (see Section 9.8.1)
	Execution Management (see Section 9.2.1)
	Identity and Access Management (see Section 9.5.2)
	Log and Trace (see Section 9.2.3)
	Network Management (see Section 9.3.2)
	Persistency (see Section 9.4.1)
	Platform Health Management (see Section 9.6.1)
	Time Synchronization (see Section 9.3.3)
Update and Configuration Management (see Section 9.7.1)	

**Table 9.24: Interfaces provided by Registry to other Functional Clusters**



### 9.7.2.3 Required interfaces

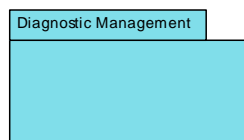


**Figure 9.79: Interfaces required by Registry**

<i>Interface</i>	<i>Purpose</i>
Non-volatile Storage	Registry shall use this interface to read the information from the Manifest(s).

**Table 9.25: Interfaces required by Registry**

## 9.8 Diagnostics



**Figure 9.80: Overview of Diagnostics and its building blocks**

### 9.8.1 Diagnostic Management

<b>Name:</b>	Diagnostic Management
<b>Short name:</b>	diag
<b>Category:</b>	Diagnostics
<b>Daemon-based:</b>	Yes

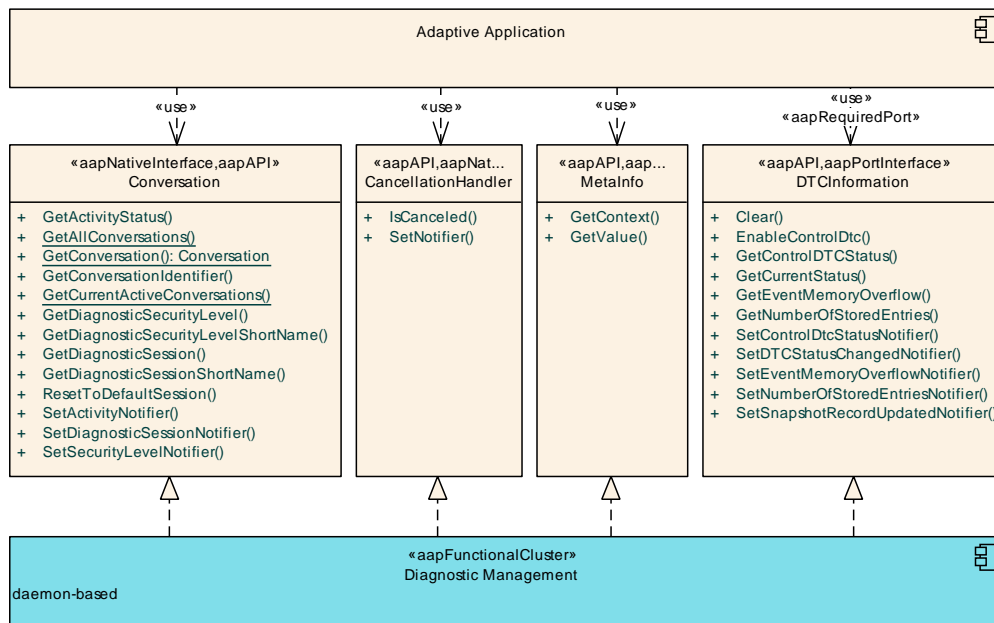




<b>Responsibilities:</b>	Diagnostic Management is responsible for handling diagnostic events produced by the individual Processes running in an AUTOSAR Runtime for Adaptive Applications. Diagnostic Management stores such events and the associated data persistently according to rendition policies. Diagnostic Management also provides access to diagnostic data for external Diagnostic Clients via standardized network protocols (ISO 14229-5 (UDSonIP) which is based on the ISO 14229-1 (UDS) and ISO 13400-2 (DoIP)).
--------------------------	---

### 9.8.1.1 Defined interfaces

#### 9.8.1.1.1 Common interfaces



**Figure 9.81: Common interfaces of Diagnostic Management**

<b>Name:</b>	Conversation	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides functionality to handle diagnostic conversations.	
<b>Operations:</b>	GetActivityStatus	Represents the status of an active conversation.
	GetAllConversations	Get all possible conversations.
	GetConversation	Get one conversation based on given meta information.
	GetConversationIdentifier	Getter for the current identification properties of the active conversation.





	GetCurrentActiveConversations	Get all currently active conversations.
	GetDiagnosticSecurityLevel	Represents the current active diagnostic <code>SecurityLevel</code> of an active conversation.
	GetDiagnosticSecurityLevelShortName	Converts the given diagnostic <code>SecurityLevel</code> into the <code>ShortName</code> .
	GetDiagnosticSession	Represents the current active diagnostic session of an active conversation.
	GetDiagnosticSessionShortName	Converts the given diagnostic session into the <code>ShortName</code> .
	ResetToDefaultSession	Method to reset the current session to the default session.
	SetActivityNotifier	Register a notifier function which is called if the activity is changed.
	SetDiagnosticSessionNotifier	Register a notifier function which is called if the <code>Session</code> is changed.
	SetSecurityLevelNotifier	Register a notifier function which is called if the <code>SecurityLevel</code> is changed.

<b>Name:</b>	CancellationHandler	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface holds a shared state if the processing should be canceled.	
<b>Operations:</b>	IsCanceled	Returns true in if the diagnostic service execution is canceled.
	SetNotifier	Registers a notifier function which is called if the diagnostic service execution is canceled.

<b>Name:</b>	MetaInfo	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface specifies a mechanism to provide meta information, i.e. from transport protocol layer, to an interested application.	
<b>Operations:</b>	GetContext	Get the context of the invocation.
	GetValue	Get the metainfo value for a given key.

<b>Name:</b>	DTCInformation	
<b>Technology:</b>	Port interface	
<b>Generated:</b>	No	
<b>Meta-model interface type:</b>	DiagnosticDTCInformationInterface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides operations on DTC information per configured <code>DiagnosticMemory</code> destination.	
<b>Operations:</b>	Clear	Method for Clearing a DTC or a group of DTCs.
	EnableControlDtc	Enforce restoring ControlDTCStatus setting to enabled in case the monitor has some conditions or states demands to do so.
	GetControlDTCStatus	Contains the current status of the ControlDTCStatus.
	GetCurrentStatus	Retrieves the current UDS DTC status byte of the given DTC identifier.
	GetEventMemoryOverflow	Contains the current event memory overflow status.
	GetNumberOfStoredEntries	Contains the number of currently stored fault memory entries.
	SetControlDtcStatusNotifier	Registers a notifier function which is called if the control DTC setting is changed.
	SetDTCStatusChangedNotifier	Register a notifier function which is called if a UDS DTC status is changed.
	SetEventMemoryOverflowNotifier	Register a notifier function which is called if the current event memory overflow status changed.
	SetNumberOfStoredEntriesNotifier	Register a notifier function which is called if the number of currently stored fault memory entries changed.
	SetSnapshotRecordUpdatedNotifier	Register a notifier function which is called if the SnapshotRecord is changed.

### 9.8.1.1.2 Interfaces for request handling

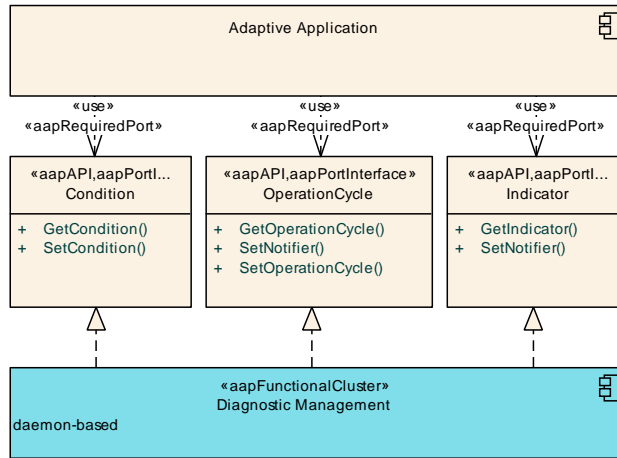
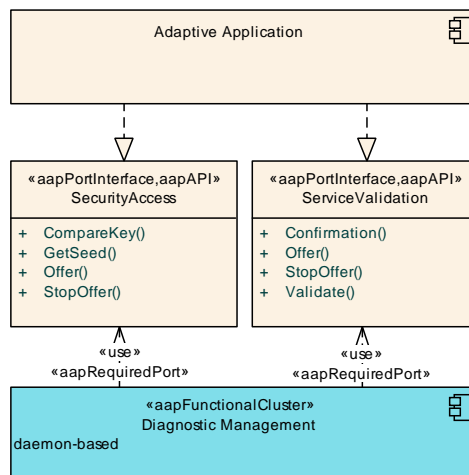


Figure 9.82: Interfaces for request handling

<b>Name:</b>	Condition	
<b>Technology:</b>	Port interface	
<b>Generated:</b>	No	
<b>Meta-model interface type:</b>	DiagnosticConditionInterface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides functionality for condition management.	
<b>Operations:</b>	GetCondition	Get the current condition.
	SetCondition	Set the current condition.

<b>Name:</b>	OperationCycle	
<b>Technology:</b>	Port interface	
<b>Generated:</b>	No	
<b>Meta-model interface type:</b>	DiagnosticOperationCycleInterface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides functionality for handling of operation cycles.	
<b>Operations:</b>	GetOperationCycle	Get the current OperationCycle.
	SetNotifier	Registers a notifier function which is called if the OperationCycle is changed.
	SetOperationCycle	Set the current OperationCycle.

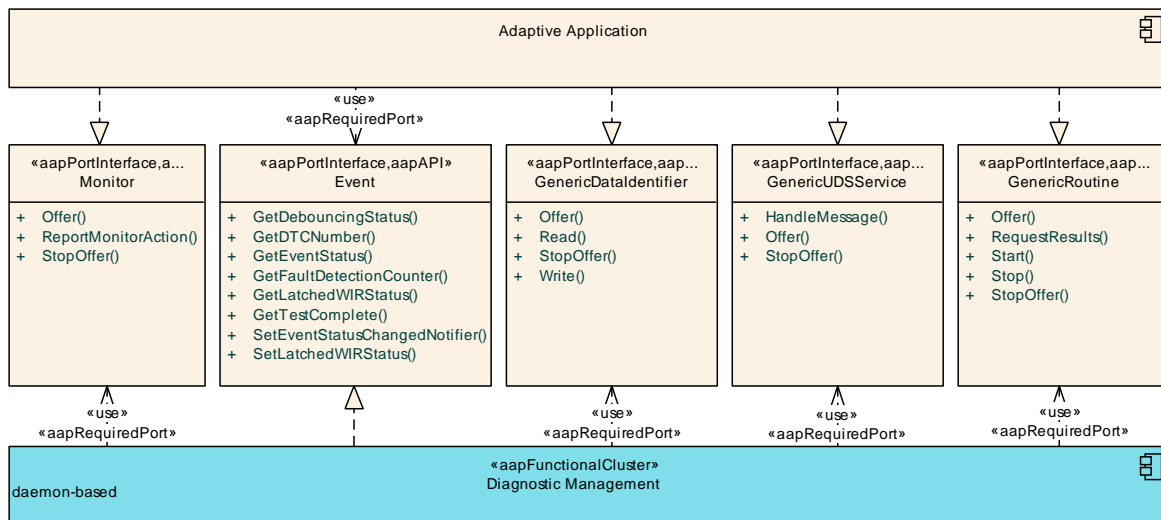
<b>Name:</b>	Indicator	
<b>Technology:</b>	Port interface	
<b>Generated:</b>	No	
<b>Meta-model interface type:</b>	DiagnosticIndicatorInterface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides functionality for handling indicators.	
<b>Operations:</b>	GetIndicator	Get current Indicator.
	SetNotifier	Register a notifier function which is called if the indicator is updated.



**Figure 9.83: Interfaces for request handling**

<b>Name:</b>	SecurityAccess	
<b>Technology:</b>	Port interface	
<b>Generated:</b>	No	
<b>Meta-model interface type:</b>	DiagnosticSecurityLevelInterface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides functionality for handling <code>SecurityAccess</code> requests.	
<b>Operations:</b>	CompareKey	This method is called, when a diagnostic request has been finished, to notify about the outcome.
	GetSeed	Called for any request message.
	Offer	Enable forwarding of request messages from Diagnostic Management.
	StopOffer	Disable forwarding of request messages from Diagnostic Management.

<b>Name:</b>	ServiceValidation	
<b>Technology:</b>	Port interface	
<b>Generated:</b>	No	
<b>Meta-model interface type:</b>	DiagnosticServiceValidationInterface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides functionality for handling <code>ServiceValidation</code> requests.	
<b>Operations:</b>	Confirmation	This method is called, when a diagnostic request has been finished, to notify about the outcome.
	Offer	Enable forwarding of request messages from <code>Diagnostic Management</code> .
	StopOffer	Disable forwarding of request messages from <code>Diagnostic Management</code> .
	Validate	Called for any request message.



**Figure 9.84: Interfaces for generic request handling**

<b>Name:</b>	Monitor	
<b>Technology:</b>	Port interface	
<b>Generated:</b>	No	
<b>Meta-model interface type:</b>	DiagnosticMonitorInterface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides functionality to report qualified and unqualified test results and to control debouncing options.	
<b>Operations:</b>	Offer	Enable forwarding of request messages from <code>Diagnostic Management</code> .





	ReportMonitorAction	Report the status information being relevant for error monitoring paths.
	StopOffer	Disable forwarding of request messages from Diagnostic Management.

<b>Name:</b>	Event	
<b>Technology:</b>	Port interface	
<b>Generated:</b>	No	
<b>Meta-model interface type:</b>	DiagnosticEventInterface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface defines functionality for diagnostic events.	
<b>Operations:</b>	GetDTCNumber	Returns the DTC-ID related to this event instance.
	GetDebouncingStatus	Get the current debouncing status.
	GetEventStatus	Returns the current diagnostic event status.
	GetFaultDetectionCounter	Returns the current value of Fault Detection Counter of this event.
	GetLatchedWIRStatus	Returns the current warning indicator status.
	GetTestComplete	Get the status if the event has matured to test completed (corresponds to FDC = -128 or FDC = 127).
	SetEventStatusChangedNotifier	Register a notifier function which is called if a diagnostic event is changed.
	SetLatchedWIRStatus	Set the warning indicator status.

<b>Name:</b>	GenericDataIdentifier	
<b>Technology:</b>	Port interface	
<b>Generated:</b>	No	
<b>Meta-model interface type:</b>	DiagnosticDataIdentifierGenericInterface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Generic interface to handle <code>ReadDataByIdentifier</code> and <code>WriteDataByIdentifier</code> requests.	
<b>Operations:</b>	Offer	Enable forwarding of request messages from Diagnostic Management.
	Read	Called for a <code>ReadDataByIdentifier</code> request for this <code>DiagnosticDataIdentifier</code> .



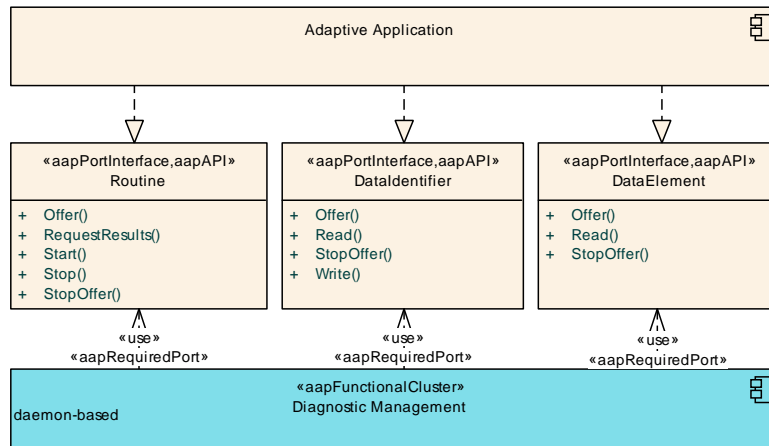


△

	StopOffer	Disable forwarding of request messages from Diagnostic Management.
	Write	Called for a WriteDataByIdentifier request for this DiagnosticDataIdentifier.

<b>Name:</b>	GenericUDSService	
<b>Technology:</b>	Port interface	
<b>Generated:</b>	No	
<b>Meta-model interface type:</b>	DiagnosticGenericUdsInterface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Generic interface to handle UDS messages.	
<b>Operations:</b>	HandleMessage	Handles an UDS request message.
	Offer	Enable forwarding of request messages from Diagnostic Management.
	StopOffer	Disable forwarding of request messages from Diagnostic Management.

<b>Name:</b>	GenericRoutine	
<b>Technology:</b>	Port interface	
<b>Generated:</b>	No	
<b>Meta-model interface type:</b>	DiagnosticRoutineGenericInterface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Generic interface to handle RoutineControl requests.	
<b>Operations:</b>	Offer	Enable forwarding of request messages from Diagnostic Management.
	RequestResults	Called for RoutineControl with SubFunction RequestResults request for this DiagnosticRoutineIdentifier.
	Start	Called for RoutineControl with SubFunction Start request for this DiagnosticRoutineIdentifier.
	Stop	Called for RoutineControl with SubFunction Stop request for this DiagnosticRoutineIdentifier.
	StopOffer	Disable forwarding of request messages from Diagnostic Management.



**Figure 9.85: Generated interfaces for request handling**

<b>Name:</b>	DataIdentifier	
<b>Technology:</b>	Port interface	
<b>Generated:</b>	Yes	
<b>Meta-model interface type:</b>	DiagnosticDataIdentifierInterface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Generated interface to handle ReadDataByIdentifier and WriteDataByIdentifier requests.	
<b>Operations:</b>	Offer	Enable forwarding of request messages from Diagnostic Management.
	Read	Called for a ReadDataByIdentifier request for this DiagnosticDataIdentifier.
	StopOffer	Disable forwarding of request messages from Diagnostic Management.
	Write	Called for a WriteDataByIdentifier request for this DiagnosticDataIdentifier.

<b>Name:</b>	DataElement	
<b>Technology:</b>	Port interface	
<b>Generated:</b>	Yes	
<b>Meta-model interface type:</b>	DiagnosticDataElementInterface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Generated interface to handle read requests for DataElements.	
<b>Operations:</b>	Offer	Enable forwarding of request messages from Diagnostic Management.

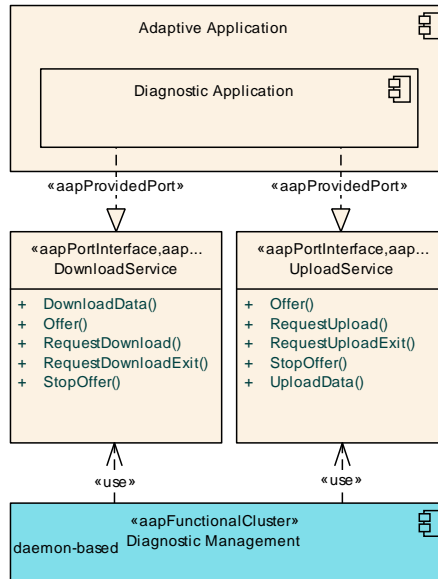




	Read	Called for reading a <code>DataElement</code> .
	StopOffer	Disable forwarding of request messages from <code>Diagnostic Management</code> .

<b>Name:</b>	Routine	
<b>Technology:</b>	Port interface	
<b>Generated:</b>	Yes	
<b>Meta-model interface type:</b>	<code>DiagnosticRoutineInterface</code>	
<b>Usage:</b>	Public API	
<b>Description:</b>	Generated interface to handle <code>RoutineControl</code> requests.	
<b>Operations:</b>	Offer	Enable forwarding of request messages from <code>Diagnostic Management</code> .
	RequestResults	Called for <code>RoutineControl</code> with <code>SubFunction RequestResults</code> request for this <code>DiagnosticRoutineIdentifier</code> .
	Start	Called for <code>RoutineControl</code> with <code>SubFunction Start</code> request for this <code>DiagnosticRoutineIdentifier</code> .
	Stop	Called for <code>RoutineControl</code> with <code>SubFunction Stop</code> request for this <code>DiagnosticRoutineIdentifier</code> .
	StopOffer	Disable forwarding of request messages from <code>Diagnostic Management</code> .

**9.8.1.1.3 Interfaces for Upload and Download**



**Figure 9.86: Interfaces for Upload and Download**

<b>Name:</b>	UploadService	
<b>Technology:</b>	Port interface	
<b>Generated:</b>	No	
<b>Meta-model interface type:</b>	DiagnosticUploadInterface	
<b>Usage:</b>	Public API	
<b>Description:</b>	Upload service interface.	
<b>Operations:</b>	Offer	Enable forwarding of request messages from Diagnostic Management.
	RequestUpload	Called for RequestUpload.
	RequestUploadExit	Called for RequestTransferExit.
	StopOffer	Disable forwarding of request messages from Diagnostic Management.
	UploadData	Called for TransferData following a previous RequestUpload.

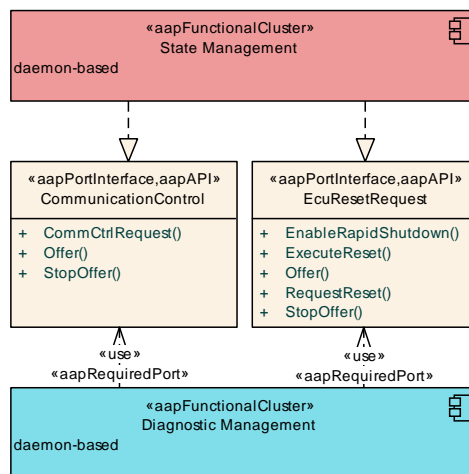
<b>Name:</b>	DownloadService	
<b>Technology:</b>	Port interface	
<b>Generated:</b>	No	
<b>Meta-model interface type:</b>	DiagnosticDownloadInterface	





<b>Usage:</b>	Public API	
<b>Description:</b>	Download service interface.	
<b>Operations:</b>	DownloadData	Called for TransferData following a previous RequestDownload.
	Offer	Enable forwarding of request messages from Diagnostic Management.
	RequestDownload	Called for RequestDownload.
	RequestDownloadExit	Called for RequestTransferExit.
	StopOffer	Disable forwarding of request messages from Diagnostic Management.

### 9.8.1.1.4 Interfaces for State Management



**Figure 9.87: Interfaces for State Management**

<b>Name:</b>	CommunicationControl	
<b>Technology:</b>	Port interface	
<b>Generated:</b>	No	
<b>Meta-model interface type:</b>	DiagnosticComControl	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides functionality for CommunicationControl.	
<b>Operations:</b>	CommCtrlRequest	Called for CommunicationControl (x028) with any subfunction as subfunction value is part of argument list.
	Offer	Enable forwarding of request messages from Diagnostic Management.

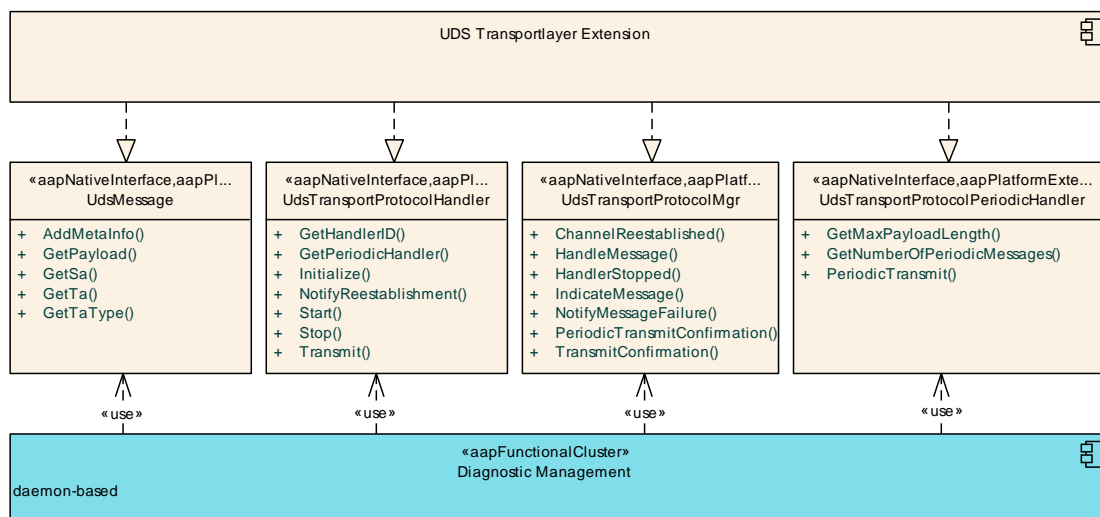




	StopOffer	Disable forwarding of request messages from Diagnostic Management.
--	-----------	--

<b>Name:</b>	EcuResetRequest	
<b>Technology:</b>	Port interface	
<b>Generated:</b>	Yes	
<b>Meta-model interface type:</b>	DiagnosticEcuResetInterface	
<b>Usage:</b>	Public API	
<b>Description:</b>	This interface provides functionality for EcuReset requests.	
<b>Operations:</b>	EnableRapidShutdown	Called for subfunction En-/DisableRapidShutdown.
	ExecuteReset	Execute the requested reset.
	Offer	Enable forwarding of request messages from Diagnostic Management.
	RequestReset	Called for any EcuReset subfunction, except En-/DisableRapidShutdown.
	StopOffer	Disable forwarding of request messages from Diagnostic Management.

### 9.8.1.1.5 Interfaces for UDS Transportlayer API



**Figure 9.88: Interfaces for the UDS Transportlayer API**

<b>Name:</b>	UdsMessage	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Platform extension	
<b>Description:</b>	Represents an UDS message exchanged between Diagnostic Management's generic core (UdsTransportProtocolMgr) and a specific implementation of UdsTransportProtocolHandler on diagnostic request reception path or diagnostic response transmission path.	
<b>Operations:</b>	AddMetaInfo	Called by the transport plugin to add channel specific meta-info.
	GetPayload	Get the UDS message data starting with the SID (A_Data as per ISO).
	GetSa	Get the source address of the UDS message.
	GetTa	Get the target address of the UDS message.
	GetTaType	Get the target address type (phys/func) of the UDS message.

<b>Name:</b>	UdsTransportProtocolHandler	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Platform extension	
<b>Description:</b>	This interface provides functionality for general Transport Protocol handling.	
<b>Operations:</b>	GetHandlerID	Return the UdsTransportProtocolHandlerID, which was given to the implementation during construction (ctor call).
	GetPeriodicHandler	Returns the corresponding periodic Transport Protocol handler.
	Initialize	Initializes the handler.
	NotifyReestablishment	Notify the DiagnosticManagement core if the given channel has been re-established.
	Start	Start processing the implemented UDS Transport Protocol.
	Stop	Indicate that this instance should terminate.
	Transmit	Transmit a UDS message via the underlying UDS Transport Protocol channel.

<b>Name:</b>	UdsTransportProtocolMgr	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Platform extension	





<b>Description:</b>	This interface provides functionality to manage messages and their handling.	
<b>Operations:</b>	ChannelReestablished	Notification call from the given transport channel, that it has been reestablished since the last (Re)Start from the <a href="#">UdsTransportProtocolHandler</a> to which this channel belongs.
	HandleMessage	Hands over a valid received UDS message (currently this is only a request type) from transport layer to session layer.
	HandlerStopped	Notification from handler, that it has stopped now (e.g. closed down network connections, freed resources, etc...)
	IndicateMessage	Indicates a message start.
	NotifyMessageFailure	Indicates, that the message indicated via <a href="#">IndicateMessage()</a> has failure and will not lead to a final <a href="#">HandleMessage()</a> call.
	PeriodicTransmitConfirmation	Confirmation of sent messages and number.
	TransmitConfirmation	Notification about the outcome of a transmit request called by core <code>Diagnostic Management</code> at the handler via <a href="#">Transmit()</a>

<b>Name:</b>	UdsTransportProtocolPeriodicHandler	
<b>Technology:</b>	Native interface	
<b>Usage:</b>	Platform extension	
<b>Description:</b>	This interface provides functionality for Transport Protocol handling of periodic messages.	
<b>Operations:</b>	GetMaxPayloadLength	Reports the maximum payload length supported for a single periodic transmission on the channel.
	GetNumberOfPeriodicMessages	Reports the Transport Protocol implementation and connection specific number of periodic messages.
	PeriodicTransmit	Sends all the messages in the list in the given order.



### 9.8.1.1.6 Interfaces for DoIP API

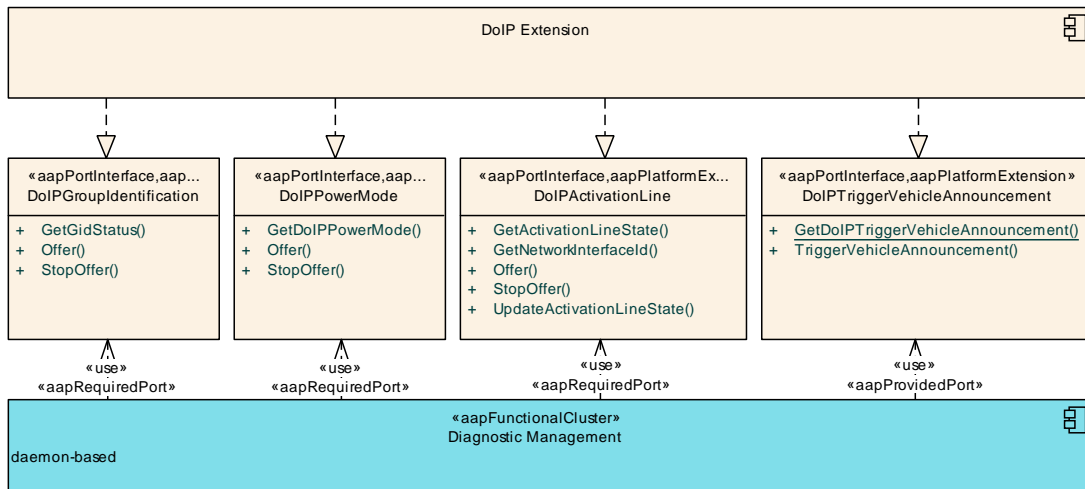


Figure 9.89: Interfaces for the DoIP API

<b>Name:</b>	DoIPGroupIdentification	
<b>Technology:</b>	Port interface	
<b>Generated:</b>	No	
<b>Meta-model interface type:</b>	DiagnosticDoIPGroupIdentificationInterface	
<b>Usage:</b>	Platform extension	
<b>Description:</b>	This interface provides functionality to get the GID state of the DoIP protocol.	
<b>Operations:</b>	GetGidStatus	Called to get the current GID state for the DoIP protocol.
	Offer	Enable forwarding of request messages from Diagnostic Management.
	StopOffer	Disables forwarding of request messages from Diagnostic Management.

<b>Name:</b>	DoIPPowerMode	
<b>Technology:</b>	Port interface	
<b>Generated:</b>	No	
<b>Meta-model interface type:</b>	DiagnosticDoIPPowerModeInterface	
<b>Usage:</b>	Platform extension	
<b>Description:</b>	This interface provides functionality to control the power mode via DoIP.	
<b>Operations:</b>	GetDoIPPowerMode	Called to get the current Power Mode for the DoIP protocol.





	Offer	Enable forwarding of request messages from Diagnostic Management.
	StopOffer	Disable forwarding of request messages from Diagnostic Management.

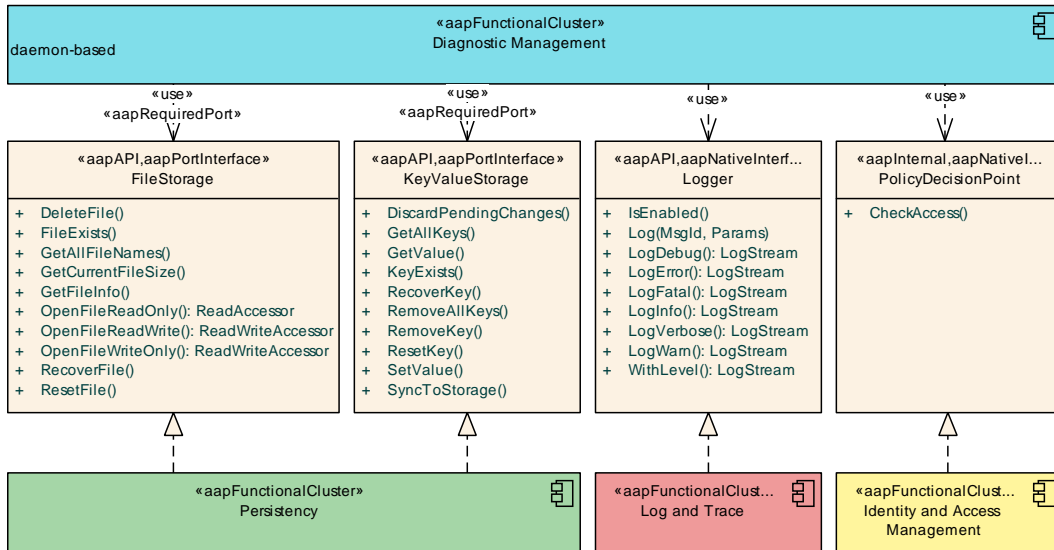
<b>Name:</b>	DoIPActivationLine	
<b>Technology:</b>	Port interface	
<b>Generated:</b>	No	
<b>Meta-model interface type:</b>	DiagnosticDoIPActivationLineInterface	
<b>Usage:</b>	Platform extension	
<b>Description:</b>	This interface provides functionality to control the DoIP activation line.	
<b>Operations:</b>	GetActivationLineState	Get the current activation line state.
	GetNetworkInterfaceId	Get the network interface Id for which this instance is responsible.
	Offer	Enable provision of the activation line state to Diagnostic Management.
	StopOffer	Disable provision of the activation line state to Diagnostic Management.
	UpdateActivationLineState	Update current activation line state.

<b>Name:</b>	DoIPTriggerVehicleAnnouncement	
<b>Technology:</b>	Port interface	
<b>Generated:</b>	No	
<b>Meta-model interface type:</b>	DiagnosticDoIPTriggerVehicleAnnouncementInterface	
<b>Usage:</b>	Platform extension	
<b>Description:</b>	This interface provides functionality to trigger a vehicle announcement via DoIP.	
<b>Operations:</b>	GetDoIPTriggerVehicleAnnouncement	Get the <a href="#">DoIPTriggerVehicleAnnouncement</a> interface.
	TriggerVehicleAnnouncement	Send out vehicle announcements on the given network interface Id.

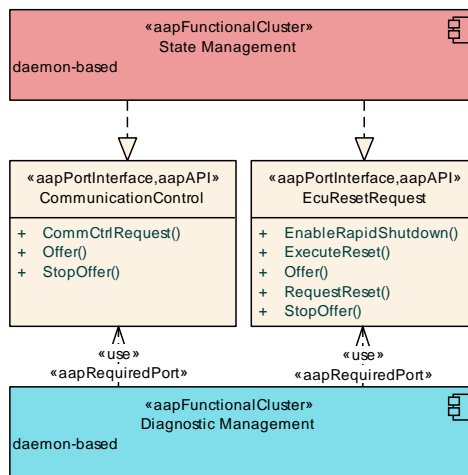
### 9.8.1.2 Provided interfaces

Diagnostic Management does not provide any interfaces to other Functional Clusters.

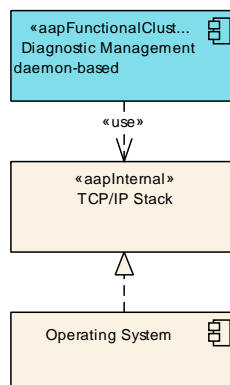
**9.8.1.3 Required interfaces**



**Figure 9.90: Interfaces required by Diagnostic Management**



**Figure 9.91: Interfaces required by Diagnostic Management from State Management**



**Figure 9.92: Interfaces required by Diagnostic Management from external components**

<b>Interface</b>	<b>Purpose</b>
n/a	This interface should be used to persist key-value data.
n/a	This interface is used to read and write data to files.
n/a	This interface is used for general persistency handling.
CommunicationControl	This interface should be used to realize UDS Service 0x28 - CommunicationControl.
DataElement	This interface is used to handle read <code>DataElement</code> requests.
DataIdentifier	This interface is used to handle <code>ReadDataByIdentifier</code> and <code>WriteDataByIdentifier</code> requests.
DiagnosticReset	This interface is used to handle diagnostic reset requests.
DoIPActivationLine	This interface is used to control a DoIP Transport Layer implementation.
DoIPGroupIdentification	This interface is used to control a DoIP Transport Layer implementation.
DoIPPowerMode	This interface is used to control a DoIP Transport Layer implementation.
DoIPTriggerVehicleAnnouncement	This interface is used to control a DoIP Transport Layer implementation.
DownloadService	This interface is used to handle download requests.
EcuResetRequest	This interface is used to handle reset requests.
ExecutionClient	This interface is used to report the status of the <code>Diagnostic Management</code> daemon process(es).
FileStorage	This interface is used to read and write data to files.
GenericDataIdentifier	This interface is used to handle <code>ReadDataByIdentifier</code> and <code>WriteDataByIdentifier</code> requests.
GenericRoutine	This interface is used to handle <code>RoutineControl</code> requests.
GenericUDSService	This interface is used to handle UDS requests.
KeyValueStorage	This interface should be used to persist key-value data.
Logger	<code>Diagnostic Management</code> shall use this interface to log standardized messages.
Manifest Accessor	<code>Diagnostic Management</code> shall use this interface to read its configuration information from the <code>Manifests</code> .
Monitor	This interface is used to report qualified and unqualified test results.
PolicyDecisionPoint	<code>Diagnostic Management</code> shall use this interface to check access from <code>Diagnostic Clients</code> .
ReadAccessor	This interface is used to read and write data to files.
ReadWriteAccessor	This interface is used to read and write data to files.
Routine	This interface is used to handle <code>RoutineControl</code> requests.
SecurityAccess	This interface is used to handle <code>SecurityAccess</code> requests.





<a href="#">ServiceValidation</a>	This interface is used to handle <code>ServiceValidation</code> requests.
<a href="#">SupervisedEntity</a>	This interface is used to supervise the <code>Diagnostic Management</code> daemon process(es).
TCP/IP Stack	This interface is used for DoIP Transport Protocols.
<a href="#">UdsMessage</a>	This interface is used to access an UDS Transport Layer implementation.
<a href="#">UdsTransportProtocolHandler</a>	This interface is used to access an UDS Transport Layer implementation.
<a href="#">UdsTransportProtocolMgr</a>	This interface is used to access an UDS Transport Layer implementation.
<a href="#">UdsTransportProtocolPeriodicHandler</a>	This interface is used to access an UDS Transport Layer implementation.
<a href="#">UploadService</a>	This interface is used to handle upload requests.

**Table 9.26: Interfaces required by Diagnostic Management**

## 10 Runtime View

This chapter shows the original design approach of the AUTOSAR Adaptive Platform for implementing selected use cases. The presented use cases currently cover just a small part of the functionality of the AUTOSAR Adaptive Platform. More use cases will be added in future versions of this document. Please note that individual implementations of the AUTOSAR Adaptive Platform may always choose a different design internally. Thus, interaction partners, the type of messages, and their order may differ.

### 10.1 Overview

The use cases are categorized in the subsequent sections. Section 10.2 groups the use cases that control the lifecycle of an AUTOSAR Runtime for Adaptive Applications. Section 10.3 lists use cases for communication with external systems. Section 10.4 demonstrates how Adaptive Applications can be installed and how they and the AUTOSAR Runtime for Adaptive Applications can be updated.

### 10.2 AUTOSAR Runtime for Adaptive Applications Lifecycle

#### 10.2.1 Machine Startup

During the startup of a machine the `Operating System` performs initialization steps in an implementation-specific way. These steps include starting any middleware related to the `Operating System`, including device-drivers and services handling low-level middleware. In addition, `Execution Management` is started as the entry point of the AUTOSAR Runtime for Adaptive Applications. `Execution Management` then controls the startup of the AUTOSAR Runtime for Adaptive Applications by starting `State Management` and `Platform Health Management Processes`.

After `State Management` and `Platform Health Management` are started, `State Management` takes control over the initialization of the AUTOSAR Adaptive Platform by requesting a transition to the standardized `Machine State Startup` from `Execution Management`. After the rest of the AUTOSAR Adaptive Platform has been initialized, `State Management` requests application-specific states for the other `Function Groups` on the Machine from `Execution Management` in the same way. `Platform Health Management` always supervises the `Processes` of `Execution Management` and `State Management` with a (probably fixed) implementation-specific set of rules. `Platform Health Management` itself is supervised by the `Watchdog`. In addition, `Platform Health Management` supervises application `Processes` according to the configuration in the `Machine Manifest`.

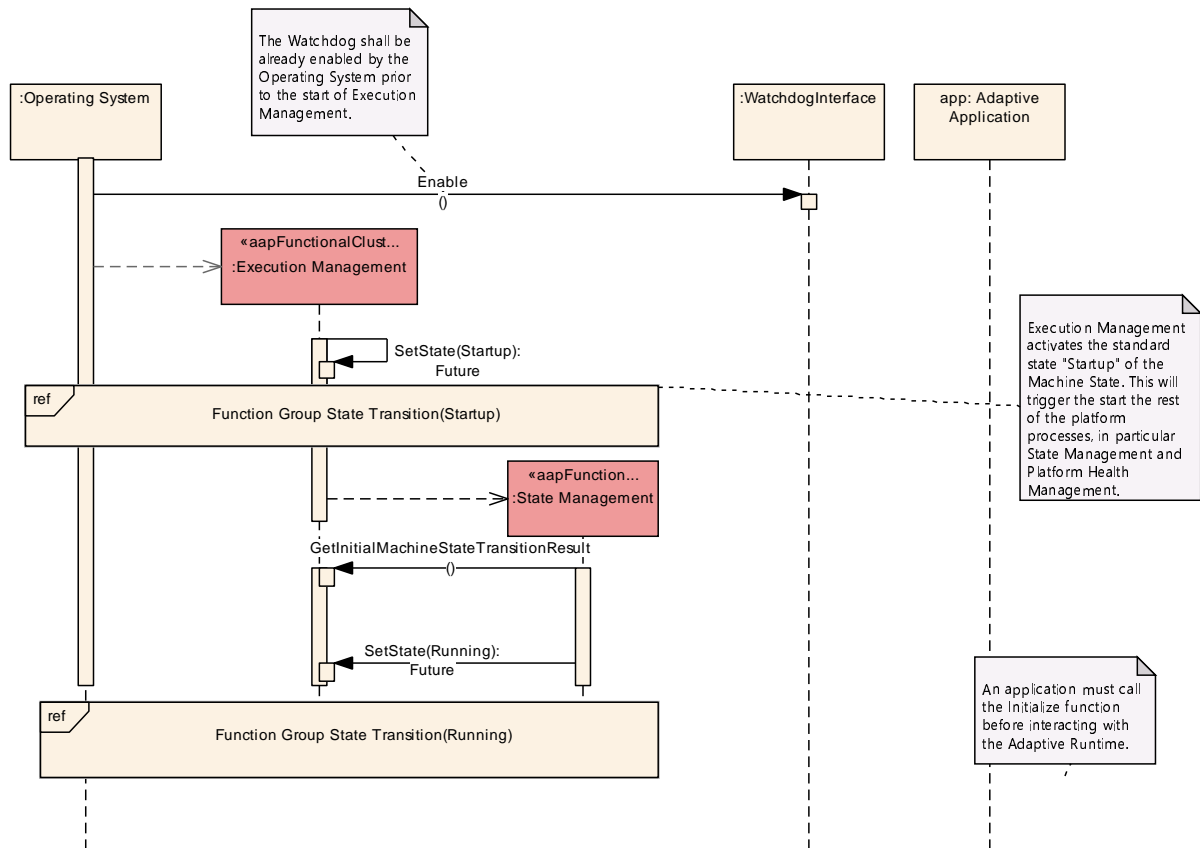
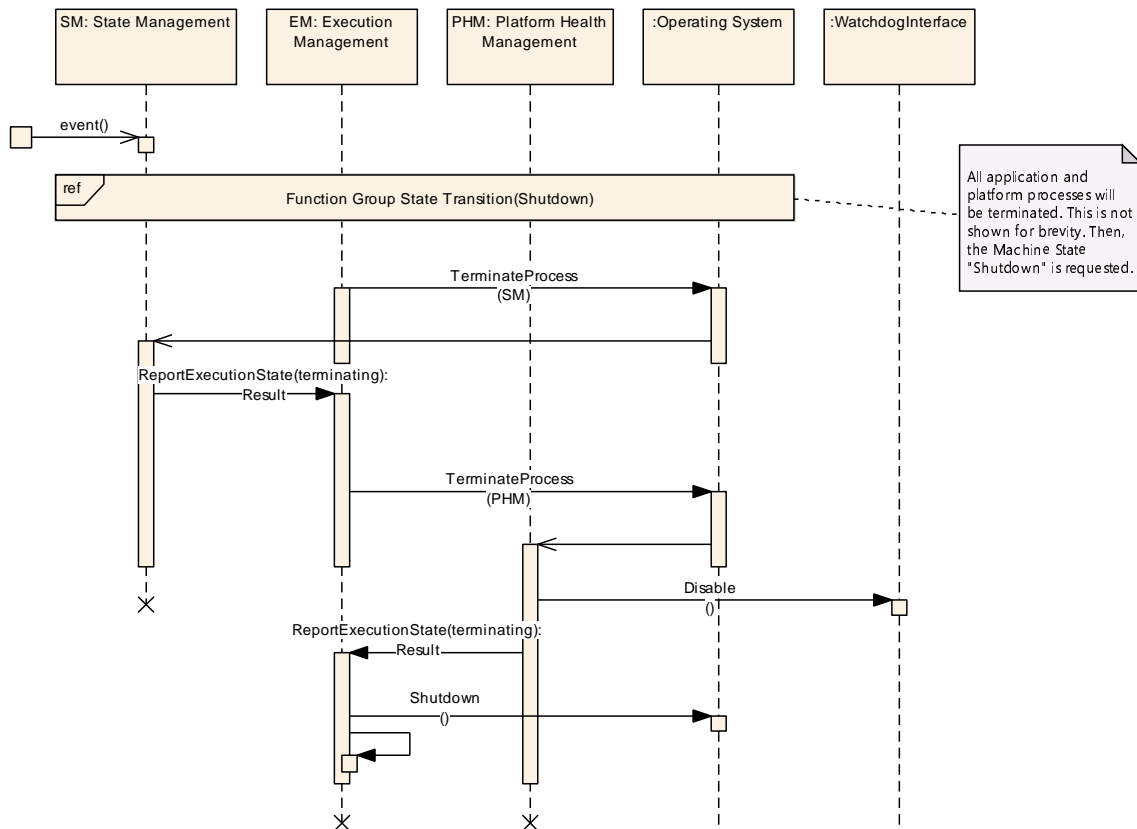


Figure 10.1: Startup of the AUTOSAR Runtime for Adaptive Applications

## 10.2.2 Machine Shutdown

A shutdown is requested by State Management after an application-specific event. First, the application Function Groups may be brought to a corresponding state (not shown). Afterwards, State Management triggers a transition to the Shutdown Machine State. The shutdown procedure is controlled by Execution Management. Execution Management stops all platform Processes. Then, Execution Management terminates the State Management and Platform Health Management Processes and requests a shutdown of the underlying operating system.



**Figure 10.2: Shutdown of the AUTOSAR Runtime for Adaptive Applications**

### 10.2.3 Function Group State Transition

A switch to another Function Group State is requested by State Management based on its inputs and internal state. The transition to the new Function Group State is controlled by Execution Management. First, all Processes are terminated that are either not active in the target state, or do have a different Startup Configuration in the target state. The latter may also include different startup dependencies. Then, all Processes are started by Execution Management in the order imposed by their dependencies. During the state transition, Execution Management notifies Platform Health Management on any change of the state of the Processes. Platform Health Management adapts its supervisions accordingly.



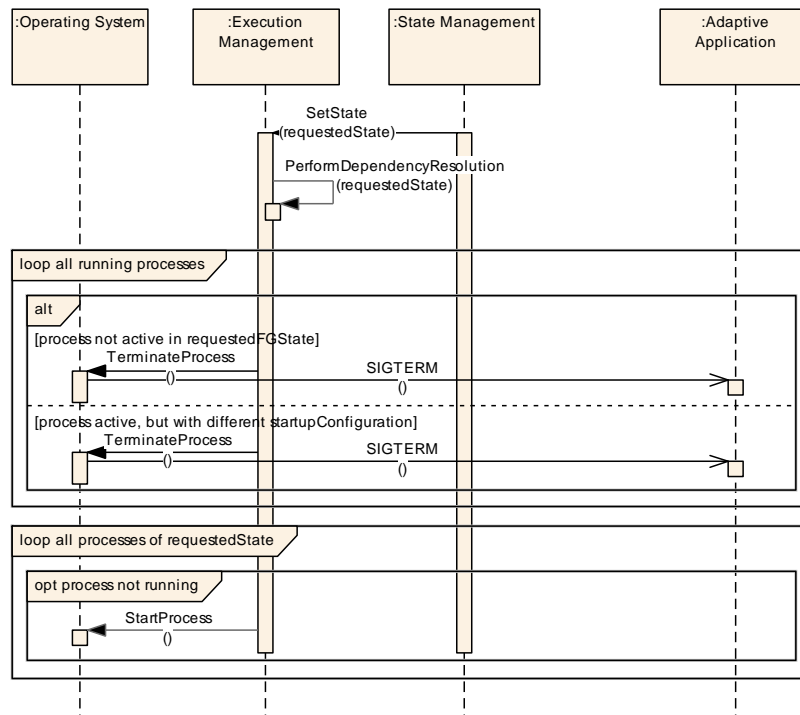


Figure 10.3: Transition to another Function Group State

## 10.2.4 Failure Recovery

In case Platform Health Management detects a failure in an entity it supervises it informs Execution Management about the supervision failure. Execution Management maps the supervised Process to the corresponding Function Group and delegates to State Management to handle that failure in the Function Group and perform recovery actions. State Management is an application-specific component that, depending on its various inputs, internal state etc., may decide upon actions to be taken to recover from a failure. There are two main possibilities:

- recover by switching Function Group State to another Function Group State (e.g., for degradation)
- recover by re-entering the same Function Group State and essentially restarting all Processes in the Function Group
- as a last resort, advise Platform Health Management to reset the Machine

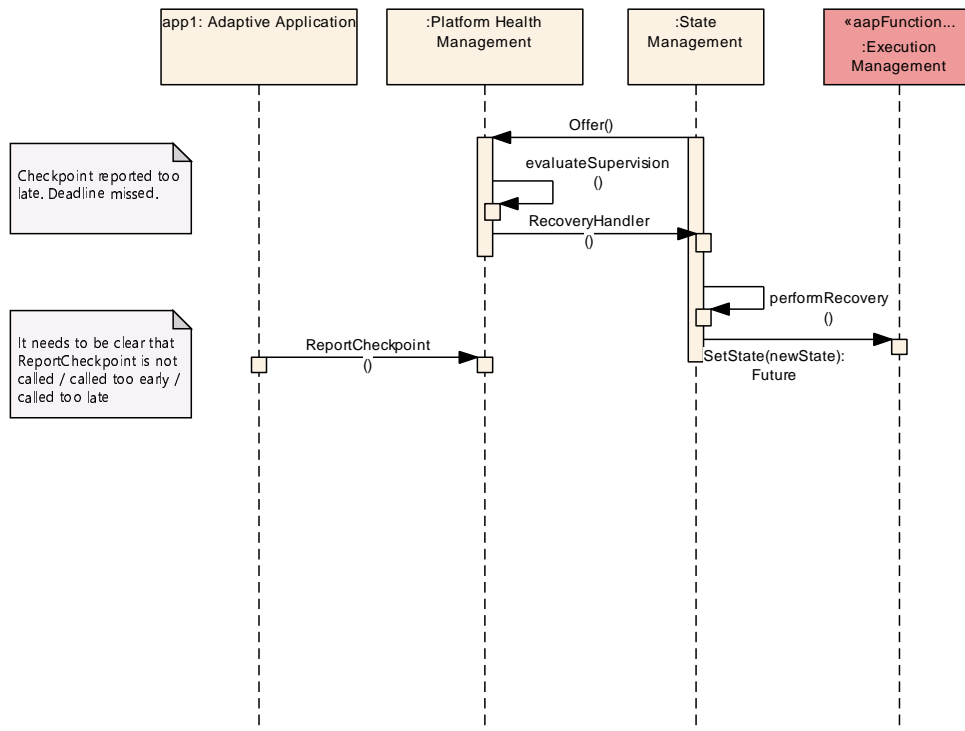


Figure 10.4: Failure recovery scenarios

### 10.3 Communication

Service-oriented communication in the AUTOSAR Adaptive Platform is guarded by Identity and Access Management that provides access control. All service-requests are handled by Communication Management. Communication Management determines the Adaptive Application Identity (AAID) of the sender Process using Execution Management. Then, Communication Management requests an access control decision from Identity and Access Management using the identity of the sender and information about the called service. Communication Management enforces the access control decision by forwarding the request to the service in case the access was granted or dropping the request in case the access was denied.

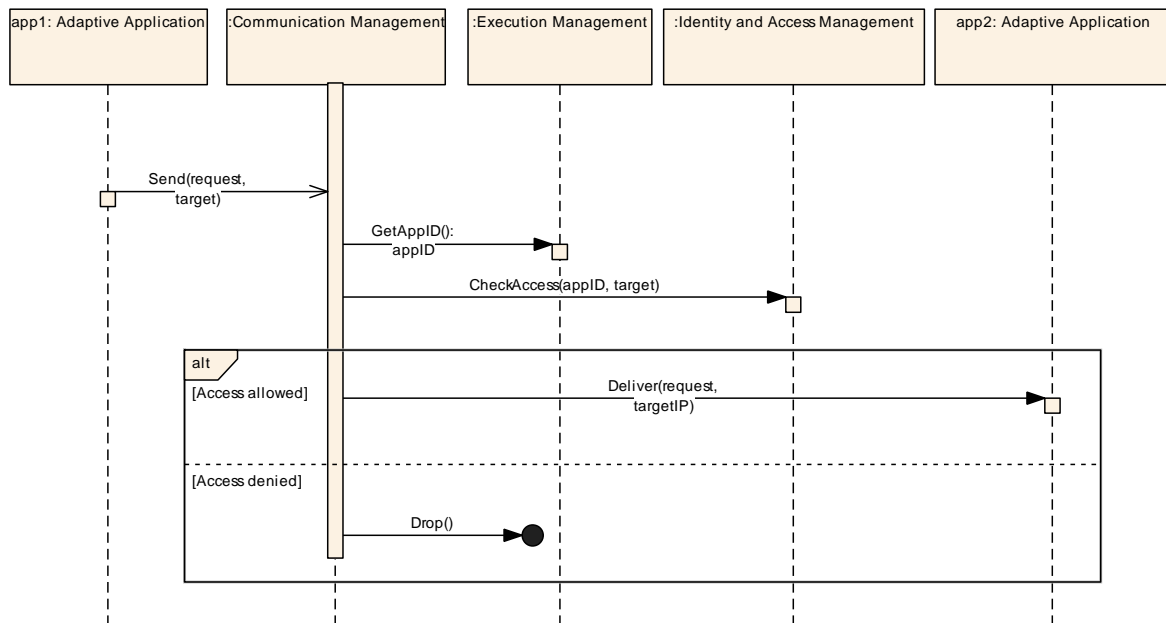


Figure 10.5: Access control in service-oriented communication

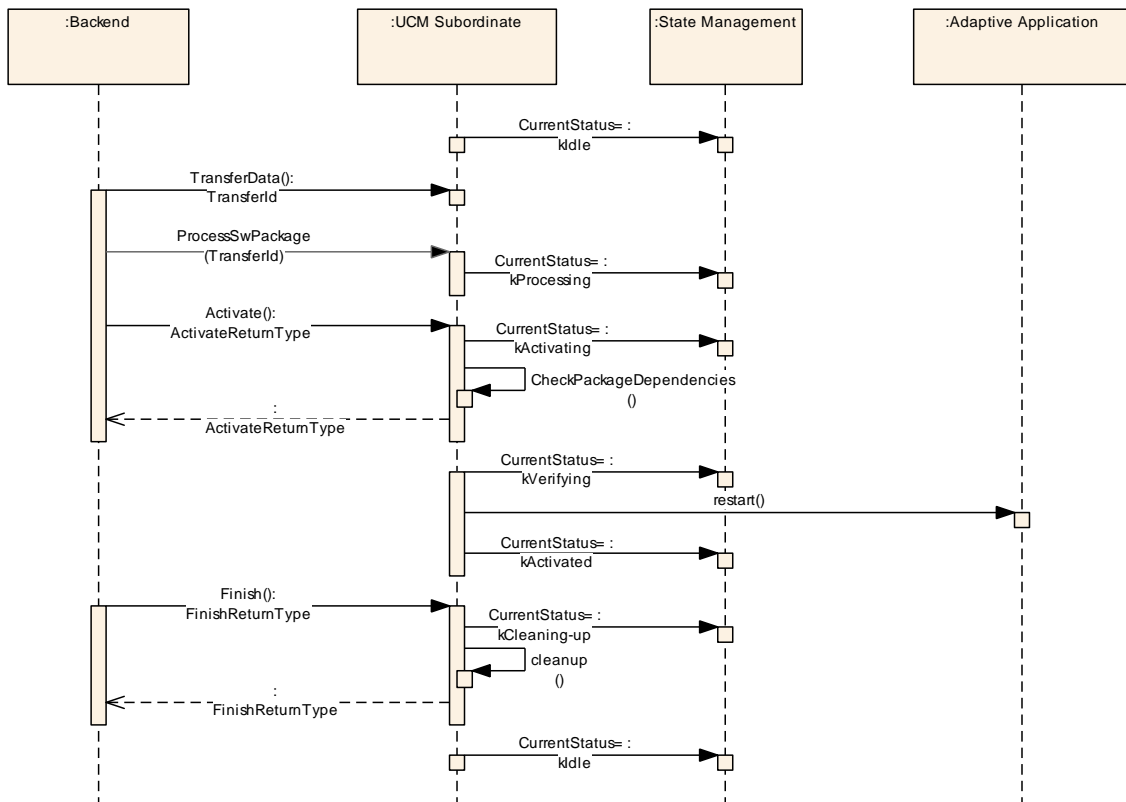
## 10.4 Update and Configuration Management

### 10.4.1 Update of an Adaptive Application

When an Adaptive Application has to be updated, the new version of the application first has to be transferred to the Update and Configuration Management instance. The state of the application as well as Update and Configuration Management are monitored by State Management. Only if there is nothing else running (state `kIdle`), the actual update process starts.

Update and Configuration Management then receives the signal to start processing the transferred data. After successful processing, the update enters activation phase and Update and Configuration Management checks dependencies in the transferred Software Packages. After successful activation, the Adaptive Application is restarted and after successful restart, running the updated software.

The last step for Update and Configuration Management is to finish the update process by cleaning up and deleting temporary data, old software version or stored data which are no longer required for the execution of the Adaptive Application.

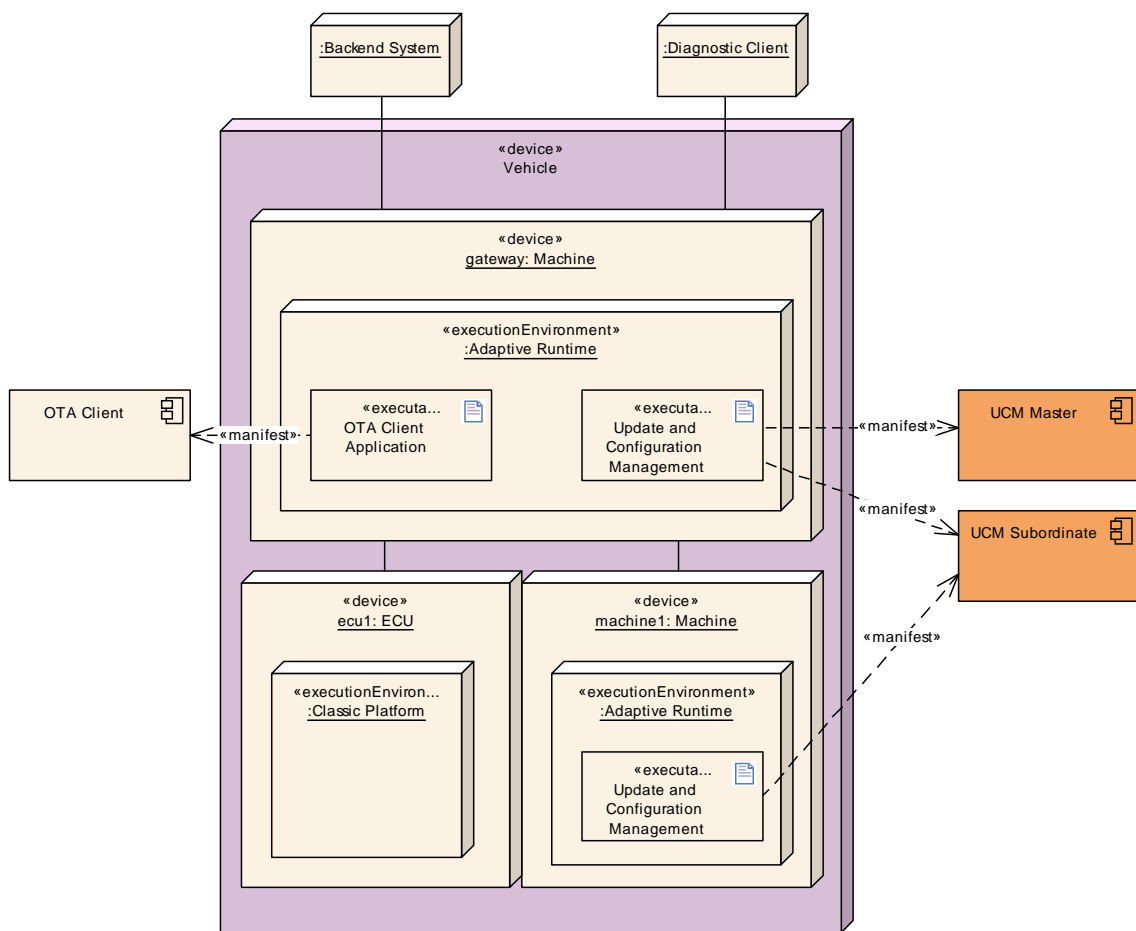


**Figure 10.6: Successful update of an Adaptive Application**

## 11 Deployment View

This chapter provides an overview of exemplary deployment scenarios for an AUTOSAR Adaptive Platform. Since the AUTOSAR Adaptive Platform is highly configurable in its deployment, this section rather provides constraints on supported deployments and a selection of relevant deployment scenarios.

### 11.1 Vehicle Software Deployment



**Figure 11.1: Exemplary vehicle software update scenario**

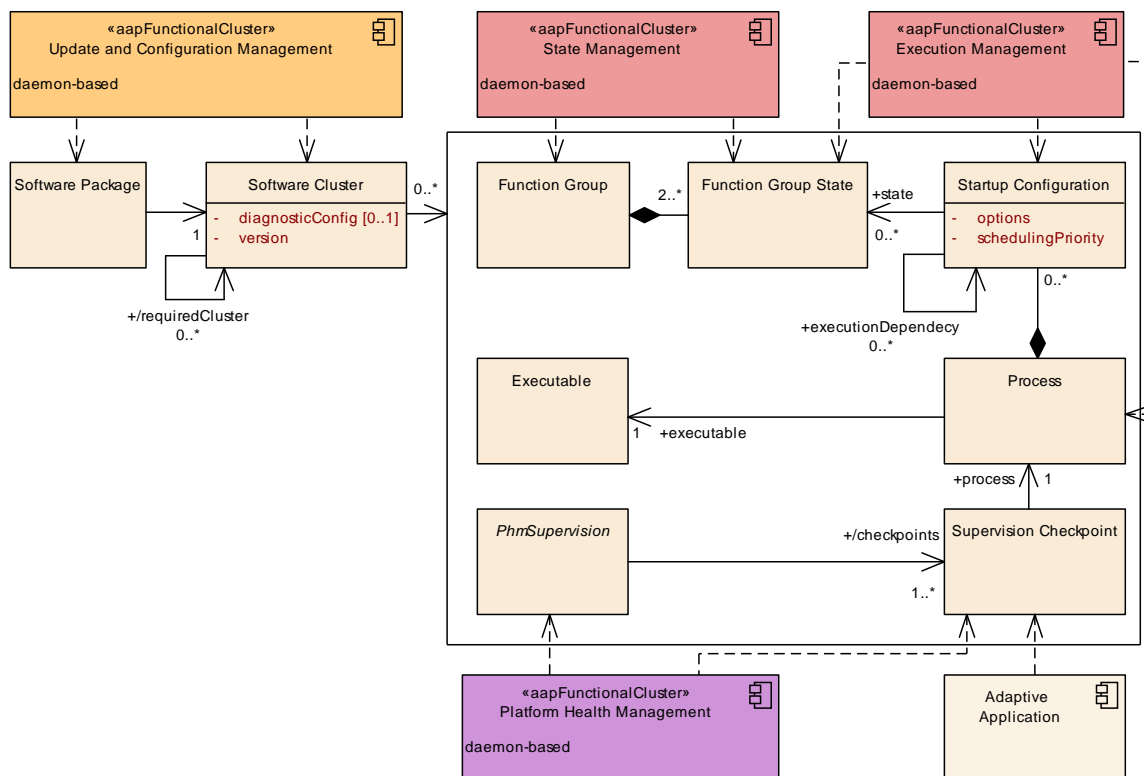
Update and Configuration Management allows to install and update software on the AUTOSAR Adaptive Platform and AUTOSAR Classic Platform. For the AUTOSAR Adaptive Platform, Update and Configuration Management also allows to remove software. The software packages can be received either from a Diagnostic Client or from a specific Backend System for over-the-air updates. In a vehicle, one Adaptive Application takes the role of a master that controls the update process in the vehicle and distributes individual software packages to the Machines and ECUs within a vehicle.

## 12 Cross-cutting Concepts

This section provides an overview of cross-cutting concepts and patterns used in the AUTOSAR Adaptive Platform.

### 12.1 Overview of Platform Entities

The AUTOSAR Adaptive Platform defines design entities that several Functional Clusters depend on. Figure 12.1 provides an overview of these entities, their logical relationships, and the Functional Clusters that depend on them. For the sake of brevity, this overview uses simplifications and abstractions over the actual specifications in the [10, manifest specification].



**Figure 12.1: Overview of platform entities and their logical relationships**

A `Software Package` is a digitally signed package that can be installed/uninstalled via `Update and Configuration Management`. A `Software Package` contains exactly one `Software Cluster` (see Section 12.4 for details). A `Software Cluster` refers to a set of `Executables` (and other files). The corresponding executable file then holds the executable code for the `Machine` that the AUTOSAR Adaptive Platform runs on.

Additionally, a `Software Cluster` configuration collects a set of `Processes` (cf. Section 12.4) and related entities. A `Process` refers to an `Executable` and provides different `Startup Configuration` values, for example parameters, a scheduling

priority, and resource constraints. A Startup Configuration of a Process applies to one or more Function Group States. Function Group States belong to a Function Group.

During runtime, State Management requests to enter a Function Group State from Execution Management. Execution Management then terminates and starts the Processes accordingly using the underlying Operating System.

For safety-critical systems, Platform Health Management performs supervision of Processes according to rules (logical sequence, deadlines) defined in PhmSupervisions. A PhmSupervision refers to a number of Supervision Checkpoints. During runtime, a process reports whenever it has reached such a checkpoint in its control flow.

## 12.2 Function Group

A Function Group is (logically) made up a set of modeled Processes that provide a certain functionality. For example, a Function Group could be an application, or a service. A special Function Group is the Machine State that groups the Process of the AUTOSAR Adaptive Platform itself. A Function Group contains a set of Function Group States.

## 12.3 Function Group State

A Function Group State defines which Processes of a Function Group with what configuration parameters shall be running or not. The Machine State (that refers to the Processes of the AUTOSAR Adaptive Platform itself) defines at least the following Function Group States: Off, Startup, Shutdown, and Restart.

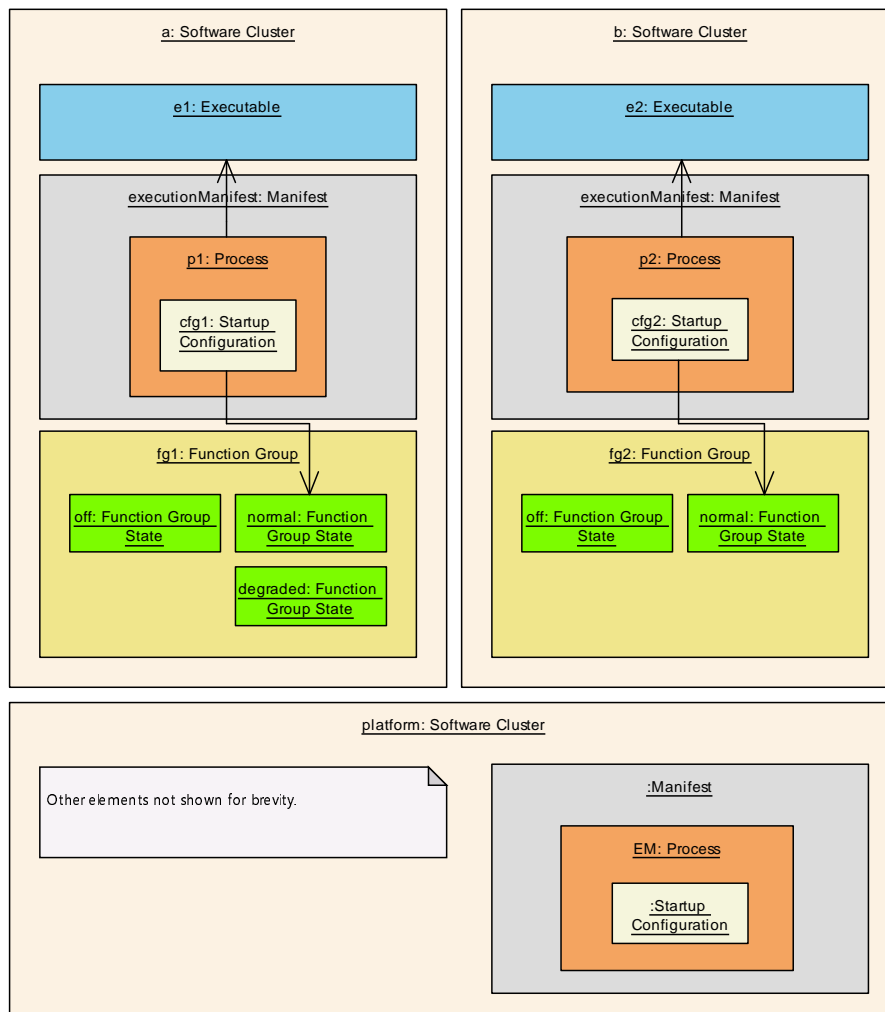
## 12.4 Software Cluster

A Software Cluster configuration refers to a set of modeled Processes. Those Processes are (transitively) used by one or more Function Group(s). Hereby, a Function Group and its associated entities shall be part of only one Software Cluster. In other words, Function Groups that span several Software Clusters are invalid. A Software Cluster is packaged into one Software Package - the atomic installable/updateable unit managed by Update and Configuration Management. A Software Cluster may depend on other Software Clusters. Such dependencies are expressed by version constraints. A Software Cluster may also specify structural dependencies to Sub Software Clusters in order to build larger installable units. The top of such a structural dependency hierarchy is called a Root Software Cluster. Please note that a Software Cluster is only used to configure deployment aspects. A Software Cluster is not a runtime entity.

A Root Software Cluster may specify a diagnostic configuration, in particular, a diagnostic address. In contrast, a Sub Software Cluster may depend on a diagnostic configuration of its Root Software Cluster. The diagnostic configuration applies to Processes that are (transitively) contained in a Root Software Cluster and its Sub Software Cluster(s). That means, at runtime, any diagnostic event produced by those Processes will be associated with the diagnostic address. Please refer to Section 12.11 for further details on the diagnostic deployment.

An exemplary Software Cluster during application design is shown in Figure 12.2. The application Software Cluster(s) are modeled/configured in the same way as the platform Software Cluster by defining Function Groups with Function Group States and associating StartupConfigurations of Processes to them.

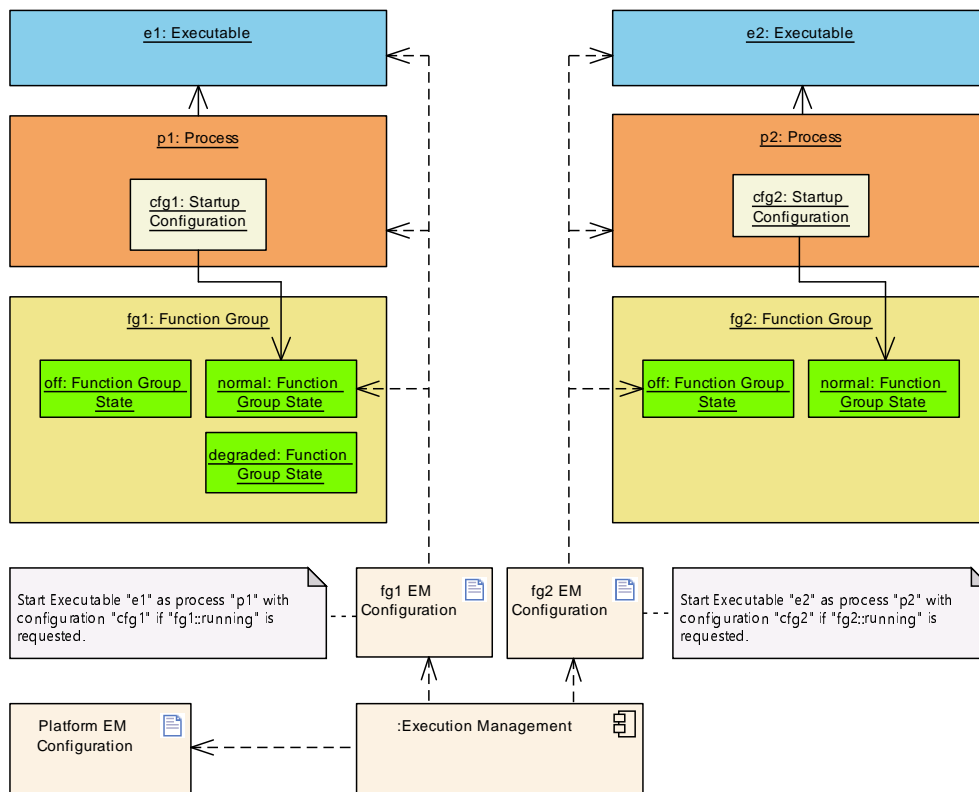
A Software Cluster serves as a grouping entity during application design. As a result, entities within a Software Cluster, in particular the Function Groups, do not need to have a unique (simple) name within the overall model because their path is still unique. This allows to design Software Clusters independently, for example, by external suppliers.



**Figure 12.2: Exemplary Software Cluster during application design**



From such a standardized model, an equivalent implementation-specific configuration for Execution Management is derived that is used during runtime (see Figure 12.3). That configuration advises Execution Management to start and configure processes accordingly when a Function Group State is requested. Hereby, Execution Management (logically) merges configurations contributed by all installed Software Packages. Other Functional Clusters that depend on configuration provided in the Manifests merge the configurations contributed by all installed Software Packages in the same way. Please also note that there is no corresponding runtime entity for a Software Cluster (see Figure 12.3).



**Figure 12.3: Impact of exemplary Software Cluster during runtime**

All Processes related to the Functional Clusters of the AUTOSAR Adaptive Platform should be referenced only in Software Clusters of category PLATFORM\_CORE or PLATFORM. This allows for platform-independent development of Software Clusters of category APPLICATION\_LAYER.

In case a Functional Cluster may need multiple logical instances (for example, Diagnostic Management has a logical instance per diagnostic address), an implementation of the Functional Cluster should still use a single physical (daemon) process.

An AUTOSAR Adaptive Platform vendor may deviate from this design guide but should provide additional countermeasures to keep Adaptive Applications portable.

## 12.5 Machine

The AUTOSAR Adaptive Platform regards hardware it runs on as a `Machine`. The rationale behind that is to achieve a consistent platform view regardless of any virtualization technology which might be used. The `Machine` might be a real physical machine, a fully-virtualized machine, a para-virtualized OS, an OS-level-virtualized container or any other virtualized environment.

On hardware, there can be one or more `Machine`, and only a single instance of AUTOSAR Adaptive Platform runs on a machine. It is generally assumed that this hardware includes a single chip, hosting a single or multiple `Machines`. However, it is also possible that multiple chips form a single `Machine` if the AUTOSAR Adaptive Platform implementation allows it.

## 12.6 Manifest

A `Manifest` represents a piece of AUTOSAR model description that is created to support the configuration of an AUTOSAR Adaptive Platform product and which is uploaded to the AUTOSAR Adaptive Platform product, potentially in combination with other artifacts (like binary files) that contain executable code to which the `Manifest` applies. Please note that a typical `Adaptive Application` will make use of several distinct but interrelated `Manifests`. Hereby, the individual `Manifests` contribute information to the complete application model. For example, each `Software Cluster` may contribute a self-contained set of `Manifests` to configure its functionality.

The usage of a `Manifest` is limited to the AUTOSAR Adaptive Platform. This does not mean, however, that all ARXML produced in a development project that targets the AUTOSAR Adaptive Platform is automatically considered a `Manifest`. In fact, the AUTOSAR Adaptive Platform is usually not exclusively used in a vehicle project. A typical vehicle will most likely be also equipped with a number of ECUs developed on the AUTOSAR Classic Platform and the system design for the entire vehicle will, therefore, have to cover both, ECUs built on top of the AUTOSAR Classic Platform and `Machines` created on top of the AUTOSAR Adaptive Platform.

In principle, the term `Manifest` could be defined such that there is conceptually just one "Manifest" and every deployment aspect would be handled in this context. This does not seem appropriate because it became apparent that `Manifest`-related model-elements exist that are relevant in entirely different phases of a typical development project.

This aspect is taken as the main motivation that next to the application design it is necessary to subdivide the definition of the term `Manifest` in three different partitions:

**Application Design** This kind of description specifies all design-related aspects that apply to the creation of application software for the AUTOSAR Adaptive Platform. It is not necessarily required to be deployed to the adaptive platform machine, but the application design aids the definition of the deployment of application software in the

Execution Manifest and Service Instance Manifest. See Section 12.7 for details.

**Execution Manifest** This kind of Manifest is used to specify the deployment-related information of applications running on the AUTOSAR Adaptive Platform. An Execution Manifest is bundled with the actual executable code to support the integration of the executable code onto the machine. See Section 12.8 for details.

**Service Instance Manifest** This kind of Manifest is used to specify how service-oriented communication is configured in terms of the requirements of the underlying transport protocols. A Service Instance Manifest is bundled with the actual executable code that implements the respective usage of service-oriented communication. See Section 12.9 for details.

**Machine Manifest** This kind of Manifest is supposed to describe deployment-related content that applies to the configuration of just the underlying machine (i.e. without any applications running on the machine) that runs an AUTOSAR Adaptive Platform. A Machine Manifest is bundled with the software taken to establish an instance of the AUTOSAR Adaptive Platform. See Section 12.10 for details.

The temporal division between the definition (and usage) of different kinds of Manifest leads to the conclusion that in most cases different physical files will be used to store the content of the three kinds of Manifest. In addition to the Application Design and the different kinds of Manifest, the AUTOSAR Methodology supports a System Design with the possibility to describe Software Components of both AUTOSAR Platforms that will be used in a System in one single model. The Software Components of the different AUTOSAR platforms may communicate in a service-oriented way with each other. But it is also possible to describe a mapping of Signals to Services to create a bridge between the service-oriented communication and the signal-based communication.

## 12.7 Application Design

The application design describes all design-related modeling that applies to the creation of application software for the AUTOSAR AP. Application Design focuses on the following aspects:

- Data types used to classify information for the software design and implementation
- Service interfaces as the pivotal element for service-oriented communication
- Definition how service-oriented communication is accessible by the application
- Persistency Interfaces as the pivotal element to access persistent data and files
- Definition how persistent storage is accessible by the application
- Definition how files are accessible by the application

- Definition how crypto software is accessible by the application
- Definition how the Platform Health Management is accessible by the application
- Definition how Time Bases are accessible by the application
- Serialization properties to define the characteristics of how data is serialized for the transport on the network
- Description of client and server capabilities
- Grouping of applications in order to ease the deployment of software.

The artifacts defined in the application design are independent of a specific deployment of the application software and thus ease the reuse of application implementations for different deployment scenarios.

## 12.8 Execution Manifest

The purpose of the execution manifest is to provide information that is needed for the actual deployment of an application onto the AUTOSAR AP. The general idea is to keep the application software code as independent as possible from the deployment scenario to increase the odds that the application software can be reused in different deployment scenarios. With the execution manifest the instantiation of applications is controlled, thus it is possible to

- instantiate the same application software several times on the same machine, or to
- deploy the application software to several machines and instantiate the application software per machine.

The Execution manifest focuses on the following aspects:

- Startup configuration to define how the application instance shall be started. The startup includes the definition of startup options and access roles. Each startup may be dependent on machines states and/or function group states.
- Resource Management, in particular resource group assignments.

## 12.9 Service Instance Manifest

The implementation of service-oriented communication on the network requires configuration which is specific to the used communication technology (e.g. SOME/IP). Since the communication infrastructure shall behave the same on the provider and the requesters of a service, the implementation of the service shall be compatible on both sides.

The Service Instance Manifest focuses on the following aspects:

- Service interface deployment to define how a service shall be represented on the specific communication technology.
- Service instance deployment to define for specific provided and required service instances the required credentials for the communication technology.
- The configuration of E2E protection
- The configuration of Security protection
- The configuration of Log and Trace

## 12.10 Machine Manifest

The machine manifest allows to configure the actual adaptive platform instance running on specific hardware (machine).

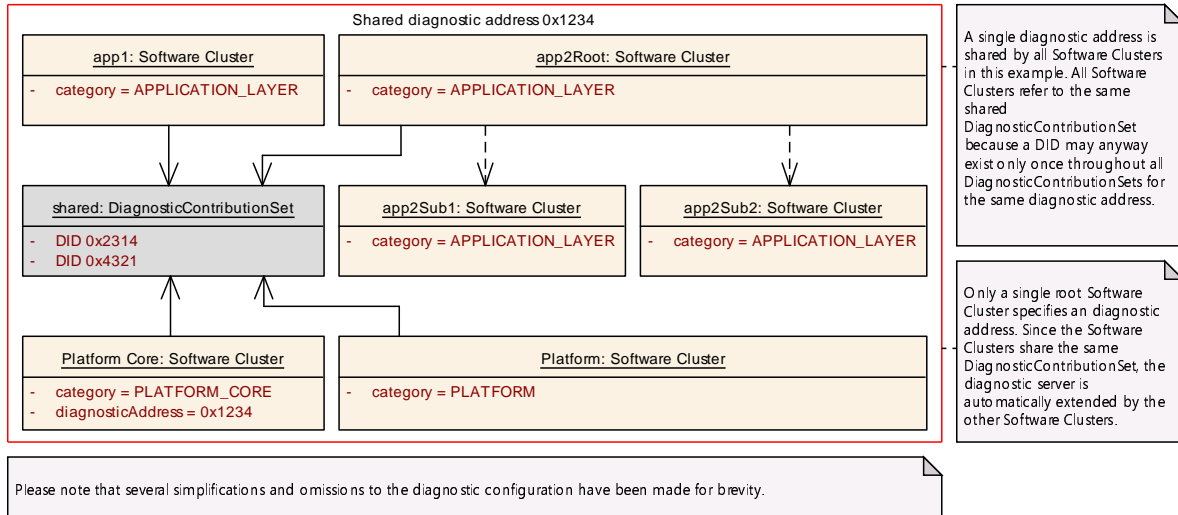
The Machine Manifest focuses on the following aspects:

- Configuration of the network connection and defining the basic credentials for the network technology (e.g. for Ethernet this involves setting of a static IP address or the definition of DHCP).
- Configuration of the service discovery technology (e.g. for SOME/IP this involves the definition of the IP port and IP multi-cast address to be used).
- Definition of the used machine states.
- Definition of the used function groups.
- Configuration of the adaptive platform functional cluster implementations (e.g. the operating system provides a list of OS users with specific rights).
- The configuration of the Crypto platform Module.
- The configuration of Platform Health Management.
- The configuration of Time Synchronization.
- Documentation of available hardware resources (e.g. how much RAM is available; how many processor cores are available).

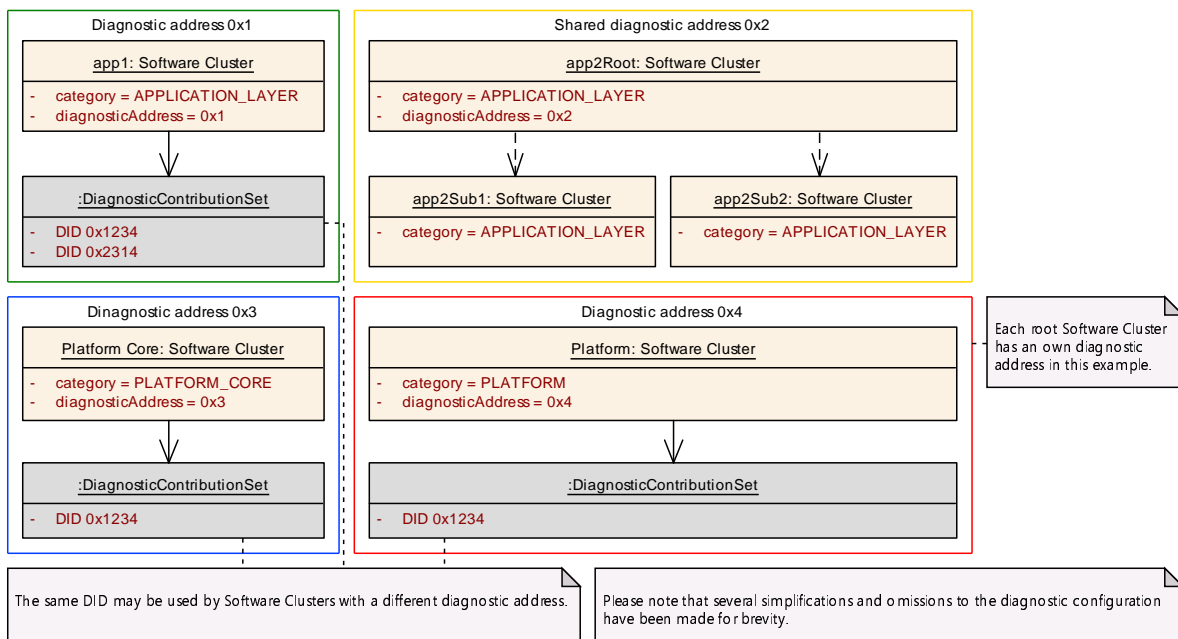
## 12.11 Diagnostics deployment

A diagnostic configuration, in particular a diagnostic address, may only be assigned to a `Root Software Cluster` in the `Manifest`. Nevertheless, the mapped `DiagnosticContributionSet(s)` may be distributed across several `Software Clusters`. This concept provides a lot of flexibility in assignment of a single diagnostic address to `Software Clusters`. For example, in one extreme this allows to use a single diagnostic address for the whole `Machine` (see Figure 12.4), in another extreme

a single diagnostic address per Software Cluster could be used (see Figure 12.5). Deployment scenarios in between those extremes are also possible.



**Figure 12.4: Example defining a single diagnostic address for the whole Machine**



**Figure 12.5: Example using one diagnostic address for each Software Cluster**

In the case of a distributed DiagnosticContributionSet, each Software Cluster shall include its related diagnostic configuration objects (for example, the Data Identifier configuration). The merge of such a split DiagnosticContributionSet is done internally by the AUTOSAR Adaptive Platform (e.g. during installation or during start up the Diagnostic Management daemon).

## 12.12 Error Handling

Proper handling of errors during runtime is an important aspect to build safe and secure systems. The AUTOSAR Adaptive Platform does provide means for raising and handling of such errors on different levels in the platform.

Platform Health Management detects errors (errors in the logical control flow, missed deadlines, and missed liveness reporting) at the level of Processes and performs recovery actions (for example, degradation) according to rules defined in the Manifest. Execution Management detects unexpected termination of Processes and reports to State Management for handling of such errors.

During execution of a Process of an Adaptive Application, different abnormal conditions might be detected and need to be handled and/or reported. The following types of unsuccessful operations are distinguished within the AUTOSAR Adaptive Platform:

- An **Error** is the inability of an AUTOSAR Runtime for Adaptive Applications API function to fulfill its specified purpose. An **Error** it is often a consequence of invalid and/or unexpected input data. An **Error** is considered to be recoverable and therefore shall be handled by applications.
- A **Violation** is the consequence of failed pre- or post-conditions of internal state of the AUTOSAR Runtime for Adaptive Applications. A **Violation** is considered to be non-recoverable.
- A **Corruption** is the consequence of the corruption of a system resource, e.g. stack or heap overflow, or a hardware memory flaw (for example, a detected bit flip). A **Corruption** is considered to be non-recoverable.
- A **failed default allocation** is the inability of the AUTOSAR Runtime for Adaptive Applications's default memory allocation mechanism to satisfy an allocation request (for example, there is not enough free memory available).

It is expected that a **Violation** or **Corruption** will not be experienced by a user of the AUTOSAR Adaptive Platform (i.e. an application developer), unless there is something seriously wrong in the overall system. For example, faulty hardware may lead to a **Corruption**. A **Violation** may occur if basic assumptions about resource requirements are violated, or the user runs the AUTOSAR Runtime for Adaptive Applications in a configuration that is not supported by its vendor.

## 12.13 Trusted Platform

To guarantee the correct function of the system, it is crucial to ensure that the code executed on the AUTOSAR Adaptive Platform is unaltered (integrity) and has legitimate origin (authenticity). Keeping this property allows the integrator to build a **Trusted Platform**. A key property of a system that implements a **Trusted Platform** is a **Trust Anchor** (also called **Root of Trust**). A **Trust Anchor** is often realized

as a public key that is stored in a secure environment, e.g. in non-modifiable persistent memory or in an `Hardware Security Module`. A system designer is responsible to ensure that the system starts beginning with a `Trust Anchor` and that the chain of trust is kept until the `Execution Management` is launched. Depending on the mechanism that is chosen by the system designer to establish the chain of trust, the integrity and authenticity of the entire system (including all executables) may be checked during system start-up. Alternatively, the system designer might only ensure that the already executed software has been checked regarding integrity and authenticity and `Execution Management` takes over responsibility on continuing the chain of trust when it takes over control of the system. In the latter case, the system integrator is responsible to ensure that the `Execution Management` is configured accordingly.

Passing trust requires that a trusted entity checks (using trusted functionality) that the entity down the chain is authentic. The `Trust Anchor` (the first entity in the chain) is authentic by definition. An example of such a chain of trust could look like this: The `Trust Anchor` authenticates the bootloader before the bootloader is being started. In each subsequent step in the boot process, the to-be-started executable shall be authenticated first, for example by the executable started previously or by some external entity like an `Hardware Security Module`. After the relevant parts of the `Operating System` have been authentically started, it shall launch `Execution Management` as one of its first processes in the same manner passing trust to the `AUTOSAR Adaptive Platform`. Then, `Execution Management` takes over the responsibility of authenticating `Adaptive Applications` before launching them.

As stated above, if authenticity is not checked by the functionality of the `Trust Anchor` itself, which is authentic by definition, the functionality that shall be applied to verify authenticity of an executable has to be authenticated as well before it is applied. For instance, if the `Crypto Functional Cluster` shall be used to verify authenticity of executables, the `Crypto Functional Cluster` itself shall be authenticated by some trusted entity before it is used.

## 12.14 Secure Communication

AUTOSAR supports different protocols that provide communication security over a network. Integrity of messages can be ensured by the end-to-end protection offered by the [11, AUTOSAR E2E library]. End-to-end protection assumes that safety- and security-related data exchange shall be protected at runtime against the effects of faults within the communication link. Such faults include random hardware faults (e.g. corrupt registers of a transceiver), interference (e.g. electromagnetic interference), and systematic faults in the communication stack. The configuration of end-to-end-protection is done via `Service Instance Manifest` on level of `Service` events, methods, and fields (notifier, get, and set methods). Confidentiality and authenticity of messages can be ensured by dedicated configurations for the individual transport protocols (e.g. TLS, SecOC) in the `Service Instance Manifest` on level of `Service` events, methods, and fields (notifier, get, and set methods).



## 13 Risks and Technical Debt

This chapter lists and rates risks associated with the overall architecture of the AUTOSAR Adaptive Platform in Section 13.1. These risks usually might cause that some of the quality attributes of the AUTOSAR Adaptive Platform are not (fully) met. Section 13.2 lists technical debt of the AUTOSAR Adaptive Platform that may impact its maintainability.

### 13.1 Risks

#### 13.1.1 Risk Assessment

This document categorizes risks according to their severity. The severity is a function of the probability and the impact of a risk. The probabilities are categorized as follows:

- **very low** - probability is less than 1 thousandth
- **low** - probability is between 1 thousandth and 1 percent
- **medium** - probability is between 1 percent and 10 percent
- **high** - probability is between 10 percent and 50 percent
- **very high** - probability is more than 50 percent

The impact of a risk is categorized as follows:

- **very low** - at most one quality scenario will take additional significant effort to be satisfied
- **low** - more than one quality scenario will take additional significant effort to be satisfied
- **medium** - at most one quality scenario is not satisfied with small gaps
- **high** - at most one quality scenario is not satisfied with big gaps
- **very high** - more than one quality scenario is not satisfied with big gaps

The final severity of a risk is then calculated according to table 13.1.

Impact	Probability				
	very low	low	medium	high	very high
very low	low (1)	low (2)	low (3)	medium (4)	medium (5)
low	low (2)	medium (4)	medium (6)	high (8)	high (10)
medium	low (3)	medium (6)	high (9)	high (12)	high (15)
high	medium (4)	high (8)	high (12)	extreme (16)	extreme (20)
very high	medium (5)	high (10)	high (15)	extreme (20)	extreme (25)

**Table 13.1: Risk Severity Matrix**

### **13.1.2 Risk List**

No architectural risks were identified yet.

## **13.2 Technical Debt**

No technical debt has been identified yet.

## References

- [1] ISO 42010:2011 – Systems and software engineering – Architecture description  
<http://www.iso.org>
- [2] Explanation of Adaptive Platform Software Architectural Decisions  
AUTOSAR\_EXP\_SWArchitecturalDecisions
- [3] Glossary  
AUTOSAR\_TR\_Glossary
- [4] Main Requirements  
AUTOSAR\_RS\_Main
- [5] General Requirements specific to Adaptive Platform  
AUTOSAR\_RS\_General
- [6] ATAM<sup>SM</sup>: Method for Architecture Evaluation  
[https://resources.sei.cmu.edu/asset\\_files/TechnicalReport/2000\\_005\\_001\\_13706.pdf](https://resources.sei.cmu.edu/asset_files/TechnicalReport/2000_005_001_13706.pdf)
- [7] Agile Software Development: Principles, Patterns, and Practices
- [8] Guide to the Software Engineering Body of Knowledge, Version 3.0  
[www.swebok.org](http://www.swebok.org)
- [9] API standards for Open Systems  
<http://www.opengroup.org/austin/papers/wp-apis.txt>
- [10] Specification of Manifest  
AUTOSAR\_TPS\_ManifestSpecification
- [11] Specification of SW-C End-to-End Communication Protection Library  
AUTOSAR\_SWS\_E2ELibrary