Document Title	Specification of Persistency
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	858

Document Status	published
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	R21-11

Document Change History

Date	Release	Changed by	Description
2021-11-25	R21-11	AUTOSAR Release Management	• Clarified and extended specification of Persistency behavior • Improved configuration of storage location and versioning • kNotInitialized was removed • Deleted move constructors/operators
2020-11-30	R20-11	AUTOSAR Release Management	• Replaced POSIX based file access API and improved error handling and symmetry of other APIs • Full support for encryption and redundancy by hashes using Crypto API • Added information to application about safety related problems • Improved installation/update and redundancy
2019-11-28	R19-11	AUTOSAR Release Management	• Introduced reset and restore of storages • Introduced storage statistics • Improved compliance with general AUTOSAR concepts • Improved naming and consistency of classes / methods / functions / constants • Changed Document Status from Final to published

2019-03-29	19-03	AUTOSAR Release Management	• Improved naming of classes / methods / functions • Reworked installation/update • Support for parallel execution in multiple threads • Cleaned up usage of ara::core concepts
2018-10-31	18-10	AUTOSAR Release Management	• Introduction of ara::core types and switch to exceptionless API • Rework of redundancy approach • Support for resource limitation • Improvements and harmonization of KeyValueStorage and FileProxy API
2018-03-29	18-03	AUTOSAR Release Management	• Installation / update of persistent data • Data types supported by KeyValueStorage API
2017-10-27	17-10	AUTOSAR Release Management	• Introduction of AUTOSAR model • Security added • Redundancy added • Rework of FileProxy / Stream API
2017-03-31	17-03	AUTOSAR Release Management	• Initial release

Table of Contents

1 Introduction and Functional Overview

This document is the software specification of the `Persistency functional cluster` within the `Adaptive Platform`. The `Persistency functional cluster` will be referenced as `Persistency` in the remainder of this document.

`Persistency` offers mechanisms to `Adaptive Applications` and other `functional clusters` to store information in the non-volatile memory of a machine. The data is available over boot and ignition cycles.

The `Persistency` will typically be implemented as a library that runs within a `Process` of an `Adaptive Application`, with the rights of that `Process`.

2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant to the Persistency that are not included in the [1, AUTOSAR glossary].

Abbreviation / Acronym	Description
FS	File Storage
KVS	Key-Value Storage

Terms	Description
Adaptive Application	Refers to the Adaptive Application defined in [1].
Adaptive Platform	Refers to the AUTOSAR Adaptive Platform defined in [1].
Adaptive Platform Foundation	Refers to the Adaptive Platform Foundation defined in [1].
Element	Refers to either a key-value pair of a Key-Value Storage or a file of a File Storage. Used in the specification where something applies to all kinds of storage elements.
Execution Manifest	Refers to the Execution Manifest defined in [1].
File	A binary or text file to be stored in a File Storage.
File Name	The file name uniquely identifies a file within a File Storage.
File Storage	A set of files that are stored persistently.
Functional Cluster	Refers to the Functional Cluster defined in [1].
Integrity	Persistency distinguishes data integrity, which is ensured by the configured redundancy, from structural integrity, i.e. the readability of the structure of a Key-Value Storage or File Storage.
Key	The key uniquely identifies a key-value pair within a Key-Value Storage.
Key-Value Pair	A key with an associated value, to be stored in a Key-Value Storage together with the type of the value.
Key-Value Storage	A set of key-value pairs that are stored persistently.
Persistency	The functional cluster described in this document, which handles persistent data of AUTOSAR Adaptive Applications and other functional clusters in File Storages and Key-Value Storages.
Persistent Data	Data that is stored in the persistent memory that can be accessed by one Process. Persistency supports different mechanisms to access data in persistent memory. Concurrent access to the data by several Processes is not supported as the data is owned exclusively by one Process.
Redundancy	Redundancy is used by Persistency to ensure the integrity of stored data. It can be configured to use replication of stored data, CRCs, or Hashes. Typically, only replication will allow to repair corrupted data.
Service Interface	Refers to the Service Interface defined in [1].
Software Package	Refers to the Software Package defined in [1].
Storage	Refers to either a Key-Value Storage or a File Storage. Used in the specification where something applies to all kinds of storages.
Value	A value of a key-value pair stored in a Key-Value Storage.

3 Related Documentation

3.1 Input Documents & Related Standards and Norms

[1] Glossary
AUTOSAR_TR_Glossary

[2] Specification of Adaptive Platform Core
AUTOSAR_SWS_AdaptivePlatformCore

[3] Specification of Manifest
AUTOSAR_TPS_ManifestSpecification

[4] Specification of Execution Management
AUTOSAR_SWS_ExecutionManagement

[5] Specification of Cryptography
AUTOSAR_SWS_Cryptography

[6] Specification of Update and Configuration Management
AUTOSAR_SWS_UpdateAndConfigurationManagement

[7] Requirements on Persistency
AUTOSAR_RS_Persistency

[8] General Requirements specific to Adaptive Platform
AUTOSAR_RS_General

[9] Explanation of Adaptive Platform Design
AUTOSAR_EXP_PlatformDesign

[10] Specification of Platform Types for Adaptive Platform
AUTOSAR_SWS_AdaptivePlatformTypes

3.2 Further Applicable Specifications

AUTOSAR provides a core specification [2] which is also applicable for the `Persistency`. The chapter “General requirements for all FunctionalClusters” of this specification shall be considered as an additional and required specification for implementation of the `Persistency`.

4 Constraints and Assumptions

4.1 Known Limitations

- Although a `Key-Value Storage` and `File Storage` can be configured as write-only, the current API always allows read access. Read access is even possible when a `file` has been opened with `ara::per::FileStorage::OpenFileWriteOnly`.

4.2 Constraints on Configuration

There are several constraints on the `Persistency` configuration that need to be observed by the tooling which creates/processes this part of the `Execution Manifest`. These constraints are defined in [3].

4.3 Direct Access to Storage Hardware

Modern embedded controllers use flash memory and similar hardware to store data. These devices have the intrinsic problem that the signal that can be read from each memory cell is reduced over time, mainly influenced by the number of write accesses. In the end, the cell will produce arbitrary values on each read access.

Unfortunately, the distribution of write accesses in typical systems is very uneven. Some parameters might be updated a few times a second, while some code may stay untouched for the whole life time of the ECU. To avoid early read errors, wear leveling should be deployed, such that frequent updates of single data elements are distributed over the whole memory area.

On the other hand, most operating systems include a file system or at least a flash driver that takes care of wear leveling, such that a typical implementation of the `Persistency` will not have to care about the wear leveling. This use case is therefore not described in any detail in this specification.

5 Dependencies to Other Functional Clusters

5.1 Protocol Layer Dependencies

The `Persistency` is (at least partially) compiled as part of an `Executable` of an `Adaptive Application`, and therefore also executed as part of a `Process`, which creates an implicit dependency on the `Execution Management` [4].

For the implementation of redundancy and security purposes, the `Persistency` accesses services of the `Cryptography` [5].

For the installation, update, and deletion of persisted data, the `Persistency` interacts with the `Update and Configuration Management` [6].

6 Requirements Tracing

The following table references the requirements specified in the AUTOSAR RS Persistency [7] and the AUTOSAR RS General [8], and links to the fulfillments of these. Please note that if column “Satisfied by” is empty for a specific requirement, this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[RS_AP_00111]	The AUTOSAR Adaptive Platform shall support source code portability for AUTOSAR Adaptive applications.	[SWS_PER_NA]
[RS_AP_00114]	C++ interface shall be compatible with C++14.	[SWS_PER_NA]
[RS_AP_00115]	Public namespaces.	[SWS_PER_00002]
[RS_AP_00116]	Header file name.	[SWS_PER_NA]
[RS_AP_00119]	Return values / application errors.	[SWS_PER_00042] [SWS_PER_00043] [SWS_PER_00046] [SWS_PER_00047] [SWS_PER_00048] [SWS_PER_00049] [SWS_PER_00052] [SWS_PER_00107] [SWS_PER_00110] [SWS_PER_00111] [SWS_PER_00112] [SWS_PER_00113] [SWS_PER_00114] [SWS_PER_00115] [SWS_PER_00116] [SWS_PER_00119] [SWS_PER_00122] [SWS_PER_00125] [SWS_PER_00144] [SWS_PER_00162] [SWS_PER_00163] [SWS_PER_00164] [SWS_PER_00165] [SWS_PER_00166] [SWS_PER_00167] [SWS_PER_00168] [SWS_PER_00313] [SWS_PER_00314] [SWS_PER_00315] [SWS_PER_00323] [SWS_PER_00325] [SWS_PER_00327] [SWS_PER_00329] [SWS_PER_00332] [SWS_PER_00333] [SWS_PER_00334] [SWS_PER_00335] [SWS_PER_00336] [SWS_PER_00337] [SWS_PER_00338] [SWS_PER_00351] [SWS_PER_00352] [SWS_PER_00357] [SWS_PER_00358] [SWS_PER_00360] [SWS_PER_00361] [SWS_PER_00363] [SWS_PER_00364] [SWS_PER_00365] [SWS_PER_00368] [SWS_PER_00370] [SWS_PER_00372] [SWS_PER_00375] [SWS_PER_00376] [SWS_PER_00377] [SWS_PER_00398] [SWS_PER_00399] [SWS_PER_00400] [SWS_PER_00401] [SWS_PER_00402] [SWS_PER_00403] [SWS_PER_00405] [SWS_PER_00406] [SWS_PER_00407] [SWS_PER_00414] [SWS_PER_00416] [SWS_PER_00418] [SWS_PER_00419] [SWS_PER_00420] [SWS_PER_00421] [SWS_PER_00422] [SWS_PER_00423]

Requirement	Description	Satisfied by
		[SWS_PER_00424] [SWS_PER_00426] [SWS_PER_00427] [SWS_PER_00428] [SWS_PER_00429] [SWS_PER_00430] [SWS_PER_00431] [SWS_PER_00434] [SWS_PER_00438]
[RS_AP_00120]	Method and Function names.	[SWS_PER_00042] [SWS_PER_00043] [SWS_PER_00046] [SWS_PER_00047] [SWS_PER_00048] [SWS_PER_00049] [SWS_PER_00050] [SWS_PER_00052] [SWS_PER_00107] [SWS_PER_00110] [SWS_PER_00111] [SWS_PER_00112] [SWS_PER_00113] [SWS_PER_00114] [SWS_PER_00115] [SWS_PER_00116] [SWS_PER_00119] [SWS_PER_00122] [SWS_PER_00125] [SWS_PER_00144] [SWS_PER_00162] [SWS_PER_00163] [SWS_PER_00164] [SWS_PER_00165] [SWS_PER_00166] [SWS_PER_00167] [SWS_PER_00168] [SWS_PER_00313] [SWS_PER_00314] [SWS_PER_00315] [SWS_PER_00322] [SWS_PER_00323] [SWS_PER_00324] [SWS_PER_00325] [SWS_PER_00326] [SWS_PER_00327] [SWS_PER_00328] [SWS_PER_00329] [SWS_PER_00330] [SWS_PER_00332] [SWS_PER_00333] [SWS_PER_00334] [SWS_PER_00335] [SWS_PER_00336] [SWS_PER_00337] [SWS_PER_00338] [SWS_PER_00350] [SWS_PER_00351] [SWS_PER_00352] [SWS_PER_00355] [SWS_PER_00356] [SWS_PER_00357] [SWS_PER_00358] [SWS_PER_00365] [SWS_PER_00367] [SWS_PER_00368] [SWS_PER_00369] [SWS_PER_00370] [SWS_PER_00371] [SWS_PER_00372] [SWS_PER_00373] [SWS_PER_00374] [SWS_PER_00375] [SWS_PER_00376] [SWS_PER_00377] [SWS_PER_00405] [SWS_PER_00406] [SWS_PER_00407] [SWS_PER_00413] [SWS_PER_00414] [SWS_PER_00415] [SWS_PER_00416] [SWS_PER_00417] [SWS_PER_00418] [SWS_PER_00419] [SWS_PER_00420] [SWS_PER_00421] [SWS_PER_00422] [SWS_PER_00423] [SWS_PER_00424] [SWS_PER_00426] [SWS_PER_00427] [SWS_PER_00428] [SWS_PER_00429] [SWS_PER_00430] [SWS_PER_00431] [SWS_PER_00433] [SWS_PER_00434] [SWS_PER_00438] [SWS_PER_00459] [SWS_PER_00460] [SWS_PER_00461] [SWS_PER_00462]

Requirement	Description	Satisfied by
[RS_AP_00121]	Parameter names.	[SWS_PER_00043] [SWS_PER_00046] [SWS_PER_00047] [SWS_PER_00052] [SWS_PER_00111] [SWS_PER_00112] [SWS_PER_00113] [SWS_PER_00114] [SWS_PER_00115] [SWS_PER_00116] [SWS_PER_00119] [SWS_PER_00125] [SWS_PER_00144] [SWS_PER_00163] [SWS_PER_00164] [SWS_PER_00165] [SWS_PER_00166] [SWS_PER_00315] [SWS_PER_00322] [SWS_PER_00323] [SWS_PER_00326] [SWS_PER_00327] [SWS_PER_00332] [SWS_PER_00333] [SWS_PER_00334] [SWS_PER_00335] [SWS_PER_00336] [SWS_PER_00337] [SWS_PER_00338] [SWS_PER_00350] [SWS_PER_00351] [SWS_PER_00355] [SWS_PER_00356] [SWS_PER_00367] [SWS_PER_00368] [SWS_PER_00369] [SWS_PER_00370] [SWS_PER_00371] [SWS_PER_00372] [SWS_PER_00375] [SWS_PER_00376] [SWS_PER_00377] [SWS_PER_00405] [SWS_PER_00406] [SWS_PER_00407] [SWS_PER_00413] [SWS_PER_00414] [SWS_PER_00420] [SWS_PER_00421] [SWS_PER_00422] [SWS_PER_00423] [SWS_PER_00424] [SWS_PER_00426] [SWS_PER_00427] [SWS_PER_00429] [SWS_PER_00430] [SWS_PER_00431] [SWS_PER_00433] [SWS_PER_00434] [SWS_PER_00438]
[RS_AP_00122]	Type names.	[SWS_PER_00146] [SWS_PER_00147] [SWS_PER_00311] [SWS_PER_00312] [SWS_PER_00339] [SWS_PER_00340] [SWS_PER_00342] [SWS_PER_00343] [SWS_PER_00354] [SWS_PER_00359] [SWS_PER_00362] [SWS_PER_00411] [SWS_PER_00412] [SWS_PER_00432] [SWS_PER_00435] [SWS_PER_00436] [SWS_PER_00437]
[RS_AP_00124]	Variable names.	[SWS_PER_NA]
[RS_AP_00127]	Usage of ara::core types.	[SWS_PER_00042] [SWS_PER_00043] [SWS_PER_00046] [SWS_PER_00047] [SWS_PER_00048] [SWS_PER_00049] [SWS_PER_00052] [SWS_PER_00110] [SWS_PER_00111] [SWS_PER_00112] [SWS_PER_00113] [SWS_PER_00114] [SWS_PER_00115] [SWS_PER_00116] [SWS_PER_00119] [SWS_PER_00122] [SWS_PER_00125] [SWS_PER_00165] [SWS_PER_00166] [SWS_PER_00311] [SWS_PER_00312] [SWS_PER_00332] [SWS_PER_00333] [SWS_PER_00334]

Requirement	Description	Satisfied by
		[SWS_PER_00335] [SWS_PER_00336] [SWS_PER_00337] [SWS_PER_00338] [SWS_PER_00354] [SWS_PER_00356] [SWS_PER_00357] [SWS_PER_00358] [SWS_PER_00365] [SWS_PER_00375] [SWS_PER_00376] [SWS_PER_00377] [SWS_PER_00405] [SWS_PER_00406] [SWS_PER_00407] [SWS_PER_00420] [SWS_PER_00421] [SWS_PER_00422] [SWS_PER_00423] [SWS_PER_00424] [SWS_PER_00426] [SWS_PER_00427] [SWS_PER_00428] [SWS_PER_00429] [SWS_PER_00430] [SWS_PER_00431] [SWS_PER_00433] [SWS_PER_00438]
[RS_AP_00128]	Error reporting.	[SWS_PER_00046] [SWS_PER_00047] [SWS_PER_00048] [SWS_PER_00049] [SWS_PER_00052] [SWS_PER_00111] [SWS_PER_00113] [SWS_PER_00114] [SWS_PER_00115] [SWS_PER_00116] [SWS_PER_00122] [SWS_PER_00332] [SWS_PER_00333] [SWS_PER_00334] [SWS_PER_00335] [SWS_PER_00336] [SWS_PER_00337] [SWS_PER_00338] [SWS_PER_00353] [SWS_PER_00357] [SWS_PER_00358] [SWS_PER_00365] [SWS_PER_00375] [SWS_PER_00376] [SWS_PER_00377] [SWS_PER_00405] [SWS_PER_00406] [SWS_PER_00407] [SWS_PER_00424] [SWS_PER_00426] [SWS_PER_00427] [SWS_PER_00428] [SWS_PER_00429] [SWS_PER_00430] [SWS_PER_00431] [SWS_PER_00438] [SWS_PER_00472] [SWS_PER_00473] [SWS_PER_00474] [SWS_PER_00475] [SWS_PER_00476]
[RS_AP_00129]	Public types defined by functional clusters shall be designed to allow implementation without dynamic memory allocation.	[SWS_PER_00042] [SWS_PER_00046] [SWS_PER_00047] [SWS_PER_00048] [SWS_PER_00049] [SWS_PER_00050] [SWS_PER_00052] [SWS_PER_00110] [SWS_PER_00111] [SWS_PER_00113] [SWS_PER_00114] [SWS_PER_00115] [SWS_PER_00116] [SWS_PER_00119] [SWS_PER_00122] [SWS_PER_00322] [SWS_PER_00326] [SWS_PER_00330] [SWS_PER_00332] [SWS_PER_00333] [SWS_PER_00334] [SWS_PER_00335] [SWS_PER_00336] [SWS_PER_00337]

Requirement	Description	Satisfied by
		[SWS_PER_00338] [SWS_PER_00360] [SWS_PER_00361] [SWS_PER_00363] [SWS_PER_00364] [SWS_PER_00365] [SWS_PER_00367] [SWS_PER_00369] [SWS_PER_00371] [SWS_PER_00375] [SWS_PER_00376] [SWS_PER_00377] [SWS_PER_00398] [SWS_PER_00399] [SWS_PER_00400] [SWS_PER_00401] [SWS_PER_00402] [SWS_PER_00403] [SWS_PER_00405] [SWS_PER_00406] [SWS_PER_00407] [SWS_PER_00413] [SWS_PER_00417] [SWS_PER_00424] [SWS_PER_00426] [SWS_PER_00427] [SWS_PER_00428] [SWS_PER_00429] [SWS_PER_00430] [SWS_PER_00431] [SWS_PER_00438] [SWS_PER_00459] [SWS_PER_00460] [SWS_PER_00461] [SWS_PER_00462]
[RS_AP_00130]	AUTOSAR Adaptive Platform shall represent a rich and modern programming environment.	[SWS_PER_NA]
[RS_AP_00132]	noexcept behavior of API functions	[SWS_PER_00042] [SWS_PER_00043] [SWS_PER_00046] [SWS_PER_00047] [SWS_PER_00048] [SWS_PER_00049] [SWS_PER_00050] [SWS_PER_00052] [SWS_PER_00107] [SWS_PER_00110] [SWS_PER_00111] [SWS_PER_00112] [SWS_PER_00113] [SWS_PER_00114] [SWS_PER_00115] [SWS_PER_00116] [SWS_PER_00119] [SWS_PER_00122] [SWS_PER_00125] [SWS_PER_00162] [SWS_PER_00163] [SWS_PER_00164] [SWS_PER_00165] [SWS_PER_00166] [SWS_PER_00167] [SWS_PER_00168] [SWS_PER_00313] [SWS_PER_00314] [SWS_PER_00315] [SWS_PER_00322] [SWS_PER_00323] [SWS_PER_00326] [SWS_PER_00327] [SWS_PER_00330] [SWS_PER_00332] [SWS_PER_00333] [SWS_PER_00334] [SWS_PER_00335] [SWS_PER_00336] [SWS_PER_00337] [SWS_PER_00338] [SWS_PER_00351] [SWS_PER_00352] [SWS_PER_00355] [SWS_PER_00356] [SWS_PER_00357] [SWS_PER_00358] [SWS_PER_00360]

Requirement	Description	Satisfied by
		[SWS_PER_00361] [SWS_PER_00363] [SWS_PER_00364] [SWS_PER_00365] [SWS_PER_00367] [SWS_PER_00368] [SWS_PER_00369] [SWS_PER_00370] [SWS_PER_00371] [SWS_PER_00372] [SWS_PER_00375] [SWS_PER_00376] [SWS_PER_00377] [SWS_PER_00398] [SWS_PER_00399] [SWS_PER_00400] [SWS_PER_00401] [SWS_PER_00402] [SWS_PER_00403] [SWS_PER_00405] [SWS_PER_00406] [SWS_PER_00407] [SWS_PER_00413] [SWS_PER_00414] [SWS_PER_00417] [SWS_PER_00418] [SWS_PER_00419] [SWS_PER_00420] [SWS_PER_00421] [SWS_PER_00422] [SWS_PER_00423] [SWS_PER_00424] [SWS_PER_00426] [SWS_PER_00427] [SWS_PER_00428] [SWS_PER_00429] [SWS_PER_00430] [SWS_PER_00431] [SWS_PER_00433] [SWS_PER_00438]
[RS_AP_00134]	noexcept behavior of class destructors	[SWS_PER_00050] [SWS_PER_00330] [SWS_PER_00417]
[RS_AP_00144]	Availability of a named constructor.	[SWS_PER_00052] [SWS_PER_00113] [SWS_PER_00114] [SWS_PER_00115] [SWS_PER_00116] [SWS_PER_00375] [SWS_PER_00376] [SWS_PER_00377] [SWS_PER_00429] [SWS_PER_00430] [SWS_PER_00431]
[RS_AP_00146]	Classes whose construction requires interaction by the ARA framework.	[SWS_PER_00339] [SWS_PER_00340] [SWS_PER_00342] [SWS_PER_00343] [SWS_PER_00459] [SWS_PER_00460] [SWS_PER_00461] [SWS_PER_00462]
[RS_PER_00001]	Persistency shall support storage of persistent data	[SWS_PER_00107] [SWS_PER_00110] [SWS_PER_00111] [SWS_PER_00112] [SWS_PER_00113] [SWS_PER_00114] [SWS_PER_00115] [SWS_PER_00116] [SWS_PER_00119] [SWS_PER_00122] [SWS_PER_00125] [SWS_PER_00144] [SWS_PER_00162] [SWS_PER_00163] [SWS_PER_00164] [SWS_PER_00165] [SWS_PER_00166] [SWS_PER_00167] [SWS_PER_00168] [SWS_PER_00302] [SWS_PER_00303] [SWS_PER_00304] [SWS_PER_00309] [SWS_PER_00335]

Requirement	Description	Satisfied by
		[SWS_PER_00336] [SWS_PER_00337] [SWS_PER_00338] [SWS_PER_00360] [SWS_PER_00361] [SWS_PER_00363] [SWS_PER_00364] [SWS_PER_00375] [SWS_PER_00376] [SWS_PER_00377] [SWS_PER_00398] [SWS_PER_00399] [SWS_PER_00400] [SWS_PER_00401] [SWS_PER_00402] [SWS_PER_00403] [SWS_PER_00418] [SWS_PER_00419] [SWS_PER_00420] [SWS_PER_00421] [SWS_PER_00422] [SWS_PER_00423] [SWS_PER_00425] [SWS_PER_00428] [SWS_PER_00429] [SWS_PER_00430] [SWS_PER_00431] [SWS_PER_00434] [SWS_PER_00494] [SWS_PER_00495] [SWS_PER_00499] [SWS_PER_00501] [SWS_PER_00502] [SWS_PER_00503] [SWS_PER_00534] [SWS_PER_00535]
[RS_PER_00002]	Persistency shall support to retrieve data that has been persistently stored on a platform instance	[SWS_PER_00049] [SWS_PER_00050] [SWS_PER_00322] [SWS_PER_00323] [SWS_PER_00324] [SWS_PER_00325] [SWS_PER_00339] [SWS_PER_00359] [SWS_PER_00360] [SWS_PER_00361] [SWS_PER_00362] [SWS_PER_00363] [SWS_PER_00364] [SWS_PER_00365] [SWS_PER_00371] [SWS_PER_00372] [SWS_PER_00373] [SWS_PER_00374] [SWS_PER_00398] [SWS_PER_00399] [SWS_PER_00400] [SWS_PER_00401] [SWS_PER_00402] [SWS_PER_00403] [SWS_PER_00459] [SWS_PER_00496] [SWS_PER_00497] [SWS_PER_00498] [SWS_PER_00506]
[RS_PER_00003]	Persistency shall support identification of data using a unique identifier	[SWS_PER_00042] [SWS_PER_00043] [SWS_PER_00046] [SWS_PER_00047] [SWS_PER_00048] [SWS_PER_00052] [SWS_PER_00146] [SWS_PER_00147] [SWS_PER_00331] [SWS_PER_00332] [SWS_PER_00333] [SWS_PER_00334] [SWS_PER_00360] [SWS_PER_00361] [SWS_PER_00363] [SWS_PER_00364] [SWS_PER_00398] [SWS_PER_00399] [SWS_PER_00400] [SWS_PER_00401] [SWS_PER_00402] [SWS_PER_00403] [SWS_PER_00426] [SWS_PER_00427] [SWS_PER_00496] [SWS_PER_00497] [SWS_PER_00498] [SWS_PER_00499] [SWS_PER_00501] [SWS_PER_00502] [SWS_PER_00504] [SWS_PER_00505] [SWS_PER_00534] [SWS_PER_00535]

Requirement	Description	Satisfied by
[RS_PER_00004]	Persistency shall support access to file-like structures	[SWS_PER_00107] [SWS_PER_00110] [SWS_PER_00111] [SWS_PER_00112] [SWS_PER_00113] [SWS_PER_00114] [SWS_PER_00115] [SWS_PER_00116] [SWS_PER_00119] [SWS_PER_00122] [SWS_PER_00125] [SWS_PER_00144] [SWS_PER_00162] [SWS_PER_00163] [SWS_PER_00164] [SWS_PER_00165] [SWS_PER_00166] [SWS_PER_00167] [SWS_PER_00168] [SWS_PER_00326] [SWS_PER_00327] [SWS_PER_00328] [SWS_PER_00329] [SWS_PER_00330] [SWS_PER_00335] [SWS_PER_00336] [SWS_PER_00337] [SWS_PER_00338] [SWS_PER_00340] [SWS_PER_00342] [SWS_PER_00343] [SWS_PER_00367] [SWS_PER_00368] [SWS_PER_00369] [SWS_PER_00370] [SWS_PER_00375] [SWS_PER_00376] [SWS_PER_00377] [SWS_PER_00413] [SWS_PER_00414] [SWS_PER_00415] [SWS_PER_00416] [SWS_PER_00417] [SWS_PER_00418] [SWS_PER_00419] [SWS_PER_00420] [SWS_PER_00421] [SWS_PER_00422] [SWS_PER_00423] [SWS_PER_00428] [SWS_PER_00429] [SWS_PER_00430] [SWS_PER_00431] [SWS_PER_00434] [SWS_PER_00435] [SWS_PER_00436] [SWS_PER_00437] [SWS_PER_00438] [SWS_PER_00440] [SWS_PER_00441] [SWS_PER_00442] [SWS_PER_00443] [SWS_PER_00444] [SWS_PER_00445] [SWS_PER_00457] [SWS_PER_00458] [SWS_PER_00460] [SWS_PER_00461] [SWS_PER_00462] [SWS_PER_00507] [SWS_PER_00508] [SWS_PER_00509] [SWS_PER_00510] [SWS_PER_00511] [SWS_PER_00512] [SWS_PER_00513] [SWS_PER_00514] [SWS_PER_00515] [SWS_PER_00516] [SWS_PER_00517] [SWS_PER_00518] [SWS_PER_00519] [SWS_PER_00520] [SWS_PER_00521] [SWS_PER_00522] [SWS_PER_00523] [SWS_PER_00524] [SWS_PER_00525] [SWS_PER_00526] [SWS_PER_00527] [SWS_PER_00528] [SWS_PER_00529] [SWS_PER_00530] [SWS_PER_00531] [SWS_PER_00532] [SWS_PER_00533]
[RS_PER_00005]	Persistency shall support encryption/decryption of persistent data	[SWS_PER_00210] [SWS_PER_00211] [SWS_PER_00449] [SWS_PER_00450] [SWS_PER_00451] [SWS_PER_00464] [SWS_PER_00465] [SWS_PER_00466] [SWS_PER_00467] [SWS_PER_00468]

Requirement	Description	Satisfied by
[RS_PER_00008]	Persistency shall support detection of data corruption in persistent memory	[SWS_PER_00221] [SWS_PER_00317] [SWS_PER_00318] [SWS_PER_00319] [SWS_PER_00432] [SWS_PER_00433] [SWS_PER_00439] [SWS_PER_00447] [SWS_PER_00448] [SWS_PER_00480] [SWS_PER_00481] [SWS_PER_00482] [SWS_PER_00483] [SWS_PER_00484] [SWS_PER_00485] [SWS_PER_00486] [SWS_PER_00487] [SWS_PER_00488] [SWS_PER_00489] [SWS_PER_00490]
[RS_PER_00009]	Persistency shall support data recovery mechanisms if persistent data was corrupted	[SWS_PER_00317] [SWS_PER_00318] [SWS_PER_00319] [SWS_PER_00333] [SWS_PER_00334] [SWS_PER_00335] [SWS_PER_00336] [SWS_PER_00337] [SWS_PER_00338] [SWS_PER_00358] [SWS_PER_00426] [SWS_PER_00427] [SWS_PER_00439] [SWS_PER_00447] [SWS_PER_00448] [SWS_PER_00452] [SWS_PER_00453] [SWS_PER_00454] [SWS_PER_00455] [SWS_PER_00456] [SWS_PER_00477] [SWS_PER_00478] [SWS_PER_00479]
[RS_PER_00010]	The layout of persistent data shall be configurable	[SWS_PER_00046] [SWS_PER_00047] [SWS_PER_00048] [SWS_PER_00052] [SWS_PER_00113] [SWS_PER_00114] [SWS_PER_00115] [SWS_PER_00116] [SWS_PER_00210] [SWS_PER_00211] [SWS_PER_00251] [SWS_PER_00252] [SWS_PER_00253] [SWS_PER_00254] [SWS_PER_00265] [SWS_PER_00266] [SWS_PER_00267] [SWS_PER_00275] [SWS_PER_00277] [SWS_PER_00281] [SWS_PER_00283] [SWS_PER_00304] [SWS_PER_00317] [SWS_PER_00318] [SWS_PER_00319] [SWS_PER_00320] [SWS_PER_00321] [SWS_PER_00332] [SWS_PER_00333] [SWS_PER_00334] [SWS_PER_00335] [SWS_PER_00336] [SWS_PER_00375] [SWS_PER_00376] [SWS_PER_00377] [SWS_PER_00378] [SWS_PER_00379] [SWS_PER_00380] [SWS_PER_00382] [SWS_PER_00383] [SWS_PER_00384] [SWS_PER_00385] [SWS_PER_00386] [SWS_PER_00387] [SWS_PER_00388] [SWS_PER_00389] [SWS_PER_00390] [SWS_PER_00391]

Requirement	Description	Satisfied by
		[SWS_PER_00392] [SWS_PER_00393] [SWS_PER_00394] [SWS_PER_00395] [SWS_PER_00426] [SWS_PER_00427] [SWS_PER_00429] [SWS_PER_00430] [SWS_PER_00431] [SWS_PER_00439] [SWS_PER_00447] [SWS_PER_00448] [SWS_PER_00449] [SWS_PER_00450] [SWS_PER_00451] [SWS_PER_00463] [SWS_PER_00464] [SWS_PER_00465] [SWS_PER_00466] [SWS_PER_00467] [SWS_PER_00468] [SWS_PER_00469] [SWS_PER_00470] [SWS_PER_00471] [SWS_PER_CONSTR_00001] [SWS_PER_CONSTR_00002] [SWS_PER_CONSTR_00003] [SWS_PER_CONSTR_00004]
[RS_PER_00011]	Persistency shall be able to ensure and limit the amount of storage used by persisted data	[SWS_PER_00320] [SWS_PER_00321] [SWS_PER_00491] [SWS_PER_00492] [SWS_PER_00493]
[RS_PER_00012]	Persistency shall support installation of persistent data	[SWS_PER_00251] [SWS_PER_00252] [SWS_PER_00253] [SWS_PER_00254] [SWS_PER_00265] [SWS_PER_00266] [SWS_PER_00267] [SWS_PER_00379] [SWS_PER_00380] [SWS_PER_00382] [SWS_PER_00383] [SWS_PER_00384] [SWS_PER_00385] [SWS_PER_00463] [SWS_PER_00469] [SWS_PER_00470] [SWS_PER_00471] [SWS_PER_CONSTR_00001] [SWS_PER_CONSTR_00002] [SWS_PER_CONSTR_00003] [SWS_PER_CONSTR_00004]
[RS_PER_00013]	Persistency shall support update of persistent data	[SWS_PER_00251] [SWS_PER_00275] [SWS_PER_00277] [SWS_PER_00281] [SWS_PER_00283] [SWS_PER_00356] [SWS_PER_00357] [SWS_PER_00378] [SWS_PER_00379] [SWS_PER_00380] [SWS_PER_00386] [SWS_PER_00387] [SWS_PER_00388] [SWS_PER_00389] [SWS_PER_00390] [SWS_PER_00391] [SWS_PER_00392] [SWS_PER_00393] [SWS_PER_00394] [SWS_PER_00395] [SWS_PER_00463] [SWS_PER_00469] [SWS_PER_00470] [SWS_PER_00471]
[RS_PER_00014]	Persistency shall support roll-back of persistent data	[SWS_PER_00378] [SWS_PER_00396] [SWS_PER_00463] [SWS_PER_00469] [SWS_PER_00470] [SWS_PER_00471]
[RS_PER_00016]	Persistency shall support finalization of an update of persistent data	[SWS_PER_00446] [SWS_PER_00463] [SWS_PER_00470] [SWS_PER_00471]
[RS_PER_00017]	Persistency shall be able to report the amount of currently used storage	[SWS_PER_00405] [SWS_PER_00406] [SWS_PER_00407] [SWS_PER_00424]

Requirement	Description	Satisfied by
[RS_PER_00018]	Persistency shall support central initialization and shutdown	[SWS_PER_00408] [SWS_PER_00409] [SWS_PER_00410]

7 Functional Specification

7.1 The Architecture of Persistency

The Persistency offers two different mechanisms to access persistent memory: Key-Value Storages offer access to a set of keys with associated values (similar to a database), while File Storages offer access to a set of files (similar to a directory of a file system).

The typical usage of the Persistency within an Adaptive Application is depicted in Figure 7.1. As shown there, an Adaptive Application can use a combination of multiple Key-Value Storages and multiple File Storages. Of course, the same applies to other functional clusters using Persistency.

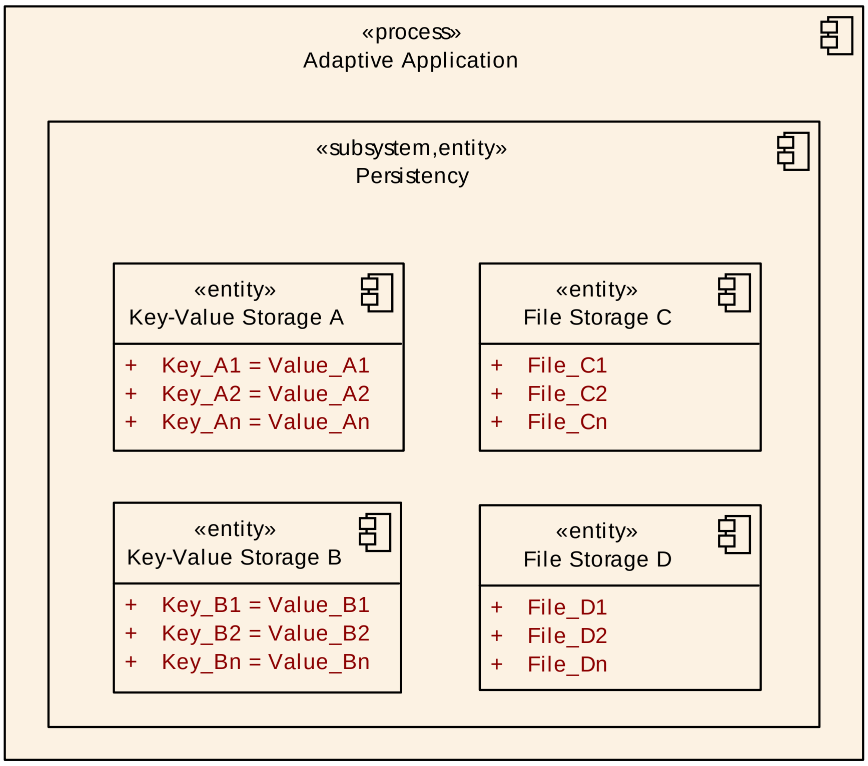

Figure 7.1: Typical usage of Persistency within an Adaptive Application

7.1.1 Persistency in the Manifest

The Persistency usage of an Adaptive Application is modeled in the Execution Manifest (furtheron simply referred to as the "manifest") as part of the AdaptiveApplicationSwComponentTypes of an Executable. The model has two principal parts: The application design information, aggregated by the PersistencyKeyValueStorageInterface and the PersistencyFileStorageInterface, and the deployment information, aggregated by the PersistencyKeyValueStorage and the PersistencyFileStorage.

The API specification holds the classes ara::per::KeyValueStorage and ara::per::FileStorage for access to a Key-Value Storage or a File Storage, respectively. The global functions of these classes receive the identifier (the fully qualified shortName path) of a PortPrototype typed by a PersistencyInterface as an ara::core::InstanceSpecifier input parameter (see subsection 8.2.1 and subsection 8.3.1). Depending on the nature of the PortPrototype, the Key-Value Storage or File Storage will be accessible as:

Read Only if the PortPrototype is instantiated as RPortPrototype, or

Read/Write if the PortPrototype is instantiated as PRPortPrototype, or

Write Only if the PortPrototype is instantiated as PPortPrototype.

The manifest contains separate deployment data for each Process that references the Executable. The Process is bound to the deployment data by specialization of the class PersistencyPortPrototypeToDeploymentMapping, which refers to a PortPrototype typed by a PersistencyInterface, a PersistencyDeployment, and the Process.

Usage of base classes in the manifest

For simplification reasons, the information that applies to both the Key-Value Storages and the File Storages is collected in base classes in the manifest, namely in PersistencyInterface for PersistencyKeyValueStorageInterface and PersistencyFileStorageInterface, and in PersistencyDeployment for PersistencyKeyValueStorage and PersistencyFileStorage.

Likewise, the common information about key-value pairs and files is collected in PersistencyInterfaceElement for PersistencyDataElement and PersistencyFileElement, and in PersistencyDeploymentElement for PersistencyKeyValuePair and PersistencyFile.

And the link between application design and deployment information, represented by PersistencyPortPrototypeToDeploymentMapping, is specialized as PersistencyPortPrototypeToKeyValueStorageMapping and PersistencyPortPrototypeToFileStorageMapping.

7.1.2 Key-Value Storages in the Manifest

Every Key-Value Storage is represented by a PortPrototype typed by a PersistencyKeyValueStorageInterface in the application design for the respective AdaptiveApplicationSwComponentType, and by a PersistencyKeyValueStorage containing deployment information. Every Key-Value Storage can hold multiple key-value pairs. Key-value pairs can be added and removed at run-time by the Adaptive Application using the Persistency API (see subsubsection 8.2.5.7 and subsubsection 8.2.5.8).

A Key-Value Storage with predefined key-value pairs can be deployed with default data during installation or update of an Adaptive Application. This operation is (indirectly) triggered by the Update and Configuration Management [6] during installation or update using the deployment information and data provided by the software package of the Adaptive Application. See subsection 7.2.6.

The link between application design and deployment information of a Key-Value Storage is represented by PersistencyPortPrototypeToKeyValueStorageMapping, which refers to a PortPrototype typed by a PersistencyKeyValueStorageInterface, the corresponding PersistencyKeyValueStorage, and a Process.

7.1.3 File Storages in the Manifest

Every File Storage is represented by a PortPrototype typed by a PersistencyFileStorageInterface in the application design for the respective AdaptiveApplicationSwComponentType, and by a PersistencyFileStorage containing deployment information. Every File Storage can hold multiple files as described in [3]. Similar to the key-value pairs mentioned above, files can be created and deleted at run-time by the Adaptive Application using the Persistency API (see subsubsection 8.3.11.11, subsubsection 8.3.11.13, and subsubsection 8.3.11.5).

A File Storage with predefined files with initial content can be deployed during installation or update. This operation is also (indirectly) triggered by the Update and Configuration Management [6]. All needed deployment information and files come with the software package of the Adaptive Application. See subsection 7.2.6.

The link between application design and deployment information of a File Storage is represented by PersistencyPortPrototypeToFileStorageMapping, which refers to a PortPrototype typed by a PersistencyFileStorageInterface, the corresponding PersistencyFileStorage, and a Process.

7.2 General Features of Persistency

[SWS_PER_00002] ⌈All specified classes within the Persistency shall reside within the C++ namespace `ara::per`.⌋*(RS_AP_00115)*

7.2.1 Functional Cluster Lifecycle

7.2.1.1 Initialization and Shutdown of Persistency

Using `ara::core::Initialize` and `ara::core::Deinitialize`, the application can start and shut down all functional clusters with direct ARA interfaces (i.e. the Adaptive Platform Foundation).

[SWS_PER_00408] ⌈When `ara::core::Initialize` is called, the Persistency shall read in the manifest information and prepare the access structures to all Key-Value Storages and File Storages that are defined in the manifest.⌋*(RS_PER_00018)*

[SWS_PER_00409] ⌈When `ara::core::Deinitialize` is called, the Persistency shall implicitly ensure that all open files of all File Storages are persisted as though `ara::per::ReadWriteAccessor::SyncToFile` was called and closed as though the `ara::per::UniqueHandle`s were destructed, and that not persisted values in all Key-Value Storages are dropped as though `ara::per::KeyValueStorage::DiscardPendingChanges` was called. Afterwards, all access structures shall be freed.⌋*(RS_PER_00018)*

The application is expected not to call any API of Persistency (directly or indirectly through other functional clusters) before `ara::core::Initialize` or after `ara::core::Deinitialize`, but Persistency needs to protect itself against such eventualities.

[SWS_PER_00410]{DRAFT} ⌈All functions of Persistency and all methods of its classes shall call `ara::core::Abort` when they are called after static initialization but before `ara::core::Initialize` was called or after `ara::core::Deinitialize` was called.⌋*(RS_PER_00018)*

7.2.2 Error Handling

Error handling in `Persistency` is aligned with the guidelines described in [2]. To this end, the `Persistency` has to implement a set of standard classes and APIs, which are described in this section.

[SWS_PER_00472] ⌈`Persistency` shall use the error codes defined in `ara::per::PerErrc` to report problems to the calling `application` via `ara::core::Result`. Vendors of `Persistency` may add their own errors to `ara::per::PerErrc`, using codes above 255.⌋*(RS_AP_00128)*

`ara::per::PerErrc` belongs to the `ara::per::PerErrorDomain`, which can be used by an `application` to classify returned errors.

[SWS_PER_00473] ⌈`ara::per::GetPerDomain` shall return the global `ara::per::PerErrorDomain` object.⌋*(RS_AP_00128)*

To create its own `Persistency` error codes, the `application` may use `ara::per::MakeErrorCode`.

[SWS_PER_00474] ⌈`ara::per::MakeErrorCode` shall return an `ara::core::ErrorCode` when called with an error code from `ara::per::PerErrc`.⌋*(RS_AP_00128)*

[SWS_PER_00353] ⌈`ara::per::PerErrorDomain::Name` shall return the NUL-terminated string “Per”.⌋*(RS_AP_00128)*

[SWS_PER_00475] ⌈`ara::per::PerErrorDomain::Message` shall return the error message associated with the passed `ara::core::ErrorCode`.⌋*(RS_AP_00128)*

The whole `Persistency` API has been designed to be exception-less. If an `application` prefers to use exceptions, it may use `ara::per::PerErrorDomain::ThrowAsException`, or simply `ara::core::ErrorCode::ThrowAsException`.

[SWS_PER_00476] ⌈`ara::per::PerErrorDomain::ThrowAsException` shall throw an `ara::per::PerException` that is created from the passed error code.⌋*(RS_AP_00128)*

7.2.3 Parallel Access to Persistent Data

According to [9], the persistent data is local to one Process. Therefore, Persistency will never share persistent data between two (or more) Processes, even of the same Executable. The background of this decision is that Persistency should not provide an additional communication path for applications besides the mechanisms provided by the functional cluster Communication Management (e.g. using ara::com).

[SWS_PER_00309] ⌈Persistent data shall always be local to one Process.⌋ *(RS_PER_00001)*

If persistent data needs to be accessed by multiple Processes (of the same or different applications), it is the duty of the application designer to provide Service Interfaces for communication.

Persistency is, on the other hand, prepared to handle concurrent access from multiple threads of the same application, running in the context of the same Process. To create shared access to a Key-Value Storage or File Storage, either the ara::per::SharedHandle returned by ara::per::OpenKeyValueStorage and ara::per::OpenFileStorage can be passed on (i.e. copied) to another thread, or ara::per::OpenKeyValueStorage and ara::per::OpenFileStorage can be called in independent threads for the same Key-Value Storage or File Storage, respectively. All operations of the Key-Value Storage and File Storage support concurrent access from multiple threads, though operations like ara::per::RecoverKeyValueStorage and ara::per::ResetKeyValueStorage or ara::per::RecoverAllFiles and ara::per::ResetAllFiles will only succeed when the corresponding Key-Value Storage or File Storage is not opened.

Access to single key-value pairs of a Key-Value Storage is possible from multiple threads at the same time, because the operation of ara::per::KeyValueStorage::GetValue and ara::per::KeyValueStorage::SetValue are atomic, as are those of ara::per::KeyValueStorage::RemoveKey, ara::per::KeyValueStorage::RemoveAllKeys, ara::per::KeyValueStorage::SyncToStorage, and ara::per::KeyValueStorage::DiscardPendingChanges.

Access to single files of a File Storage cannot be shared between multiple threads, because it would be impossible to synchronize read and write accesses and the corresponding change of the seek position in a file. Accordingly, the ara::per::UniqueHandle returned by the OpenFile* APIs can only be moved to another thread, and trying to open an already opened file will fail. Likewise, operations like ara::per::FileStorage::DeleteFile, ara::per::FileStorage::RecoverFile, and ara::per::FileStorage::ResetFile will also not be possible on open files.

Files are implicitly closed when their ara::per::UniqueHandle goes out of scope, or when the File Storage to which they belong is closed.

[SWS_PER_00425] ⌈When a `File Storage` is closed, because all related `ara::per::SharedHandle`s go out of scope, any `files` which are still open are also closed.⌋*(RS_PER_00001)*

Accessing a `ara::per::UniqueHandle` of a `file` of a closed `File Storage` will result in undefined behavior.

7.2.4 Security Concepts

The Persistency supports encryption and authentication of data stored in a Key--Value Storage or File Storage. Whether encryption and/or authentication is applied, is decided at deployment time. The application is not aware of this fact.

In general, a storage or an element of a storage are encrypted after the creation of the storage and when the storage is saved, and are decrypted when a storage is opened. The signed hash used for the authentication of a storage is likewise verified when opening a storage, and calculated during installation or when saving a storage.

In case of a read-only storage, encryption is done only once during installation. A signed hash used for authentication of a read-only storage (or an element therein) is either provided as PersistencyDeploymentToCryptoKeySlotMapping.verificationHash (or PersistencyDeploymentElementToCryptoKeySlotMapping.verificationHash) in the manifest, or calculated during installation.

[SWS_PER_00210] ⌈If a PersistencyDeploymentToCryptoKeySlotMapping exists in the manifest, and PersistencyDeploymentToCryptoKeySlotMapping.keySlotUsage is set to encryption, the Persistency shall encrypt all data related to the storage before storing it to the persistent memory.⌋*(RS_PER_00005, RS_PER_00010)*

[SWS_PER_00464] ⌈If a PersistencyDeploymentElementToCryptoKeySlotMapping exists in the manifest, and PersistencyDeploymentElementToCryptoKeySlotMapping.keySlotUsage is set to encryption, the Persistency shall encrypt the element data before storing it to the persistent memory.⌋*(RS_PER_00005, RS_PER_00010)*

[SWS_PER_00211] ⌈If a PersistencyDeploymentToCryptoKeySlotMapping exists in the manifest, and PersistencyDeploymentToCryptoKeySlotMapping.keySlotUsage is set to encryption, the Persistency shall decrypt all data related to the storage after reading it from persistent memory.⌋*(RS_PER_00005, RS_PER_00010)*

[SWS_PER_00465] ⌈If a PersistencyDeploymentElementToCryptoKeySlotMapping exists in the manifest, and PersistencyDeploymentElementToCryptoKeySlotMapping.keySlotUsage is set to encryption, the Persistency shall decrypt the element data after reading it from persistent memory.⌋*(RS_PER_00005, RS_PER_00010)*

[SWS_PER_00449] ⌈If a PersistencyDeploymentToCryptoKeySlotMapping exists in the manifest, and PersistencyDeploymentToCryptoKeySlotMapping.keySlotUsage is set to verification, the Persistency shall sign all data related to the storage before storing it to the persistent memory.⌋*(RS_PER_00005, RS_PER_00010)*

[SWS_PER_00466] ⌈If a PersistencyDeploymentElementToCryptoKeySlotMapping exists in the manifest, and PersistencyDeploymentElementToCryptoKeySlotMapping.keySlotUsage is set to verification, the Persistency shall sign the element data before storing it to the persistent memory.⌋*(RS_PER_00005, RS_PER_00010)*

[SWS_PER_00450] ⌈If a PersistencyDeploymentToCryptoKeySlotMapping exists in the manifest, and PersistencyDeploymentToCryptoKeySlotMapping.keySlotUsage is set to verification, the Persistency shall verify the signature of all data related to the storage after reading it from persistent memory.⌋*(RS_PER_00005, RS_PER_00010)*

[SWS_PER_00467] ⌈If a PersistencyDeploymentElementToCryptoKeySlotMapping exists in the manifest, and PersistencyDeploymentElementToCryptoKeySlotMapping.keySlotUsage is set to verification, the Persistency shall verify the signature of the element data after reading it from persistent memory.⌋*(RS_PER_00005, RS_PER_00010)*

[SWS_PER_00451] ⌈If PersistencyDeploymentToCryptoKeySlotMapping.verificationHash is available, the Persistency shall use this hash to verify all data related to the storage.⌋*(RS_PER_00005, RS_PER_00010)*

[SWS_PER_00468] ⌈If PersistencyDeploymentElementToCryptoKeySlotMapping.verificationHash is available, the Persistency shall use this hash to verify the element data.⌋*(RS_PER_00005, RS_PER_00010)*

The Persistency will use the services of the Cryptography [5] for encryption and decryption and for creating and verifying signed hashes. It will derive the algorithms and keys to be used from the CryptoKeySlot referenced by PersistencyDeploymentToCryptoKeySlotMapping or PersistencyDeploymentElementToCryptoKeySlotMapping, and will use them for the access to the Cryptography.

7.2.5 Redundancy Concepts

The Persistency shall take care of the integrity of the stored data, both for safety purposes and to prevent data loss. This can be achieved by calculating CRCs or hash values of the stored data, and by creating redundant copies. All these measures effectively create some redundancy for the stored data. The concrete measures to be taken are configurable: The application designer can use PersistencyInterface.redundancy to request redundancy (by setting it to redundant or redundantPerElement), or use PersistencyInterface.redundancyHandling to preselect the actual measures to be taken. During deployment, the integrator can define the actual measures taken to ensure data integrity using PersistencyDeployment.redundancyHandling. If PersistencyInterface.redundancyHandling is configured, the integrator shall use it as a guidance, but may also choose other, more appropriate measures based on superior knowledge of the final system.

[SWS_PER_00317] ⌈The Persistency shall store redundant information for every storage represented by a PersistencyDeployment where PersistencyDeployment.redundancyHandling is configured.⌋*(RS_PER_00008, RS_PER_00009, RS_PER_00010)*

The actual handling of the redundancy configured during deployment is described in the following sections, see also [SWS_PER_00318], [SWS_PER_00319], and [SWS_PER_00447].

[SWS_PER_00221] ⌈Persistency shall check the redundant data when accessing stored data. When the stored data is corrupted, Persistency shall try to restore it using the available redundancy. If Persistency is not able to recover using the redundancy, it shall report kValidationFailed.⌋*(RS_PER_00008)*

Depending on the actual implementation, Persistency might access the stored data at different times, e.g. when ara::core::Initialize is called, when a Key-Value Storage is opened, or when a file is accessed. The question whether the redundancy is sufficient for recovery is also implementation specific and can only be safely assumed for M out of N.

When the recovery failed, the application can choose to use ara::per::RecoverKeyValueStorage, ara::per::KeyValueStorage::RecoverKey, ara::per::RecoverAllFiles, or ara::per::FileStorage::RecoverFile to recover as much as possible and set the corresponding Key-Value Storage or File Storage again into a consistent state.

[SWS_PER_00452] ⌈When ara::per::RecoverKeyValueStorage is called, Persistency shall restore the Key-Value Storage to a consistent state, including redundancy. First, the infrastructure of the whole Key-Value Storage shall be restored, then Persistency shall try to recover all key-value pairs available in the Key-Value Storage as described in [SWS_PER_00453]. Depending on available information, the whole Key-Value Storage might be reset to the initial state as described in [SWS_PER_00456], losing all updated values of its key-value pairs, or may contain outdated key-value pairs after the operation.⌋*(RS_PER_00009)*

[SWS_PER_00453] ⌈When ara::per::KeyValueStorage::RecoverKey is called, Persistency shall try to restore the given key to a consistent state, including redundancy. Depending on available information, the key might be removed, reset to the initial value as described in [SWS_PER_00477], or might contain an outdated value after the operation.⌋*(RS_PER_00009)*

[SWS_PER_00454]{DRAFT} ⌈When ara::per::RecoverAllFiles is called, Persistency shall restore the File Storage to a consistent state, including redundancy. First, the infrastructure of the whole File Storage shall be restored as described in [SWS_PER_00478], then Persistency shall try to recover all currently available files as described in [SWS_PER_00455]. Depending on available information, the whole File Storage might be reset to the initial state, losing all updated content of its files, or may contain outdated files after the operation.⌋*(RS_PER_00009)*

[SWS_PER_00455]{DRAFT} ⌈When ara::per::FileStorage::RecoverFile is called, Persistency shall try to restore the given file to a consistent state, including redundancy. Depending on available information, the file might be removed, reset to the initial state as described in [SWS_PER_00479], or might contain outdated content after the operation.⌋*(RS_PER_00009)*

Of course the application has to validate the restored data in this case.

Or it can use ara::per::ResetKeyValueStorage, ara::per::KeyValueStorage::ResetKey, ara::per::ResetAllFiles, or ara::per::FileStorage::ResetFile to reset the corrupted item to the initial state according to the current manifest.

[SWS_PER_00456] ⌈When ara::per::ResetKeyValueStorage is called, Persistency shall reset the Key-Value Storage to the state it would have after installation of the application using the current manifest information.⌋*(RS_PER_00009)*

[SWS_PER_00477] ⌈When ara::per::KeyValueStorage::ResetKey is called, Persistency shall reset the given key to the state it would have after installation of the application using the current manifest information. If the key is not available in the manifest, the call shall fail with kInitValueNotAvailable.⌋*(RS_PER_00009)*

[SWS_PER_00478] ⌈When ara::per::ResetAllFiles is called, Persistency shall reset the File Storage to the state it would have after installation of the application using the current manifest information.⌋*(RS_PER_00009)*

[SWS_PER_00479] ⌈When ara::per::FileStorage::ResetFile is called, Persistency shall reset the given file to the state it would have after installation of the application using the current manifest information. If the file is not available in the manifest, the call shall fail with kInitValueNotAvailable.⌋*(RS_PER_00009)*

The application may want to monitor its storages for any problem detected by redundancy, even if Persistency is able to recover by itself. This might be required to e.g. get an early indication of hardware problems or for safety critical applications. This monitoring is supported by Persistency, which will trigger a callback function of the application in case of any problems with the storages. To activate this monitoring, the application has to register that callback function using ara::per::RegisterRecoveryReportCallback.

[SWS_PER_00480]{DRAFT} ⌈When ara::per::RegisterRecoveryReportCallback is called, Persistency shall register the provided function and enable reporting of redundancy problems in all storages of this application.⌋*(RS_PER_00008)*

Persistency may check redundancy at different places, e.g. when ara::core::Initialize is called, when a storage is opened, or when elements of the storage are accessed. Whenever a problem is detected with redundancy, independently of the situation in which the problem appeared or whether the problem could be handled, Persistency will inform the application about these problems via the registered callback, stating kKeyValueStorageRecovered, kKeyRecovered, kFileStorageRecovered, or kFileRecovered when recovery of a Key-Value Storage, a File Storage, a key-value pair, or a file was possible, and kKeyValueStorageRecoveryFailed, kKeyRecoveryFailed, kFileStorageRecoveryFailed, or kFileRecoveryFailed if not. The callback also reports the affected storage, the affected elements, and how many copies of these elements were affected (the latter only in case PersistencyRedundancyMOutOfN is configured).

[SWS_PER_00481]{DRAFT} ⌈When a Key-Value Storage is accessed, and a redundancy problem affecting the whole Key-Value Storage is detected that cannot be handled by Persistency (i.e. kValidationFailed is returned), Persistency shall call the registered callback with storage set to the ara::core::InstanceSpecifier of the Key-Value Storage, recoveryReportKind set to kKeyValueStorageRecoveryFailed, an empty ara::core::Vector for reportedElements, and an ara::core::Vector with the indices of the affected redundant instances of the Key-Value Storage in reportedInstances.⌋*(RS_PER_00008)*

[SWS_PER_00482]{DRAFT} ⌈When a Key-Value Storage is accessed, and a redundancy problem affecting the whole Key-Value Storage is detected that can be handled by Persistency (i.e. the operation succeeds), Persistency shall call the registered callback with storage set to the ara::core::InstanceSpecifier of the Key-Value Storage, recoveryReportKind set to kKeyValueStorageRecovered, an empty ara::core::Vector for reportedElements, and an ara::core::Vector with the indices of the affected redundant instances of the Key-Value Storage in reportedInstances.⌋*(RS_PER_00008)*

[SWS_PER_00483]{DRAFT} ⌈When a File Storage is accessed, and a redundancy problem affecting the whole File Storage is detected that cannot be handled by Persistency (i.e. kValidationFailed is returned), Persistency shall call the registered callback with storage set to the ara::core::InstanceSpecifier of the File Storage, recoveryReportKind set to kFileStorageRecoveryFailed, an empty ara::core::Vector for reportedElements, and an ara::core::Vector with the indices of the affected redundant instances of the File Storage in reportedInstances.⌋*(RS_PER_00008)*

[SWS_PER_00484]{DRAFT} ⌈When a File Storage is accessed, and a redundancy problem affecting the whole File Storage is detected that can be handled by Persistency (i.e. the operation succeeds), Persistency shall call the registered callback with storage set to the ara::core::InstanceSpecifier of the File Storage, recoveryReportKind set to kFileStorageRecovered, an empty ara::core::Vector for reportedElements, and an ara::core::Vector with the indices of the affected redundant instances of the File Storage in reportedInstances.⌋*(RS_PER_00008)*

[SWS_PER_00485]{DRAFT} ⌈When a Key-Value Storage or one of its keys is accessed, and a redundancy problem affecting a set of keys is detected that cannot be handled by Persistency (i.e. kValidationFailed is returned), Persistency shall call the registered callback with storage set to the ara::core::InstanceSpecifier of the Key-Value Storage, recoveryReportKind set to kKeyRecoveryFailed, an ara::core::Vector with the affected keys in reportedElements, and an ara::core::Vector with the indices of the affected redundant instances of the keys in reportedInstances.⌋*(RS_PER_00008)*

[SWS_PER_00486]{DRAFT} ⌈When a Key-Value Storage or one of its keys is accessed, and a redundancy problem affecting a set of keys is detected that can be handled by Persistency (i.e. the operation succeeds), Persistency shall call the registered callback with storage set to the ara::core::InstanceSpecifier of the Key-Value Storage, recoveryReportKind set to kKeyRecovered, an ara::core::Vector with the affected keys in reportedElements, and an ara::core::Vector with the indices of the affected redundant instances of the keys in reportedInstances.⌋*(RS_PER_00008)*

[SWS_PER_00487]{DRAFT} ⌈When a redundancy problem of single keys is reported according to [SWS_PER_00485] or [SWS_PER_00486], Persistency shall in general ensure that each entry in reportedElements matches an entry in reportedInstances at the same positions, the two ara::core::Vectors shall have the same size. If several instances of a key are affected, the key may appear several times in reportedElements. As an optimization, if only one key is affected, reportedElements may contain the affected key as single entry, related to all entries of reportedInstances.⌋*(RS_PER_00008)*

[SWS_PER_00488]{DRAFT} ⌈When a File Storage or one of its files is accessed, and a redundancy problem affecting a set of files is detected that cannot be handled by Persistency (i.e. kValidationFailed is returned), Persistency shall call the registered callback with storage set to the ara::core::InstanceSpecifier of the File Storage, recoveryReportKind set to kFileRecoveryFailed, an ara::core::Vector with the affected files in reportedElements, and an ara::core::Vector with the indices of the affected redundant instances of the files in reportedInstances.⌋*(RS_PER_00008)*

[SWS_PER_00489]{DRAFT} ⌈When a File Storage or one of its files is accessed, and a redundancy problem affecting a set of files is detected that can be handled by Persistency (i.e. the operation succeeds), Persistency shall call the registered callback with storage set to the ara::core::InstanceSpecifier of the File Storage, recoveryReportKind set to kFileRecovered, an ara::core::Vector with the affected files in reportedElements, and an ara::core::Vector with the indices of the affected redundant instances of the files in reportedInstances.⌋*(RS_PER_00008)*

[SWS_PER_00490]{DRAFT} ⌈When a redundancy problem of single file is reported according to [SWS_PER_00488] or [SWS_PER_00489], Persistency shall in general ensure that each entry in reportedElements matches an entry in reportedInstances at the same positions, the two ara::core::Vectors shall have the same size. If several instances of a file are affected, the file may appear several times in reportedElements. As an optimization, if only one file is affected, reportedElements may contain the affected file as single entry, related to all entries of reportedInstances.⌋*(RS_PER_00008)*

7.2.5.1 Redundancy Types

The type of redundancy that is applied by the Persistency is defined by the set of PersistencyRedundancyHandling classes aggregated as PersistencyDeployment.redundancyHandling. The level to which redundancy is applied is defined by the possible values of the PersistencyRedundancyHandlingScopeEnum, which are persistencyRedundancyHandlingScopeStorage and persistencyRedundancyHandlingScopeElement for a Key-Value Storage and its key-value pairs, or a File Storage and its files, respectively.

[SWS_PER_00318] ⌈In case a PersistencyRedundancyHandling aggregated as PersistencyDeployment.redundancyHandling is derived as PersistencyRedundancyCrc, the Persistency shall calculate a CRC value when persisting the storage or an element of the storage (depending on PersistencyDeployment.redundancyHandling.scope), and shall use this CRC to check the storage or the element when it is read back.⌋*(RS_PER_00008, RS_PER_00009, RS_PER_00010)*

[SWS_PER_00439] ⌈Persistency shall calculate the CRC value using the algorithm defined by PersistencyRedundancyCrc.algorithmFamily with the bit width defined by PersistencyRedundancyCrc.length.⌋*(RS_PER_00008, RS_PER_00009, RS_PER_00010)*

[SWS_PER_00319] ⌈In case a PersistencyRedundancyHandling aggregated as PersistencyDeployment.redundancyHandling is derived as PersistencyRedundancyMOutOfN, the Persistency shall store N copies when persisting the storage or an element of the storage (depending on PersistencyDeployment.redundancyHandling.scope), and shall check that at least M of the N copies of the storage or the element are identical when it is read back. N is defined by n, and M is defined by m.⌋*(RS_PER_00008, RS_PER_00009, RS_PER_00010)*

[SWS_PER_00447]{DRAFT} ⌈In case a PersistencyRedundancyHandling aggregated as PersistencyDeployment.redundancyHandling is derived as PersistencyRedundancyHash, the Persistency shall calculate a hash value when persisting the storage or an element of the storage (depending on PersistencyDeployment.redundancyHandling.scope), and shall use this hash value to check the storage or the element when it is read back.⌋*(RS_PER_00008, RS_PER_00009, RS_PER_00010)*

[SWS_PER_00448]{DRAFT} ⌈Persistency shall calculate the hash value using the algorithm defined by PersistencyRedundancyHash.algorithmFamily with the bit width defined by PersistencyRedundancyHash.length. If PersistencyRedundancyHash.initializationVectorLength is configured, an initialization vector of this length shall be calculated containing random data and passed to the hash algorithm.⌋*(RS_PER_00008, RS_PER_00009, RS_PER_00010)*

A possible approach to calculate the hash value and the random data would be to use the Cryptography [5]. The integration will have to take care that the configured PersistencyRedundancyHash.length and PersistencyRedundancyHash.initializationVectorLength are supported by the configured PersistencyRedundancyHash.algorithmFamily.

7.2.6 Installation and Update of Persistent Data

The Update and Configuration Management [6] handles the life cycle of Adaptive Applications with the following phases:

- Installation of new software
- Update of already installed software
- Finalization of updated software after the update succeeded
- Roll-back of updated software after the update failed
- Removal of installed software

For all these phases, persistent data needs to be handled alongside the application. The Adaptive Application may trigger this handling explicitly by calling ara::per::UpdatePersistency during the verification phase that follows the installation or update, or rely on the Persistency to do this implicitly when persistent data is accessed (ara::per::OpenKeyValueStorage/ara::per::OpenFileStorage). In both cases, the Persistency will compare the stored manifest version against the current manifest version, and perform the required action.

Persistency stores information about already installed storages together with version information in a central location.

[SWS_PER_00463]{DRAFT} ⌈Persistency shall store information about the installed Key-Value Storages and File Storages in the location denoted by ProcessToMachineMapping.persistencyCentralStorageURI of the ProcessToMachineMapping that refers to the Process that is referenced by PersistencyPortPrototypeToDeploymentMappings. It shall also store the current manifest version in this location.⌋*(RS_PER_00010, RS_PER_00012, RS_PER_00013, RS_PER_00014, RS_PER_00016)*

[SWS_PER_00469] ⌈When ara::per::UpdatePersistency is called, the Persistency shall follow [SWS_PER_00382] (for installation), [SWS_PER_00386] and [SWS_PER_00387] (for update), or [SWS_PER_00396] (for roll-back) for each storage configured as PersistencyDeployment in the deployment data.⌋*(RS_PER_00010, RS_PER_00012, RS_PER_00013, RS_PER_00014)*

[SWS_PER_00470] ⌈When a Key-Value Storage is opened by the application using ara::per::OpenKeyValueStorage, the Persistency shall follow [SWS_PER_00382] (for installation), [SWS_PER_00386] and [SWS_PER_00387] (for update), [SWS_PER_00446] (for finalization), or [SWS_PER_00396] (for roll-back) for this Key-Value Storage configured as PersistencyKeyValueStorage in the deployment data.⌋*(RS_PER_00010, RS_PER_00012, RS_PER_00013, RS_PER_00014, RS_PER_00016)*

[SWS_PER_00471] ⌈When a File Storage is opened by the application using ara::per::OpenFileStorage, the Persistency shall follow [SWS_PER_00382] (for installation), [SWS_PER_00386] and [SWS_PER_00387] (for update), [SWS_PER_00446] (for finalization), or [SWS_PER_00396] (for roll-back) for each File Storage configured as PersistencyFileStorage in the deployment data.⌋*(RS_PER_00010, RS_PER_00012, RS_PER_00013, RS_PER_00014, RS_PER_00016)*

[SWS_PER_00378] ⌈Persistency shall extract the Executable.version and the PersistencyDeployment.version from the manifest, and store them persistently in the location denoted by ProcessToMachineMapping.persistencyCentralStorageURI.⌋*(RS_PER_00010, RS_PER_00013, RS_PER_00014)*

The Executable.version is used by Persistency to detect a change of the application (see [SWS_PER_00387]), while the PersistencyDeployment.version is used to detect a change of the deployed persistent data (see [SWS_PER_00386] and [SWS_PER_00396]).

[SWS_PER_CONSTR_00001]{DRAFT} ⌈When the Executable.version is increased, the PersistencyDeployment.version needs to be increased, too.⌋*(RS_-PER_00010, RS_PER_00012)*

The PersistencyDeployment.version and Executable.version are StrongRevisionLabelStrings. These strings consists of a MajorVersion, a MinorVersion, a PatchVersion, and additional labels for pre-release version and build metadata. It is assumed that at least one of the first three will be incremented when the version is changed, while the additional labels might be arbitrary.

[SWS_PER_CONSTR_00002]{DRAFT} ⌈When the PersistencyDeployment.version or Executable.version is increased, the MajorVersion, MinorVersion, or PatchVersion have to be incremented.⌋*(RS_PER_00010, RS_PER_-00012)*

After installation of the Adaptive Application, the Persistency will install predefined persistent data from the manifest. There are different possibilities how this persistent data can be defined in the manifest:

- Persistent data can be defined by an application designer within PersistencyKeyValueStorageInterface or PersistencyFileStorageInterface.
- Persistent data that was defined by an application designer can be changed and fine-tuned by an integrator within PersistencyKeyValueStorage or PersistencyFileStorage.
- Persistent data can be directly defined by an integrator within PersistencyKeyValueStorage or PersistencyFileStorage.

[SWS_PER_00379] ⌈Elements defined in the deployment data (PersistencyDeploymentElement) shall always be preferred over the elements defined in the application design (PersistencyInterfaceElement). The latter shall only be used if the former does not exist.⌋*(RS_PER_00010, RS_PER_00012, RS_PER_00013)*

After an update of the Adaptive Application or the manifest, the Persistency will create a backup of the persistent data, and then update the existing persistent data using one of the following strategies:

- Existing persistent data is kept unchanged (keepExisting).
- Existing persistent data is replaced (overwrite).
- Existing persistent data is removed (delete).
- New persistent data is added (keepExisting and overwrite).

The update strategy can be set during application design or deployment, and can be defined for the whole Key-Value Storage or File Storage (PersistencyCollectionLevelUpdateStrategyEnum – keepExisting or delete) and for a single key-value pair or file (PersistencyElementLevelUpdateStrategyEnum – keepExisting, overwrite, or delete).

[SWS_PER_00251] ⌈An update strategy defined in the deployment data (PersistencyDeploymentElement.updateStrategy) shall always be preferred over the update strategy defined in the application design (PersistencyInterfaceElement.updateStrategy). The latter shall only be used if the former does not exist.⌋ *(RS_PER_00010, RS_PER_00012, RS_PER_00013)*

PersistencyDeployment.updateStrategy is a mandatory attribute and therefore PersistencyInterface.updateStrategy is just a recommendation for the deployment and never used by Persistency.

[SWS_PER_00380] ⌈An update strategy defined for a single element (PersistencyDeploymentElement.updateStrategy, PersistencyInterfaceElement.updateStrategy) shall always be preferred over the update strategy defined for the enclosing storage (PersistencyDeployment.updateStrategy, PersistencyInterface.updateStrategy). The latter shall only be used if the former does not exist.⌋ *(RS_PER_00010, RS_PER_00012, RS_PER_00013)*

When the update succeeded, the Update and Configuration Management will finalize the new Adaptive Application. The Persistency will free the resources allocated by the last backup when it is opened the first time after the update succeeded.

When the update failed, the Update and Configuration Management will revert to the old Adaptive Application and/or manifest. The Persistency will then replace the currently used persistent data by the backup created during the update.

Finally, when the Adaptive Application is removed, the Update and Configuration Management is responsible to remove the related persistent data as well.

7.2.6.1 Installation of Persistent Data

[SWS_PER_00382] ⌈When a storage is opened by the application, the Persistency shall check for the existence of any persistent data of this Process. If no persistent data is found, the Persistency shall initialize the persistent data.⌋*(RS_PER_00010, RS_PER_00012)*

Initialization of persistent data is described in paragraph 7.2.6.1.1 and paragraph 7.2.6.1.2.

7.2.6.1.1 Installation of Key-Value Storage

[SWS_PER_00383] ⌈Persistency shall create a Key-Value Storage for each PortPrototype typed by a PersistencyKeyValueStorageInterface that is found in the manifest of a newly installed Adaptive Application.⌋*(RS_PER_00010, RS_PER_00012)*

The Key-Value Storages created by [SWS_PER_00383] are identified at runtime by the shortName path of the PortPrototype, passed as ara::core::InstanceSpecifier to ara::per::OpenKeyValueStorage.

[SWS_PER_00252] ⌈Persistency shall create an entry in the Key-Value Storage for each PersistencyKeyValueStorageInterface.dataElement and PersistencyKeyValueStorage.keyValuePair that is found in the manifest of a newly installed or updated Adaptive Application, and for which the update strategy is not delete.⌋*(RS_PER_00010, RS_PER_00012)*

Key-Value Storage entries are identified by the key. An entry with identical key might be defined both in the PersistencyKeyValueStorageInterface as dataElement and the PersistencyKeyValueStorage as keyValuePair, in which case [SWS_PER_00379] applies. The update strategy is determined according to [SWS_PER_00251] and [SWS_PER_00380].

[SWS_PER_00253] ⌈Entries in the Key-Value Storage shall use the shortName of the PersistencyDataElement and/or PersistencyKeyValuePair as key.⌋*(RS_PER_00010, RS_PER_00012)*

[SWS_PER_00254] ⌈Entries in the Key-Value Storage shall be created with the data type defined by the CppImplementationDataType which types the PersistencyDataElement and/or by the CppImplementationDataType referenced as PersistencyKeyValuePair.valueDataType.⌋*(RS_PER_00010, RS_PER_00012)*

[SWS_PER_00384] ⌈Entries in the Key-Value Storage shall be created with the value taken from the PersistencyKeyValuePair.initValue or, if that does not exist, from the PersistencyDataRequiredComSpec.initValue.⌋*(RS_PER_00010, RS_PER_00012)*

[SWS_PER_CONSTR_00003] ⌈A manifest is not valid if the value or data type of any PersistencyKeyValuePair or PersistencyDataElement cannot be determined, or if the determined data types are conflicting.⌋*(RS_PER_00010, RS_PER_00012)*

Invalid manifests should be rejected by the tooling.

7.2.6.1.2 Installation of File Storage

[SWS_PER_00385] ⌈Persistency shall create a File Storage for each PortPrototype typed by a PersistencyFileStorageInterface that is found in the manifest of a newly installed Adaptive Application.⌋*(RS_PER_00010, RS_PER_00012)*

The File Storages created by [SWS_PER_00385] are identified at run-time by the shortName path of the PortPrototype, passed as ara::core::InstanceSpecifier to ara::per::OpenFileStorage.

[SWS_PER_00265] ⌈Persistency shall create a file in the File Storage for each PersistencyFileStorageInterface.fileElement and PersistencyFileStorage.file that is found in the manifest of a newly installed or updated Adaptive Application, and for which the update strategy is not delete.⌋*(RS_PER_00010, RS_PER_00012)*

The files within a File Storage are identified by their file name. A file with the same file name might be defined both in the PersistencyFileStorageInterface as fileElement and the PersistencyFileStorage as file, in which case [SWS_PER_00379] applies. The update strategy is determined according to [SWS_PER_00251] and [SWS_PER_00380].

[SWS_PER_00266] ⌈Files in the File Storage shall use the file name identified by PersistencyFileElement.fileName and/or PersistencyFile.fileName.⌋*(RS_PER_00010, RS_PER_00012)*

[SWS_PER_00267] ⌈Files in the File Storage shall be created with the content taken from the resource (within the installed SoftwarePackage) that is addressed by PersistencyFile.contentUri or, if that does not exist, by PersistencyFileElement.contentUri. If that does not exist either, an empty file shall be created.⌋*(RS_PER_00010, RS_PER_00012)*

[SWS_PER_CONSTR_00004] ⌈A manifest is invalid if the shortNames of a PersistencyFileElement and a PersistencyFile with the same file name differs.⌋*(RS_PER_00010, RS_PER_00012)*

Invalid manifests should be rejected by the tooling.

7.2.6.2 Update of Persistent Data

[SWS_PER_00386] ⌈When a storage is opened by the application, the Persistency shall compare the PersistencyDeployment.version in the manifest against the stored version. If the version in the manifest is higher than the stored version, the Persistency shall first create a backup of all the persistent data of this Process and then update the data.⌋*(RS_PER_00010, RS_PER_00013)*

Only one set of backup data needs to be kept at any time. When a new update is performed, old backup data could be overwritten. Update of persistent data is described in paragraph 7.2.6.2.1 and paragraph 7.2.6.2.2.

[SWS_PER_00387] ⌈If the application registered a function using ara::per::RegisterApplicationDataUpdateCallback, and if the Persistency had to update at least one of its storages according to [SWS_PER_00386], it shall compare the Executable.version in the manifest against the stored version. If the version in the manifest is higher than the stored version, the Persistency shall call the registered function for each storage that was updated according to [SWS_PER_00386].⌋*(RS_PER_00010, RS_PER_00013)*

The function registered by the application using ara::per::RegisterApplicationDataUpdateCallback can be used by the application to update elements of a storage manually. The storage is identified by the ara::core::InstanceSpecifier provided to this function. The application might then, based on the Executable.version of the stored data provided as second argument to the function, read in the stored data in the old format or with the old type, convert the data, and store it again with the new format or new type expected by the current version.

Example: Version 1 of the application stored the maximum speed in mph as uint8, but version 2 expects the maximum speed in km/h as uint16. The update callback function will then see that a Key-Value Storage from version 1 of the Executable has been updated to the current version, and can read in the old maximum speed by ara::per::KeyValueStorage::GetValue as uint8, convert it, and store it as uint16 with ara::per::KeyValueStorage::SetValue after removing the old value with ara::per::KeyValueStorage::RemoveKey.

7.2.6.2.1 Update of Key-Value Storage

[SWS_PER_00388] ⌈When a new PortPrototype typed by a PersistencyKeyValueStorageInterface is detected in an updated manifest, the Persistency shall create a Key-Value Storage as specified in [SWS_PER_00383].⌋*(RS_PER_00010, RS_PER_00013)*

[SWS_PER_00389] ⌈When a PortPrototype typed by a PersistencyKeyValueStorageInterface is missing in an updated manifest, the Persistency shall remove the corresponding Key-Value Storage.⌋*(RS_PER_00010, RS_PER_00013)*

[SWS_PER_00390] ⌈When a PersistencyKeyValueStorageInterface.dataElement and/or a PersistencyKeyValueStorage.keyValuePair with a new key is detected in an updated manifest, the Persistency shall create a new entry in the Key-Value Storage as specified in [SWS_PER_00252], [SWS_PER_00253], [SWS_PER_00254], and [SWS_PER_00384].⌋*(RS_PER_00010, RS_PER_00013)*

[SWS_PER_00391] ⌈When an existing key-value pair cannot be associated with any PersistencyKeyValueStorageInterface.dataElement or PersistencyKeyValueStorage.keyValuePair in an updated manifest, and the update strategy of the PersistencyKeyValueStorage corresponding to the Key-Value Storage is delete, the Persistency shall remove that key-value pair from the Key-Value Storage.⌋*(RS_PER_00010, RS_PER_00013)*

The update strategy is determined according to [SWS_PER_00251].

[SWS_PER_00275] ⌈When an existing key-value pair can be associated with a PersistencyKeyValueStorageInterface.dataElement or PersistencyKeyValueStorage.keyValuePair in an updated manifest, and the update strategy is overwrite, the Persistency shall replace that key-value pair with the new type and value as specified in [SWS_PER_00254] and [SWS_PER_00384].⌋*(RS_PER_00010, RS_PER_00013)*

An entry with identical key might be defined both in the PersistencyKeyValueStorageInterface and the PersistencyKeyValueStorage, in which case [SWS_PER_00379] applies. The update strategy is determined according to [SWS_PER_00251] and [SWS_PER_00380].

[SWS_PER_00277] ⌈When an existing key-value pair can be associated with a PersistencyKeyValueStorageInterface.dataElement or PersistencyKeyValueStorage.keyValuePair in an updated manifest, and the update strategy is delete, the Persistency shall remove that key-value pair from the Key-Value Storage.⌋*(RS_PER_00010, RS_PER_00013)*

Updated key-value pairs with the update strategy keepExisting will not be touched during an update. Persistency will neither check the value nor the type of the existing entry.

7.2.6.2.2 Update of File Storage

[SWS_PER_00392] ⌈When a new PortPrototype typed by a PersistencyFileStorageInterface is detected in an updated manifest, the Persistency shall create a File Storage as specified in [SWS_PER_00385].⌋*(RS_PER_00010, RS_PER_00013)*

[SWS_PER_00393] ⌈When a PortPrototype typed by a PersistencyFileStorageInterface is missing in an updated manifest, the Persistency shall remove the corresponding File Storage.⌋*(RS_PER_00010, RS_PER_00013)*

[SWS_PER_00394] ⌈When a PersistencyFileStorageInterface.fileElement and/or PersistencyFileStorage.file with a new file name is detected in an updated manifest, the Persistency shall create a new file in the File Storage as specified in [SWS_PER_00265], [SWS_PER_00266], and [SWS_PER_00267].⌋*(RS_PER_00010, RS_PER_00013)*

[SWS_PER_00395] ⌈When an existing file cannot be associated with any PersistencyFileStorageInterface.fileElement or PersistencyFileStorage.file in an updated manifest, and the update strategy of the PersistencyFileStorage corresponding to the File Storage is delete, the Persistency shall remove that file from the File Storage.⌋*(RS_PER_00010, RS_PER_00013)*

The update strategy is determined according to [SWS_PER_00251].

[SWS_PER_00281] ⌈When an existing file can be associated with a PersistencyFileStorageInterface.fileElement or PersistencyFileStorage.file in an updated manifest, and the update strategy is overwrite, the Persistency shall replace the content of that file with the new content as specified in [SWS_PER_00267].⌋*(RS_PER_00010, RS_PER_00013)*

A file with the same file name might be defined both in the PersistencyFileStorageInterface and the PersistencyFileStorage, in which case [SWS_PER_00379] applies. The update strategy is determined according to [SWS_PER_00251] and [SWS_PER_00380].

[SWS_PER_00283] ⌈When an existing file can be associated with a PersistencyFileStorageInterface.fileElement or PersistencyFileStorage.file in an updated manifest, and the update strategy is delete, the Persistency shall remove that file from the File Storage.⌋*(RS_PER_00010, RS_PER_00013)*

Updated files with the update strategy keepExisting will not be touched during an update. Persistency will not check the content of the existing file.

7.2.6.3 Finalization of Persistent Data after Successful Update

After installation and update, Persistency will usually be called with ara::per::UpdatePersistency within the verification phase of the application. When this succeeded, the application will be finalized by Update and Configuration Management and then started again in normal execution mode. In this case, Persistency should remove any backups that were created during a preceding update.

[SWS_PER_00446] ⌈When a storage is opened by the application, and ara::per::UpdatePersistency has not been called since Persistency was initialized, the Persistency shall compare the PersistencyDeployment.version in the manifest against the stored version. If the two versions are identical, the Persistency shall remove all backup data of the storage.⌋*(RS_PER_00016)*

Update of persistent data is described in subsubsection 7.2.6.2.

7.2.6.4 Roll-Back of Persistent Data after Failed Update

[SWS_PER_00396] ⌈When a storage is opened by the application, the Persistency shall compare the PersistencyDeployment.version in the manifest against the stored version. If the version in the manifest is lower than the stored version, the Persistency shall compare the version in the manifest against the version stored in backup data. If the versions match, the Persistency shall restore the backup. Otherwise, it shall remove all storages, and re-install the persistent data from the manifest.⌋*(RS_PER_00014)*

Initialization of persistent data is described in subsubsection 7.2.6.1.

7.2.6.5 Removal of Persistent Data

Persistency is not able to remove its own data when the Update and Configuration Management removes an application, because the application will not be executed in this case, and therefore Persistency does not run. On the other hand, the Update and Configuration Management may use the information in the manifest (ProcessToMachineMapping.persistencyCentralStorageURI, PersistencyFileStorage.uri, and PersistencyKeyValueStorage.uri) to obtain the locations of persistent data, and, if it has access to the locations, remove it.

7.2.7 Resource Management Concepts

The Persistency supports configuration of both an upper and a lower limit for the resources used by a Key-Value Storage or a File Storage.

The lower limit may already be defined by the application developer using PersistencyInterface.minimumSustainedSize.

During deployment, the integrator may update the lower limit using PersistencyDeployment.minimumSustainedSize and add an upper limit using PersistencyDeployment.maximumAllowedSize.

[SWS_PER_00320] ⌈The Persistency shall ensure that the space configured by PersistencyDeployment.minimumSustainedSize is always available for the storage.⌋*(RS_PER_00010, RS_PER_00011)*

One possibility to achieve this would be to initially allocate the minimum size during deployment, and never reduce the size below this value when persistent data is removed. But the implementation of the Persistency is free to chose other appropriate measures.

[SWS_PER_00321] ⌈The Persistency shall ensure that the space actually allocated by a storage never surpasses the amount configured by PersistencyDeployment.maximumAllowedSize.⌋*(RS_PER_00010, RS_PER_00011)*

This could be ensured by supervising all write accesses to persistent data. But again, the implementation of the Persistency is free to chose other appropriate measures.

The application can also poll the amount of storage space currently occupied by a complete Key-Value Storage or File Storage by using ara::per::GetCurrentKeyValueStorageSize or ara::per::GetCurrentFileStorageSize, respectively. Naturally, the returned values will not drop below a configured minimum size (PersistencyDeployment.minimumSustainedSize) or rise above a configured maximum size (PersistencyDeployment.maximumAllowedSize).

[SWS_PER_00491]{DRAFT} ⌈ara::per::GetCurrentKeyValueStorageSize shall return the total size of the storage space currently allocated to a Key-Value Storage, including administrative data (apart from data stored in ProcessToMachineMapping.persistencyCentralStorageURI), redundant data, and backup data.⌋*(RS_PER_00011)*

[SWS_PER_00492] ⌈ara::per::GetCurrentFileStorageSize shall return the total size of the storage space currently allocated to a File Storage, including administrative data (apart from data stored in ProcessToMachineMapping.persistencyCentralStorageURI), all its files, redundant data, and backup data.⌋*(RS_PER_00011)*

In addition, the application can poll the amount of storage space currently occupied by a single file using ara::per::FileStorage::GetCurrentFileSize of an open File Storage.

[SWS_PER_00493]{DRAFT} ⌈`ara::per::FileStorage::GetCurrentFileSize` shall return the current size of the the passed `file`. This size shall reflect only the data contained in the `file`. In case several redundant instances of the `file` exist, the size of the currently accessed instance shall be returned.⌋*(RS_PER_00011)*

7.3 Key-Value Storage specific Features

To access a Key-Value Storage, the application has to call ara::per::OpenKeyValueStorage with the ara::core::InstanceSpecifier derived from the manifest (a shortName path from the Executable to a PortPrototype or a mapping derived from FunctionalClusterInteractsWithFunctionalClusterMapping). This call will return an ara::per::SharedHandle of an ara::per::KeyValueStorage. The Key-Value Storage is closed when the ara::per::SharedHandle and all of its copies go out of scope, or when ara::core::Deinitialize is called.

[SWS_PER_00506] ⌈When ara::per::OpenKeyValueStorage is called, and Persistency is properly initialized as described in [SWS_PER_00408], Persistency shall create a temporary storage that provides access to the Key-Value Storage identified by the ara::core::InstanceSpecifier, and shall create and return an ara::per::SharedHandle of an ara::per::KeyValueStorage.⌋ *(RS_PER_00002)*

If ara::per::OpenKeyValueStorage is called without proper initialization, [SWS_PER_00410] applies.

All operations on a Key-Value Storage will be done in a temporary storage created during the call to ara::per::OpenKeyValueStorage, which the application can persist using ara::per::KeyValueStorage::SyncToStorage, or reset to the last stored state with ara::per::KeyValueStorage::DiscardPendingChanges.

Therefore, if the Key-Value Storage is just destructed (also implicitly when the Process terminates), the Key-Value Storage is not updated, and the next time the Key-Value Storage is accessed, the application will see the last saved state.

[SWS_PER_00331] ⌈Modifications of a Key-Value Storage that have not been persisted with a call to ara::per::KeyValueStorage::SyncToStorage shall be discarded when the Key-Value Storage is closed or the system is restarted, just as if ara::per::KeyValueStorage::DiscardPendingChanges had been called.⌋ *(RS_PER_00003)*

Changes done by any thread (using a copy of the ara::per::SharedHandle) will be immediately visible in all other threads. This also applies to ara::per::KeyValueStorage::DiscardPendingChanges, which resets the key-value pairs in all threads, and to ara::per::KeyValueStorage::SyncToStorage, which persists all changes done by any thread.

[SWS_PER_00494] ⌈When ara::per::KeyValueStorage::SyncToStorage is called, Persistency shall store all changes permanently that have been done to the Key-Value Storage since the last call to this method or since the Key-Value Storage was opened. Persistency shall also update any configured redundancy within this call.⌋ *(RS_PER_00001)*

The handling of redundancy is described in detail in subsection 7.2.5.

[SWS_PER_00495] ⌈When ara::per::KeyValueStorage::DiscardPendingChanges is called, Persistency shall reset the Key-Value Storage to the last persisted state, which is the state after the last call to ara::per::KeyValueStorage::SyncToStorage or after opening the Key-Value Storage.⌋ *(RS_PER_00001)*

Single key-value pairs of the Key-Value Storage are accessed using ara::per::KeyValueStorage::GetValue and ara::per::KeyValueStorage::SetValue. ara::per::KeyValueStorage::SetValue may also be used to create a key-value pair.

[SWS_PER_00496] ⌈When ara::per::KeyValueStorage::GetValue is called, Persistency shall first check whether the key-value pair is present in the temporary storage, and otherwise return directly with kKeyNotFound.⌋*(RS_PER_00002, RS_PER_00003)*

[SWS_PER_00497] ⌈When ara::per::KeyValueStorage::GetValue is called for an existing key-value pair, Persistency shall check whether the templated data type matches the stored data type, and otherwise return directly with kDataTypeMismatch.⌋*(RS_PER_00002, RS_PER_00003)*

[SWS_PER_00498] ⌈When ara::per::KeyValueStorage::GetValue is called for an existing key-value pair with the correct templated data type, Persistency shall return the stored value of the key-value pair, or, if the value was recently changed by ara::per::KeyValueStorage::SetValue (also in another thread), this new temporary value.⌋*(RS_PER_00002, RS_PER_00003)*

[SWS_PER_00499] ⌈When ara::per::KeyValueStorage::SetValue is called for an existing key-value pair, Persistency shall check whether the templated data type matches the stored data type, and otherwise return directly with kDataTypeMismatch.⌋*(RS_PER_00001, RS_PER_00003)*

[SWS_PER_00534] ⌈When ara::per::KeyValueStorage::SetValue is called for an existing key-value pair with the correct templated data type, Persistency shall store the new value of the key-value pair in the temporary storage.⌋*(RS_PER_00001, RS_PER_00003)*

[SWS_PER_00501] ⌈When ara::per::KeyValueStorage::SetValue is called, and the key-value pair does not exist in the temporary storage, Persistency shall create the key-value pair with the templated data type and the provided value in the temporary storage.⌋*(RS_PER_00001, RS_PER_00003)*

To remove a single key-value pair, the application may use ara::per::KeyValueStorage::RemoveKey, while ara::per::KeyValueStorage::RemoveAllKeys empties the Key-Value Storage. The type of a key-value pair may be changed by first removing it, and then creating it with the new type.

[SWS_PER_00502]{DRAFT} ⌈When ara::per::KeyValueStorage::RemoveKey is called, Persistency shall first check whether the key-value pair is present in the temporary storage, and otherwise return directly with kKeyNotFound.⌋ *(RS_PER_00001, RS_PER_00003)*

[SWS_PER_00535] ⌈When ara::per::KeyValueStorage::RemoveKey is called for an existing key-value pair, Persistency shall remove the key-value pair from the temporary storage.⌋*(RS_PER_00001, RS_PER_00003)*

[SWS_PER_00503] ⌈When ara::per::KeyValueStorage::RemoveAllKeys is called, Persistency shall remove all key-value pairs from the temporary storage, resulting in an empty Key-Value Storage.⌋*(RS_PER_00001)*

Finally, the application can check for the existence of a single key with ara::per::KeyValueStorage::KeyExists, and acquire a list of all currently available keys using ara::per::KeyValueStorage::GetAllKeys.

[SWS_PER_00504] ⌈ara::per::KeyValueStorage::KeyExists shall return true if the key is present in the temporary storage, otherwise it shall return false.⌋ *(RS_PER_00003)*

[SWS_PER_00505] ⌈ara::per::KeyValueStorage::GetAllKeys shall return an ara::core::Vector of ara::core::String, containing all the keys that are present in the temporary storage. If the temporary storage is empty, an empty ara::core::Vector shall be returned.⌋*(RS_PER_00003)*

7.3.1 Supported Data Types in Key-Value Storages

The Persistency supports the following classes of data types in the functions ara::per::KeyValueStorage::GetValue (templated via T) and ara::per::KeyValueStorage::SetValue (templated via T) of a Key-Value Storage.

[SWS_PER_00302] ⌈The Persistency shall be able to store all data types described in [10] in a Key-Value Storage.⌋*(RS_PER_00001)*

[SWS_PER_00303] ⌈The Persistency shall be able to store serialized binary data in a Key-Value Storage. Serialized binary data has to be presented as ara::core::Vector of ara::core::Byte.⌋*(RS_PER_00001)*

This allows the application to store custom data types.

[SWS_PER_00304] ⌈The Persistency shall be able to store all CppImplementationDataTypes referred via PersistencyKeyValueStorageInterface.dataTypeForSerialization or via PersistencyKeyValueStorageInterface.dataElement in the application design of a PersistencyKeyValueStorage in the corresponding Key-Value Storage. See [3].⌋*(RS_PER_00001, RS_PER_00010)*

7.4 File Storage specific Features

To access a File Storage, the application has to call ara::per::OpenFileStorage with the ara::core::InstanceSpecifier derived from the manifest (a shortName path from the Executable to a PortPrototype or a mapping derived from FunctionalClusterInteractsWithFunctionalClusterMapping). This call will return an ara::per::SharedHandle of an ara::per::FileStorage. The File Storage is closed when the ara::per::SharedHandle and all of its copies go out of scope, or when ara::core::Deinitialize is called.

[SWS_PER_00507] ⌈When ara::per::OpenFileStorage is called, and Persistency is properly initialized as described in [SWS_PER_00408], Persistency shall create the necessary structures to access the File Storage identified by the ara::core::InstanceSpecifier, and create and return an ara::per::SharedHandle of an ara::per::FileStorage.⌋*(RS_PER_00004)*

If ara::per::OpenFileStorage is called without proper initialization, [SWS_PER_00410] applies.

To check for the existence of a single file, the application may call ara::per::FileStorage::FileExists, and ara::per::FileStorage::GetAllFileNames will return a list of all currently available files of the File Storage.

[SWS_PER_00508] ⌈ara::per::FileStorage::FileExists shall return true if the file is present in the File Storage, otherwise it shall return false.⌋*(RS_PER_00004)*

[SWS_PER_00509] ⌈ara::per::FileStorage::GetAllFileNames shall return an ara::core::Vector of ara::core::String, containing the file names of all the files that are present in the File Storage. If the File Storage is empty, an empty ara::core::Vector shall be returned.⌋*(RS_PER_00004)*

Files may be have been installed with the application or may have been created during an update. To create new files, the application may use ara::per::FileStorage::OpenFileReadWrite or ara::per::FileStorage::OpenFileWriteOnly, and it can use ara::per::FileStorage::DeleteFile to remove any file.

[SWS_PER_00510]{DRAFT} ⌈When ara::per::FileStorage::DeleteFile is called, Persistency shall first check whether the file is present in the File Storage, and otherwise return directly with kFileNotFound.⌋*(RS_PER_00004)*

[SWS_PER_00511] ⌈When ara::per::FileStorage::DeleteFile is called for an existing file, Persistency shall remove the file from the File Storage.⌋*(RS_PER_00004)*

To access a file of a File Storage, the application has to call ara::per::FileStorage::OpenFileReadWrite, ara::per::FileStorage::OpenFileReadOnly, or ara::per::FileStorage::OpenFileWriteOnly with the file name of the file. These calls will return an ara::per::UniqueHandle of an ara::per::ReadAccessor or ara::per::ReadWriteAccessor.

[SWS_PER_00512] ⌈When ara::per::FileStorage::OpenFileReadOnly (or one of the overloaded versions ara::per::FileStorage::OpenFileReadOnly with ara::per::OpenMode or ara::per::FileStorage::OpenFileReadOnly with ara::per::OpenMode and separate buffer) is called, Persistency shall create the necessary structures to access the file identified by the file name, and create and return an ara::per::UniqueHandle of an ara::per::ReadAccessor.⌋*(RS_PER_00004)*

[SWS_PER_00513] ⌈When ara::per::FileStorage::OpenFileReadWrite or ara::per::FileStorage::OpenFileWriteOnly (or one of their overloaded versions ara::per::FileStorage::OpenFileReadWrite with ara::per::OpenMode, ara::per::FileStorage::OpenFileReadWrite with ara::per::OpenMode and separate buffer, ara::per::FileStorage::OpenFileWriteOnly with ara::per::OpenMode, or ara::per::FileStorage::OpenFileWriteOnly with ara::per::OpenMode and separate buffer) are called, Persistency shall create the necessary structures to access the file identified by the file name, and create and return an ara::per::UniqueHandle of an ara::per::ReadWriteAccessor.⌋*(RS_PER_00004)*

The file is closed when the ara::per::UniqueHandle goes out of scope, or when ara::core::Deinitialize is called.

[SWS_PER_00457] ⌈When a file is closed, Persistency shall ensure that all changes to the file are persisted. This does not need to be done immediately like when ara::per::ReadWriteAccessor::SyncToFile is called, but may happen at a later time, latest when the file is opened again, or ara::core::Deinitialize is called.⌋*(RS_PER_00004)*

Some of the overloads of the file opening functions receive an ara::per::OpenMode as an argument. OpenModes can be combined using the operators “|” and “|=”.

[SWS_PER_00514] ⌈ara::per::operator “|” and ara::per::operator “|=” take two ara::per::OpenMode arguments and return the combined ara::per::OpenMode.⌋*(RS_PER_00004)*

All files of Persistency are implicitly readable, even when opened as "write only", which is expressed by ara::per::ReadWriteAccessor inheriting from ara::per::ReadAccessor. The ara::per::ReadAccessor class consequently also offers the methods related to file positions.

[SWS_PER_00515]{DRAFT} ⌈ara::per::ReadAccessor::SetPosition shall set the file position to the provided position. If the provided position is located outside of the current content of the file (including the position at the end of the file), ara::per::ReadAccessor::SetPosition shall keep the previous file position and return kInvalidPosition.⌋*(RS_PER_00004)*

[SWS_PER_00516]{DRAFT} ⌈ara::per::ReadAccessor::MovePosition shall move the file position to offset bytes according to the provided origin. If the new position would be located outside of the current content of the file (including the position at the end of the file), ara::per::ReadAccessor::MovePosition shall keep the previous file position and return kInvalidPosition.⌋*(RS_PER_00004)*

[SWS_PER_00517] ⌈ara::per::ReadAccessor::GetPosition shall return the current read/write position in the file. In the case of an empty file, the position shall be returned as 0.⌋*(RS_PER_00004)*

[SWS_PER_00518]{DRAFT} ⌈ara::per::ReadAccessor::IsEof shall return true if the position is the last possible position in the file, i.e. the position directly after the last character in the file, or 0 in case the file is empty.⌋*(RS_PER_00004)*

ara::per::ReadAccessor::IsEof will return true if the current position corresponds to the total file size, which can be obtained separately using ara::per::ReadAccessor::GetSize.

[SWS_PER_00519]{DRAFT} ⌈ara::per::ReadAccessor::GetSize shall return the current total size of the file.⌋*(RS_PER_00004)*

Persistency does not care whether the content of a file is text or some binary data, and therefore offers separate methods to access the file content as text or as binary data. To read content from a text file, the application may use one of the following methods of the ara::per::ReadAccessor class:

[SWS_PER_00520] ⌈ara::per::ReadAccessor::PeekChar shall return the character at the current file position without changing the position.⌋*(RS_PER_00004)*

[SWS_PER_00521] ⌈ara::per::ReadAccessor::GetChar shall return the character at the current file position and advance the position by one.⌋*(RS_PER_00004)*

[SWS_PER_00522] ⌈ara::per::ReadAccessor::ReadText shall read the text from the current position to the end of the file and return it as an ara::core::String. The position shall be set to the end of the file.⌋*(RS_PER_00004)*

[SWS_PER_00523] ⌈ara::per::ReadAccessor::ReadText shall read the n characters of text from the current position and return them as an ara::core::String. The position shall be incremented by n. In case the end of the file is reached during this operation, the available characters shall be returned, and the position shall be set to the end of the file.⌋*(RS_PER_00004)*

[SWS_PER_00524] ⌈ara::per::ReadAccessor::ReadLine shall read all characters until the delimiter (defaulting to the newline character) or the end of the file is reached, and return them as a ara::core::String. The delimiter shall not be included in the returned ara::core::String. The position shall be set to the character following the delimiter or the end of the file.⌋*(RS_PER_00004)*

All these functions return characters with a size of eight bits, which are just so-called code units in case of UTF-8, not code points. Persistency itself does not change or interpret the content of a file when accessing it in text mode. It is assumed, though, that files in the File Storage are encoded as UTF-8 (see [RS_AP_00136]; this is also in line with the constraint for StdCppImplementationDataType of category STRING in [3], see [constr_1674]). It is also assumed that line endings are handled according to UNIX conventions, i.e. just LF ("\n").

The following methods of the ara::per::ReadAccessor class can be used by an application to read binary content from a file:

[SWS_PER_00525] ⌈ara::per::ReadAccessor::PeekByte shall return the byte at the current file position without changing the position.⌋*(RS_PER_00004)*

[SWS_PER_00526] ⌈ara::per::ReadAccessor::GetByte shall return the byte at the current file position and advance the position by one.⌋*(RS_PER_00004)*

[SWS_PER_00527] ⌈ara::per::ReadAccessor::ReadBinary shall read binary data from the current position to the end of the file and return it as an ara::core::Vector of ara::core::Byte. The position shall be set to the end of the file.⌋*(RS_PER_00004)*

[SWS_PER_00528] ⌈ara::per::ReadAccessor::ReadBinary shall read the n characters of text from the current position and return them as an ara::core::Vector of ara::core::Byte. The position shall be incremented by n. In case the end of the file is reached during this operation, the available bytes shall be returned, and the position shall be set to the end of the file.⌋*(RS_PER_00004)*

To write text to files, the application may use the ara::per::ReadWriteAccessor::WriteText method or the ara::per::ReadWriteAccessor::operator "«" of the ara::per::ReadWriteAccessor class, which treat text in the same way as described above for e.g. ara::per::ReadAccessor::ReadText.

[SWS_PER_00529]{DRAFT} ⌈ara::per::ReadWriteAccessor::WriteText shall write the characters provided as ara::core::StringView to the file, overwriting current content and advancing the end of the file if necessary. The position shall be set to character following the last character that was written during this operation, or to the end of the file.⌋*(RS_PER_00004)*

[SWS_PER_00530]{DRAFT} ⌈ara::per::ReadWriteAccessor::operator "«" shall write the characters provided as ara::core::StringView to the file, overwriting current content and advancing the end of the file if necessary. The position shall be set to character following the last character that was written during this operation, or to the end of the file. If an error occurs during this operation, the file content might be partially updated and the resulting file position might not be as expected.⌋*(RS_PER_00004)*

To write binary data to a file, the application may use the method ara::per::ReadWriteAccessor::WriteBinary of the ara::per::ReadWriteAccessor class.

[SWS_PER_00531]{DRAFT} ⌈ara::per::ReadWriteAccessor::WriteBinary shall write the bytes provided as ara::core::Span of ara::core::Byte to the file, overwriting current content and advancing the end of the file if necessary. The position shall be set to byte following the last byte that was written during this operation, or to the end of the file.⌋*(RS_PER_00004)*

The application may use ara::per::ReadWriteAccessor::SetFileSize to explicitly set the file size to a defined value in order to truncate a file or to empty it. Enlarging files is not supported by ara::per::ReadWriteAccessor::SetFileSize.

[SWS_PER_00532]{DRAFT} ⌈ara::per::ReadWriteAccessor::SetFileSize shall set the file size to the provided value. The read/write position shall be set to the end of the file if the current position is higher than the new file size. If the provided value is larger than the current file size, ara::per::ReadWriteAccessor::SetFileSize shall return kInvalidSize.⌋*(RS_PER_00004)*

When the application changed a file, Persistency will ensure that these changes are persisted. This can happen at any time, and latest when the file is closed. To trigger an additional synchronization of the file content to the persistent storage, the application may call ara::per::ReadWriteAccessor::SyncToFile.

[SWS_PER_00533] ⌈When ara::per::ReadWriteAccessor::SyncToFile is called, Persistency shall start writing the content of the file to the persistent storage. The actual update of the persistent storage may still be ongoing or may not even have started when this call returns.⌋*(RS_PER_00004)*

7.4.1 Access to Additional Information about Files

To gain information about stored files, the Persistency provides the method ara::per::FileStorage::GetFileInfo. This method returns information about the time the file was created (creationTime), last modified (modificationTime), and last accessed (accessTime), and how and by whom it was created (fileCreationState) and last modified (fileModificationState).

[SWS_PER_00440] ⌈The method ara::per::FileStorage::GetFileInfo shall gather the required information into a ara::per::FileInfo struct and return it to the application.⌋*(RS_PER_00004)*

In case the Persistency uses a file system of the underlying OS, part of that information (like the creation or access time) can be obtained from the file system. This information might not be accurate if the file is currently open.

[SWS_PER_00458]{DRAFT} ⌈If creationTime, modificationTime, or accessTime are not available, they shall be set to 0.⌋*(RS_PER_00004)*

As an example, the accessTime is not available for a read-only File Storage, and would therefore be reported as “midnight 1970-01-01”.

8 API Specification

The APIs for accessing File Storages and Key-Value Storage are completely separate, and therefore divided into separate sections. Additional sections describe common functionality.

The API of Persistency is designed around the ara::per::SharedHandle and ara::per::UniqueHandle, which are returned by factory functions like ara::per::OpenKeyValueStorage or ara::per::FileStorage::OpenFileReadWrite. The classes defined in this chapter cannot be constructed directly by the Adaptive Application, and consequently the default constructors are considered to be not publicly accessible (i.e. to be deleted, private, or protected).

8.1 General Features of Persistency

8.1.1 ara::core Types

The ara::per API is based heavily on the ara::core types defined in [2].

ara::core::Result is used wherever possible, and because of this, most methods are defined as noexcept.

Consequently, in situations where memory cannot be allocated for new objects, the Persistency shall terminate the process by calling ara::core::Abort (see [2]).

8.1.2 Update and Removal of Persistent Data

The Persistency allows for updating and resetting/removing all installed Key-Value Storages and File Storages. And the application may also register a callback function that is called after the update of any Key-Value Storage and File Storage.

8.1.2.1 RegisterApplicationDataUpdateCallback

[SWS_PER_00356] ⌈

Kind:	function	
Symbol:	RegisterApplicationDataUpdateCallback(std::function< void(const ara::core::InstanceSpecifier &storage, ara::core::String version)> appDataUpdateCallback)	
Scope:	namespace ara::per	
Syntax:	`void RegisterApplicationDataUpdateCallback (std::function< void(const ara::core::InstanceSpecifier &storage, ara::core::String version)> appDataUpdateCallback) noexcept;`	
Parameters (in):	appDataUpdateCallback	The callback function to be called by Persistency after an update of persistent data took place. The function will be called with the shortName path of an updated Key-Value Storage or File Storage, and with the Executable version with which the Persistency was last accessed.
Return value:	None	
Exception Safety:	noexcept	
Thread Safety:	no	
Header file:	#include "ara/per/update.h"	
Description:	Registers an application data update callback with Persistency. The provided callback function will be called by Persistency if an update of stored application data might be necessary. This decision is based on the Executable versions. The version that last accessed Persistency is provided as an argument to the callback, as well as the InstanceSpecifier referring to the updated Key-Value Storage or File Storage. Based on this information, the application can decide which updates are actually necessary, e.g. a migration from any older version could be supported, with different steps required for each of these. The provided function will be called from the context of UpdatePersistency(), OpenKeyValue Storage(), or OpenFileStorage().	

⌋*(RS_PER_00013, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00132)*

8.1.2.2 UpdatePersistency

[SWS_PER_00357] ⌈

Kind:	function
Symbol:	UpdatePersistency()

▽

△

Scope:	namespace ara::per	
Syntax:	`ara::core::Result<void> UpdatePersistency () noexcept;`	
Return value:	ara::core::Result< void >	A Result of void. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	no	
Errors:	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails during the update operation.
	PerErrc::kIntegrityCorrupted	Returned if stored data cannot be read because the structural integrity is corrupted.
	PerErrc::kValidationFailed	Returned if the validity of stored data cannot be ensured.
	PerErrc::kEncryptionFailed	Returned if the encryption or decryption of stored data fails during the update operation.
	PerErrc::kResourceBusy	Returned if ResetPersistency is currently being executed, or if RecoverKeyValueStorage or Reset KeyValueStorage is currently being executed for any Key-Value Storage, or if RecoverAllFiles or ResetAll Files is currently being executed for any File Storage, or a SharedHandle of a Key-Value Storage or a File Storage is currently in use.
	PerErrc::kOutOfStorageSpace	Returned if the available storage space is insufficient for the update.
Header file:	#include "ara/per/update.h"	
Description:	Updates all Persistency File Storages and Key-Value Storages after a new manifest was installed. This method can be used to update the persistent data of the application during verification phase.	

⌋*(RS_PER_00013, RS_AP_00119, RS_AP_00120, RS_AP_00127, RS_AP_00128, RS_AP_00132)*

8.1.2.3 ResetPersistency

[SWS_PER_00358] ⌈

Kind:	function	
Symbol:	ResetPersistency()	
Scope:	namespace ara::per	
Syntax:	`ara::core::Result<void> ResetPersistency () noexcept;`	
Return value:	ara::core::Result< void >	A Result of void. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	no	
Errors:	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails during the reset operation.

▽

△

	PerErrc::kResourceBusy	Returned if UpdatePersistency is currently being executed, or if RecoverKeyValueStorage or Reset KeyValueStorage is currently being executed for any Key-Value Storage, or if RecoverAllFiles or ResetAll Files is currently being executed for any File Storage, or a SharedHandle of a Key-Value Storage or a File Storage is currently in use.
Header file:	#include "ara/per/update.h"	
Description:	Resets all File Storages and Key-Value Storages by entirely removing their content. The File Storages and Key-Value Storages will be re-created when OpenFileStorage or Open KeyValueStorage is called next time.	

⌋*(RS_PER_00009, RS_AP_00119, RS_AP_00120, RS_AP_00127, RS_AP_00128, RS_AP_00132)*

8.1.3 Redundancy Handling

The Persistency supports redundant storage of Key-Value Storages, File Storages, and the key-value pairs and files contained in these. An error in the stored data that can be fixed using the redundantly stored data will be implicitly fixed when the Key-Value Storage or File Storage is accessed, an error is only returned by Persistency when the redundancy fails. To be able to track whether storage errors have been fixed using the available redundancy, the application can register the following callback function.

8.1.3.1 RecoveryReportKind

[SWS_PER_00432] ⌈

Kind:	enumeration	
Symbol:	RecoveryReportKind	
Scope:	namespace ara::per	
Underlying type:	std::uint32_t	
Syntax:	`enum class RecoveryReportKind : std::uint32_t {...};`	
Values:	kKeyValueStorageRecoveryFailed= 1	A Key-Value Storage was corrupted, an insufficient number of valid copies existed. storage contains the short-name path of the Key-Value Storage, reported Elements is empty, reportedInstances contains the indices of the affected Key-Value Storage copies.
	kKeyValueStorageRecovered= 2	A Key-Value Storage was corrupted, but a sufficient number of valid copies existed. storage contains the short-name path of the Key-Value Storage, reported Elements is empty, reportedInstances contains the indices of the affected Key-Value Storage copies.
	kKeyRecoveryFailed= 3	A set of key-value pairs was corrupted, an insufficient number of valid copies existed. storage contains the short-name path of the Key-Value Storage, reportedElements contains the list of affected keys, reportedInstances contains the indices of the affected Key-Value Storage or key-value pair copies. In general, the nth key in reportedElements corresponds to the nth index in reportedInstances, i.e. a key may be reported several times if several copies are broken. In case only one key-value pair is affected, reported Elements may be provided containing just this key.
	kKeyRecovered= 4	A set of key-value pairs was corrupted, but a sufficient number of valid copies existed. storage contains the short-name path of the Key-Value Storage, reportedElements contains the list of affected keys, reportedInstances contains the indices of the affected Key-Value Storage or key-value pair copies. In general, the nth key in reportedElements corresponds to the nth index in reportedInstances, i.e. a key may be reported several times if several copies are broken. In case only one key-value pair is affected, reported Elements may be provided containing just this key.

▽

△

	kFileStorageRecoveryFailed= 5	A File Storage was corrupted, an insufficient number of valid copies existed. storage contains the short-name path of the File Storage, reported Elements is empty, reportedInstances contains the indices of the affected File Storage copies.
	kFileStorageRecovered= 6	A File Storage was corrupted, but a sufficient number of valid copies existed. storage contains the short-name path of the File Storage, reported Elements is empty, reportedInstances contains the indices of the affected File Storage copies.
	kFileRecoveryFailed= 7	A set of files was corrupted, an insufficient number of valid copies existed. storage contains the short-name path of the File Storage, reported Elements contains the list of affected file names, reportedInstances contains the indices of the affected File Storage or file copies. In general, the nth file name in reportedElements corresponds to the nth index in reportedInstances, i.e. a file name may be reported several times if several copies are broken. In case only one file is affected, reported Elements may be provided containing just this file name.
	kFileRecovered= 8	A set of files was corrupted, but a sufficient number of valid copies existed. storage contains the short-name path of the File Storage, reported Elements contains the list of affected file names, reportedInstances contains the indices of the affected File Storage or file copies. In general, the nth file name in reportedElements corresponds to the nth index in reportedInstances, i.e. a file name may be reported several times if several copies are broken. In case only one file is affected, reported Elements may be provided containing just this file name.
Header file:	#include "ara/per/recovery.h"	
Description:	Defines the reported recovery actions.	

⌋*(RS_PER_00008, RS_AP_00122)*

8.1.3.2 RegisterRecoveryReportCallback

[SWS_PER_00433] ⌈

Kind:	function
Symbol:	RegisterRecoveryReportCallback(std::function< void(const ara::core::InstanceSpecifier &storage, ara::per::RecoveryReportKind recoveryReportKind, ara::core::Vector< ara::core::String > reportedElements, ara::core::Vector< std::uint8_t > reportedInstances)> recoveryReportCallback)
Scope:	namespace ara::per
Syntax:	`void RegisterRecoveryReportCallback (std::function< void(const ara::core::InstanceSpecifier &storage, ara::per::RecoveryReportKind recoveryReportKind, ara::core::Vector< ara::core::String > reported Elements, ara::core::Vector< std::uint8_t > reportedInstances)> recoveryReportCallback) noexcept;`

▽

△

Parameters (in):	recoveryReportCallback	The callback function to be called by Persistency to report errors in the stored data that were corrected using the available redundancy. The function will be called with the shortName path of the affected Key-Value Storage or File Storage in storage and information on what has been corrected, placed in the parameters recoveryReportKind, reported Elements, and reportedInstances.
Return value:	None	
Exception Safety:	noexcept	
Thread Safety:	no	
Header file:	#include "ara/per/recovery.h"	
Description:	Register a recovery reporting callback with Persistency. This callback can be used in safety-aware applications to detect actions of the Persistency that are related to the correctness of the persisted data and the reliability of the storage.	

⌋*(RS_PER_00008, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00132)*

8.1.4 Handle Classes

This section contains the definition of the handle classes used in the API of the `Persistency`. The `ara::per::SharedHandle` (templated via `typenameT`) is used to provide shared access to either a `ara::per::KeyValueStorage` or a `ara::per::FileStorage`, while the `ara::per::UniqueHandle` (templated via `typenameT`) is used to provide non-shared access to either a `ara::per::ReadAccessor` or a `ara::per::ReadWriteAccessor` to a `File Storage`.

8.1.4.1 SharedHandle Class

[SWS_PER_00362] ⌈

Kind:	class	
Symbol:	SharedHandle	
Scope:	namespace ara::per	

Syntax:	`template <typename T>`
	`class SharedHandle final {...};`	

Template param:	typename T	–
Header file:	#include "ara/per/shared_handle.h"	
Description:	Handle to a File Storage or Key-Value Storage. A SharedHandle is returned by the functions OpenFileStorage() and OpenKeyValueStorage() and can be passed between threads as needed. It provides the abstraction that is necessary to allow thread-safe implementation of OpenFile Storage() and OpenKeyValueStorage().	

⌋*(RS_PER_00002, RS_AP_00122)*

8.1.4.1.1 SharedHandle::SharedHandle

[SWS_PER_00367] ⌈

Kind:	function	
Symbol:	SharedHandle(SharedHandle &&sh)	
Scope:	class ara::per::SharedHandle	
Syntax:	`SharedHandle (SharedHandle &&sh) noexcept;`	
Parameters (in):	sh	The SharedHandle object to be moved.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Header file:	#include "ara/per/shared_handle.h"	
Description:	Move constructor for SharedHandle. The source handle object is invalidated and cannot be used anymore. The operator bool() shall be used to check the state of a handle object before using any other operators of the handle object.	

⌋(*RS_PER_00004, RS_AP_00120, RS_AP_00121, RS_AP_00129, RS_AP_00132*)

[SWS_PER_00369] ⌈

Kind:	function	
Symbol:	SharedHandle(const SharedHandle &sh)	
Scope:	class ara::per::SharedHandle	
Syntax:	`SharedHandle (const SharedHandle &sh) noexcept;`	
Parameters (in):	sh	The SharedHandle object to be copied.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Header file:	#include "ara/per/shared_handle.h"	
Description:	Copy constructor for SharedHandle.	

⌋(*RS_PER_00004, RS_AP_00120, RS_AP_00121, RS_AP_00129, RS_AP_00132*)

8.1.4.1.2 SharedHandle::operator=

[SWS_PER_00368] ⌈

Kind:	function	
Symbol:	operator=(SharedHandle &&sh)	
Scope:	class ara::per::SharedHandle	
Syntax:	`SharedHandle& operator= (SharedHandle &&sh) &noexcept;`	
Parameters (in):	sh	The SharedHandle object to be moved.
Return value:	SharedHandle &	The moved SharedHandle object.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Header file:	#include "ara/per/shared_handle.h"	
Description:	Move assignment operator for SharedHandle. The source handle object is invalidated and cannot be used anymore. The operator bool() shall be used to check the state of a handle object before using any other operators of the handle object.	

⌋(*RS_PER_00004, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00132*)

[SWS_PER_00370] ⌈

Kind:	function	
Symbol:	operator=(const SharedHandle &sh)	
Scope:	class ara::per::SharedHandle	
Syntax:	`SharedHandle& operator= (const SharedHandle &sh) &noexcept;`	
Parameters (in):	sh	The SharedHandle object to be copied.
Return value:	SharedHandle &	The moved SharedHandle object.
Exception Safety:	noexcept	

▽

△

Thread Safety:	re-entrant
Header file:	#include "ara/per/shared_handle.h"
Description:	Copy assignment operator for SharedHandle.

⌋*(RS_PER_00004, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00132)*

8.1.4.1.3 SharedHandle::operator bool

[SWS_PER_00398] ⌈

Kind:	function
Symbol:	operator bool()
Scope:	class ara::per::SharedHandle
Syntax:	`explicit operator bool () const noexcept;`
Exception Safety:	noexcept
Thread Safety:	re-entrant
Header file:	#include "ara/per/shared_handle.h"
Description:	Handle state. True if the handle represents a valid object of the templated class, False if the handle is empty (e.g. after a move operation). Using other operators than bool() of an empty handle will result in undefined behavior.

⌋*(RS_PER_00001, RS_PER_00002, RS_PER_00003, RS_AP_00119, RS_AP_-00129, RS_AP_00132)*

8.1.4.1.4 SharedHandle::Operator->

[SWS_PER_00363] ⌈

Kind:	function	
Symbol:	operator->()	
Scope:	class ara::per::SharedHandle	
Syntax:	`T* operator-> () noexcept;`	
Return value:	T *	–
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Header file:	#include "ara/per/shared_handle.h"	
Description:	Non-constant arrow operator.	

⌋*(RS_PER_00001, RS_PER_00002, RS_PER_00003, RS_AP_00119, RS_AP_-00129, RS_AP_00132)*

[SWS_PER_00364] ⌈

Kind:	function	
Symbol:	operator->()	
Scope:	class ara::per::SharedHandle	
Syntax:	`const T* operator-> () const noexcept;`	
Return value:	const T *	–
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Header file:	#include "ara/per/shared_handle.h"	
Description:	Constant arrow operator.	

⌋*(RS_PER_00001, RS_PER_00002, RS_PER_00003, RS_AP_00119, RS_AP_-00129, RS_AP_00132)*

8.1.4.1.5 SharedHandle::Operator*

[SWS_PER_00402] ⌈

Kind:	function	
Symbol:	operator*()	
Scope:	class ara::per::SharedHandle	
Syntax:	`T& operator* () noexcept;`	
Return value:	T &	–
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Header file:	#include "ara/per/shared_handle.h"	
Description:	Non-constant dereference operator.	

⌋*(RS_PER_00001, RS_PER_00002, RS_PER_00003, RS_AP_00119, RS_AP_-00129, RS_AP_00132)*

[SWS_PER_00403] ⌈

Kind:	function	
Symbol:	operator*()	
Scope:	class ara::per::SharedHandle	
Syntax:	`const T& operator* () const noexcept;`	
Return value:	const T &	–
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Header file:	#include "ara/per/shared_handle.h"	
Description:	Constant dereference operator.	

⌋*(RS_PER_00001, RS_PER_00002, RS_PER_00003, RS_AP_00119, RS_AP_-00129, RS_AP_00132)*

8.1.4.2 UniqueHandle Class

[SWS_PER_00359] ⌈

Kind:	class	
Symbol:	UniqueHandle	
Scope:	namespace ara::per	
Syntax:	`template <typename T>` `class UniqueHandle final {...};`	
Template param:	typename T	–
Header file:	#include "ara/per/unique_handle.h"	
Description:	Handle to a ReadAccessor or ReadWriteAccessor. A UniqueHandle is returned by the functions OpenFileReadOnly(), OpenFileWriteOnly(), and OpenFileReadWrite().	

⌋*(RS_PER_00002, RS_AP_00122)*

8.1.4.2.1 UniqueHandle::UniqueHandle

[SWS_PER_00371] ⌈

Kind:	function	
Symbol:	UniqueHandle(UniqueHandle &&uh)	
Scope:	class ara::per::UniqueHandle	
Syntax:	`UniqueHandle (UniqueHandle &&uh) noexcept;`	
Parameters (in):	uh	The UniqueHandle object to be moved.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Header file:	#include "ara/per/unique_handle.h"	
Description:	Move constructor for UniqueHandle. The source handle object is invalidated and cannot be used anymore. The operator bool() shall be used to check the state of a handle object before using any other operators of the handle object.	

⌋*(RS_PER_00002, RS_AP_00120, RS_AP_00121, RS_AP_00129, RS_AP_00132)*

[SWS_PER_00373] ⌈

Kind:	function
Symbol:	UniqueHandle(const UniqueHandle &)
Scope:	class ara::per::UniqueHandle
Syntax:	`UniqueHandle (const UniqueHandle &)=delete;`
Header file:	#include "ara/per/unique_handle.h"
Description:	The copy constructor for UniqueHandle shall not be used.

⌋*(RS_PER_00002, RS_AP_00120)*

8.1.4.2.2 UniqueHandle::operator=

[SWS_PER_00372] ⌈

Kind:	function	
Symbol:	operator=(UniqueHandle &&uh)	
Scope:	class ara::per::UniqueHandle	
Syntax:	`UniqueHandle& operator= (UniqueHandle &&uh) &noexcept;`	
Parameters (in):	uh	The UniqueHandle object to be moved.
Return value:	UniqueHandle &	The moved UniqueHandle object.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Header file:	#include "ara/per/unique_handle.h"	
Description:	Move assignment operator for UniqueHandle. The source handle object is invalidated and cannot be used anymore. The operator bool() shall be used to check the state of a handle object before using any other operators of the handle object.	

⌋*(RS_PER_00002, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00132)*

[SWS_PER_00374] ⌈

Kind:	function
Symbol:	operator=(const UniqueHandle &)
Scope:	class ara::per::UniqueHandle
Syntax:	`UniqueHandle& operator= (const UniqueHandle &)=delete;`
Header file:	#include "ara/per/unique_handle.h"
Description:	The copy assignment operator for UniqueHandle shall not be used.

⌋*(RS_PER_00002, RS_AP_00120)*

8.1.4.2.3 UniqueHandle::operator bool

[SWS_PER_00399] ⌈

Kind:	function
Symbol:	operator bool()
Scope:	class ara::per::UniqueHandle
Syntax:	`explicit operator bool () const noexcept;`
Exception Safety:	noexcept
Thread Safety:	re-entrant
Header file:	#include "ara/per/unique_handle.h"

▽

△

Description:	Handle state. True if the handle represents a valid object of the templated class, False if the handle is empty (e.g. after a move operation). Using other operators than bool() of an empty handle will result in undefined behavior.

⌋*(RS_PER_00001, RS_PER_00002, RS_PER_00003, RS_AP_00119, RS_AP_-00129, RS_AP_00132)*

8.1.4.2.4 UniqueHandle::Operator->

[SWS_PER_00360] ⌈

Kind:	function	
Symbol:	operator->()	
Scope:	class ara::per::UniqueHandle	
Syntax:	`T* operator-> () noexcept;`	
Return value:	T *	–
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Header file:	#include "ara/per/unique_handle.h"	
Description:	Non-constant arrow operator.	

⌋*(RS_PER_00001, RS_PER_00002, RS_PER_00003, RS_AP_00119, RS_AP_-00129, RS_AP_00132)*

[SWS_PER_00361] ⌈

Kind:	function	
Symbol:	operator->()	
Scope:	class ara::per::UniqueHandle	
Syntax:	`const T* operator-> () const noexcept;`	
Return value:	const T *	–
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Header file:	#include "ara/per/unique_handle.h"	
Description:	Constant arrow operator.	

⌋*(RS_PER_00001, RS_PER_00002, RS_PER_00003, RS_AP_00119, RS_AP_-00129, RS_AP_00132)*

8.1.4.2.5 UniqueHandle::Operator*

[SWS_PER_00400] ⌈

Kind:	function	
Symbol:	operator*()	
Scope:	class ara::per::UniqueHandle	
Syntax:	`T& operator* () noexcept;`	
Return value:	T &	–
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Header file:	#include "ara/per/unique_handle.h"	
Description:	Non-constant dereference operator.	

⌋*(RS_PER_00001, RS_PER_00002, RS_PER_00003, RS_AP_00119, RS_AP_-00129, RS_AP_00132)*

[SWS_PER_00401] ⌈

Kind:	function	
Symbol:	operator*()	
Scope:	class ara::per::UniqueHandle	
Syntax:	`const T& operator* () const noexcept;`	
Return value:	const T &	–
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Header file:	#include "ara/per/unique_handle.h"	
Description:	Constant dereference operator.	

⌋*(RS_PER_00001, RS_PER_00002, RS_PER_00003, RS_AP_00119, RS_AP_-00129, RS_AP_00132)*

8.1.5 Errors

The Persistency implements an error handling based on `ara::core::Result`. The errors supported by the Persistency are listed in subsubsection 8.1.5.1.

8.1.5.1 PerErrc

[SWS_PER_00311] ⌈

Kind:	enumeration	
Symbol:	PerErrc	
Scope:	namespace ara::per	
Underlying type:	ara::core::ErrorDomain::CodeType	
Syntax:	`enum class PerErrc : ara::core::ErrorDomain::CodeType {...};`	
Values:	kStorageNotFound= 1	The requested Key-Value Storage or File Storage is not configured in the AUTOSAR model.
	kKeyNotFound= 2	The provided key cannot be not found in the Key-Value Storage.
	kIllegalWriteAccess= 3	Opening a file for writing or changing, or synchronizing a key-value pair failed, because the storage is configured read-only.
	kPhysicalStorageFailure= 4	An error occurred when accessing the physical storage, e.g. because of a corrupted file system or corrupted hardware, or because of insufficient access rights.
	kIntegrityCorrupted= 5	The structural integrity of the storage could not be established. This can happen when the internal structure of a Key-Value Storage or the meta data of a File Storage is corrupted.
	kValidationFailed= 6	The validation of redundancy measures failed for a single key-value pair, for the whole Key-Value Storage, for a single file, or for the whole File Storage.
	kEncryptionFailed= 7	The encryption or decryption failed for a single key-value pair, for the whole Key-Value Storage, for a single file, or for the whole File Storage.
	kDataTypeMismatch= 8	The provided data type does not match the stored data type.
	kInitValueNotAvailable= 9	The operation could not be performed because no initial value is available.
	kResourceBusy= 10	The operation could not be performed because the resource is currently busy.
	kOutOfStorageSpace= 12	The allocated storage quota was exceeded.
	kFileNotFound= 13	The requested file name cannot be not found in the File Storage.
	kInvalidPosition= 15	SetPosition tried to move to a position that is not reachable (i.e. which is smaller than zero or greater than the current size of the file).
	kIsEof= 16	The application tried to read from the end of the file or from an empty file.

▽

△

	kInvalidOpenMode= 17	Opening a file failed because the requested combination of OpenModes is invalid.
	kInvalidSize= 18	SetFileSize tried to set a new size that is bigger than the current file size.
Header file:	#include "ara/per/per_error_domain.h"	
Description:	Defines the errors for Persistency. The enumeration values 0 - 255 are reserved for AUTOSAR assigned errors, the stack provider is free to define additional errors starting from 256.	

⌋*(RS_AP_00122, RS_AP_00127)*

8.1.5.2 GetPerDomain

[SWS_PER_00352] ⌈

Kind:	function	
Symbol:	GetPerDomain()	
Scope:	namespace ara::per	
Syntax:	`constexpr const ara::core::ErrorDomain& GetPerDomain () noexcept;`	
Return value:	const ara::core::ErrorDomain &	The global PerErrorDomain object.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Header file:	#include "ara/per/per_error_domain.h"	
Description:	Returns the global PerErrorDomain object.	

⌋*(RS_AP_00119, RS_AP_00120, RS_AP_00132)*

8.1.5.3 MakeErrorCode

[SWS_PER_00351] ⌈

Kind:	function	
Symbol:	MakeErrorCode(PerErrc code, ara::core::ErrorDomain::SupportDataType data)	
Scope:	namespace ara::per	
Syntax:	`constexpr ara::core::ErrorCode MakeErrorCode (PerErrc code, ara::core::ErrorDomain::SupportDataType data) noexcept;`	
Parameters (in):	code	Error code number.
	data	Vendor defined data associated with the error.
Return value:	ara::core::ErrorCode	An ErrorCode object.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Header file:	#include "ara/per/per_error_domain.h"	

▽

△

Description:	Creates an error code.

⌋*(RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00132)*

8.1.5.4 PerException Class

[SWS_PER_00354] ⌈

Kind:	class
Symbol:	PerException
Scope:	namespace ara::per
Base class:	ara::core::Exception
Syntax:	`class PerException :  public Exception {...};`
Header file:	#include "ara/per/per_error_domain.h"
Description:	Exception type thrown by Persistency.

⌋*(RS_AP_00122, RS_AP_00127)*

8.1.5.4.1 PerException::PerException

[SWS_PER_00355] ⌈

Kind:	function	
Symbol:	PerException(ara::core::ErrorCode errorCode)	
Scope:	class ara::per::PerException	
Syntax:	`explicit PerException (ara::core::ErrorCode errorCode) noexcept;`	
Parameters (in):	errorCode	The error code.
Exception Safety:	noexcept	
Header file:	#include "ara/per/per_error_domain.h"	
Description:	Construct a new Persistency exception object containing an error code.	

⌋*(RS_AP_00120, RS_AP_00121, RS_AP_00132)*

8.1.5.5 PerErrorDomain Class

The error handling requires an `ara::core::ErrorDomain`, which can be used to check the errors returned via `ara::core::Result`.

[SWS_PER_00312] ⌈

Kind:	class
Symbol:	PerErrorDomain
Scope:	namespace ara::per
Base class:	ara::core::ErrorDomain
Syntax:	`class PerErrorDomain final : public ErrorDomain {...};`
Unique ID:	0x8000'0000'0000'0101
Header file:	#include "ara/per/per_error_domain.h"
Description:	Defines the error domain for Persistency.

⌋*(RS_AP_00122, RS_AP_00127)*

8.1.5.5.1 PerErrorDomain::Errc

[SWS_PER_00411] ⌈

Kind:	type alias
Symbol:	Errc
Scope:	class ara::per::PerErrorDomain
Derived from:	PerErrc
Syntax:	`using Errc = PerErrc;`
Header file:	#include "ara/per/per_error_domain.h"
Description:	Alias for the error code value enumeration.

⌋*(RS_AP_00122)*

8.1.5.5.2 PerErrorDomain::Exception

[SWS_PER_00412] ⌈

Kind:	type alias
Symbol:	Exception
Scope:	class ara::per::PerErrorDomain
Derived from:	PerException
Syntax:	`using Exception = PerException;`
Header file:	#include "ara/per/per_error_domain.h"
Description:	Alias for the exception base class.

⌋*(RS_AP_00122)*

8.1.5.5.3 PerErrorDomain::PerErrorDomain

[SWS_PER_00313] ⌈

Kind:	function	
Symbol:	PerErrorDomain()	
Scope:	class ara::per::PerErrorDomain	
Syntax:	`PerErrorDomain () noexcept;`	
Exception Safety:	noexcept	
Thread Safety:	no	
Header file:	#include "ara/per/per_error_domain.h"	
Description:	Creates a PerErrorDomain instance.	

⌋*(RS_AP_00119, RS_AP_00120, RS_AP_00132)*

8.1.5.5.4 PerErrorDomain::Name

[SWS_PER_00314] ⌈

Kind:	function	
Symbol:	Name()	
Scope:	class ara::per::PerErrorDomain	
Syntax:	`const char* Name () const noexcept override;`	
Return value:	const char *	The name of the error domain.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Header file:	#include "ara/per/per_error_domain.h"	
Description:	Returns the name of the error domain.	

⌋*(RS_AP_00119, RS_AP_00120, RS_AP_00132)*

8.1.5.5.5 PerErrorDomain::Message

[SWS_PER_00315] ⌈

Kind:	function	
Symbol:	Message(CodeType errorCode)	
Scope:	class ara::per::PerErrorDomain	
Syntax:	`const char* Message (CodeType errorCode) const noexcept override;`	
Parameters (in):	errorCode	The error code number.
Return value:	const char *	The message associated with the error code.
Exception Safety:	noexcept	
Thread Safety:	no	
Header file:	#include "ara/per/per_error_domain.h"	
Description:	Returns the message associated with the error code.	

⌋*(RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00132)*

8.1.5.5.6 PerErrorDomain::ThrowAsException

[SWS_PER_00350] ⌈

Kind:	function	
Symbol:	ThrowAsException(const ara::core::ErrorCode &errorCode)	
Scope:	class ara::per::PerErrorDomain	
Syntax:	`void ThrowAsException (const ara::core::ErrorCode &errorCode) const override;`	
Parameters (in):	errorCode	The error to throw.
Return value:	None	
Thread Safety:	no	
Header file:	#include "ara/per/per_error_domain.h"	
Description:	Throws the exception associated with the error code.	

⌋*(RS_AP_00120, RS_AP_00121)*

8.2 Key-Value Storage

This section lists all functions and classes that are required to operate a Key-Value Storage.

The following functions are used to get access to a Key-Value Storage, to recover as much as possible after it was corrupted, to reset it to the deployed defaults, and to get the amount of storage space allocated to the Key-Value Storage.

8.2.1 OpenKeyValueStorage

[SWS_PER_00052] ⌈

Kind:	function	
Symbol:	OpenKeyValueStorage(const ara::core::InstanceSpecifier &kvs)	
Scope:	namespace ara::per	
Syntax:	`ara::core::Result<SharedHandle<KeyValueStorage> > OpenKeyValueStorage (const ara::core::InstanceSpecifier &kvs) noexcept;`	
Parameters (in):	kvs	The shortName path of a PortPrototype typed by a PersistencyKeyValueStorageInterface.
Return value:	ara::core::Result< SharedHandle< Key ValueStorage > >	A Result containing a SharedHandle for the Key ValueStorage. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	PerErrc::kStorageNotFound	Returned if the passed InstanceSpecifier does not match any PersistencyKeyValueStorageInterface configured for this Executable.
	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kIntegrityCorrupted	Returned if stored data cannot be read because the structural integrity is corrupted.
	PerErrc::kValidationFailed	Returned if the validity of stored data cannot be ensured.
	PerErrc::kEncryptionFailed	Returned if the decryption of stored data fails.
	PerErrc::kResourceBusy	Returned if UpdatePersistency or ResetPersistency is currently being executed, or if RecoverKeyValue Storage or ResetKeyValueStorage is currently being executed for the same Key-Value Storage.
	PerErrc::kOutOfStorageSpace	Returned if the available storage space is insufficient for the added/updated values.
Header file:	#include "ara/per/key_value_storage.h"	
Description:	Opens a Key-Value Storage. OpenKeyValueStorage will fail with kResourceBusy when the Key-Value Storage is currently being modified by a call from another thread to UpdatePersistency, ResetPersistency, Recover KeyValueStorage, or ResetKeyValueStorage. Because multiple threads can access the same Key-Value Storage concurrently, the Key-Value Storage might not be closed when the SharedHandle returned by this function goes out of scope. It will only be closed when all SharedHandles that refer to the same Key-Value Storage went out of scope.	

⌋*(RS_PER_00003, RS_PER_00010, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00129, RS_AP_00132, RS_AP_00144)*

8.2.2 RecoverKeyValueStorage

[SWS_PER_00333] ⌈

Kind:	function	
Symbol:	RecoverKeyValueStorage(const ara::core::InstanceSpecifier &kvs)	
Scope:	namespace ara::per	
Syntax:	`ara::core::Result<void> RecoverKeyValueStorage (const ara::core::InstanceSpecifier &kvs) noexcept;`	
Parameters (in):	kvs	The shortName path of a PortPrototype typed by a PersistencyKeyValueStorageInterface.
Return value:	ara::core::Result< void >	A Result of void. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	PerErrc::kStorageNotFound	Returned if the passed InstanceSpecifier does not match any PersistencyKeyValueStorageInterface configured for this Executable.
	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kEncryptionFailed	Returned if the encryption of stored data fails.
	PerErrc::kResourceBusy	Returned if UpdatePersistency or ResetPersistency is currently being executed, or if ResetKeyValue Storage is currently being executed for the same Key-Value Storage, or a SharedHandle of the same Key-Value Storage is currently in use.
	PerErrc::kOutOfStorageSpace	Returned if the available storage space is insufficient for the added/updated values.
Header file:	#include "ara/per/key_value_storage.h"	
Description:	Recovers a Key-ValueStorage. RecoverKeyValueStorage allows to recover a Key-Value Storage when the redundancy checks fail. It will fail with kResourceBusy when the Key-Value Storage is currently open, or when it is modified by a call from another thread to UpdatePersistency, ResetPersistency, RecoverKey ValueStorage, or ResetKeyValueStorage. This method does a best-effort recovery of all key-value pairs. After recovery, keys might show outdated or initial value, or might be lost.	

⌋*(RS_PER_00003, RS_PER_00009, RS_PER_00010, RS_AP_00119, RS_AP_-00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00129, RS_AP_-00132)*

8.2.3 ResetKeyValueStorage

[SWS_PER_00334] ⌈

Kind:	function
Symbol:	ResetKeyValueStorage(const ara::core::InstanceSpecifier &kvs)
Scope:	namespace ara::per

▽

△

Syntax:	`ara::core::Result<void> ResetKeyValueStorage (const ara::core::InstanceSpecifier &kvs) noexcept;`	
Parameters (in):	kvs	The shortName path of a PortPrototype typed by a PersistencyKeyValueStorageInterface.
Return value:	ara::core::Result< void >	A Result of void. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	PerErrc::kStorageNotFound	Returned if the passed InstanceSpecifier does not match any PersistencyKeyValueStorageInterface configured for this Executable.
	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kEncryptionFailed	Returned if the encryption of stored data fails.
	PerErrc::kResourceBusy	Returned if UpdatePersistency or ResetPersistency is currently being executed, or if RecoverKeyValue Storage is currently being executed for the same Key-Value Storage, or a SharedHandle of the same Key-Value Storage is currently in use.
	PerErrc::kOutOfStorageSpace	Returned if the available storage space is insufficient for the added/updated values.
Header file:	#include "ara/per/key_value_storage.h"	
Description:	Resets a Key-Value Storage to the initial state. ResetKeyValueStorage allows to reset a Key-Value Storage to the initial state, containing only key-value pairs which were deployed from the manifest, with their initial values. Afterwards, the Key-Value Storage will appear as if it was newly installed from the current manifest. It will fail with kResourceBusy when the Key-Value Storage is currently open, or when it is modified by a call from another thread to UpdatePersistency, ResetPersistency, RecoverKey ValueStorage, or ResetKeyValueStorage.	

⌋*(RS_PER_00003, RS_PER_00009, RS_PER_00010, RS_AP_00119, RS_AP_-00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00129, RS_AP_-00132)*

8.2.4 GetCurrentKeyValueStorageSize

[SWS_PER_00405] ⌈

Kind:	function	
Symbol:	GetCurrentKeyValueStorageSize(const ara::core::InstanceSpecifier &kvs)	
Scope:	namespace ara::per	
Syntax:	`ara::core::Result<std::uint64_t> GetCurrentKeyValueStorageSize (const ara::core::InstanceSpecifier &kvs) noexcept;`	
Parameters (in):	kvs	The shortName path of a PortPrototype typed by a PersistencyKeyValueStorageInterface.
Return value:	ara::core::Result< std::uint64_t >	A Result containing the occupied space in bytes. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	

▽

△

Thread Safety:	re-entrant	
Errors:	PerErrc::kStorageNotFound	Returned if the passed InstanceSpecifier does not match any PersistencyKeyValueStorageInterface configured for this Executable.
	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
Header file:	#include "ara/per/key_value_storage.h"	
Description:	Returns the space in bytes currently occupied by a Key-Value Storage. The returned size includes all meta data and the space used for redundancy and backups. The returned size is only accurate if no other operation on the Key-Value Storage takes place at the same time.	

⌋*(RS_PER_00017, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00129, RS_AP_00132)*

8.2.5 KeyValueStorage Class

This section shows the methods available for a `ara::per::KeyValueStorage` object obtained from a call to `ara::per::OpenKeyValueStorage`.

[SWS_PER_00339] ⌈

Kind:	class
Symbol:	KeyValueStorage
Scope:	namespace ara::per
Syntax:	`class KeyValueStorage final {...};`
Header file:	#include "ara/per/key_value_storage.h"
Description:	The Key-Value Storage contains a set of keys with associated values.

⌋*(RS_PER_00002, RS_AP_00122, RS_AP_00146)*

8.2.5.1 KeyValueStorage::KeyValueStorage

[SWS_PER_00459]{DRAFT} ⌈

Kind:	function
Symbol:	KeyValueStorage()
Scope:	class ara::per::KeyValueStorage
Syntax:	`KeyValueStorage ()=delete;`
Header file:	#include "ara/per/key_value_storage.h"
Description:	The default constructor for KeyValueStorage shall not be used.

⌋*(RS_PER_00002, RS_AP_00120, RS_AP_00129, RS_AP_00146)*

[SWS_PER_00322] ⌈

Kind:	function
Symbol:	KeyValueStorage(KeyValueStorage &&kvs)
Scope:	class ara::per::KeyValueStorage
Syntax:	`KeyValueStorage (KeyValueStorage &&kvs)=delete;`
Header file:	#include "ara/per/key_value_storage.h"
Description:	The move constructor for KeyValueStorage shall not be used.

⌋*(RS_PER_00002, RS_AP_00120, RS_AP_00121, RS_AP_00129, RS_AP_00132)*

[SWS_PER_00324] ⌈

Kind:	function
Symbol:	KeyValueStorage(const KeyValueStorage &)
Scope:	class ara::per::KeyValueStorage
Syntax:	`KeyValueStorage (const KeyValueStorage &)=delete;`
Header file:	#include "ara/per/key_value_storage.h"
Description:	The copy constructor for KeyValueStorage shall not be used.

⌋*(RS_PER_00002, RS_AP_00120)*

8.2.5.2 KeyValueStorage::operator=

[SWS_PER_00323] ⌈

Kind:	function
Symbol:	operator=(KeyValueStorage &&kvs)
Scope:	class ara::per::KeyValueStorage
Syntax:	`KeyValueStorage& operator= (KeyValueStorage &&kvs) &=delete;`
Header file:	#include "ara/per/key_value_storage.h"
Description:	The move assignment operator for KeyValueStorage shall not be used.

⌋*(RS_PER_00002, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00132)*

[SWS_PER_00325] ⌈

Kind:	function
Symbol:	operator=(const KeyValueStorage &)
Scope:	class ara::per::KeyValueStorage
Syntax:	`KeyValueStorage& operator= (const KeyValueStorage &)=delete;`
Header file:	#include "ara/per/key_value_storage.h"
Description:	The copy assignment operator for KeyValueStorage shall not be used.

⌋*(RS_PER_00002, RS_AP_00119, RS_AP_00120)*

8.2.5.3 KeyValueStorage::~KeyValueStorage

[SWS_PER_00050] ⌈

Kind:	function	
Symbol:	~KeyValueStorage()	
Scope:	class ara::per::KeyValueStorage	
Syntax:	`~KeyValueStorage () noexcept;`	
Exception Safety:	noexcept	
Thread Safety:	no	
Header file:	#include "ara/per/key_value_storage.h"	
Description:	Destructor for KeyValueStorage.	

⌋*(RS_PER_00002, RS_AP_00120, RS_AP_00129, RS_AP_00132, RS_AP_00134)*

8.2.5.4 KeyValueStorage::GetAllKeys

[SWS_PER_00042] ⌈

Kind:	function	
Symbol:	GetAllKeys()	
Scope:	class ara::per::KeyValueStorage	
Syntax:	`ara::core::Result<ara::core::Vector<ara::core::String> > GetAllKeys () const noexcept;`	
Return value:	ara::core::Result< ara::core::Vector< ara::core::String > >	A Result containing a list of available keys. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kIntegrityCorrupted	Returned if stored data cannot be read because the structural integrity is corrupted.
	PerErrc::kValidationFailed	Returned if the validity of stored data cannot be ensured.
	PerErrc::kEncryptionFailed	Returned if the decryption of stored data fails.
Header file:	#include "ara/per/key_value_storage.h"	
Description:	Returns a list of all currently available keys of this Key-Value Storage. The list of keys is only accurate if no key-value pair is added or deleted at the same time.	

⌋*(RS_PER_00003, RS_AP_00119, RS_AP_00120, RS_AP_00127, RS_AP_00129, RS_AP_00132)*

8.2.5.5 KeyValueStorage::KeyExists

[SWS_PER_00043] ⌈

Kind:	function	
Symbol:	KeyExists(ara::core::StringView key)	
Scope:	class ara::per::KeyValueStorage	
Syntax:	`ara::core::Result<bool> KeyExists (ara::core::StringView key) const noexcept;`	
Parameters (in):	key	The key that shall be checked.
Return value:	ara::core::Result< bool >	A Result containing true if the key could be located or false if it couldn't. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kIntegrityCorrupted	Returned if stored data cannot be read because the structural integrity is corrupted.
	PerErrc::kValidationFailed	Returned if the validity of stored data cannot be ensured.
	PerErrc::kEncryptionFailed	Returned if the decryption of stored data fails.
Header file:	#include "ara/per/key_value_storage.h"	
Description:	Checks if a key-value pair exists in this Key-Value Storage. The result is only accurate if no key-value pair is added or deleted at the same time. E.g. when a key-value pair is removed in another thread directly after this function returned "true", the result is not valid anymore.	

⌋*(RS_PER_00003, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00132)*

8.2.5.6 KeyValueStorage::GetValue

[SWS_PER_00332] ⌈

Kind:	function	
Symbol:	GetValue(ara::core::StringView key)	
Scope:	class ara::per::KeyValueStorage	

Syntax:	`template <class T>`
	`ara::core::Result<T> GetValue (ara::core::StringView key) const noexcept;`	

Template param:	T	The type of the value that shall be retrieved.
Parameters (in):	key	The key to look up.
Return value:	ara::core::Result< T >	A Result containing the retrieved value. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	PerErrc::kKeyNotFound	Returned if the provided key does not exist in the Key-Value Storage.
	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.

▽

△

	PerErrc::kIntegrityCorrupted	Returned if stored data cannot be read because the structural integrity is corrupted.
	PerErrc::kValidationFailed	Returned if the validity of stored data cannot be ensured.
	PerErrc::kEncryptionFailed	Returned if the decryption of stored data fails.
	PerErrc::kDataTypeMismatch	Returned if the data type of stored value does not match the templated type.
Header file:	#include "ara/per/key_value_storage.h"	
Description:	Returns the value assigned to a key of this Key-Value Storage. GetValue may be delayed by an ongoing call from another thread to RemoveAllKeys or Discard PendingChanges, or to SetValue, RemoveKey, RecoverKey, or ResetKey for the same key-value pair.	

⌋*(RS_PER_00003, RS_PER_00010, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00129, RS_AP_00132)*

8.2.5.7 KeyValueStorage::SetValue

[SWS_PER_00046] ⌈

Kind:	function	
Symbol:	SetValue(ara::core::StringView key, const T &value)	
Scope:	class ara::per::KeyValueStorage	

Syntax:	`template <class T>`
	`ara::core::Result<void> SetValue (ara::core::StringView key, const T &value) noexcept;`	

Template param:	T	The type of the value that shall be set.
Parameters (in):	key	The key to assign the value to.
	value	The value to store.
Return value:	ara::core::Result< void >	A Result of void. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	PerErrc::kIllegalWriteAccess	Returned if the Key-Value Storage is configured as read-only.
	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kIntegrityCorrupted	Returned if stored data cannot be written because the structural integrity is corrupted.
	PerErrc::kEncryptionFailed	Returned if the encryption or decryption of stored data fails.
	PerErrc::kDataTypeMismatch	Returned if the data type of an already stored value does not match the templated type.
	PerErrc::kOutOfStorageSpace	Returned if the available storage space is insufficient for the added/updated value.
Header file:	#include "ara/per/key_value_storage.h"	

▽

△

Description:	Stores a key-value pair in this Key-Value Storage. If a value already exists and has the same data type as the new value, it is overwritten. If the new value has a different data type than the stored value, kDataTypeMismatch is returned. SetValue may be delayed by an ongoing call from another thread to RemoveAllKeys, SyncToStorage, or DiscardPendingChanges, or to SetValue, GetValue, RemoveKey, RecoverKey, or ResetKey for the same key-value pair.	

⌋*(RS_PER_00003, RS_PER_00010, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00129, RS_AP_00132)*

8.2.5.8 KeyValueStorage::RemoveKey

[SWS_PER_00047] ⌈

Kind:	function	
Symbol:	RemoveKey(ara::core::StringView key)	
Scope:	class ara::per::KeyValueStorage	
Syntax:	`ara::core::Result<void> RemoveKey (ara::core::StringView key) noexcept;`	
Parameters (in):	key	The key to be removed.
Return value:	ara::core::Result< void >	A Result of void. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	PerErrc::kKeyNotFound	Returned if the provided key does not exist in the Key-Value Storage.
	PerErrc::kIllegalWriteAccess	Returned if the Key-Value Storage is configured as read-only.
	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kIntegrityCorrupted	Returned if stored data cannot be written because the structural integrity is corrupted.
	PerErrc::kEncryptionFailed	Returned if the encryption or decryption of stored data fails.
Header file:	#include "ara/per/key_value_storage.h"	
Description:	Removes a key and the associated value from this Key-Value Storage. RemoveKey may be delayed by an ongoing call from another thread to RemoveAllKeys, SyncToStorage, or DiscardPendingChanges, or to SetValue, GetValue, RemoveKey, RecoverKey, or ResetKey for the same key-value pair.	

⌋*(RS_PER_00003, RS_PER_00010, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00129, RS_AP_00132)*

8.2.5.9 KeyValueStorage::RecoverKey

[SWS_PER_00427] ⌈

Kind:	function	
Symbol:	RecoverKey(ara::core::StringView key)	
Scope:	class ara::per::KeyValueStorage	
Syntax:	`ara::core::Result<void> RecoverKey (ara::core::StringView key) noexcept;`	
Parameters (in):	key	The key to be recovered.
Return value:	ara::core::Result< void >	A Result of void. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	PerErrc::kKeyNotFound	Returned if the provided key does not exist in the Key-Value Storage.
	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kIntegrityCorrupted	Returned if stored data cannot be written because the structural integrity is corrupted.
	PerErrc::kEncryptionFailed	Returned if the encryption or decryption of stored data fails.
	PerErrc::kOutOfStorageSpace	Returned if the available storage space is insufficient for the restored value.
Header file:	#include "ara/per/key_value_storage.h"	
Description:	Recovers a single key-value pair of this Key Value Storage. This method allows to recover a single key-value pair when the redundancy checks fail. This method does a best-effort recovery of the key-value pair. After recovery, the key-value pair might contain outdated or initial content, or might be lost. RecoverKey may be delayed by an ongoing call from another thread to RemoveAllKeys, SyncToStorage, or DiscardPendingChanges, or to SetValue, GetValue, RemoveKey, RecoverKey, or ResetKey for the same key-value pair.	

⌋*(RS_PER_00003, RS_PER_00009, RS_PER_00010, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00129, RS_AP_00132)*

8.2.5.10 KeyValueStorage::ResetKey

[SWS_PER_00426] ⌈

Kind:	function	
Symbol:	ResetKey(ara::core::StringView key)	
Scope:	class ara::per::KeyValueStorage	
Syntax:	`ara::core::Result<void> ResetKey (ara::core::StringView key) noexcept;`	
Parameters (in):	key	The key to be reset.
Return value:	ara::core::Result< void >	A Result of void. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	

▽

△

Errors:	PerErrc::kIllegalWriteAccess	Returned if the Key-Value Storage is configured as read-only.
	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kIntegrityCorrupted	Returned if stored data cannot be written because the structural integrity is corrupted.
	PerErrc::kEncryptionFailed	Returned if the encryption or decryption of stored data fails.
	PerErrc::kInitValueNotAvailable	Returned if no intitial value was configured for this key.
	PerErrc::kOutOfStorageSpace	Returned if the available storage space is insufficient for the restored value.
Header file:	#include "ara/per/key_value_storage.h"	
Description:	Resets a key of this Key-Value Storage to its initial value. ResetKey allows to reset a single key to its initial value. If the key is currently not available in the Key-Value Storage, it is re-created. Afterwards, the key-value pair will appear in both cases as if it was newly installed from the current manifest. ResetKey will fail with kInitValueNotAvailable when neither design nor deployment define an initial value for the key. ResetKey may be delayed by an ongoing call from another thread to RemoveAllKeys, SyncToStorage, or DiscardPendingChanges, or to SetValue, GetValue, RemoveKey, RecoverKey, or ResetKey for the same key-value pair.	

⌋*(RS_PER_00003, RS_PER_00009, RS_PER_00010, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00129, RS_AP_00132)*

8.2.5.11 KeyValueStorage::RemoveAllKeys

[SWS_PER_00048] ⌈

Kind:	function	
Symbol:	RemoveAllKeys()	
Scope:	class ara::per::KeyValueStorage	
Syntax:	`ara::core::Result<void> RemoveAllKeys () noexcept;`	
Return value:	ara::core::Result< void >	A Result of void. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	PerErrc::kIllegalWriteAccess	Returned if the Key-Value Storage is configured as read-only.
	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kIntegrityCorrupted	Returned if stored data cannot be written because the structural integrity is corrupted.
	PerErrc::kEncryptionFailed	Returned if the encryption or decryption of stored data fails.
Header file:	#include "ara/per/key_value_storage.h"	

▽

△

Description:	Removes all key-value pairs and associated values from this Key-Value Storage. RemoveAllKeys may be delayed by an ongoing call from another thread to RemoveAllKeys, SyncToStorage, DiscardPendingChanges, SetValue, GetValue, RemoveKey, RecoverKey, or ResetKey.

⌋*(RS_PER_00003, RS_PER_00010, RS_AP_00119, RS_AP_00120, RS_AP_00127, RS_AP_00128, RS_AP_00129, RS_AP_00132)*

8.2.5.12 KeyValueStorage::SyncToStorage

[SWS_PER_00049] ⌈

Kind:	function	
Symbol:	SyncToStorage()	
Scope:	class ara::per::KeyValueStorage	
Syntax:	`ara::core::Result<void> SyncToStorage () noexcept;`	
Return value:	ara::core::Result< void >	A Result of void. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	PerErrc::kIllegalWriteAccess	Returned if the Key-Value Storage is configured as read-only.
	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kIntegrityCorrupted	Returned if stored data cannot be written because the structural integrity is corrupted.
	PerErrc::kEncryptionFailed	Returned if the encryption of stored data fails.
	PerErrc::kOutOfStorageSpace	Returned if the available storage space is insufficient for the added/updated values.
Header file:	#include "ara/per/key_value_storage.h"	
Description:	Triggers flushing of changed key-value pairs of the Key-Value Storage to the physical storage. SyncToStorage may be delayed by an ongoing call from another thread to RemoveAllKeys, DiscardPendingChanges, SetValue, RemoveKey, RecoverKey, or ResetKey.	

⌋*(RS_PER_00002, RS_AP_00119, RS_AP_00120, RS_AP_00127, RS_AP_00128, RS_AP_00129, RS_AP_00132)*

8.2.5.13 KeyValueStorage::DiscardPendingChanges

[SWS_PER_00365] ⌈

Kind:	function
Symbol:	DiscardPendingChanges()

▽

△

Scope:	class ara::per::KeyValueStorage	
Syntax:	`ara::core::Result<void> DiscardPendingChanges () noexcept;`	
Return value:	ara::core::Result< void >	A Result of void. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kIntegrityCorrupted	Returned if stored data cannot be read because the structural integrity is corrupted.
	PerErrc::kValidationFailed	Returned if the validity of stored data cannot be ensured.
	PerErrc::kEncryptionFailed	Returned if the decryption of stored data fails.
Header file:	#include "ara/per/key_value_storage.h"	
Description:	Removes all pending changes to this Key-Value Storage since the last call to SyncToStorage() or since this Key-Value Storage was opened using OpenKeyValueStorage(). DiscardPendingChanges may be delayed by an ongoing call from another thread to RemoveAll Keys, SyncToStorage, DiscardPendingChanges, SetValue, GetValue, RemoveKey, RecoverKey, or ResetKey.	

⌋*(RS_PER_00002, RS_AP_00119, RS_AP_00120, RS_AP_00127, RS_AP_00128, RS_AP_00129, RS_AP_00132)*

8.3 File Storage

This section lists all functions and classes that are required to operate a File Storage.

The following functions are used to get access to a File Storage, to recover as much as possible after it was corrupted, to reset it to the deployed defaults, and to get the amount of storage space allocated to the File Storage. In addition, operators are present to combine the ara::per::OpenMode values passed as mode to the OpenFile* functions.

8.3.1 OpenFileStorage

[SWS_PER_00116] ⌈

Kind:	function	
Symbol:	OpenFileStorage(const ara::core::InstanceSpecifier &fs)	
Scope:	namespace ara::per	
Syntax:	`ara::core::Result<SharedHandle<FileStorage> > OpenFileStorage (const ara::core::InstanceSpecifier &fs) noexcept;`	
Parameters (in):	fs	The shortName path of a PortPrototype typed by a PersistencyFileStorageInterface.
Return value:	ara::core::Result< SharedHandle< File Storage > >	A Result containing a SharedHandle for the File Storage. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	PerErrc::kStorageNotFound	Returned if the passed InstanceSpecifier does not match any PersistencyFileStorageInterface configured for this Executable.
	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kIntegrityCorrupted	Returned if stored data cannot be read because the structural integrity is corrupted.
	PerErrc::kValidationFailed	Returned if the validity of stored data cannot be ensured.
	PerErrc::kEncryptionFailed	Returned if the decryption of stored data fails.
	PerErrc::kResourceBusy	Returned if UpdatePersistency or ResetPersistency is currently being executed, or if RecoverAllFiles or ResetAllFiles is currently being executed for the same File Storage.
	PerErrc::kOutOfStorageSpace	Returned if the available storage space is insufficient for the added/updated files.
Header file:	#include "ara/per/file_storage.h"	
Description:	Opens a File Storage. OpenFileStorage will fail with kResourceBusy when the File Storage is currently being modified by a call from another thread to UpdatePersistency, ResetPersistency, RecoverAllFiles, or ResetAllFiles. Because multiple threads can access the same File Storage concurrently, the File Storage might not be closed when the SharedHandle returned by this function goes out of scope. It will only be closed when all SharedHandles that refer to the same File Storage went out of scope.	

⌋*(RS_PER_00001, RS_PER_00004, RS_PER_00010, RS_AP_00119, RS_AP_-00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00129, RS_AP_-00132, RS_AP_00144)*

8.3.2 RecoverAllFiles

[SWS_PER_00335] ⌈

Kind:	function	
Symbol:	RecoverAllFiles(const ara::core::InstanceSpecifier &fs)	
Scope:	namespace ara::per	
Syntax:	`ara::core::Result<void> RecoverAllFiles (const ara::core::InstanceSpecifier &fs) noexcept;`	
Parameters (in):	fs	The shortName path of a PortPrototype typed by a PersistencyFileStorageInterface.
Return value:	ara::core::Result< void >	A Result of void. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	PerErrc::kStorageNotFound	Returned if the passed InstanceSpecifier does not match any PersistencyFileStorageInterface configured for this Executable.
	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kEncryptionFailed	Returned if the encryption of stored data fails.
	PerErrc::kResourceBusy	Returned if UpdatePersistency or ResetPersistency is currently being executed, or if ResetAllFiles is currently being executed for the same File Storage, or a SharedHandle of the same File Storage is currently in use.
	PerErrc::kOutOfStorageSpace	Returned if the available storage space is insufficient for the restored files.
Header file:	#include "ara/per/file_storage.h"	
Description:	Recovers a File Storage, including all files. RecoverAllFiles recovers a File Storage when the redundancy checks fail. It will fail with kResourceBusy when the File Storage is currently open, or when it is modified by a call from another thread to UpdatePersistency, ResetPersistency, RecoverAllFiles, or ResetAll Files. This method does a best-effort recovery of all files. After recovery, files might show outdated or initial content, or might be lost.	

⌋*(RS_PER_00001, RS_PER_00004, RS_PER_00009, RS_PER_00010, RS_AP_-00119, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_-00129, RS_AP_00132)*

8.3.3 ResetAllFiles

[SWS_PER_00336] ⌈

Kind:	function	
Symbol:	ResetAllFiles(const ara::core::InstanceSpecifier &fs)	
Scope:	namespace ara::per	
Syntax:	`ara::core::Result<void> ResetAllFiles (const ara::core::InstanceSpecifier &fs) noexcept;`	
Parameters (in):	fs	The shortName path of a PortPrototype typed by a PersistencyFileStorageInterface.
Return value:	ara::core::Result< void >	A Result of void. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	PerErrc::kStorageNotFound	Returned if the passed InstanceSpecifier does not match any PersistencyFileStorageInterface configured for this Executable.
	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kEncryptionFailed	Returned if the encryption of stored data fails.
	PerErrc::kResourceBusy	Returned if UpdatePersistency or ResetPersistency is currently being executed, or if RecoverAllFiles is currently being executed for the same File Storage, or a SharedHandle of the same File Storage is currently in use.
	PerErrc::kOutOfStorageSpace	Returned if the available storage space is insufficient for the restored files.
Header file:	#include "ara/per/file_storage.h"	
Description:	Resets a File Storage, including all files. ResetAllFiles resets a File Storage to the initial state, containing only the files which were deployed from the manifest, with their initial content. Afterwards, the File Storage will appear as if it was newly installed from the current manifest. It will fail with kResourceBusy when the File Storage is currently open, or when it is modified by a call from another thread to UpdatePersistency, ResetPersistency, RecoverAllFiles, or ResetAllFiles.	

⌋*(RS_PER_00001, RS_PER_00004, RS_PER_00009, RS_PER_00010, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00129, RS_AP_00132)*

8.3.4 GetCurrentFileStorageSize

[SWS_PER_00406] ⌈

Kind:	function	
Symbol:	GetCurrentFileStorageSize(const ara::core::InstanceSpecifier &fs)	
Scope:	namespace ara::per	
Syntax:	`ara::core::Result<std::uint64_t> GetCurrentFileStorageSize (const ara::core::InstanceSpecifier &fs) noexcept;`	
Parameters (in):	fs	The shortName path of a PortPrototype typed by a PersistencyFileStorageInterface.

▽

△

Return value:	ara::core::Result< std::uint64_t >	A Result containing the occupied space in bytes. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	PerErrc::kStorageNotFound	Returned if the passed InstanceSpecifier does not match any PersistencyFileStorageInterface configured for this Executable.
	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
Header file:	#include "ara/per/file_storage.h"	
Description:	Returns the space in bytes currently occupied by a File Storage. The returned size includes all meta data and the space used for redundancy and backups. The returned size is only accurate if no other operation on the File Storage takes place at the same time.	

⌋*(RS_PER_00017, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00129, RS_AP_00132)*

8.3.5 OpenMode

[SWS_PER_00147] ⌈

Kind:	enumeration	
Symbol:	OpenMode	
Scope:	namespace ara::per	
Underlying type:	std::uint32_t	
Syntax:	`enum class OpenMode : std::uint32_t {...};`	
Values:	kAtTheBeginning= 1 << 0	Sets the seek position to the beginning of the file when the file is opened. This mode cannot be combined with kAtTheEnd.
	kAtTheEnd= 1 << 1	Sets the seek position to the end of the file when the file is opened. This mode cannot be combined with kAtTheBeginning or kTruncate.
	kTruncate= 1 << 2	Removes existing content when the file is opened. This mode cannot be combined with kAtTheEnd.
	kAppend= 1 << 3	Append to the end. Always seeks to the end of the file before writing.
Header file:	#include "ara/per/file_storage.h"	
Description:	This enumeration defines how a file shall be opened. The values can be combined (using | and |=) as long as they do not contradict each other.	

⌋*(RS_PER_00003, RS_AP_00122)*

8.3.6 operator| for FileStorage::OpenMode

[SWS_PER_00144] ⌈

Kind:	function	
Symbol:	operator|(OpenMode left, OpenMode right)	
Scope:	namespace ara::per	
Syntax:	`constexpr OpenMode operator| (OpenMode left, OpenMode right);`	
Parameters (in):	left	First OpenMode modifiers.
	right	Second OpenMode modifiers.
Return value:	OpenMode	returns Merged OpenMode modifiers.
Thread Safety:	re-entrant	
Header file:	#include "ara/per/file_storage.h"	
Description:	Merges two OpenMode modifiers into one.	

⌋*(RS_PER_00001, RS_PER_00004, RS_AP_00119, RS_AP_00120, RS_AP_00121)*

8.3.7 operator|= for FileStorage::OpenMode

[SWS_PER_00434] ⌈

Kind:	function	
Symbol:	operator|=(OpenMode &left, const OpenMode &right)	
Scope:	namespace ara::per	
Syntax:	`OpenMode& operator|= (OpenMode &left, const OpenMode &right);`	
Parameters (in):	left	Left OpenMode modifiers.
	right	Right OpenMode modifiers.
Return value:	OpenMode &	returns The modified OpenMode.
Thread Safety:	re-entrant	
Header file:	#include "ara/per/file_storage.h"	
Description:	Merges an OpenMode modifier into this OpenMode.	

⌋*(RS_PER_00001, RS_PER_00004, RS_AP_00119, RS_AP_00120, RS_AP_00121)*

8.3.8 FileCreationState

[SWS_PER_00435] ⌈

Kind:	enumeration	
Symbol:	FileCreationState	
Scope:	namespace ara::per	
Underlying type:	std::uint32_t	
Syntax:	`enum class FileCreationState : std::uint32_t {...};`	
Values:	kCreatedDuringInstallion= 1	The file was created by Persistency after installation of the application or after ResetPersistency.

▽

△

	kCreatedDuringUpdate= 2	The file was created by Persistency during an update.
	kCreatedDuringReset= 3	The file was re-created due to a call to ResetFile or ResetAllFiles.
	kCreatedDuringRecovery= 4	The file was re-created by Persistency after a corruption was detected.
	kCreatedByApplication= 5	The file was created by the application.
Header file:	#include "ara/per/file_storage.h"	
Description:	This enumeration describes how and when a file was created.	

⌋*(RS_PER_00004, RS_AP_00122)*

8.3.9 FileModificationState

[SWS_PER_00436] ⌈

Kind:	enumeration	
Symbol:	FileModificationState	
Scope:	namespace ara::per	
Underlying type:	std::uint32_t	
Syntax:	`enum class FileModificationState : std::uint32_t {...};`	
Values:	kModifiedDuringUpdate= 2	The file was last modified by Persistency during an update.
	kModifiedDuringReset= 3	The file was last modified by Persistency due to a call to ResetFile or ResetAllFiles.
	kModifiedDuringRecovery= 4	The file was last modified by Persistency after a corruption was detected.
	kModifiedByApplication= 5	The file was last modified by the application.
Header file:	#include "ara/per/file_storage.h"	
Description:	This enumeration describes how and when a file was last modified.	

⌋*(RS_PER_00004, RS_AP_00122)*

8.3.10 FileInfo

[SWS_PER_00437] ⌈

Kind:	struct
Symbol:	FileInfo
Scope:	namespace ara::per
Syntax:	`struct FileInfo {...};`
Header file:	#include "ara/per/file_storage.h"
Description:	This structure contains additional information on a file returned by GetFileInfo.

⌋*(RS_PER_00004, RS_AP_00122)*

8.3.10.1 FileInfo.creationTime

[SWS_PER_00441] ⌈

Kind:	variable
Symbol:	creationTime
Scope:	struct ara::per::FileInfo
Type:	std::uint64_t
Syntax:	`std::uint64_t creationTime;`
Header file:	#include "ara/per/file_storage.h"
Description:	Time in nanoseconds since midnight 1970-01-01 UTC at which the file was created.

⌋*(RS_PER_00004)*

8.3.10.2 FileInfo.modificationTime

[SWS_PER_00442] ⌈

Kind:	variable
Symbol:	modificationTime
Scope:	struct ara::per::FileInfo
Type:	std::uint64_t
Syntax:	`std::uint64_t modificationTime;`
Header file:	#include "ara/per/file_storage.h"
Description:	Time in nanoseconds since midnight 1970-01-01 UTC at which the file was last modified.

⌋*(RS_PER_00004)*

8.3.10.3 FileInfo.accessTime

[SWS_PER_00443] ⌈

Kind:	variable
Symbol:	accessTime
Scope:	struct ara::per::FileInfo
Type:	std::uint64_t
Syntax:	`std::uint64_t accessTime;`
Header file:	#include "ara/per/file_storage.h"
Description:	Time in nanoseconds since midnight 1970-01-01 UTC at which the file was last accessed.

⌋(RS_PER_00004)

8.3.10.4 FileInfo.fileCreationState

[SWS_PER_00444] ⌈

Kind:	variable
Symbol:	fileCreationState
Scope:	struct ara::per::FileInfo
Type:	FileCreationState
Syntax:	`FileCreationState fileCreationState;`
Header file:	#include "ara/per/file_storage.h"
Description:	Information on how and by whom the file was created.

⌋(RS_PER_00004)

8.3.10.5 FileInfo.fileModificationState

[SWS_PER_00445] ⌈

Kind:	variable
Symbol:	fileModificationState
Scope:	struct ara::per::FileInfo
Type:	FileModificationState
Syntax:	`FileModificationState fileModificationState;`
Header file:	#include "ara/per/file_storage.h"
Description:	Information on how and by whom the file was last modified.

⌋(RS_PER_00004)

8.3.11 FileStorage Class

This section shows the methods available for a `ara::per::FileStorage` object obtained from a call to `ara::per::OpenFileStorage`.

[SWS_PER_00340] ⌈

Kind:	class
Symbol:	FileStorage
Scope:	namespace ara::per
Syntax:	`class FileStorage final {...};`

▽

△

Header file:	#include "ara/per/file_storage.h"
Description:	The File Storage contains a set of files identified by their file names.

⌋*(RS_PER_00004, RS_AP_00122, RS_AP_00146)*

8.3.11.1 FileStorage::FileStorage

[SWS_PER_00460]{DRAFT} ⌈

Kind:	function
Symbol:	FileStorage()
Scope:	class ara::per::FileStorage
Syntax:	`FileStorage ()=delete;`
Header file:	#include "ara/per/file_storage.h"
Description:	The default constructor for FileStorage shall not be used.

⌋*(RS_PER_00004, RS_AP_00120, RS_AP_00129, RS_AP_00146)*

[SWS_PER_00326] ⌈

Kind:	function
Symbol:	FileStorage(FileStorage &&fs)
Scope:	class ara::per::FileStorage
Syntax:	`FileStorage (FileStorage &&fs)=delete;`
Header file:	#include "ara/per/file_storage.h"
Description:	The move constructor for FileStorage shall not be used.

⌋*(RS_PER_00004, RS_AP_00120, RS_AP_00121, RS_AP_00129, RS_AP_00132)*

[SWS_PER_00328] ⌈

Kind:	function
Symbol:	FileStorage(const FileStorage &)
Scope:	class ara::per::FileStorage
Syntax:	`FileStorage (const FileStorage &)=delete;`
Header file:	#include "ara/per/file_storage.h"
Description:	The copy constructor for FileStorage shall not be used.

⌋*(RS_PER_00004, RS_AP_00120)*

8.3.11.2 FileStorage::operator=

[SWS_PER_00327] ⌈

Kind:	function
Symbol:	operator=(FileStorage &&fs)
Scope:	class ara::per::FileStorage
Syntax:	`FileStorage& operator= (FileStorage &&fs) &=delete;`
Header file:	#include "ara/per/file_storage.h"
Description:	The move assignment operator for FileStorage shall not be used.

⌋*(RS_PER_00004, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00132)*

[SWS_PER_00329] ⌈

Kind:	function
Symbol:	operator=(const FileStorage &)
Scope:	class ara::per::FileStorage
Syntax:	`FileStorage& operator= (const FileStorage &)=delete;`
Header file:	#include "ara/per/file_storage.h"
Description:	The copy assignment operator for FileStorage shall not be used.

⌋*(RS_PER_00004, RS_AP_00119, RS_AP_00120)*

8.3.11.3 FileStorage::~FileStorage

[SWS_PER_00330] ⌈

Kind:	function
Symbol:	~FileStorage()
Scope:	class ara::per::FileStorage
Syntax:	`~FileStorage () noexcept;`
Exception Safety:	noexcept
Thread Safety:	no
Header file:	#include "ara/per/file_storage.h"
Description:	Destructor for FileStorage.

⌋*(RS_PER_00004, RS_AP_00120, RS_AP_00129, RS_AP_00132, RS_AP_00134)*

8.3.11.4 FileStorage::GetAllFileNames

[SWS_PER_00110] ⌈

Kind:	function
Symbol:	GetAllFileNames()

▽

△

Scope:	class ara::per::FileStorage	
Syntax:	`ara::core::Result<ara::core::Vector<ara::core::String> > GetAllFileNames () const noexcept;`	
Return value:	ara::core::Result< ara::core::Vector< ara::core::String > >	A Result containing a list of available file names. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kIntegrityCorrupted	Returned if stored data cannot be read because the structural integrity is corrupted.
	PerErrc::kValidationFailed	Returned if the validity of stored data cannot be ensured.
	PerErrc::kEncryptionFailed	Returned if the decryption of stored data fails.
Header file:	#include "ara/per/file_storage.h"	
Description:	Returns a list of all currently available file names of this File Storage. The list of file names is only accurate if no file is added or deleted at the same time.	

⌋*(RS_PER_00001, RS_PER_00004, RS_AP_00119, RS_AP_00120, RS_AP_00127, RS_AP_00129, RS_AP_00132)*

8.3.11.5 FileStorage::DeleteFile

[SWS_PER_00111] ⌈

Kind:	function	
Symbol:	DeleteFile(ara::core::StringView fileName)	
Scope:	class ara::per::FileStorage	
Syntax:	`ara::core::Result<void> DeleteFile (ara::core::StringView fileName) noexcept;`	
Parameters (in):	fileName	File name of the file. May correspond to the PersistencyFile.fileName of a configured file.
Return value:	ara::core::Result< void >	A Result of void. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	PerErrc::kIllegalWriteAccess	Returned if the File Storage is configured as read-only.
	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kIntegrityCorrupted	Returned if stored data cannot be written because the structural integrity is corrupted.
	PerErrc::kEncryptionFailed	Returned if the encryption or decryption of stored data fails.
	PerErrc::kResourceBusy	Returned if the file is open, or if RecoverFile or ResetFile with the same file name is currently being executed.

▽

△

	PerErrc::kFileNotFound	Returned if the provided file does not exist in the File Storage.
Header file:	#include "ara/per/file_storage.h"	
Description:	Deletes a file from this File Storage. This operation will fail with kResourceBusy when the file is currently open.	

⌋*(RS_PER_00001, RS_PER_00004, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00129, RS_AP_00132)*

8.3.11.6 FileStorage::FileExists

[SWS_PER_00112] ⌈

Kind:	function	
Symbol:	FileExists(ara::core::StringView fileName)	
Scope:	class ara::per::FileStorage	
Syntax:	`ara::core::Result<bool> FileExists (ara::core::StringView fileName) const noexcept;`	
Parameters (in):	fileName	File name of the file. May correspond to the PersistencyFile.fileName of a configured file.
Return value:	ara::core::Result< bool >	A Result containing true if the file could be located or false if it couldn't. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kIntegrityCorrupted	Returned if stored data cannot be read because the structural integrity is corrupted.
	PerErrc::kValidationFailed	Returned if the validity of stored data cannot be ensured.
	PerErrc::kEncryptionFailed	Returned if the decryption of stored data fails.
Header file:	#include "ara/per/file_storage.h"	
Description:	Checks if a file exists in this File Storage. The result is only accurate if no file is added or deleted at the same time. E.g. when a file is removed in another thread directly after this function returned "true", the result is not valid anymore.	

⌋*(RS_PER_00001, RS_PER_00004, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00132)*

8.3.11.7 FileStorage::RecoverFile

[SWS_PER_00337] ⌈

Kind:	function	
Symbol:	RecoverFile(ara::core::StringView fileName)	
Scope:	class ara::per::FileStorage	
Syntax:	`ara::core::Result<void> RecoverFile (ara::core::StringView fileName) noexcept;`	
Parameters (in):	fileName	File name of the file. May correspond to the PersistencyFile.fileName of a configured file.
Return value:	ara::core::Result< void >	A Result of void. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kEncryptionFailed	Returned if the encryption or decryption of stored data fails.
	PerErrc::kResourceBusy	Returned if the file is open, or if DeleteFile or Reset File with the same file name is currently being executed.
	PerErrc::kOutOfStorageSpace	Returned if the available storage space is insufficient for the restored file.
	PerErrc::kFileNotFound	Returned if the provided file does not exist in the File Storage.
Header file:	#include "ara/per/file_storage.h"	
Description:	Recovers a file of this File Storage. This method allows to recover a single file when the redundancy checks fail. It will fail with kResourceBusy when the file is currently open. This method does a best-effort recovery of the file. After recovery, the file might show outdated or initial content, or might be lost.	

⌋*(RS_PER_00001, RS_PER_00004, RS_PER_00009, RS_AP_00119, RS_AP_-00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00129, RS_AP_-00132)*

8.3.11.8 FileStorage::ResetFile

[SWS_PER_00338] ⌈

Kind:	function	
Symbol:	ResetFile(ara::core::StringView fileName)	
Scope:	class ara::per::FileStorage	
Syntax:	`ara::core::Result<void> ResetFile (ara::core::StringView fileName) noexcept;`	
Parameters (in):	fileName	File name of the file. May correspond to the PersistencyFile.fileName of a configured file.
Return value:	ara::core::Result< void >	A Result of void. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	

▽

△

Thread Safety:	re-entrant	
Errors:	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kEncryptionFailed	Returned if the encryption or decryption of stored data fails.
	PerErrc::kInitValueNotAvailable	Returned if no intitial value was configured for this file.
	PerErrc::kResourceBusy	Returned if the file is open, or if DeleteFile or RecoverFile with the same file name is currently being executed.
	PerErrc::kOutOfStorageSpace	Returned if the available storage space is insufficient or the number of files would get larger than the configured maxNumberOfFiles when the file is restored.
Header file:	#include "ara/per/file_storage.h"	
Description:	Resets a file of this File Storage to its initial content. ResetFile allows to reset a single file to its initial content. If the file is currently not available in the File Storage, it is re-created. Afterwards, the file will appear in both cases as if it was newly installed from the current manifest. It will fail with kResourceBusy when the file is currently open, and with kInitValueNotAvailable when neither design nor deployment define an initial content for the file.	

⌋*(RS_PER_00001, RS_PER_00004, RS_PER_00009, RS_AP_00119, RS_AP_-00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00129, RS_AP_-00132)*

8.3.11.9 FileStorage::GetCurrentFileSize

[SWS_PER_00407] ⌈

Kind:	function	
Symbol:	GetCurrentFileSize(ara::core::StringView fileName)	
Scope:	class ara::per::FileStorage	
Syntax:	`ara::core::Result<std::uint64_t> GetCurrentFileSize (ara::core::StringView fileName) const noexcept;`	
Parameters (in):	fileName	File name of the file. May correspond to the PersistencyFile.fileName of a configured file.
Return value:	ara::core::Result< std::uint64_t >	A Result containing the occupied space in bytes. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kIntegrityCorrupted	Returned if stored data cannot be read because the structural integrity is corrupted.
	PerErrc::kFileNotFound	Returned if the provided file does not exist in the File Storage.
Header file:	#include "ara/per/file_storage.h"	

▽

△

Description:	Returns the space in bytes currently occupied by the content of a file of this File Storage. The returned size is only accurate if no other operation on the file takes place at the same time.

⌋*(RS_PER_00017, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00129, RS_AP_00132)*

8.3.11.10 FileStorage::GetFileInfo

[SWS_PER_00438] ⌈

Kind:	function	
Symbol:	GetFileInfo(ara::core::StringView fileName)	
Scope:	class ara::per::FileStorage	
Syntax:	`ara::core::Result<FileInfo> GetFileInfo (ara::core::StringView fileName) const noexcept;`	
Parameters (in):	fileName	File name of the file. May correspond to the PersistencyFile.fileName of a configured file.
Return value:	ara::core::Result< FileInfo >	A Result containing a FileInfo struct. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kIntegrityCorrupted	Returned if stored data cannot be read because the structural integrity is corrupted.
	PerErrc::kFileNotFound	Returned if the provided file does not exist in the File Storage.
Header file:	#include "ara/per/file_storage.h"	
Description:	Returns additional information on a file of this File Storage. The returned FileInfo struct contains information about the times when the file was created, last modified, and last accessed, and about how and by whom the file was created and last modified. The modificationTime, accessTime, and fileModificationState returned in the FileInfo are only accurate if the file is currently not open.	

⌋*(RS_PER_00004, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00129, RS_AP_00132)*

8.3.11.11 FileStorage::OpenFileReadWrite

[SWS_PER_00375] ⌈

Kind:	function
Symbol:	OpenFileReadWrite(ara::core::StringView fileName)

▽

△

Scope:	class ara::per::FileStorage	
Syntax:	`ara::core::Result<UniqueHandle<ReadWriteAccessor> > OpenFileReadWrite (ara::core::StringView fileName) noexcept;`	
Parameters (in):	fileName	File name of the file. May correspond to the PersistencyFile.fileName of a configured file.
Return value:	ara::core::Result< UniqueHandle< ReadWriteAccessor > >	A Result containing a UniqueHandle for the file. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	PerErrc::kIllegalWriteAccess	Returned if the File Storage is configured as read-only.
	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kIntegrityCorrupted	Returned if stored data cannot be read because the structural integrity is corrupted.
	PerErrc::kValidationFailed	Returned if the validity of stored data cannot be ensured.
	PerErrc::kEncryptionFailed	Returned if the decryption of stored data fails.
	PerErrc::kResourceBusy	Returned if the file is already open, or if DeleteFile, RecoverFile, or ResetFile with the same file name is currently being executed.
	PerErrc::kOutOfStorageSpace	Returned if the available storage space is insufficient or the number of files would get larger than the configured maxNumberOfFiles when the file is created.
Header file:	#include "ara/per/file_storage.h"	
Description:	Opens a file of this File Storage for reading and writing. The file is opened with the seek position set to the beginning (corresponding to kAtThe Beginning). If the file does not exist, it is created. The file will be closed when the returned UniqueHandle goes out of scope.	

⌋*(RS_PER_00001, RS_PER_00004, RS_PER_00010, RS_AP_00119, RS_AP_-00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00129, RS_AP_-00132, RS_AP_00144)*

[SWS_PER_00113] ⌈

Kind:	function	
Symbol:	OpenFileReadWrite(ara::core::StringView fileName, OpenMode mode)	
Scope:	class ara::per::FileStorage	
Syntax:	`ara::core::Result<UniqueHandle<ReadWriteAccessor> > OpenFileReadWrite (ara::core::StringView fileName, OpenMode mode) noexcept;`	
Parameters (in):	fileName	File name of the file. May correspond to the PersistencyFile.fileName of a configured file.
	mode	Mode with which the file shall be opened.
Return value:	ara::core::Result< UniqueHandle< ReadWriteAccessor > >	A Result containing a UniqueHandle for the file. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	

▽

△

Thread Safety:	re-entrant	
Errors:	PerErrc::kIllegalWriteAccess	Returned if the File Storage is configured as read-only.
	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kIntegrityCorrupted	Returned if stored data cannot be read because the structural integrity is corrupted.
	PerErrc::kValidationFailed	Returned if the validity of stored data cannot be ensured.
	PerErrc::kEncryptionFailed	Returned if the decryption of stored data fails.
	PerErrc::kResourceBusy	Returned if the file is already open, or if DeleteFile, RecoverFile, or ResetFile with the same file name is currently being executed.
	PerErrc::kOutOfStorageSpace	Returned if the available storage space is insufficient or the number of files would get larger than the configured maxNumberOfFiles when the file is created.
	PerErrc::kInvalidOpenMode	Returned if the passed mode contains an invalid combination of modes.
Header file:	#include "ara/per/file_storage.h"	
Description:	Opens a file of this File Storage for reading and writing with a defined mode. If not otherwise specified by the provided mode, the file is opened with the seek position set to the beginning (corresponding to kAtTheBeginning). If the file does not exist, it is created. The file will be closed when the returned UniqueHandle goes out of scope.	

⌋*(RS_PER_00001, RS_PER_00004, RS_PER_00010, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00129, RS_AP_00132, RS_AP_00144)*

[SWS_PER_00429] ⌈

Kind:	function	
Symbol:	OpenFileReadWrite(ara::core::StringView fileName, OpenMode mode, ara::core::Span< ara::core::Byte > buffer)	
Scope:	class ara::per::FileStorage	
Syntax:	`ara::core::Result<UniqueHandle<ReadWriteAccessor> > OpenFileReadWrite (ara::core::StringView fileName, OpenMode mode, ara::core::Span< ara::core::Byte > buffer) noexcept;`	
Parameters (in):	fileName	File name of the file. May correspond to the PersistencyFile.fileName of a configured file.
	mode	Mode with which the file shall be opened.
	buffer	Memory to be used for block-wise reading/writing.
Return value:	ara::core::Result< UniqueHandle< ReadWriteAccessor > >	A Result containing a UniqueHandle for the file. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	PerErrc::kIllegalWriteAccess	Returned if the File Storage is configured as read-only.
	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.

▽

△

	PerErrc::kIntegrityCorrupted	Returned if stored data cannot be read because the structural integrity is corrupted.
	PerErrc::kValidationFailed	Returned if the validity of stored data cannot be ensured.
	PerErrc::kEncryptionFailed	Returned if the decryption of stored data fails.
	PerErrc::kResourceBusy	Returned if the file is already open, or if DeleteFile, RecoverFile, or ResetFile with the same file name is currently being executed.
	PerErrc::kOutOfStorageSpace	Returned if the available storage space is insufficient or the number of files would get larger than the configured maxNumberOfFiles when the file is created.
	PerErrc::kInvalidOpenMode	Returned if the passed mode contains an invalid combination of modes.
Header file:	#include "ara/per/file_storage.h"	
Description:	Opens a file of this File Storage for reading and writing with a user provided buffer. If not otherwise specified by the provided mode, the file is opened with the seek position set to the beginning (corresponding to kAtTheBeginning). The provided buffer will be used by the ReadWriteAccessor to implement block-wise reading and writing to speed up multiple small accesses to the file. If the file does not exist, it is created. The file will be closed when the returned UniqueHandle goes out of scope.	

⌋*(RS_PER_00001, RS_PER_00004, RS_PER_00010, RS_AP_00119, RS_AP_-00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00129, RS_AP_-00132, RS_AP_00144)*

8.3.11.12 FileStorage::OpenFileReadOnly

[SWS_PER_00376] ⌈

Kind:	function	
Symbol:	OpenFileReadOnly(ara::core::StringView fileName)	
Scope:	class ara::per::FileStorage	
Syntax:	`ara::core::Result<UniqueHandle<ReadAccessor> > OpenFileReadOnly ( ara::core::StringView fileName) noexcept;`	
Parameters (in):	fileName	File name of the file. May correspond to the PersistencyFile.fileName of a configured file.
Return value:	ara::core::Result< UniqueHandle< ReadAccessor > >	A Result containing a UniqueHandle for the file. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kIntegrityCorrupted	Returned if stored data cannot be read because the structural integrity is corrupted.
	PerErrc::kValidationFailed	Returned if the validity of stored data cannot be ensured.

▽

△

	PerErrc::kEncryptionFailed	Returned if the decryption of stored data fails.
	PerErrc::kResourceBusy	Returned if the file is already open, or if DeleteFile, RecoverFile, or ResetFile with the same file name is currently being executed.
	PerErrc::kFileNotFound	Returned if the provided file does not exist in the File Storage.
Header file:	#include "ara/per/file_storage.h"	
Description:	Opens a file of this File Storage for reading. The file is opened with the seek position set to the beginning (corresponding to kAtThe Beginning). The file will be closed when the returned UniqueHandle goes out of scope.	

⌋*(RS_PER_00001, RS_PER_00004, RS_PER_00010, RS_AP_00119, RS_AP_-00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00129, RS_AP_-00132, RS_AP_00144)*

[SWS_PER_00114] ⌈

Kind:	function	
Symbol:	OpenFileReadOnly(ara::core::StringView fileName, OpenMode mode)	
Scope:	class ara::per::FileStorage	
Syntax:	`ara::core::Result<UniqueHandle<ReadAccessor> > OpenFileReadOnly (ara::core::StringView fileName, OpenMode mode) noexcept;`	
Parameters (in):	fileName	File name of the file. May correspond to the PersistencyFile.fileName of a configured file.
	mode	Mode with which the file shall be opened.
Return value:	ara::core::Result< UniqueHandle< ReadAccessor > >	A Result containing a UniqueHandle for the file. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kIntegrityCorrupted	Returned if stored data cannot be read because the structural integrity is corrupted.
	PerErrc::kValidationFailed	Returned if the validity of stored data cannot be ensured.
	PerErrc::kEncryptionFailed	Returned if the decryption of stored data fails.
	PerErrc::kResourceBusy	Returned if the file is already open, or if DeleteFile, RecoverFile, or ResetFile with the same file name is currently being executed.
	PerErrc::kFileNotFound	Returned if the provided file does not exist in the File Storage.
	PerErrc::kInvalidOpenMode	Returned if the passed mode contains an invalid combination of modes.
Header file:	#include "ara/per/file_storage.h"	
Description:	Opens a file of this File Storage for reading with a defined mode. If not otherwise specified by the provided mode, the file is opened with the seek position set to the beginning (corresponding to kAtTheBeginning). The file will be closed when the returned UniqueHandle goes out of scope.	

⌋*(RS_PER_00001, RS_PER_00004, RS_PER_00010, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00129, RS_AP_00132, RS_AP_00144)*

[SWS_PER_00430] ⌈

Kind:	function	
Symbol:	OpenFileReadOnly(ara::core::StringView fileName, OpenMode mode, ara::core::Span< ara::core::Byte > buffer)	
Scope:	class ara::per::FileStorage	
Syntax:	`ara::core::Result<UniqueHandle<ReadAccessor> > OpenFileReadOnly ( ara::core::StringView fileName, OpenMode mode, ara::core::Span< ara::core::Byte > buffer) noexcept;`	
Parameters (in):	fileName	File name of the file. May correspond to the PersistencyFile.fileName of a configured file.
	mode	Mode with which the file shall be opened.
	buffer	Memory to be used for block-wise reading.
Return value:	ara::core::Result< UniqueHandle< ReadAccessor > >	A Result containing a UniqueHandle for the file. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kIntegrityCorrupted	Returned if stored data cannot be read because the structural integrity is corrupted.
	PerErrc::kValidationFailed	Returned if the validity of stored data cannot be ensured.
	PerErrc::kEncryptionFailed	Returned if the decryption of stored data fails.
	PerErrc::kResourceBusy	Returned if the file is already open, or if DeleteFile, RecoverFile, or ResetFile with the same file name is currently being executed.
	PerErrc::kFileNotFound	Returned if the provided file does not exist in the File Storage.
	PerErrc::kInvalidOpenMode	Returned if the passed mode contains an invalid combination of modes.
Header file:	#include "ara/per/file_storage.h"	
Description:	Opens a file of this File Storage for reading with a user provided buffer. If not otherwise specified by the provided mode, the file is opened with the seek position set to the beginning (corresponding to kAtTheBeginning). The provided buffer will be used by the ReadAccessor to implement block-wise reading to speed up multiple small accesses to the file. The file will be closed when the returned UniqueHandle goes out of scope.	

⌋*(RS_PER_00001, RS_PER_00004, RS_PER_00010, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00129, RS_AP_00132, RS_AP_00144)*

8.3.11.13 FileStorage::OpenFileWriteOnly

[SWS_PER_00377] ⌈

Kind:	function	
Symbol:	OpenFileWriteOnly(ara::core::StringView fileName)	
Scope:	class ara::per::FileStorage	
Syntax:	`ara::core::Result<UniqueHandle<ReadWriteAccessor> > OpenFileWriteOnly (ara::core::StringView fileName) noexcept;`	
Parameters (in):	fileName	File name of the file. May correspond to the PersistencyFile.fileName of a configured file.
Return value:	ara::core::Result< UniqueHandle< ReadWriteAccessor > >	A Result containing a UniqueHandle for the file. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	PerErrc::kIllegalWriteAccess	Returned if the File Storage is configured as read-only.
	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kIntegrityCorrupted	Returned if stored data cannot be read because the structural integrity is corrupted.
	PerErrc::kValidationFailed	Returned if the validity of stored data cannot be ensured.
	PerErrc::kEncryptionFailed	Returned if the decryption of stored data fails.
	PerErrc::kResourceBusy	Returned if the file is already open, or if DeleteFile, RecoverFile, or ResetFile with the same file name is currently being executed.
	PerErrc::kOutOfStorageSpace	Returned if the available storage space is insufficient or the number of files would get larger than the configured maxNumberOfFiles when the file is created.
Header file:	#include "ara/per/file_storage.h"	
Description:	Opens a file of this File Storage for writing. The file is truncated (corresponding to kTruncate). If the file does not exist, it is created. The file will be closed when the returned UniqueHandle goes out of scope.	

⌋*(RS_PER_00001, RS_PER_00004, RS_PER_00010, RS_AP_00119, RS_AP_-00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00129, RS_AP_-00132, RS_AP_00144)*

[SWS_PER_00115] ⌈

Kind:	function	
Symbol:	OpenFileWriteOnly(ara::core::StringView fileName, OpenMode mode)	
Scope:	class ara::per::FileStorage	
Syntax:	`ara::core::Result<UniqueHandle<ReadWriteAccessor> > OpenFileWriteOnly (ara::core::StringView fileName, OpenMode mode) noexcept;`	
Parameters (in):	fileName	File name of the file. May correspond to the PersistencyFile.fileName of a configured file.
	mode	Mode with which the file shall be opened.
Return value:	ara::core::Result< UniqueHandle< ReadWriteAccessor > >	A Result containing a UniqueHandle for the file. In case of an error, it contains any of the errors defined below, or a vendor specific error.

▽

△

Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	PerErrc::kIllegalWriteAccess	Returned if the File Storage is configured as read-only.
	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kIntegrityCorrupted	Returned if stored data cannot be read because the structural integrity is corrupted.
	PerErrc::kValidationFailed	Returned if the validity of stored data cannot be ensured.
	PerErrc::kEncryptionFailed	Returned if the decryption of stored data fails.
	PerErrc::kResourceBusy	Returned if the file is already open, or if DeleteFile, RecoverFile, or ResetFile with the same file name is currently being executed.
	PerErrc::kOutOfStorageSpace	Returned if the available storage space is insufficient or the number of files would get larger than the configured maxNumberOfFiles when the file is created.
	PerErrc::kInvalidOpenMode	Returned if the passed mode contains an invalid combination of modes.
Header file:	#include "ara/per/file_storage.h"	
Description:	Opens a file of this File Storage for writing with a defined mode. If not otherwise specified by the provided mode, the file is truncated (corresponding to k Truncate). If the file does not exist, it is created. The file will be closed when the returned UniqueHandle goes out of scope.	

⌋*(RS_PER_00001, RS_PER_00004, RS_PER_00010, RS_AP_00119, RS_AP_-00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00129, RS_AP_-00132, RS_AP_00144)*

[SWS_PER_00431] ⌈

Kind:	function	
Symbol:	OpenFileWriteOnly(ara::core::StringView fileName, OpenMode mode, ara::core::Span< ara::core::Byte > buffer)	
Scope:	class ara::per::FileStorage	
Syntax:	`ara::core::Result<UniqueHandle<ReadWriteAccessor> > OpenFileWriteOnly (ara::core::StringView fileName, OpenMode mode, ara::core::Span< ara::core::Byte > buffer) noexcept;`	
Parameters (in):	fileName	File name of the file. May correspond to the PersistencyFile.fileName of a configured file.
	mode	Mode with which the file shall be opened.
	buffer	Memory to be used for block-wise writing.
Return value:	ara::core::Result< UniqueHandle< ReadWriteAccessor > >	A Result containing a UniqueHandle for the file. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Errors:	PerErrc::kIllegalWriteAccess	Returned if the File Storage is configured as read-only.

▽

△

	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kIntegrityCorrupted	Returned if stored data cannot be read because the structural integrity is corrupted.
	PerErrc::kValidationFailed	Returned if the validity of stored data cannot be ensured.
	PerErrc::kEncryptionFailed	Returned if the decryption of stored data fails.
	PerErrc::kResourceBusy	Returned if the file is already open, or if DeleteFile, RecoverFile, or ResetFile with the same file name is currently being executed.
	PerErrc::kOutOfStorageSpace	Returned if the available storage space is insufficient or the number of files would get larger than the configured maxNumberOfFiles when the file is created.
	PerErrc::kInvalidOpenMode	Returned if the passed mode contains an invalid combination of modes.
Header file:	#include "ara/per/file_storage.h"	
Description:	Opens a file of this File Storage for writing with a user provided buffer. If not otherwise specified by the provided mode, the file is truncated (corresponding to k Truncate). The provided buffer will be used by the ReadWriteAccessor to implement block-wise writing to speed up multiple small accesses to the file. If the file does not exist, it is created. The file will be closed when the returned UniqueHandle goes out of scope.	

⌋*(RS_PER_00001, RS_PER_00004, RS_PER_00010, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00129, RS_AP_00132, RS_AP_00144)*

8.3.12 Origin

[SWS_PER_00146] ⌈

Kind:	enumeration	
Symbol:	Origin	
Scope:	namespace ara::per	
Underlying type:	std::uint32_t	
Syntax:	`enum class Origin : std::uint32_t {...};`	
Values:	kBeginning= 0	Seek from the beginning of the file.
	kCurrent= 1	Seek from the current position.
	kEnd= 2	Seek from the end of the file.
Header file:	#include "ara/per/read_accessor.h"	
Description:	Specification of origin used in MovePosition.	

⌋*(RS_PER_00003, RS_AP_00122)*

8.3.13 ReadAccessor Class

This section shows the methods available for a `ara::per::ReadAccessor` object obtained from a call to `ara::per::FileStorage::OpenFileReadOnly`, and for the inheriting `ara::per::ReadWriteAccessor` object obtained from a call to `ara::per::FileStorage::OpenFileWriteOnly` or `ara::per::FileStorage::OpenFileReadWrite`.

[SWS_PER_00342] ⌈

Kind:	class
Symbol:	ReadAccessor
Scope:	namespace ara::per
Syntax:	`class ReadAccessor {...};`
Header file:	#include "ara/per/read_accessor.h"
Description:	ReadAccessor is used to read file data. It provides binary and text mode methods for checking or getting the current byte/character (PeekByte/PeekChar, GetByte/GetChar) methods for reading a section of a binary/text file (ReadBinary/ReadText), a method to read a line of text (ReadLine), and methods for checking and setting the current position in the file (GetPosition, SetPosition, MovePosition, IsEof) and for checking the current size of the file (GetSize).

⌋*(RS_PER_00004, RS_AP_00122, RS_AP_00146)*

8.3.13.1 ReadAccessor::ReadAccessor

[SWS_PER_00461]{DRAFT} ⌈

Kind:	function
Symbol:	ReadAccessor()
Scope:	class ara::per::ReadAccessor
Syntax:	`ReadAccessor ()=delete;`
Header file:	#include "ara/per/read_accessor.h"
Description:	The default constructor for ReadAccessor shall not be used.

⌋*(RS_PER_00004, RS_AP_00120, RS_AP_00129, RS_AP_00146)*

[SWS_PER_00413] ⌈

Kind:	function
Symbol:	ReadAccessor(ReadAccessor &&ra)
Scope:	class ara::per::ReadAccessor
Syntax:	`ReadAccessor (ReadAccessor &&ra)=delete;`
Header file:	#include "ara/per/read_accessor.h"
Description:	The move constructor for ReadAccessor shall not be used.

⌋*(RS_PER_00004, RS_AP_00120, RS_AP_00121, RS_AP_00129, RS_AP_00132)*

[SWS_PER_00415] ⌈

Kind:	function
Symbol:	ReadAccessor(const ReadAccessor &)
Scope:	class ara::per::ReadAccessor
Syntax:	`ReadAccessor (const ReadAccessor &)=delete;`
Header file:	#include "ara/per/read_accessor.h"
Description:	The copy constructor for ReadAccessor shall not be used.

⌋*(RS_PER_00004, RS_AP_00120)*

8.3.13.2 ReadAccessor::operator=

[SWS_PER_00414] ⌈

Kind:	function
Symbol:	operator=(ReadAccessor &&ra)
Scope:	class ara::per::ReadAccessor
Syntax:	`ReadAccessor& operator= (ReadAccessor &&ra) &=delete;`
Header file:	#include "ara/per/read_accessor.h"
Description:	The move assignment operator for ReadAccessor shall not be used.

⌋*(RS_PER_00004, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00132)*

[SWS_PER_00416] ⌈

Kind:	function
Symbol:	operator=(const ReadAccessor &)
Scope:	class ara::per::ReadAccessor
Syntax:	`ReadAccessor& operator= (const ReadAccessor &)=delete;`
Header file:	#include "ara/per/read_accessor.h"
Description:	The copy assignment operator for ReadAccessor shall not be used.

⌋*(RS_PER_00004, RS_AP_00119, RS_AP_00120)*

8.3.13.3 ReadAccessor::~ReadAccessor

[SWS_PER_00417] ⌈

Kind:	function
Symbol:	~ReadAccessor()
Scope:	class ara::per::ReadAccessor

▽

△

Syntax:	`~ReadAccessor () noexcept;`
Exception Safety:	noexcept
Thread Safety:	no
Header file:	#include "ara/per/read_accessor.h"
Description:	Destructor for ReadAccessor.

⌋*(RS_PER_00004, RS_AP_00120, RS_AP_00129, RS_AP_00132, RS_AP_00134)*

8.3.13.4 ReadAccessor::PeekChar

[SWS_PER_00167] ⌈

Kind:	function	
Symbol:	PeekChar()	
Scope:	class ara::per::ReadAccessor	
Syntax:	`ara::core::Result<char> PeekChar () const noexcept;`	
Return value:	ara::core::Result< char >	A Result containing a character. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	no	
Errors:	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kValidationFailed	Returned if the validity of stored data cannot be ensured.
	PerErrc::kEncryptionFailed	Returned if the decryption of stored data fails.
	PerErrc::kIsEof	Returned if the current position is at the end of the file or if the file is empty.
Header file:	#include "ara/per/read_accessor.h"	
Description:	Returns the character at the current position of the file. The current position is not changed.	

⌋*(RS_PER_00001, RS_PER_00004, RS_AP_00119, RS_AP_00120, RS_AP_00132)*

8.3.13.5 ReadAccessor::PeekByte

[SWS_PER_00418] ⌈

Kind:	function
Symbol:	PeekByte()
Scope:	class ara::per::ReadAccessor
Syntax:	`ara::core::Result<ara::core::Byte> PeekByte () const noexcept;`

▽

△

Return value:	ara::core::Result< ara::core::Byte >	A Result containing a byte. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	no	
Errors:	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kValidationFailed	Returned if the validity of stored data cannot be ensured.
	PerErrc::kEncryptionFailed	Returned if the decryption of stored data fails.
	PerErrc::kIsEof	Returned if the current position is at the end of the file or if the file is empty.
Header file:	#include "ara/per/read_accessor.h"	
Description:	Returns the byte at the current position of the file. The current position is not changed.	

⌋*(RS_PER_00001, RS_PER_00004, RS_AP_00119, RS_AP_00120, RS_AP_00132)*

8.3.13.6 ReadAccessor::GetChar

[SWS_PER_00168] ⌈

Kind:	function	
Symbol:	GetChar()	
Scope:	class ara::per::ReadAccessor	
Syntax:	`ara::core::Result<char> GetChar () noexcept;`	
Return value:	ara::core::Result< char >	A Result containing a character. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	no	
Errors:	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kValidationFailed	Returned if the validity of stored data cannot be ensured.
	PerErrc::kEncryptionFailed	Returned if the decryption of stored data fails.
	PerErrc::kIsEof	Returned if the current position is at the end of the file or if the file is empty.
Header file:	#include "ara/per/read_accessor.h"	
Description:	Returns the character at the current position of the file, advancing the current position. In case of an error, the current position is not changed.	

⌋*(RS_PER_00001, RS_PER_00004, RS_AP_00119, RS_AP_00120, RS_AP_00132)*

8.3.13.7 ReadAccessor::GetByte

[SWS_PER_00419] ⌈

Kind:	function	
Symbol:	GetByte()	
Scope:	class ara::per::ReadAccessor	
Syntax:	`ara::core::Result<ara::core::Byte> GetByte () noexcept;`	
Return value:	ara::core::Result< ara::core::Byte >	A Result containing a byte. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	no	
Errors:	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kValidationFailed	Returned if the validity of stored data cannot be ensured.
	PerErrc::kEncryptionFailed	Returned if the decryption of stored data fails.
	PerErrc::kIsEof	Returned if the current position is at the end of the file or if the file is empty.
Header file:	#include "ara/per/read_accessor.h"	
Description:	Returns the byte at the current position of the file, advancing the current position. In case of an error, the current position is not changed.	

⌋*(RS_PER_00001, RS_PER_00004, RS_AP_00119, RS_AP_00120, RS_AP_00132)*

8.3.13.8 ReadAccessor::ReadText

[SWS_PER_00420] ⌈

Kind:	function	
Symbol:	ReadText()	
Scope:	class ara::per::ReadAccessor	
Syntax:	`ara::core::Result<ara::core::String> ReadText () noexcept;`	
Return value:	ara::core::Result< ara::core::String >	A Result containing a String. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	no	
Errors:	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kValidationFailed	Returned if the validity of stored data cannot be ensured.
	PerErrc::kEncryptionFailed	Returned if the decryption of stored data fails.
	PerErrc::kIsEof	Returned if the current position is at the end of the file or if the file is empty.
Header file:	#include "ara/per/read_accessor.h"	
Description:	Reads all remaining characters into a String, starting from the current position. The current position is set to the end of the file. In case of an error, the current position is not changed.	

⌋*(RS_PER_00001, RS_PER_00004, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00132)*

[SWS_PER_00165] ⌈

Kind:	function	
Symbol:	ReadText(std::uint64_t n)	
Scope:	class ara::per::ReadAccessor	
Syntax:	`ara::core::Result<ara::core::String> ReadText (std::uint64_t n) noexcept;`	
Parameters (in):	n	Number of characters to read.
Return value:	ara::core::Result< ara::core::String >	A Result containing a String. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	no	
Errors:	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kValidationFailed	Returned if the validity of stored data cannot be ensured.
	PerErrc::kEncryptionFailed	Returned if the decryption of stored data fails.
	PerErrc::kIsEof	Returned if the current position is at the end of the file or if the file is empty.
Header file:	#include "ara/per/read_accessor.h"	
Description:	Reads a number of characters into a String, starting from the current position. The current position is advanced accordingly. If the end of the file is reached, the number of returned characters can be less than the requested number, and the current position is set to the end of the file. In case of an error, the current position is not changed.	

⌋*(RS_PER_00001, RS_PER_00004, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00132)*

8.3.13.9 ReadAccessor::ReadBinary

[SWS_PER_00421] ⌈

Kind:	function	
Symbol:	ReadBinary()	
Scope:	class ara::per::ReadAccessor	
Syntax:	`ara::core::Result<ara::core::Vector<ara::core::Byte> > ReadBinary () noexcept;`	
Return value:	ara::core::Result< ara::core::Vector< ara::core::Byte > >	A Result containing a Vector of Byte. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	no	
Errors:	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kValidationFailed	Returned if the validity of stored data cannot be ensured.
	PerErrc::kEncryptionFailed	Returned if the decryption of stored data fails.

▽

△

	PerErrc::kIsEof	Returned if the current position is at the end of the file or if the file is empty.
Header file:	#include "ara/per/read_accessor.h"	
Description:	Reads all remaining bytes into a Vector of Byte, starting from the current position. The current position is set to the end of the file. In case of an error, the current position is not changed.	

⌋*(RS_PER_00001, RS_PER_00004, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00132)*

[SWS_PER_00422] ⌈

Kind:	function	
Symbol:	ReadBinary(std::uint64_t n)	
Scope:	class ara::per::ReadAccessor	
Syntax:	`ara::core::Result<ara::core::Vector<ara::core::Byte> > ReadBinary (std::uint64_t n) noexcept;`	
Parameters (in):	n	Number of bytes to read.
Return value:	ara::core::Result< ara::core::Vector< ara::core::Byte > >	A Result containing a Vector of Byte. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	no	
Errors:	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kValidationFailed	Returned if the validity of stored data cannot be ensured.
	PerErrc::kEncryptionFailed	Returned if the decryption of stored data fails.
	PerErrc::kIsEof	Returned if the current position is at the end of the file or if the file is empty.
Header file:	#include "ara/per/read_accessor.h"	
Description:	Reads a number of bytes into a Vector of Byte, starting from the current position. The current position is advanced accordingly. If the end of the file is reached, the number of returned bytes can be less than the requested number, and the current position is set to the end of the file. In case of an error, the current position is not changed.	

⌋*(RS_PER_00001, RS_PER_00004, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00132)*

8.3.13.10 ReadAccessor::ReadLine

[SWS_PER_00119] ⌈

Kind:	function	
Symbol:	ReadLine(char delimiter='\n')	
Scope:	class ara::per::ReadAccessor	
Syntax:	`ara::core::Result<ara::core::String> ReadLine (char delimiter='\n') noexcept;`	
Parameters (in):	delimiter	The character that is used as delimiter.
Return value:	ara::core::Result< ara::core::String >	A Result containing a String. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	no	
Errors:	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kValidationFailed	Returned if the validity of stored data cannot be ensured.
	PerErrc::kEncryptionFailed	Returned if the decryption of stored data fails.
	PerErrc::kIsEof	Returned if the current position is at the end of the file or if the file is empty.
Header file:	#include "ara/per/read_accessor.h"	
Description:	Reads a complete line of characters into a String, advancing the current position accordingly. The end of the line is demarcated by the delimiter, or by "\\n" (ASCII 0x0a) if that parameter is omitted. The delimiter itself is not included in the returned String. If the end of the file is reached, the remaining characters are returned and the current position is set to the end of the file. In case of an error, the current position is not changed.	

⌋*(RS_PER_00001, RS_PER_00004, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00129, RS_AP_00132)*

8.3.13.11 ReadAccessor::GetSize

[SWS_PER_00424] ⌈

Kind:	function	
Symbol:	GetSize()	
Scope:	class ara::per::ReadAccessor	
Syntax:	`std::uint64_t GetSize () const noexcept;`	
Return value:	std::uint64_t	The current size of the file in bytes.
Exception Safety:	noexcept	
Thread Safety:	no	
Header file:	#include "ara/per/read_accessor.h"	
Description:	Returns the current size of a file in bytes.	

⌋*(RS_PER_00017, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00129, RS_AP_00132)*

8.3.13.12 ReadAccessor::GetPosition

[SWS_PER_00162] ⌈

Kind:	function	
Symbol:	GetPosition()	
Scope:	class ara::per::ReadAccessor	
Syntax:	`std::uint64_t GetPosition () const noexcept;`	
Return value:	std::uint64_t	The current position in the file in bytes from the beginning of the file.
Exception Safety:	noexcept	
Thread Safety:	no	
Header file:	#include "ara/per/read_accessor.h"	
Description:	Returns the current position relative to the beginning of the file. The returned position may be at the end of the file.	

⌋*(RS_PER_00001, RS_PER_00004, RS_AP_00119, RS_AP_00120, RS_AP_00132)*

8.3.13.13 ReadAccessor::SetPosition

[SWS_PER_00163] ⌈

Kind:	function	
Symbol:	SetPosition(std::uint64_t position)	
Scope:	class ara::per::ReadAccessor	
Syntax:	`ara::core::Result<void> SetPosition (std::uint64_t position) noexcept;`	
Parameters (in):	position	Current position in the file in bytes from the beginning of the file.
Return value:	ara::core::Result< void >	A Result of void. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	no	
Errors:	PerErrc::kInvalidPosition	Returned if the given position is beyond the end of the file.
Header file:	#include "ara/per/read_accessor.h"	
Description:	Sets the current position relative to the beginning of the file. In case of an error, the current position is not changed.	

⌋*(RS_PER_00001, RS_PER_00004, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00132)*

8.3.13.14 ReadAccessor::MovePosition

[SWS_PER_00164] ⌈

Kind:	function	
Symbol:	MovePosition(Origin origin, std::int64_t offset)	
Scope:	class ara::per::ReadAccessor	
Syntax:	`ara::core::Result<std::uint64_t> MovePosition (Origin origin, std::int64_t offset) noexcept;`	
Parameters (in):	origin	Starting point from which to move 'offset' bytes.
	offset	Offset in bytes relative to 'origin'. Can be positive in case of kBeginning and kCurrent and negative in case of kCurrent and kEnd. In case of kCurrent, an offset of zero will not change the current position. In case of kEnd, an offset of zero will set the position to the end of the file.
Return value:	ara::core::Result< std::uint64_t >	A Result containing the new position in bytes from the beginning of the file. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	no	
Errors:	PerErrc::kInvalidPosition	Returned if the resulting position is lower than zero or beyond the end of the file.
Header file:	#include "ara/per/read_accessor.h"	
Description:	Moves the current position in the file relative to the Origin. In case of an error, the current position is not changed.	

⌋*(RS_PER_00001, RS_PER_00004, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00132)*

8.3.13.15 ReadAccessor::IsEof

[SWS_PER_00107] ⌈

Kind:	function	
Symbol:	IsEof()	
Scope:	class ara::per::ReadAccessor	
Syntax:	`bool IsEof () const noexcept;`	
Return value:	bool	True if the current position is at the end of the file, false otherwise.
Exception Safety:	noexcept	
Thread Safety:	no	
Header file:	#include "ara/per/read_accessor.h"	
Description:	Checks if the current position is at end of file.	

⌋*(RS_PER_00001, RS_PER_00004, RS_AP_00119, RS_AP_00120, RS_AP_-00132)*

8.3.14 ReadWriteAccessor Class

This section shows the methods available for a `ara::per::ReadWriteAccessor` object obtained from a call to `ara::per::FileStorage::OpenFileWriteOnly` or `ara::per::FileStorage::OpenFileReadWrite`.

[SWS_PER_00343] ⌈

Kind:	class
Symbol:	ReadWriteAccessor
Scope:	namespace ara::per
Base class:	ReadAccessor
Syntax:	`class ReadWriteAccessor : public ReadAccessor {...};`
Header file:	#include "ara/per/read_write_accessor.h"
Description:	ReadWriteAccessor is used to read and write file data. It provides the WriteBinary and WriteText methods featuring a Result for controlled, unformatted writing, and the operator<< method for simple formatted writing. It also provides SyncToFile() to flush the buffer of the operating system to the storage.

⌋*(RS_PER_00004, RS_AP_00122, RS_AP_00146)*

8.3.14.1 ReadWriteAccessor::ReadWriteAccessor

[SWS_PER_00462]{DRAFT} ⌈

Kind:	function
Symbol:	ReadWriteAccessor()
Scope:	class ara::per::ReadWriteAccessor
Syntax:	`ReadWriteAccessor ()=delete;`
Header file:	#include "ara/per/read_write_accessor.h"
Description:	The default constructor for ReadWriteAccessor shall not be used.

⌋*(RS_PER_00004, RS_AP_00120, RS_AP_00129, RS_AP_00146)*

8.3.14.2 ReadWriteAccessor::SyncToFile

[SWS_PER_00122] ⌈

Kind:	function
Symbol:	SyncToFile()
Scope:	class ara::per::ReadWriteAccessor
Syntax:	`ara::core::Result<void> SyncToFile () noexcept;`

▽

△

Return value:	ara::core::Result< void >	A Result of void. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	no	
Errors:	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kEncryptionFailed	Returned if the encryption of stored data fails.
	PerErrc::kOutOfStorageSpace	Returned if the available storage space is insufficient for the updated file size.
Header file:	#include "ara/per/read_write_accessor.h"	
Description:	Triggers flushing of the current file content to the physical storage.	

⌋*(RS_PER_00001, RS_PER_00004, RS_AP_00119, RS_AP_00120, RS_AP_00128, RS_AP_00127, RS_AP_00129, RS_AP_00132)*

8.3.14.3 ReadWriteAccessor::SetFileSize

[SWS_PER_00428] ⌈

Kind:	function	
Symbol:	SetFileSize(std::uint64_t size)	
Scope:	class ara::per::ReadWriteAccessor	
Syntax:	`ara::core::Result<void> SetFileSize (std::uint64_t size) noexcept;`	
Parameters (in):	size	New size of the file.
Return value:	ara::core::Result< void >	A Result of void. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	no	
Errors:	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kEncryptionFailed	Returned if the encryption or decryption of stored data fails.
	PerErrc::kInvalidSize	Returned if the new size is larger than the current size.
Header file:	#include "ara/per/read_write_accessor.h"	
Description:	Reduces the size of the file to ’size’, effectively removing the current content of the file beyond this size. The current file position is unchanged if it is lower than ’size’, or set to the last valid position in the file otherwise. If ’size’ is 0, the current file position will also be set to 0.	

⌋*(RS_PER_00001, RS_PER_00004, RS_AP_00119, RS_AP_00120, RS_AP_00128, RS_AP_00127, RS_AP_00129, RS_AP_00132)*

8.3.14.4 ReadWriteAccessor::WriteText

[SWS_PER_00166] ⌈

Kind:	function	
Symbol:	WriteText(ara::core::StringView s)	
Scope:	class ara::per::ReadWriteAccessor	
Syntax:	`ara::core::Result<void> WriteText (ara::core::StringView s) noexcept;`	
Parameters (in):	s	A StringView containing the characters to be written.
Return value:	ara::core::Result< void >	A Result of void. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	no	
Errors:	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.
	PerErrc::kEncryptionFailed	Returned if the encryption or decryption of stored data fails.
	PerErrc::kOutOfStorageSpace	Returned if the available storage space is insufficient for the updated file size.
Header file:	#include "ara/per/read_write_accessor.h"	
Description:	Writes the content of a StringView to the file. The time when the content is persisted depends on the implementation of Persistency. SyncTo File can be used to force Persistency to persist the file content. In case of an error, the file content might be corrupted, and the current position might or might not have changed. The expected state of the file for each supported error can be expected to be as follows: k PhysicalStorageFailure: The state of the file is unknown. It could have been entirely destroyed. kEncryptionFailed: The content of the file and the current position will have been updated, but could not be persisted. The persisted file will reflect an older version of the file. kOutOfStorage Space: The content of the file will have been updated, but the part of the operation that exceeded the quota will have been discarded. The current position will be at the end of the file.	

⌋*(RS_PER_00001, RS_PER_00004, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00132)*

8.3.14.5 ReadWriteAccessor::WriteBinary

[SWS_PER_00423] ⌈

Kind:	function	
Symbol:	WriteBinary(ara::core::Span< const ara::core::Byte > b)	
Scope:	class ara::per::ReadWriteAccessor	
Syntax:	`ara::core::Result<void> WriteBinary (ara::core::Span< const ara::core::Byte > b) noexcept;`	
Parameters (in):	b	A Span of Byte containing the bytes to be written.
Return value:	ara::core::Result< void >	A Result of void. In case of an error, it contains any of the errors defined below, or a vendor specific error.
Exception Safety:	noexcept	
Thread Safety:	no	
Errors:	PerErrc::kPhysicalStorageFailure	Returned if access to the storage fails.

▽

△

	PerErrc::kEncryptionFailed	Returned if the encryption or decryption of stored data fails.
	PerErrc::kOutOfStorageSpace	Returned if the available storage space is insufficient for the updated file size.
Header file:	#include "ara/per/read_write_accessor.h"	
Description:	Writes the content of a Span of Byte to the file. The time when the content is persisted depends on the implementation of Persistency. SyncTo File can be used to force Persistency to persist the file content. In case of an error, the file content might be corrupted, and the current position might or might not have changed. The expected state of the file for each supported error can be expected to be as follows: k PhysicalStorageFailure: The state of the file is unknown. It could have been entirely destroyed. kEncryptionFailed: The content of the file and the current position will have been updated, but could not be persisted. The persisted file will reflect an older version of the file. kOutOfStorage Space: The content of the file will have been updated, but the part of the operation that exceeded the quota will have been discarded. The current position will be at the end of the file.	

⌋*(RS_PER_00001, RS_PER_00004, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00132)*

8.3.14.6 ReadWriteAccessor::operator<<

[SWS_PER_00125] ⌈

Kind:	function	
Symbol:	operator<<(ara::core::StringView s)	
Scope:	class ara::per::ReadWriteAccessor	
Syntax:	`ReadWriteAccessor& operator<< (ara::core::StringView s) noexcept;`	
Parameters (in):	s	The StringView containing the characters to be written.
Return value:	ReadWriteAccessor &	The ReadWriteAccessor object.
Exception Safety:	noexcept	
Thread Safety:	no	
Header file:	#include "ara/per/read_write_accessor.h"	
Description:	Writes the content of a StringView to the file. This operator is just a comfort feature for non-safety critical applications. If an error occurs during this operation, it is silently ignored.	

⌋*(RS_PER_00001, RS_PER_00004, RS_AP_00119, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00132)*

9 Service Interfaces

The `Persistency` does not provide any service interfaces via `ara::com`.

A Mentioned Class Tables

For the sake of completeness, this chapter contains a set of class tables representing meta-classes mentioned in the context of this document but which are not contained directly in the scope of describing specific meta-model semantics.

Class	**AdaptiveApplicationSwComponentType**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::ApplicationStructure			

Note	This meta-class represents the ability to support the formal modeling of application software on the AUTOSAR adaptive platform. Consequently, it shall only be used on the AUTOSAR adaptive platform.
	
	Tags:
	atp.Status=draft
	atp.recommendedPackage=AdaptiveApplicationSwComponentTypes			

Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

internalBehavior	AdaptiveSwcInternal Behavior	0..1	aggr	This aggregation represents the internal behavior of the AdaptiveApplicationSwComponentType for the AUTOSAR adaptive platform.
				
				Stereotypes: atpSplitable; atpVariation
				Tags:
				atp.Splitkey=internalBehavior.shortName, internal Behavior.variationPoint.shortLabel
				atp.Status=draft
				vh.latestBindingTime=preCompileTime

Table A.1: AdaptiveApplicationSwComponentType

Class	***CppImplementationDataType*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::CppImplementationDataType			
Note	This meta-class represents the way to specify a reusable data type definition taken as a the basis for a C++ language binding **Tags:**atp.Status=draft			
Base	*ARElement*, *ARObject*, *AbstractImplementationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *CppImplementationDataTypeContextTarget*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	CustomCppImplementationDataType, StdCppImplementationDataType			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
arraySize	PositiveInteger	0..1	attr	This attribute can be used to specify the array size if the enclosing CppImplementationDataType has array semantics. **Stereotypes:** atpVariation **Tags:** atp.Status=draft vh.latestBindingTime=preCompileTime
headerFile	String	0..1	attr	Configuration of the Header File with the custom class declaration. **Tags:**atp.Status=draft
namespace (ordered)	SymbolProps	*	aggr	This aggregation allows for the definition an own namespace for the enclosing CppImplementationData Type. **Tags:**atp.Status=draft

▽

△

Class	***CppImplementationDataType*** (abstract)			
subElement (ordered)	CppImplementation DataTypeElement	*	aggr	This represents the collection of sub-elements of the enclosing CppImplementationDataType **Tags:**atp.Status=draft
template Argument (ordered)	CppTemplateArgument	*	aggr	This aggregation allows for the specification of properties of template arguments **Tags:**atp.Status=draft
typeEmitter	NameToken	0..1	attr	This attribute can be taken to control how the respective CppImplementationDataType is contributed to the language binding. **Tags:**atp.Status=draft
typeReference	CppImplementation DataType	0..1	ref	This reference shall be defined to define a type reference (a.k.a. typedef). **Tags:**atp.Status=draft

Table A.2: CppImplementationDataType

Class	**CryptoKeySlot**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::CryptoDeployment			

Note	This meta-class represents the ability to define a concrete key to be used for a crypto operation.
	Tags:
	atp.ManifestKind=MachineManifest
	atp.Status=draft			

Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
allocateShadow Copy	Boolean	0..1	attr	This attribute defines whether a shadow copy of this Key Slot shall be allocated to enable rollback of a failed Key Slot update campaign (see interface BeginTransaction). **Tags:**atp.Status=draft
cryptoAlgId	String	0..1	attr	This attribute defines a crypto algorithm restriction (kAlgId Any means without restriction). The algorithm can be specified partially: family & length, mode, padding. Future Crypto Providers can support some crypto algorithms that are not well known/ standardized today, therefore AUTOSAR doesn't provide a concrete list of crypto algorithms' identifiers and doesn't suppose usage of numerical identifiers. Instead of this a provider supplier should provide string names of supported algorithms in accompanying documentation. The name of a crypto algorithm shall follow the rules defined in the specification of cryptography for Adaptive Platform. **Tags:**atp.Status=draft
cryptoObject Type	CryptoObjectTypeEnum	0..1	attr	Object type that can be stored in the slot. If this field contains "Undefined" then mSlotCapacity must be provided and larger then 0. **Tags:**atp.Status=draft
keySlotAllowed Modification	CryptoKeySlotAllowed Modification	0..1	aggr	Restricts how this keySlot may be used **Tags:**atp.Status=draft
keySlotContent AllowedUsage	CryptoKeySlotContent AllowedUsage	*	aggr	Restriction of allowed usage of a key stored to the slot. **Tags:**atp.Status=draft

▽

△

Class	**CryptoKeySlot**			
slotCapacity	PositiveInteger	0..1	attr	Capacity of the slot in bytes to be reserved by the stack vendor. One use case is to define this value in case that the cryptoObjectType is undefined and the slot size can not be deduced from cryptoObjectType and cryptoAlgId. "0" means slot size can be deduced from cryptoObject Type and cryptoAlgId. **Tags:**atp.Status=draft
slotType	CryptoKeySlotType Enum	0..1	attr	This attribute defines whether the keySlot is exclusively used by the Application; or whether it is used by Stack Services and managed by a Key Manager Application. **Tags:**atp.Status=draft

Table A.3: CryptoKeySlot

Enumeration	**CryptoKeySlotUsageEnum**
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::CryptoDeployment
Note	This enum defines the possible roles of the keySlotUsage. **Tags:**atp.Status=draft
Literal	***Description***

encryption	Key slot usage for encryption
	
	Tags:
	atp.EnumerationLiteralIndex=1
	atp.Status=draft

verification	Key slot usage for verification
	
	Tags:
	atp.EnumerationLiteralIndex=0
	atp.Status=draft

Table A.4: CryptoKeySlotUsageEnum

Class	**Executable**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::ApplicationStructure			

Note	This meta-class represents an executable program.
	
	Tags:
	atp.Status=draft
	atp.recommendedPackage=Executables			

Base	*ARElement*, *ARObject*, *AtpClassifier*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
buildType	BuildTypeEnum	0..1	attr	This attribute describes the buildType of a module and/or platform implementation. **Tags:**atp.Status=draft
loggingBehavior	LoggingBehaviorEnum	0..1	attr	This attribute indicates the intended logging behavior of the enclosing Executable. **Tags:**atp.Status=draft
minimumTimer Granularity	TimeValue	0..1	attr	This attribute describes the minimum timer resolution (TimeValue of one tick) that is required by the Executable. **Tags:**atp.Status=draft

▽

△

Class	**Executable**			
reporting Behavior	ExecutionState ReportingBehavior Enum	0..1	attr	this attribute controls the execution state reporting behavior of the enclosing Executable. **Tags:**atp.Status=draft
rootSw Component Prototype	RootSwComponent Prototype	0..1	aggr	This represents the root SwCompositionPrototype of the Executable. This aggregation is required (in contrast to a direct reference of a SwComponentType) in order to support the definition of instanceRefs in Executable context. **Tags:**atp.Status=draft
version	StrongRevisionLabel String	0..1	attr	Version of the executable. **Tags:**atp.Status=draft

Table A.5: Executable

Class	***FunctionalClusterInteractsWithFunctionalClusterMapping*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::CryptoDeployment			
Note	This meta-class identifies a relation between functional clusters on the adaptive platform such one functional cluster can call APIs of the other functional cluster. **Tags:**atp.Status=draft			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadablePackageElement*			
Subclasses	ArtifactChecksumToCryptoProviderMapping, ComCertificateToCryptoCertificateMapping, ComKeyTo CryptoKeySlotMapping, ComSecOcToCryptoKeySlotMapping, PersistencyDeploymentElementToCrypto KeySlotMapping, PersistencyDeploymentToCryptoKeySlotMapping, PersistencyDeploymentToDltLogSink Mapping, TimeBaseProviderToPersistencyMapping			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.6: FunctionalClusterInteractsWithFunctionalClusterMapping

Class	**PPortPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Component port providing a certain port interface.			
Base	*ARObject*, *AbstractProvidedPortPrototype*, *AtpBlueprintable*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *PortPrototype*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
provided Interface	PortInterface	0..1	tref	The interface that this port provides. **Stereotypes:** isOfType

Table A.7: PPortPrototype

Class	**PRPortPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	This kind of PortPrototype can take the role of both a required and a provided PortPrototype.			
Base	*ARObject*, *AbstractProvidedPortPrototype*, *AbstractRequiredPortPrototype*, *AtpBlueprintable*, *Atp Feature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *PortPrototype*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	**PRPortPrototype**			
provided Required Interface	PortInterface	0..1	tref	This represents the PortInterface used to type the PRPort Prototype **Stereotypes:** isOfType

Table A.8: PRPortPrototype

Enumeration	**PersistencyCollectionLevelUpdateStrategyEnum**
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface
Note	This enumeration provides possible values for the update strategy on interface/storage level. **Tags:**atp.Status=draft
Literal	*Description*

delete	The update strategy is to delete all values on the level of the respective collection.
	Tags:
	atp.EnumerationLiteralIndex=1
	atp.Status=draft

keepExisting	The update strategy is to keep the existing values on the level of the respective collection.
	Tags:
	atp.EnumerationLiteralIndex=0
	atp.Status=draft

Table A.9: PersistencyCollectionLevelUpdateStrategyEnum

Class	**PersistencyDataElement**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This meta-class represents the ability to formally specify a piece of data that is subject to persistency in the context of the enclosing PersistencyKeyValueStorageInterface. PersistencyDataElement represents also a key-value pair of the deployed PersistencyKeyValueStorage and provides an initial value. **Tags:**atp.Status=draft			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *AutosarDataPrototype*, *DataPrototype*, *Identifiable*, *Multilanguage Referrable*, *PersistencyInterfaceElement*, *Referrable*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table A.10: PersistencyDataElement

Class	**PersistencyDataRequiredComSpec**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::ComSpec			
Note	This meta-class represents the ability to define port-specific attributes for supporting use cases of data persistency on the required side. **Tags:**atp.Status=draft			
Base	*ARObject*, *RPortComSpec*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
dataElement	PersistencyData Element	1	ref	This refrence represents the PersistencyDataElement for which the PersistencyDataRequiredComSpec applies. **Tags:**atp.Status=draft

▽

△

Class	PersistencyDataRequiredComSpec			
initValue	ValueSpecification	0..1	aggr	This aggregation represents the definition of an initial value for the PersistencyDataElement referenced by the enclosing PersistencyDataRequiredComSpec **Tags:**atp.Status=draft

Table A.11: PersistencyDataRequiredComSpec

Class	***PersistencyDeployment*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::Persistency			
Note	This abstract meta-class serves as a base class for concrete classes representing different aspects of persistency. **Tags:**atp.Status=draft			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableExclusivePackageElement*, *UploadablePackageElement*			
Subclasses	PersistencyFileStorage, PersistencyKeyValueStorage			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
maximum AllowedSize	PositiveUnlimitedInteger	0..1	attr	The value of this attribute represents the maximum size allowed at deployment time for the enclosing Persistency Deployment. **Tags:**atp.Status=draft
minimum SustainedSize	PositiveInteger	0..1	attr	The value of this attribute represents the minimum size guaranteed at deployment time for the enclosing PersistencyDeployment. **Tags:**atp.Status=draft
redundancy Handling	PersistencyRedundancy Handling	*	aggr	This aggregation represents the chosen approaches to handle redundancy. **Tags:**atp.Status=draft
updateStrategy	PersistencyCollection LevelUpdateStrategy Enum	1	attr	This attribute shall be used to specify the update strategy of the respective PersistencyDeployment as a whole. **Tags:**atp.Status=draft
version	StrongRevisionLabel String	0..1	attr	The attribute represents the version of the PersistencyFile Storage or PersistencyKeyValueStorage. **Tags:**atp.Status=draft

Table A.12: PersistencyDeployment

Class	***PersistencyDeploymentElement*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::Persistency			
Note	This abstract meta-class serves as a base class for concrete classes representing different aspects of elements of a PersistencyDeployment. **Tags:**atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	PersistencyFile, PersistencyKeyValuePair			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
updateStrategy	PersistencyElement LevelUpdateStrategy Enum	0..1	attr	This attribute can be used to specify the update strategy of the respective PersistencyDeploymentElement. **Tags:**atp.Status=draft

Table A.13: PersistencyDeploymentElement

Class	**PersistencyDeploymentElementToCryptoKeySlotMapping**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::CryptoDeployment			

Note	This meta-class represents the ability to define a mapping between the PersistencyDeploymentElement and a CryptoKeySlot.
	Tags:
	atp.Status=draft
	atp.recommendedPackage=FCInteractions			

Base	*ARElement*, *ARObject*, *CollectableElement*, *FunctionalClusterInteractsWithFunctionalClusterMapping*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadablePackageElement*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
cryptoKeySlot	CryptoKeySlot	0..1	ref	This reference represents the mapped CryptoKeySlot. **Tags:**atp.Status=draft
keySlotUsage	CryptoKeySlotUsage Enum	0..1	attr	This attribute defines the role of the keySlot assignment. **Tags:**atp.Status=draft
persistency Deployment Element	PersistencyDeployment Element	0..1	ref	This reference represents the mapped Persistency Deployment. **Tags:**atp.Status=draft
verificationHash	String	0..1	attr	This attribute defines the hash of the storage used in case of verification. **Tags:**atp.Status=draft

Table A.14: PersistencyDeploymentElementToCryptoKeySlotMapping

Class	**PersistencyDeploymentToCryptoKeySlotMapping**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::CryptoDeployment			

Note	This meta-class represents the ability to define a mapping between the PersistencyDeployment and a CryptoKeySlot.
	Tags:
	atp.Status=draft
	atp.recommendedPackage=FCInteractions			

Base	*ARElement*, *ARObject*, *CollectableElement*, *FunctionalClusterInteractsWithFunctionalClusterMapping*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadablePackageElement*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
cryptoKeySlot	CryptoKeySlot	0..1	ref	This reference represents the mapped CryptoKeySlot. **Tags:**atp.Status=draft
keySlotUsage	CryptoKeySlotUsage Enum	0..1	attr	This attribute defines the role of the keySlot assignment. **Tags:**atp.Status=draft
persistency Deployment	PersistencyDeployment	1	ref	This reference represents the mapped Persistency Deployment. **Tags:**atp.Status=draft
verificationHash	String	0..1	attr	This attribute defines the hash of the storage used in case of verification. **Tags:**atp.Status=draft

Table A.15: PersistencyDeploymentToCryptoKeySlotMapping

Enumeration	**PersistencyElementLevelUpdateStrategyEnum**
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface
Note	This enumeration provides possible values for the update strategy on element level. **Tags:**atp.Status=draft
Literal	*Description*

delete	The update strategy is to delete the value of the respective data item.
	
	Tags:
	atp.EnumerationLiteralIndex=2
	atp.Status=draft

keepExisting	The update strategy is to keep the existing value of the respective data item.
	
	Tags:
	atp.EnumerationLiteralIndex=1
	atp.Status=draft

overwrite	The update strategy is to overwrite the respective data item.
	
	Tags:
	atp.EnumerationLiteralIndex=0
	atp.Status=draft

Table A.16: PersistencyElementLevelUpdateStrategyEnum

Class	**PersistencyFile**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::Persistency			

Note	This meta-class represents the model of a file as part of the persistency on deployment level.
	
	Tags:
	atp.Status=draft
	atp.recommendedPackage=PersistencyFiles			

Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *PersistencyDeploymentElement*, *Referrable*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
contentUri	UriString	0..1	attr	This attribute represents the URI that identifies the initial content of the PersistencyFile. **Tags:**atp.Status=draft
fileName	String	1	attr	This attribute holds filename part of the storage location for the PersistencyFile, e.g. file on the file system. **Tags:**atp.Status=draft

Table A.17: PersistencyFile

Class	**PersistencyFileElement**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This meta-class has the ability to represent a file at design time such that it is possible to configure the behavior for accessing the represented file at run-time. **Tags:**atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *PersistencyInterfaceElement*, *Referrable*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
contentUri	UriString	1	attr	This attribute represents the URI that identifies the initial content of the PersistencyFile. **Tags:**atp.Status=draft

▽

△

Class	**PersistencyFileElement**			
fileName	String	1	attr	This attribute holds filename part of the storage location for the PersistencyFileProxy, e.g. file on the file system. **Tags:**atp.Status=draft

Table A.18: PersistencyFileElement

Class	**PersistencyFileStorage**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::Persistency			

Note	This meta-class comes with the ability to define a collection of single files (directory) that creates the deployment-side counterpart to a PortPrototype typed by a PersistencyFileStorageInterface.
	Tags:
	atp.Status=draft
	atp.recommendedPackage=PersistencyFileStorages			

Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PersistencyDeployment*, *Referrable*, *UploadableExclusivePackageElement*, *UploadablePackageElement*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
file	PersistencyFile	*	aggr	This aggregation represents the collection of files aggregated by the PersistencyFileStorage. **Tags:**atp.Status=draft
uri	UriString	1	attr	This attribute holds the storage location for the PersistencyFileStorage, e.g. a directory on the file system. **Tags:**atp.Status=draft

Table A.19: PersistencyFileStorage

Class	**PersistencyFileStorageInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			

Note	This meta-class provides the ability to implement a PortInterface for supporting persistency use cases for files.
	Tags:
	atp.Status=draft
	atp.recommendedPackage=PersistencyFileStorageInterfaces			

Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PersistencyInterface*, *PortInterface*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
fileElement	PersistencyFileElement	*	aggr	This aggregation represents the collection of Persistency FileStorages in the context of the enclosing Persistency FileStorageInterface. **Tags:**atp.Status=draft
maxNumberOf Files	PositiveInteger	0..1	attr	This attribute represents the definition of an upper bound for the handling of files at run-time in the context of the enclosing PersistencyFileStorageInterface. **Tags:**atp.Status=draft

Table A.20: PersistencyFileStorageInterface

Class	***PersistencyInterface*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This meta-class provides the abstract ability to define a PortInterface for the support of persistency use cases. **Tags:**atp.Status=draft			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Subclasses	PersistencyFileStorageInterface, PersistencyKeyValueStorageInterface			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
minimum SustainedSize	PositiveInteger	0..1	attr	The value of this attribute represents the minimum size required at design time for the enclosing Persistency Interface. **Tags:**atp.Status=draft
redundancy	PersistencyRedundancy Enum	0..1	attr	This attribute represents a requirement towards the redundancy of storage. **Tags:**atp.Status=draft
redundancy Handling	PersistencyRedundancy Handling	*	aggr	This aggregation represents the chosen approaches to handle redundancy for the various use cases implemented by subclasses **Tags:**atp.Status=draft
updateStrategy	PersistencyCollection LevelUpdateStrategy Enum	0..1	attr	This attribute can be used to specify the update strategy of the respective PersistencyInterface as a whole. **Tags:**atp.Status=draft

Table A.21: PersistencyInterface

Class	***PersistencyInterfaceElement*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This meta-class provides the abstract ability to define an element of a PortInterface for the support of persistency use cases. **Tags:**atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	PersistencyDataElement, PersistencyFileElement			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
updateStrategy	PersistencyElement LevelUpdateStrategy Enum	0..1	attr	This attribute can be used to specify the update strategy of the respective PersistencyInterfaceElement. **Tags:**atp.Status=draft

Table A.22: PersistencyInterfaceElement

Class	**PersistencyKeyValuePair**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::Persistency			
Note	This meta-class represents the ability to formally model a key-value pair in the context of the deployment of persistency. **Tags:**atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *PersistencyDeploymentElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	PersistencyKeyValuePair			
initValue	ValueSpecification	0..1	aggr	This aggregation represents the ability to define an initial value for the value side of the key-value pair. Please note that it does not make sense to configure an initial value if the PersistencyDeploymentElement.updateStrategy is set to the value delete. **Tags:**atp.Status=draft
valueDataType	AbstractImplementation DataType	1	ref	This reference represents the data type applicable for the value of the key-value pair. **Tags:**atp.Status=draft

Table A.23: PersistencyKeyValuePair

Class	**PersistencyKeyValueStorage**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::Persistency			

Note	This meta-class represents the ability to model a key-value storage on deployment level.
	Tags:
	atp.Status=draft
	atp.recommendedPackage=PersistencyKeyValueStorages			

Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PersistencyDeployment*, *Referrable*, *UploadableExclusivePackageElement*, *UploadablePackageElement*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
keyValuePair	PersistencyKeyValue Pair	*	aggr	This aggregation represents the key-value-pairs owned by the enclosing PersistencyKeyValueStorage. **Tags:**atp.Status=draft
uri	UriString	0..1	attr	This attribute holds the storage location for the PersistencyKeyValueStorage, e.g. file on the file system. **Tags:**atp.Status=draft

Table A.24: PersistencyKeyValueStorage

Class	**PersistencyKeyValueStorageInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			

Note	This meta-class provides the ability to implement a PortInterface for supporting persistency use cases for data.
	Tags:
	atp.Status=draft
	atp.recommendedPackage=PersistencyKeyValueStorageInterfaces			

Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PersistencyInterface*, *PortInterface*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
dataElement	PersistencyData Element	*	aggr	This aggregation represents the collection of Persistency DataElements in the context of the enclosing Persistency KeyValueStorageInterface. **Tags:**atp.Status=draft
dataTypeFor Serialization	AbstractImplementation DataType	*	ref	This reference identifies the AbstractImplementationData Types that shall be supported for storing in a key-value storage in addition to the types already determined from tha aggregation of PersistencyDataElement. **Tags:**atp.Status=draft

Table A.25: PersistencyKeyValueStorageInterface

Class	***PersistencyPortPrototypeToDeploymentMapping*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::Persistency			
Note	This abstract bas class implements the shared functionality of all mapping between a PortPrototype, a Process, and a specific subclass of PersistencyDeployment. **Tags:**atp.Status=draft			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableExclusivePackageElement*, *UploadablePackageElement*			
Subclasses	PersistencyPortPrototypeToFileStorageMapping, PersistencyPortPrototypeToKeyValueStorageMapping			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
portPrototype	PortPrototype	0..1	iref	This reference represents the mapped PortPrototype. **Tags:**atp.Status=draft **InstanceRef implemented by:**PortPrototypeIn ExecutableInstanceRef
process	Process	1	ref	This reference represents the process required as context for the mapping. **Tags:**atp.Status=draft

Table A.26: PersistencyPortPrototypeToDeploymentMapping

Class	**PersistencyPortPrototypeToFileStorageMapping**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::Persistency			

Note	This meta-class represents the ability to define a mapping between a collection of files on deployment level to a given PortPrototype.
	
	Tags:
	atp.Status=draft
	atp.recommendedPackage=PersistencyPortPrototypeToFileStorageMappings			

Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *PersistencyPortPrototypeToDeploymentMapping*, *Referrable*, *UploadableExclusivePackage Element*, *UploadablePackageElement*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
fileStorage	PersistencyFileStorage	1	ref	This reference represents the mapped file storage. **Tags:**atp.Status=draft

Table A.27: PersistencyPortPrototypeToFileStorageMapping

Class	**PersistencyPortPrototypeToKeyValueStorageMapping**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::Persistency			

Note	This meta-class represents the ability to define a mapping between a PortPrototype and a key-value storage.
	
	Tags:
	atp.Status=draft
	atp.recommendedPackage=PersistencyPortPrototypeToKeyValueStorageMappings			

Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *PersistencyPortPrototypeToDeploymentMapping*, *Referrable*, *UploadableExclusivePackage Element*, *UploadablePackageElement*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
keyValue Storage	PersistencyKeyValue Storage	1	ref	This reference represents the mapped key-value storage. **Tags:**atp.Status=draft

Table A.28: PersistencyPortPrototypeToKeyValueStorageMapping

Class	***PersistencyRedundancyChecksum*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::Persistency			
Note	Abstract class that defines the common attributes for implementations of redundancy. **Tags:**atp.Status=draft			
Base	*ARObject*, *PersistencyRedundancyHandling*			
Subclasses	PersistencyRedundancyCrc, PersistencyRedundancyHash			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
algorithmFamily	String	1	attr	This attribute identifies the algorithm family that is used to execute the CRC/Hash. **Tags:**atp.Status=draft
length	PositiveInteger	1	attr	This attribute describes the length of the CRC/Hash in the unit bits. **Tags:**atp.Status=draft

Table A.29: PersistencyRedundancyChecksum

Class	**PersistencyRedundancyCrc**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::Persistency			
Note	This meta-class formally describes the usage of a CRC for the implementation of redundancy. **Tags:**atp.Status=draft			
Base	*ARObject*, *PersistencyRedundancyChecksum*, *PersistencyRedundancyHandling*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.30: PersistencyRedundancyCrc

Enumeration	**PersistencyRedundancyEnum**
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::ComSpec
Note	This meta-class provides a way to specify in which way redundancy shall be applied on collection level. **Tags:**atp.Status=draft
Literal	***Description***

none	This value represents the requirement that redundancy measures are not applied on persistency storage level.
	
	Tags:
	atp.EnumerationLiteralIndex=1
	atp.Status=draft

redundant	This value represents the requirement that redundancy measures are applied on persistency storage level. The nature of the redundant persistent storage is not further qualified and subject to integrator decisions. **Tags:** atp.EnumerationLiteralIndex=0 atp.Status=draft

▽

△

Enumeration	**PersistencyRedundancyEnum**
redundantPer Element	This value represents the requirement that redundancy measures are applied on key-value level of a key-value storage or on file level of a file storage. The nature of the redundancy used on the persistent storage is not further qualified and subject to integrator decisions. **Tags:** atp.EnumerationLiteralIndex=2 atp.Status=draft

Table A.31: PersistencyRedundancyEnum

Class	***PersistencyRedundancyHandling*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::Persistency			
Note	This abstract base class represents a formal description of redundancy. **Tags:**atp.Status=draft			
Base	*ARObject*			
Subclasses	*PersistencyRedundancyChecksum*, PersistencyRedundancyMOutOfN			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
scope	PersistencyRedundancy HandlingScopeEnum	0..1	attr	This attribute controls the scope in which the redundancy handling is applied. **Tags:**atp.Status=draft

Table A.32: PersistencyRedundancyHandling

Enumeration	**PersistencyRedundancyHandlingScopeEnum**
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::Persistency
Note	This meta-class provides values to control the scope of redundancy measures in the persistency deployment **Tags:**atp.Status=draft
Literal	*Description*

persistency Redundancy HandlingScope Element	The redundancy handling shall be applied on element level (key-value pair and file).
	Tags:
	atp.EnumerationLiteralIndex=0
	atp.Status=draft

persistency Redundancy HandlingScope Storage	The redundancy handling shall be applied on storage (key-value storage and file storage) level.
	Tags:
	atp.EnumerationLiteralIndex=1
	atp.Status=draft

Table A.33: PersistencyRedundancyHandlingScopeEnum

Class	**PersistencyRedundancyHash**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::Persistency			
Note	This meta-class formally describes the usage of a Hash for the implementation of redundancy. **Tags:**atp.Status=draft			
Base	*ARObject*, *PersistencyRedundancyChecksum*, *PersistencyRedundancyHandling*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*

▽

△

Class	**PersistencyRedundancyHash**			
initialization VectorLength	PositiveInteger	0..1	attr	Length of the initialization vector. **Tags:**atp.Status=draft

Table A.34: PersistencyRedundancyHash

Class	**PersistencyRedundancyMOutOfN**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::Persistency			
Note	This meta-class provides the ability to describe redundancy via an "M out of N" approach. In this case N is the number of copies created and M is the minimum number of identical copies to justify a reliable read access to the data. **Tags:**atp.Status=draft			
Base	*ARObject*, *PersistencyRedundancyHandling*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
m	PositiveInteger	1	attr	This attribute represents the "M" coordinate in the "M out of N" scheme. **Tags:**atp.Status=draft
n	PositiveInteger	1	attr	This attribute represents the "N" coordinate in the "M out of N" scheme. **Tags:**atp.Status=draft

Table A.35: PersistencyRedundancyMOutOfN

Class	***PortPrototype*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Base class for the ports of an AUTOSAR software component. The aggregation of PortPrototypes is subject to variability with the purpose to support the conditional existence of ports.			
Base	*ARObject*, *AtpBlueprintable*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	*AbstractProvidedPortPrototype*, *AbstractRequiredPortPrototype*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
clientServer Annotation	ClientServerAnnotation	*	aggr	Annotation of this PortPrototype with respect to client/ server communication.
delegatedPort Annotation	DelegatedPort Annotation	0..1	aggr	Annotations on this delegated port.
ioHwAbstraction Server Annotation	IoHwAbstractionServer Annotation	*	aggr	Annotations on this IO Hardware Abstraction port.
logAndTrace Message CollectionSet	LogAndTraceMessage CollectionSet	0..1	ref	Reference to a collection of Log or Trace messages that will be used by the application. **Tags:**atp.Status=draft
modePort Annotation	ModePortAnnotation	*	aggr	Annotations on this mode port.
nvDataPort Annotation	NvDataPortAnnotation	*	aggr	Annotations on this non voilatile data port.
parameterPort Annotation	ParameterPort Annotation	*	aggr	Annotations on this parameter port.
portPrototype Props	PortPrototypeProps	0..1	aggr	This attribute allows for the definition of further qualification of the semantics of a PortPrototype. **Tags:**atp.Status=draft

▽

△

Class	***PortPrototype*** (abstract)			
senderReceiver Annotation	SenderReceiver Annotation	*	aggr	Collection of annotations of this ports sender/receiver communication.
triggerPort Annotation	TriggerPortAnnotation	*	aggr	Annotations on this trigger port.

Table A.36: PortPrototype

Class	**Process**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ExecutionManifest			

Note	This meta-class provides information required to execute the referenced executable.
	Tags:
	atp.Status=draft
	atp.recommendedPackage=Processes			

Base	*ARElement*, *ARObject*, *AbstractExecutionContext*, *AtpClassifier*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadablePackageElement*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
design	ProcessDesign	0..1	ref	This reference represents the identification of the design-time representation for the Process that owns the reference. **Tags:**atp.Status=draft
deterministic Client	DeterministicClient	0..1	ref	This reference adds further execution characteristics for deterministic clients. **Tags:**atp.Status=draft
executable	Executable	0..1	ref	Reference to executable that is executed in the process. **Stereotypes:** atpUriDef **Tags:**atp.Status=draft
functionCluster Affiliation	String	0..1	attr	This attribute specifies which functional cluster the process is affiliated with. **Tags:**atp.Status=draft
numberOf RestartAttempts	PositiveInteger	0..1	attr	This attribute defines how often a process shall be restarted if the start fails. numberOfRestartAttempts = "0" OR Attribute not existing, start once numberOfRestartAttempts = "1", start a second time **Tags:**atp.Status=draft
preMapping	Boolean	0..1	attr	This attribute describes whether the executable is preloaded into the memory. **Tags:**atp.Status=draft
processState Machine	ModeDeclarationGroup Prototype	0..1	aggr	Set of Process States that are defined for the process. **Tags:**atp.Status=draft
securityEvent	SecurityEventDefinition	*	ref	The reference identifies the collection of SecurityEvents that can be reported by the enclosing SoftwareCluster. **Stereotypes:** atpSplitable; atpUriDef **Tags:** atp.Splitkey=securityEvent atp.Status=draft
stateDependent StartupConfig	StateDependentStartup Config	*	aggr	Applicable startup configurations. **Tags:**atp.Status=draft

Table A.37: Process

Class	**ProcessToMachineMapping**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::MachineManifest			
Note	This meta-class has the ability to associate a Process with a Machine. This relation involves the definition of further properties, e.g. timeouts. **Tags:**atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
design	ProcessDesignToMachineDesignMapping	0..1	ref	This reference represents the identification of the design-time representation for the ProcessToMachineMapping that owns the reference. **Tags:**atp.Status=draft
machine	Machine	0..1	ref	This reference identifies the Machine in the context of the ProcessToMachineMapping. **Tags:**atp.Status=draft
nonOsModuleInstantiation	NonOsModuleInstantiation	0..1	ref	This supports the optional case that the process represents a platform module. **Tags:**atp.Status=draft
persistencyCentralStorageURI	UriString	0..1	attr	This attribute identifies a central place for the mapped Process to store the list of available storages and version information. **Tags:**atp.Status=draft
process	Process	1	ref	This reference identifies the Process in the context of the ProcessToMachineMapping. **Tags:**atp.Status=draft
shallNotRunOn	ProcessorCore	*	ref	This reference indicates a collection of cores onto which the mapped process shall not be executing. **Tags:**atp.Status=draft
shallRunOn	ProcessorCore	*	ref	This reference indicates a collection of cores onto which the mapped process shall be executing. **Tags:**atp.Status=draft

Table A.38: ProcessToMachineMapping

Class	**RPortPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Component port requiring a certain port interface.			
Base	*ARObject*, *AbstractRequiredPortPrototype*, *AtpBlueprintable*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *PortPrototype*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
requiredInterface	PortInterface	0..1	tref	The interface that this port requires. **Stereotypes:** isOfType

Table A.39: RPortPrototype

Class	***Referrable*** (abstract)
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable
Note	Instances of this class can be referred to by their identifier (while adhering to namespace borders).
Base	*ARObject*

▽

△

Class	***Referrable*** (abstract)			
Subclasses	*AtpDefinition*, BswDistinguishedPartition, *BswModuleCallPoint*, BswModuleClientServerEntry, BswVariableAccess, CouplingPortTrafficClassAssignment, *CppImplementationDataTypeContextTarget*, *DiagnosticEnvModeElement*, EthernetPriorityRegeneration, ExclusiveAreaNestingOrder, *HwDescriptionEntity*, *ImplementationProps*, ModeTransition, *MultilanguageReferrable*, NmNetworkHandle, PncMappingIdent, *SingleLanguageReferrable*, SoConIPduIdentifier, SocketConnectionBundle, SomeipRequiredEventGroup, TimeSyncServerConfiguration, TpConnectionIdent			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
shortName	Identifier	1	attr	This specifies an identifying shortName for the object. It needs to be unique within its context and is intended for humans but even more for technical reference. **Stereotypes:** atpIdentityContributor **Tags:** xml.enforceMinMultiplicity=true xml.sequenceOffset=-100
shortNameFragment	ShortNameFragment	*	aggr	This specifies how the Referrable.shortName is composed of several shortNameFragments. **Tags:**xml.sequenceOffset=-90

Table A.40: Referrable

Class	**SoftwarePackage**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::SoftwareDistribution			

Note	This meta-class represents the ability to formalize the content of a software package.
	
	Tags:
	atp.Status=draft
	atp.recommendedPackage=SoftwarePackages			

Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
actionType	SoftwarePackageActionTypeEnum	1	attr	This attribute defines the action to be taken in the step of processing the enclosing SoftwarePackage. **Tags:**atp.Status=draft
activationAction	SoftwarePackageActivationActionEnum	0..1	attr	This attribute governs the action to be taken after the installation of the SoftwareCluster completed. **Tags:**atp.Status=draft
compressedSoftwarePackageSize	PositiveInteger	1	attr	This size represents the size of the compressed Software Package. **Tags:**atp.Status=draft
deltaPackageApplicableVersion	StrongRevisionLabelString	0..1	attr	This attribute identifies the version of the included SoftwareCluster for which the enclosing SoftwarePackage can be used as a delta update **Tags:**atp.Status=draft
estimatedDurationOfOperation	TimeValue	0..1	attr	This attribute provides an estimation about how long the operation of the SoftwarePackage is going to take. **Tags:**atp.Status=draft
minimumSupportedUcmVersion	RevisionLabelString	1	attr	This attribute identifies the minimum supported version of the UCM for this SoftwarePackage. **Tags:**atp.Status=draft
packagerId	PositiveInteger	1	attr	This attribute identifies Id of the organization that provides the packager generating the SoftwarePackage. **Tags:**atp.Status=draft

▽

△

Class	**SoftwarePackage**			
packager Signature	CryptoService Certificate	1	ref	This reference identifies the certificate that represents the packager's signature. **Tags:**atp.Status=draft
purposeOf Update	Documentation	0..1	ref	The referenced Documentation is supposed to provide a description of the purpose of the update. **Tags:**atp.Status=draft
softwareCluster	SoftwareCluster	1	ref	This reference identifies the SoftwareCluster that belongs to the SoftwarePackage. The nature of this relation is actually more like an aggregation than a reference. But the relation is still modelled as a reference because two ARElements cannot aggregate each other. **Tags:**atp.Status=draft
uncompressed SoftwareCluster Size	PositiveInteger	1	attr	This attribute gives an indication about the storage that has to be available on the target. **Tags:**atp.Status=draft

Table A.41: SoftwarePackage

Class	**StdCppImplementationDataType**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::CppImplementationDataType			

Note	This meta-class represents the way to specify a data type definition that is taken as the basis for a C++ language binding to a C++ Standard Library feature.
	Tags:
	atp.Status=draft
	atp.recommendedPackage=CppImplementationDataTypes			

Base	*ARElement*, *ARObject*, *AbstractImplementationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *CppImplementationDataType*, *CppImplementationDataTypeContextTarget*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.42: StdCppImplementationDataType

Primitive	**StrongRevisionLabelString**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes

Note	This primitive represents a revision label which identifies an object under version control. It represents a pattern which requires three integer numbers separated by a dot, representing from left to right Major Version, MinorVersion, PatchVersion and additional labels for pre-release version and build metadata.
	Legal patterns are for example: 1.0.0-alpha+001 1.0.0+20130313144700 1.0.0-beta+exp.sha.5114f85
	Tags:
	atp.Status=draft
	xml.xsd.customType=STRONG-REVISION-LABEL-STRING
	xml.xsd.pattern=(0|[1-9]\d*)\.(0|[1-9]\d*)\.(0|[1-9]\d*)(-((0|[1-9]\d*|\d*[a-zA-Z-][0-9a-zA-Z-]*)(\.(0|[1-9]\d*|\d*[a-zA-Z-][0-9a-zA-Z-]*))*))?(\+([0-9a-zA-Z-]+(\.[0-9a-zA-Z-]+)*))?
	xml.xsd.type=string

Table A.43: StrongRevisionLabelString

B Platform Extension API (normative)

The `Persistency` cluster does not provide a platform extension API. The latter would be required to defined a plugin interface for platform specific extensions of the `Persistency`.

C Interfaces to Other Functional Clusters (informative)

The `Persistency` cluster does not provide any direct interfaces to other functional clusters. Other functional clusters may use the APIs of `Persistency` just like the application.

D History of Constraints and Specification Items

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

D.1 Constraint and Specification Item History of this Document According to AUTOSAR Release 17-03

D.1.1 Added Traceables in 17-03

[SWS_PER_00002] [SWS_PER_00003] [SWS_PER_00004] [SWS_PER_00005] [SWS_PER_00006] [SWS_PER_00007] [SWS_PER_00010] [SWS_PER_00011] [SWS_PER_00012] [SWS_PER_00013] [SWS_PER_00014] [SWS_PER_00015] [SWS_PER_00016] [SWS_PER_00017] [SWS_PER_00018] [SWS_PER_00019] [SWS_PER_00020] [SWS_PER_00021] [SWS_PER_00022] [SWS_PER_00023] [SWS_PER_00024] [SWS_PER_00025] [SWS_PER_00026] [SWS_PER_00027] [SWS_PER_00028] [SWS_PER_00029] [SWS_PER_00040] [SWS_PER_00041] [SWS_PER_00042] [SWS_PER_00043] [SWS_PER_00044] [SWS_PER_00045] [SWS_PER_00046] [SWS_PER_00047] [SWS_PER_00048] [SWS_PER_00049] [SWS_PER_00050] [SWS_PER_00051] [SWS_PER_00052] [SWS_PER_00053] [SWS_PER_00054] [SWS_PER_00055] [SWS_PER_00056] [SWS_PER_00057] [SWS_PER_00058] [SWS_PER_00059] [SWS_PER_00060] [SWS_PER_00061] [SWS_PER_00062] [SWS_PER_00066] [SWS_PER_00069] [SWS_PER_00070] [SWS_PER_00071] [SWS_PER_00072] [SWS_PER_00073] [SWS_PER_00074] [SWS_PER_00075] [SWS_PER_00076] [SWS_PER_00077] [SWS_PER_00078]

D.1.2 Changed Traceables in 17-03

none

D.1.3 Deleted Traceables in 17-03

none

D.2 Constraint and Specification Item History of this Document According to AUTOSAR Release 17-10

D.2.1 Added Traceables in 17-10

[SWS_PER_00008] [SWS_PER_00100] [SWS_PER_00101] [SWS_PER_00102] [SWS_PER_00103] [SWS_PER_00104] [SWS_PER_00105] [SWS_PER_00106] [SWS_PER_00107] [SWS_PER_00108] [SWS_PER_00109] [SWS_PER_00110] [SWS_PER_00111] [SWS_PER_00112] [SWS_PER_00113] [SWS_PER_00114] [SWS_PER_00115] [SWS_PER_00116] [SWS_PER_00117] [SWS_PER_00118] [SWS_PER_00119] [SWS_PER_00120] [SWS_PER_00121] [SWS_PER_00122] [SWS_PER_00123] [SWS_PER_00124] [SWS_PER_00125] [SWS_PER_00126] [SWS_PER_00127] [SWS_PER_00128] [SWS_PER_00129] [SWS_PER_00130] [SWS_PER_00131] [SWS_PER_00132] [SWS_PER_00133] [SWS_PER_00134] [SWS_PER_00140] [SWS_PER_00141] [SWS_PER_00142] [SWS_PER_00143] [SWS_PER_00144] [SWS_PER_00145] [SWS_PER_00150] [SWS_PER_00151] [SWS_PER_00152] [SWS_PER_00153] [SWS_PER_00154] [SWS_PER_00155] [SWS_PER_00156] [SWS_PER_00157] [SWS_PER_00160] [SWS_PER_00161] [SWS_PER_00200] [SWS_PER_00201] [SWS_PER_00210] [SWS_PER_00211] [SWS_PER_00220] [SWS_PER_00221] [SWS_PER_00222] [SWS_PER_00500]

D.2.2 Changed Traceables in 17-10

[SWS_PER_00003] [SWS_PER_00004] [SWS_PER_00010] [SWS_PER_00013] [SWS_PER_00014] [SWS_PER_00016] [SWS_PER_00017] [SWS_PER_00041] [SWS_PER_00042] [SWS_PER_00043] [SWS_PER_00044] [SWS_PER_00046] [SWS_PER_00047] [SWS_PER_00048] [SWS_PER_00049] [SWS_PER_00050] [SWS_PER_00051] [SWS_PER_00060] [SWS_PER_00061] [SWS_PER_00076]

D.2.3 Deleted Traceables in 17-10

[SWS_PER_00011] [SWS_PER_00021] [SWS_PER_00022] [SWS_PER_00023] [SWS_PER_00024] [SWS_PER_00025] [SWS_PER_00026] [SWS_PER_00027] [SWS_PER_00028] [SWS_PER_00029] [SWS_PER_00040] [SWS_PER_00045] [SWS_PER_00053] [SWS_PER_00054] [SWS_PER_00055] [SWS_PER_00056] [SWS_PER_00057] [SWS_PER_00058] [SWS_PER_00059] [SWS_PER_00062] [SWS_PER_00066] [SWS_PER_00069] [SWS_PER_00070] [SWS_PER_00071] [SWS_PER_00072] [SWS_PER_00073] [SWS_PER_00074] [SWS_PER_00075] [SWS_PER_00077] [SWS_PER_00078]

D.3 Constraint and Specification Item History of this Document According to AUTOSAR Release 18-03

D.3.1 Added Traceables in 18-03

[SWS_PER_00080] [SWS_PER_00146] [SWS_PER_00147] [SWS_PER_00148] [SWS_PER_00162] [SWS_PER_00163] [SWS_PER_00164] [SWS_PER_00165] [SWS_PER_00166] [SWS_PER_00167] [SWS_PER_00168] [SWS_PER_00169] [SWS_PER_00170] [SWS_PER_00171] [SWS_PER_00172] [SWS_PER_00173] [SWS_PER_00174] [SWS_PER_00175] [SWS_PER_00176] [SWS_PER_00180] [SWS_PER_00181] [SWS_PER_00182] [SWS_PER_00250] [SWS_PER_00251] [SWS_PER_00252] [SWS_PER_00253] [SWS_PER_00254] [SWS_PER_00255] [SWS_PER_00256] [SWS_PER_00257] [SWS_PER_00258] [SWS_PER_00259] [SWS_PER_00260] [SWS_PER_00261] [SWS_PER_00262] [SWS_PER_00264] [SWS_PER_00265] [SWS_PER_00266] [SWS_PER_00267] [SWS_PER_00268] [SWS_PER_00269] [SWS_PER_00270] [SWS_PER_00271] [SWS_PER_00272] [SWS_PER_00273] [SWS_PER_00274] [SWS_PER_00275] [SWS_PER_00276] [SWS_PER_00277] [SWS_PER_00278] [SWS_PER_00279] [SWS_PER_00280] [SWS_PER_00281] [SWS_PER_00282] [SWS_PER_00283] [SWS_PER_00284] [SWS_PER_00285] [SWS_PER_00300] [SWS_PER_00301] [SWS_PER_00302] [SWS_PER_00303] [SWS_PER_00304] [SWS_PER_UNUSED]

D.3.2 Changed Traceables in 18-03

[SWS_PER_00004] [SWS_PER_00113] [SWS_PER_00114] [SWS_PER_00115] [SWS_PER_00132] [SWS_PER_00133] [SWS_PER_00134] [SWS_PER_00201] [SWS_PER_00220] [SWS_PER_00500]

D.3.3 Deleted Traceables in 18-03

[SWS_PER_00003] [SWS_PER_00005] [SWS_PER_00006] [SWS_PER_00007] [SWS_PER_00008] [SWS_PER_00010] [SWS_PER_00012] [SWS_PER_00013] [SWS_PER_00014] [SWS_PER_00015] [SWS_PER_00016] [SWS_PER_00017] [SWS_PER_00018] [SWS_PER_00019] [SWS_PER_00020] [SWS_PER_00051] [SWS_PER_00060] [SWS_PER_00061] [SWS_PER_00076] [SWS_PER_00100] [SWS_PER_00101] [SWS_PER_00102] [SWS_PER_00103] [SWS_PER_00104] [SWS_PER_00105] [SWS_PER_00109] [SWS_PER_00117] [SWS_PER_00118] [SWS_PER_00120] [SWS_PER_00121] [SWS_PER_00123] [SWS_PER_00150] [SWS_PER_00151] [SWS_PER_00152] [SWS_PER_00153] [SWS_PER_00154] [SWS_PER_00155] [SWS_PER_00156] [SWS_PER_00157]

D.4 Constraint and Specification Item History of this Document According to AUTOSAR Release 18-10

D.4.1 Added Traceables in 18-10

[SWS_PER_00309] [SWS_PER_00311] [SWS_PER_00312] [SWS_PER_00313] [SWS_PER_00314] [SWS_PER_00315] [SWS_PER_00316] [SWS_PER_00317] [SWS_PER_00318] [SWS_PER_00319] [SWS_PER_00320] [SWS_PER_00321] [SWS_PER_00322] [SWS_PER_00323] [SWS_PER_00324] [SWS_PER_00325] [SWS_PER_00326] [SWS_PER_00327] [SWS_PER_00328] [SWS_PER_00329] [SWS_PER_00330] [SWS_PER_00331] [SWS_PER_00332] [SWS_PER_00333] [SWS_PER_00334] [SWS_PER_00335] [SWS_PER_00336] [SWS_PER_00337] [SWS_PER_00338] [SWS_PER_00339] [SWS_PER_00340] [SWS_PER_00341] [SWS_PER_00342] [SWS_PER_00343] [SWS_PER_00344] [SWS_PER_00345] [SWS_PER_00346] [SWS_PER_00347] [SWS_PER_00348] [SWS_PER_NA]

D.4.2 Changed Traceables in 18-10

[SWS_PER_00042] [SWS_PER_00043] [SWS_PER_00044] [SWS_PER_00046] [SWS_PER_00047] [SWS_PER_00048] [SWS_PER_00049] [SWS_PER_00050] [SWS_PER_00052] [SWS_PER_00106] [SWS_PER_00107] [SWS_PER_00108] [SWS_PER_00110] [SWS_PER_00111] [SWS_PER_00112] [SWS_PER_00113] [SWS_PER_00114] [SWS_PER_00115] [SWS_PER_00116] [SWS_PER_00119] [SWS_PER_00122] [SWS_PER_00124] [SWS_PER_00125] [SWS_PER_00126] [SWS_PER_00127] [SWS_PER_00128] [SWS_PER_00140] [SWS_PER_00141] [SWS_PER_00142] [SWS_PER_00143] [SWS_PER_00144] [SWS_PER_00145] [SWS_PER_00147] [SWS_PER_00160] [SWS_PER_00161] [SWS_PER_00163] [SWS_PER_00164] [SWS_PER_00165] [SWS_PER_00166] [SWS_PER_00180] [SWS_PER_00181] [SWS_PER_00182] [SWS_PER_00210] [SWS_PER_00211]

D.4.3 Deleted Traceables in 18-10

[SWS_PER_00004] [SWS_PER_00041] [SWS_PER_00080] [SWS_PER_00129] [SWS_PER_00130] [SWS_PER_00131] [SWS_PER_00132] [SWS_PER_00133] [SWS_PER_00134] [SWS_PER_00148] [SWS_PER_00169] [SWS_PER_00170] [SWS_PER_00171] [SWS_PER_00172] [SWS_PER_00173] [SWS_PER_00174] [SWS_PER_00175] [SWS_PER_00176] [SWS_PER_00200] [SWS_PER_00201] [SWS_PER_00220] [SWS_PER_00250] [SWS_PER_00500] [SWS_PER_UNUSED]

D.5 Constraint and Specification Item History of this Document According to AUTOSAR Release 19-03

D.5.1 Added Traceables in 19-03

[SWS_PER_00349] [SWS_PER_00350] [SWS_PER_00351] [SWS_PER_00352] [SWS_PER_00353] [SWS_PER_00354] [SWS_PER_00355] [SWS_PER_00356] [SWS_PER_00357] [SWS_PER_00358] [SWS_PER_00359] [SWS_PER_00360] [SWS_PER_00361] [SWS_PER_00362] [SWS_PER_00363] [SWS_PER_00364] [SWS_PER_00365] [SWS_PER_00366] [SWS_PER_00367] [SWS_PER_00368] [SWS_PER_00369] [SWS_PER_00370] [SWS_PER_00371] [SWS_PER_00372] [SWS_PER_00373] [SWS_PER_00374] [SWS_PER_00375] [SWS_PER_00376] [SWS_PER_00377] [SWS_PER_00378] [SWS_PER_00379] [SWS_PER_00380] [SWS_PER_00381] [SWS_PER_00382] [SWS_PER_00383] [SWS_PER_00384] [SWS_PER_00385] [SWS_PER_00386] [SWS_PER_00387] [SWS_PER_00388] [SWS_PER_00389] [SWS_PER_00390] [SWS_PER_00391] [SWS_PER_00392] [SWS_PER_00393] [SWS_PER_00394] [SWS_PER_00395] [SWS_PER_00396] [SWS_PER_00397] [SWS_PER_CONSTR_00001] [SWS_PER_CONSTR_00002] [SWS_PER_CONSTR_00003] [SWS_PER_CONSTR_00004]

D.5.2 Changed Traceables in 19-03

[SWS_PER_00042] [SWS_PER_00043] [SWS_PER_00044] [SWS_PER_00046] [SWS_PER_00047] [SWS_PER_00048] [SWS_PER_00049] [SWS_PER_00052] [SWS_PER_00110] [SWS_PER_00111] [SWS_PER_00112] [SWS_PER_00113] [SWS_PER_00114] [SWS_PER_00115] [SWS_PER_00116] [SWS_PER_00119] [SWS_PER_00127] [SWS_PER_00128] [SWS_PER_00144] [SWS_PER_00145] [SWS_PER_00251] [SWS_PER_00252] [SWS_PER_00253] [SWS_PER_00254] [SWS_PER_00265] [SWS_PER_00266] [SWS_PER_00267] [SWS_PER_00275] [SWS_PER_00277] [SWS_PER_00281] [SWS_PER_00283] [SWS_PER_00304] [SWS_PER_00311] [SWS_PER_00312] [SWS_PER_00313] [SWS_PER_00314] [SWS_PER_00315] [SWS_PER_00322] [SWS_PER_00323] [SWS_PER_00326] [SWS_PER_00327] [SWS_PER_00328] [SWS_PER_00329] [SWS_PER_00330] [SWS_PER_00332] [SWS_PER_00333] [SWS_PER_00334] [SWS_PER_00335] [SWS_PER_00336] [SWS_PER_00337] [SWS_PER_00338] [SWS_PER_00340]

D.5.3 Deleted Traceables in 19-03

[SWS_PER_00160] [SWS_PER_00161] [SWS_PER_00255] [SWS_PER_00256] [SWS_PER_00257] [SWS_PER_00258] [SWS_PER_00259] [SWS_PER_00260] [SWS_PER_00261] [SWS_PER_00262] [SWS_PER_00264] [SWS_PER_00268] [SWS_PER_00269] [SWS_PER_00270] [SWS_PER_00271] [SWS_PER_00272] [SWS_PER_00273] [SWS_PER_00274] [SWS_PER_00276] [SWS_PER_00278]

[SWS_PER_00279] [SWS_PER_00280] [SWS_PER_00282] [SWS_PER_00284] [SWS_PER_00285] [SWS_PER_00300] [SWS_PER_00301] [SWS_PER_00316]

D.6 Constraint and Specification Item History of this Document According to AUTOSAR Release R19-11

D.6.1 Added Traceables in R19-11

[SWS_PER_00398] [SWS_PER_00399] [SWS_PER_00400] [SWS_PER_00401] [SWS_PER_00402] [SWS_PER_00403] [SWS_PER_00404] [SWS_PER_00405] [SWS_PER_00406] [SWS_PER_00407] [SWS_PER_00408] [SWS_PER_00409] [SWS_PER_00410]

D.6.2 Changed Traceables in R19-11

[SWS_PER_00049] [SWS_PER_00113] [SWS_PER_00114] [SWS_PER_00115] [SWS_PER_00144] [SWS_PER_00145] [SWS_PER_00146] [SWS_PER_00147] [SWS_PER_00163] [SWS_PER_00164] [SWS_PER_00303] [SWS_PER_00317] [SWS_PER_00318] [SWS_PER_00319] [SWS_PER_00323] [SWS_PER_00327] [SWS_PER_00345] [SWS_PER_00351] [SWS_PER_00365] [SWS_PER_00368] [SWS_PER_00370] [SWS_PER_00372]

D.6.3 Deleted Traceables in R19-11

[SWS_PER_00044] [SWS_PER_CONSTR_00001]

D.7 Constraint and Specification Item History of this Document According to AUTOSAR Release R20-11

D.7.1 Added Traceables in R20-11

[SWS_PER_00411] [SWS_PER_00412] [SWS_PER_00413] [SWS_PER_00414] [SWS_PER_00415] [SWS_PER_00416] [SWS_PER_00417] [SWS_PER_00418] [SWS_PER_00419] [SWS_PER_00420] [SWS_PER_00421] [SWS_PER_00422] [SWS_PER_00423] [SWS_PER_00424] [SWS_PER_00425] [SWS_PER_00426] [SWS_PER_00427] [SWS_PER_00428] [SWS_PER_00429] [SWS_PER_00430] [SWS_PER_00431] [SWS_PER_00432] [SWS_PER_00433] [SWS_PER_00434] [SWS_PER_00435] [SWS_PER_00436] [SWS_PER_00437] [SWS_PER_00438] [SWS_PER_00439] [SWS_PER_00440] [SWS_PER_00441] [SWS_PER_00442] [SWS_PER_00443] [SWS_PER_00444] [SWS_PER_00445] [SWS_PER_00446]

[SWS_PER_00447] [SWS_PER_00448] [SWS_PER_00449] [SWS_PER_00450] [SWS_PER_00451]

D.7.2 Changed Traceables in R20-11

[SWS_PER_00042] [SWS_PER_00043] [SWS_PER_00046] [SWS_PER_00047] [SWS_PER_00048] [SWS_PER_00049] [SWS_PER_00052] [SWS_PER_00107] [SWS_PER_00110] [SWS_PER_00111] [SWS_PER_00112] [SWS_PER_00113] [SWS_PER_00114] [SWS_PER_00115] [SWS_PER_00116] [SWS_PER_00119] [SWS_PER_00122] [SWS_PER_00125] [SWS_PER_00144] [SWS_PER_00146] [SWS_PER_00147] [SWS_PER_00162] [SWS_PER_00163] [SWS_PER_00164] [SWS_PER_00165] [SWS_PER_00166] [SWS_PER_00167] [SWS_PER_00168] [SWS_PER_00210] [SWS_PER_00211] [SWS_PER_00251] [SWS_PER_00252] [SWS_PER_00265] [SWS_PER_00266] [SWS_PER_00267] [SWS_PER_00275] [SWS_PER_00277] [SWS_PER_00281] [SWS_PER_00283] [SWS_PER_00304] [SWS_PER_00311] [SWS_PER_00312] [SWS_PER_00317] [SWS_PER_00318] [SWS_PER_00319] [SWS_PER_00332] [SWS_PER_00333] [SWS_PER_00334] [SWS_PER_00335] [SWS_PER_00336] [SWS_PER_00337] [SWS_PER_00338] [SWS_PER_00339] [SWS_PER_00340] [SWS_PER_00342] [SWS_PER_00343] [SWS_PER_00356] [SWS_PER_00357] [SWS_PER_00358] [SWS_PER_00365] [SWS_PER_00375] [SWS_PER_00376] [SWS_PER_00377] [SWS_PER_00378] [SWS_PER_00379] [SWS_PER_00380] [SWS_PER_00383] [SWS_PER_00385] [SWS_PER_00388] [SWS_PER_00389] [SWS_PER_00390] [SWS_PER_00391] [SWS_PER_00392] [SWS_PER_00393] [SWS_PER_00394] [SWS_PER_00395] [SWS_PER_00396] [SWS_PER_00405] [SWS_PER_00406] [SWS_PER_00407] [SWS_PER_00409] [SWS_PER_CONSTR_00004]

D.7.3 Deleted Traceables in R20-11

[SWS_PER_00106] [SWS_PER_00108] [SWS_PER_00124] [SWS_PER_00126] [SWS_PER_00127] [SWS_PER_00128] [SWS_PER_00140] [SWS_PER_00141] [SWS_PER_00142] [SWS_PER_00143] [SWS_PER_00145] [SWS_PER_00180] [SWS_PER_00181] [SWS_PER_00182] [SWS_PER_00341] [SWS_PER_00344] [SWS_PER_00345] [SWS_PER_00346] [SWS_PER_00347] [SWS_PER_00348] [SWS_PER_00349] [SWS_PER_00366] [SWS_PER_00381] [SWS_PER_00404] [SWS_PER_CONSTR_00002]

D.8 Constraint and Specification Item History of this Document According to AUTOSAR Release R21-11

D.8.1 Added Traceables in R21-11

[SWS_PER_00452] [SWS_PER_00453] [SWS_PER_00454] [SWS_PER_00455] [SWS_PER_00456] [SWS_PER_00457] [SWS_PER_00458] [SWS_PER_00459] [SWS_PER_00460] [SWS_PER_00461] [SWS_PER_00462] [SWS_PER_00463] [SWS_PER_00464] [SWS_PER_00465] [SWS_PER_00466] [SWS_PER_00467] [SWS_PER_00468] [SWS_PER_00469] [SWS_PER_00470] [SWS_PER_00471] [SWS_PER_00472] [SWS_PER_00473] [SWS_PER_00474] [SWS_PER_00475] [SWS_PER_00476] [SWS_PER_00477] [SWS_PER_00478] [SWS_PER_00479] [SWS_PER_00480] [SWS_PER_00481] [SWS_PER_00482] [SWS_PER_00483] [SWS_PER_00484] [SWS_PER_00485] [SWS_PER_00486] [SWS_PER_00487] [SWS_PER_00488] [SWS_PER_00489] [SWS_PER_00490] [SWS_PER_00491] [SWS_PER_00492] [SWS_PER_00493] [SWS_PER_00494] [SWS_PER_00495] [SWS_PER_00496] [SWS_PER_00497] [SWS_PER_00498] [SWS_PER_00499] [SWS_PER_00501] [SWS_PER_00502] [SWS_PER_00503] [SWS_PER_00504] [SWS_PER_00505] [SWS_PER_00506] [SWS_PER_00507] [SWS_PER_00508] [SWS_PER_00509] [SWS_PER_00510] [SWS_PER_00511] [SWS_PER_00512] [SWS_PER_00513] [SWS_PER_00514] [SWS_PER_00515] [SWS_PER_00516] [SWS_PER_00517] [SWS_PER_00518] [SWS_PER_00519] [SWS_PER_00520] [SWS_PER_00521] [SWS_PER_00522] [SWS_PER_00523] [SWS_PER_00524] [SWS_PER_00525] [SWS_PER_00526] [SWS_PER_00527] [SWS_PER_00528] [SWS_PER_00529] [SWS_PER_00530] [SWS_PER_00531] [SWS_PER_00532] [SWS_PER_00533] [SWS_PER_00534] [SWS_PER_00535] [SWS_PER_CONSTR_00001] [SWS_PER_CONSTR_00002]

D.8.2 Changed Traceables in R21-11

[SWS_PER_00042] [SWS_PER_00043] [SWS_PER_00046] [SWS_PER_00047] [SWS_PER_00048] [SWS_PER_00049] [SWS_PER_00050] [SWS_PER_00052] [SWS_PER_00107] [SWS_PER_00110] [SWS_PER_00111] [SWS_PER_00112] [SWS_PER_00113] [SWS_PER_00114] [SWS_PER_00115] [SWS_PER_00116] [SWS_PER_00119] [SWS_PER_00122] [SWS_PER_00125] [SWS_PER_00144] [SWS_PER_00146] [SWS_PER_00147] [SWS_PER_00162] [SWS_PER_00163] [SWS_PER_00164] [SWS_PER_00165] [SWS_PER_00166] [SWS_PER_00167] [SWS_PER_00168] [SWS_PER_00210] [SWS_PER_00211] [SWS_PER_00221] [SWS_PER_00251] [SWS_PER_00252] [SWS_PER_00265] [SWS_PER_00275] [SWS_PER_00277] [SWS_PER_00281] [SWS_PER_00283] [SWS_PER_00311] [SWS_PER_00312] [SWS_PER_00313] [SWS_PER_00314] [SWS_PER_00315] [SWS_PER_00317] [SWS_PER_00318] [SWS_PER_00319] [SWS_PER_00320] [SWS_PER_00321] [SWS_PER_00322] [SWS_PER_00323] [SWS_PER_00324] [SWS_PER_00325] [SWS_PER_00326] [SWS_PER_00327] [SWS_PER_00328] [SWS_PER_00329] [SWS_PER_00330] [SWS_PER_00331] [SWS_PER_00332]

[SWS_PER_00333] [SWS_PER_00334] [SWS_PER_00335] [SWS_PER_00336] [SWS_PER_00337] [SWS_PER_00338] [SWS_PER_00339] [SWS_PER_00340] [SWS_PER_00342] [SWS_PER_00343] [SWS_PER_00350] [SWS_PER_00351] [SWS_PER_00352] [SWS_PER_00354] [SWS_PER_00355] [SWS_PER_00356] [SWS_PER_00357] [SWS_PER_00358] [SWS_PER_00359] [SWS_PER_00360] [SWS_PER_00361] [SWS_PER_00362] [SWS_PER_00363] [SWS_PER_00364] [SWS_PER_00365] [SWS_PER_00367] [SWS_PER_00368] [SWS_PER_00369] [SWS_PER_00370] [SWS_PER_00371] [SWS_PER_00372] [SWS_PER_00373] [SWS_PER_00374] [SWS_PER_00375] [SWS_PER_00376] [SWS_PER_00377] [SWS_PER_00378] [SWS_PER_00379] [SWS_PER_00380] [SWS_PER_00382] [SWS_PER_00383] [SWS_PER_00385] [SWS_PER_00386] [SWS_PER_00387] [SWS_PER_00391] [SWS_PER_00395] [SWS_PER_00396] [SWS_PER_00398] [SWS_PER_00399] [SWS_PER_00400] [SWS_PER_00401] [SWS_PER_00402] [SWS_PER_00403] [SWS_PER_00405] [SWS_PER_00406] [SWS_PER_00407] [SWS_PER_00410] [SWS_PER_00411] [SWS_PER_00412] [SWS_PER_00413] [SWS_PER_00414] [SWS_PER_00415] [SWS_PER_00416] [SWS_PER_00417] [SWS_PER_00418] [SWS_PER_00419] [SWS_PER_00420] [SWS_PER_00421] [SWS_PER_00422] [SWS_PER_00423] [SWS_PER_00424] [SWS_PER_00426] [SWS_PER_00427] [SWS_PER_00428] [SWS_PER_00429] [SWS_PER_00430] [SWS_PER_00431] [SWS_PER_00432] [SWS_PER_00433] [SWS_PER_00434] [SWS_PER_00435] [SWS_PER_00436] [SWS_PER_00437] [SWS_PER_00438] [SWS_PER_00441] [SWS_PER_00442] [SWS_PER_00443] [SWS_PER_00444] [SWS_PER_00445] [SWS_PER_00446] [SWS_PER_00447] [SWS_PER_00449] [SWS_PER_00450] [SWS_PER_00451]

D.8.3 Deleted Traceables in R21-11

[SWS_PER_00222] [SWS_PER_00397]

E Not Applicable Requirements

[SWS_PER_NA]{DRAFT} ⌈These requirements are not applicable to this specification.⌋*(RS_AP_00111, RS_AP_00114, RS_AP_00116, RS_AP_00124, RS_AP_00130)*