# Contents

1. Introduction and Functional Overview 9
2. Acronyms and Abbreviations 10
3. Related Documentation 12
   1. Input Documents & Related Standards and Norms 12
   2. Further Applicable Specifications 12
4. [Constraints and Assumptions](#_bookmark0) 13
   1. [Known Limitations](#_bookmark1) 13
   2. [Constraints on Configuration](#_bookmark2) 13
   3. [Direct Access to Storage Hardware](#_bookmark3) 13
5. [Dependencies to Other Functional Clusters](#_bookmark4) 14
   1. [Protocol Layer Dependencies](#_bookmark5) 14
6. Requirements Tracing 15
7. [Functional Specification](#_bookmark6) 28
   1. [The Architecture of Persistency](#_bookmark7) 28
      1. [Persistency in the Manifest](#_bookmark9) 28
      2. [Key-Value Storages in the Manifest](#_bookmark10) 30
      3. [File Storages in the Manifest](#_bookmark11) 30
   2. [General Features of Persistency](#_bookmark12) 32
      1. [Functional Cluster Lifecycle](#_bookmark13) 32
         1. [Initialization and Shutdown of Persistency](#_bookmark14) 32
      2. [Error Handling](#_bookmark17) 33
         1. [Handling of General Errors](#_bookmark18) 33
      3. [Parallel Access to Persistent Data](#_bookmark19) 36
      4. [Security Concepts](#_bookmark20) 38
      5. [Redundancy Concepts](#_bookmark21) 41
         1. [Redundancy Types](#_bookmark33) 45
      6. [Installation and Update of Persistent Data](#_bookmark37) 48
         1. [Installation of Persistent Data](#_bookmark42) 51
            1. [Installation of Key-Value Storage](#_bookmark44) 51
            2. [Installation of File Storage](#_bookmark50) 52
         2. [Update of Persistent Data](#_bookmark55) 53
            1. [Update of Key-Value Storage](#_bookmark58) 54
            2. [Update of File Storage](#_bookmark59) 56
         3. [Finalization of Persistent Data after Successful Update](#_bookmark60) 57
         4. [Roll-Back of Persistent Data after Failed Update](#_bookmark62) 57
         5. [Removal of Persistent Data](#_bookmark64) 57
      7. [Resource Management Concepts](#_bookmark65) 59
   3. [Key-Value Storage specific Features](#_bookmark67) 61
      1. [Supported Data Types in Key-Value Storages](#_bookmark68) 63
   4. [File Storage specific Features](#_bookmark69) 65
      1. [Access to Additional Information about Files](#_bookmark74) 70
8. [API Specification](#_bookmark75) 72
   1. [General Features of Persistency](#_bookmark76) 72
      1. [ara::core Types](#_bookmark77) 72
      2. Installation and Update of Persistent Data 73
         1. RegisterApplicationDataUpdateCallback 73
         2. UpdatePersistency 74
         3. ResetPersistency 75
      3. Redundancy Handling 76
         1. RecoveryReportKind 76
         2. RegisterRecoveryReportCallback 77
      4. Handle Classes 79
         1. SharedHandle Class 79
            1. SharedHandle::SharedHandle 79
            2. SharedHandle::operator= 80
            3. SharedHandle::operator bool 81
            4. SharedHandle::Operator-> 81
            5. SharedHandle::Operator\* 82
         2. UniqueHandle Class 83
            1. UniqueHandle::UniqueHandle 83
            2. UniqueHandle::operator= 84
            3. UniqueHandle::operator bool 85
            4. UniqueHandle::Operator-> 85
            5. UniqueHandle::Operator\* 86
      5. Errors 87
         1. PerErrc 87
         2. GetPerDomain 88
         3. MakeErrorCode 88
         4. PerException Class 89
            1. PerException::PerException 89
         5. PerErrorDomain Class 89
            1. PerErrorDomain::Errc 90
            2. PerErrorDomain::Exception 90
            3. PerErrorDomain::PerErrorDomain 91
            4. PerErrorDomain::Name 91
            5. PerErrorDomain::Message 91
            6. PerErrorDomain::ThrowAsException 92
   2. Key-Value Storage 93
      1. OpenKeyValueStorage 93
      2. RecoverKeyValueStorage 94
      3. ResetKeyValueStorage 95
      4. GetCurrentKeyValueStorageSize 96
      5. KeyValueStorage Class 96
         1. KeyValueStorage::KeyValueStorage 97
         2. KeyValueStorage::operator= 98
         3. KeyValueStorage::~KeyValueStorage 98
         4. KeyValueStorage::GetAllKeys 99
         5. KeyValueStorage::KeyExists 99
         6. KeyValueStorage::GetCurrentValueSize 100
         7. KeyValueStorage::GetValue 101
         8. KeyValueStorage::SetValue 102
         9. KeyValueStorage::RemoveKey 103
         10. KeyValueStorage::RecoverKey 104
         11. KeyValueStorage::ResetKey 105
         12. KeyValueStorage::RemoveAllKeys 106
         13. KeyValueStorage::SyncToStorage 107
         14. KeyValueStorage::DiscardPendingChanges 107
   3. File Storage 109
      1. OpenFileStorage 109
      2. RecoverAllFiles 110
      3. ResetAllFiles 111
      4. GetCurrentFileStorageSize 112
      5. OpenMode 112
      6. operator| for FileStorage::OpenMode 113
      7. operator|= for FileStorage::OpenMode 113
      8. FileCreationState 114
      9. FileModificationState 114
      10. FileInfo 115
          1. FileInfo.creationTime 115
          2. FileInfo.modificationTime 115
          3. FileInfo.accessTime 116
          4. FileInfo.fileCreationState 116
          5. FileInfo.fileModificationState 116
      11. FileStorage Class 117
          1. FileStorage::FileStorage 117
          2. FileStorage::operator= 118
          3. FileStorage::~FileStorage 118
          4. FileStorage::GetAllFileNames 119
          5. FileStorage::DeleteFile 119
          6. FileStorage::FileExists 120
          7. FileStorage::RecoverFile 121
          8. FileStorage::ResetFile 122
          9. FileStorage::GetCurrentFileSize 123
          10. FileStorage::GetFileInfo 123
          11. FileStorage::OpenFileReadWrite 124
          12. FileStorage::OpenFileReadOnly 127
          13. FileStorage::OpenFileWriteOnly 129
      12. Origin 132
      13. ReadAccessor Class 132
          1. ReadAccessor::ReadAccessor 133

|  |  |  |
| --- | --- | --- |
|  | 8.3.13.2 ReadAccessor::operator= . . . . . . . . . . . . . . . | 134 |
| 8.3.13.3 ReadAccessor::~ReadAccessor . . . . . . . . . . . | 134 |
| 8.3.13.4 ReadAccessor::PeekChar . . . . . . . . . . . . . . . | 135 |
| 8.3.13.5 ReadAccessor::PeekByte . . . . . . . . . . . . . . . | 135 |
| 8.3.13.6 ReadAccessor::GetChar . . . . . . . . . . . . . . . . | 136 |
| 8.3.13.7 ReadAccessor::GetByte . . . . . . . . . . . . . . . . | 136 |
| 8.3.13.8 ReadAccessor::ReadText . . . . . . . . . . . . . . . | 137 |
| 8.3.13.9 ReadAccessor::ReadBinary . . . . . . . . . . . . . . | 138 |
| 8.3.13.10 ReadAccessor::ReadLine . . . . . . . . . . . . . . . | 140 |
| 8.3.13.11 ReadAccessor::GetSize . . . . . . . . . . . . . . . . | 140 |
| 8.3.13.12 ReadAccessor::GetPosition . . . . . . . . . . . . . . | 141 |
| 8.3.13.13 ReadAccessor::SetPosition . . . . . . . . . . . . . . | 141 |
| 8.3.13.14 ReadAccessor::MovePosition . . . . . . . . . . . . . | 142 |
| 8.3.13.15 ReadAccessor::IsEof . . . . . . . . . . . . . . . . . . | 142 |
| 8.3.14 ReadWriteAccessor Class . . . . . . . . . . . . . . . . . . . | 143 |
| 8.3.14.1 ReadWriteAccessor::ReadWriteAccessor . . . . . . | 143 |
| 8.3.14.2 ReadWriteAccessor::SyncToFile . . . . . . . . . . . | 143 |
| 8.3.14.3 ReadWriteAccessor::SetFileSize . . . . . . . . . . . | 144 |
| 8.3.14.4 ReadWriteAccessor::WriteText . . . . . . . . . . . . | 145 |
| 8.3.14.5 ReadWriteAccessor::WriteBinary . . . . . . . . . . . | 146 |
| 8.3.14.6 ReadWriteAccessor::operator<< . . . . . . . . . . . | 146 |
| 9 | Service Interfaces | 148 |
| A | Mentioned Class Tables | 149 |
| B | Platform Extension API (normative) | 176 |
| C | Interfaces to Other Functional Clusters (informative) | 177 |
| D | History of Constraints and Specification Items | 178 |
|  | D.1 Constraint and Specification Item History of this Document According to AUTOSAR Release 17-03 . . . . . . . . . . . . . . . . . . . . . . . . | 178 |
|  | D.1.1 Added Traceables in 17-03 . . . . . . . . . . . . . . . . . . . | 178 |
|  | D.1.2 Changed Traceables in 17-03 . . . . . . . . . . . . . . . . . . | 178 |
|  | D.1.3 Deleted Traceables in 17-03 . . . . . . . . . . . . . . . . . .  D.2 Constraint and Specification Item History of this Document According to AUTOSAR Release 17-10 . . . . . . . . . . . . . . . . . . . . . . . . | 178  179 |
|  | D.2.1 Added Traceables in 17-10 . . . . . . . . . . . . . . . . . . . | 179 |
|  | D.2.2 Changed Traceables in 17-10 . . . . . . . . . . . . . . . . . . | 179 |
|  | D.2.3 Deleted Traceables in 17-10 . . . . . . . . . . . . . . . . . .  D.3 Constraint and Specification Item History of this Document According to AUTOSAR Release 18-03 . . . . . . . . . . . . . . . . . . . . . . . . | 179  180 |
|  | D.3.1 Added Traceables in 18-03 . . . . . . . . . . . . . . . . . . . | 180 |
|  | D.3.2 Changed Traceables in 18-03 . . . . . . . . . . . . . . . . . . | 180 |
|  | D.3.3 Deleted Traceables in 18-03 . . . . . . . . . . . . . . . . . . | 180 |

* 1. Constraint and Specification Item History of this Document According

to AUTOSAR Release 18-10 181

* + 1. Added Traceables in 18-10 181
    2. Changed Traceables in 18-10 181
    3. Deleted Traceables in 18-10 181
  1. Constraint and Specification Item History of this Document According

to AUTOSAR Release 19-03 182

* + 1. Added Traceables in 19-03 182
    2. Changed Traceables in 19-03 182
    3. Deleted Traceables in 19-03 182
  1. Constraint and Specification Item History of this Document According

to AUTOSAR Release R19-11 183

* + 1. Added Traceables in R19-11 183
    2. Changed Traceables in R19-11 183
    3. Deleted Traceables in R19-11 183
  1. Constraint and Specification Item History of this Document According

to AUTOSAR Release R20-11 183

* + 1. Added Traceables in R20-11 183
    2. Changed Traceables in R20-11 184
    3. Deleted Traceables in R20-11 184
  1. Constraint and Specification Item History of this Document According

to AUTOSAR Release R21-11 185

* + 1. Added Traceables in R21-11 185
    2. Changed Traceables in R21-11 185
    3. Deleted Traceables in R21-11 186
  1. Constraint and Specification Item History of this Document According

to AUTOSAR Release R22-11 186

* + 1. Added Traceables in R22-11 186
    2. Changed Traceables in R22-11 186
    3. Deleted Traceables in R22-11 187

E Not Applicable Requirements 188

# Constraints and Assumptions

## 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 pos- sible when a file has been opened with ara::per::FileStorage::Open- FileWriteOnly.

## Constraints on Configuration

There are several constraints on the Persistency configuration that need to be ob- served by the tooling which creates/processes this part of the Execution Manifest. These constraints are defined in [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 Per- sistency will not have to care about the wear leveling. This use case is therefore not described in any detail in this specification.

# Dependencies to Other Functional Clusters

## 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 ac- cesses services of the Cryptography [5].

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

# Functional Specification

## 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 de- picted in [Figure](#_bookmark8) [7.1](#_bookmark8). As shown there, an Adaptive Application can use a combi- nation of multiple Key-Value Storages and multiple File Storages. Of course, the same applies to other functional clusters using Persistency.



«process» Adaptive Application



«subsystem,entity» Persistency

«entity»

Key-Value Storage A

«entity»

File Storage C

+ Key\_A1 = Value\_A1

+ Key\_A2 = Value\_A2

+ Key\_An = Value\_An

+ File\_C1

+ File\_C2

+ File\_Cn

«entity»

Key-Value Storage B

«entity»

File Storage D

+ Key\_B1 = Value\_B1

+ Key\_B2 = Value\_B2

+ Key\_Bn = Value\_Bn

+ File\_D1

+ File\_D2

+ File\_Dn



**Figure 7.1:** **Typical usage of Persistency within an Adaptive Application**

Persistency can also be used directly by other Functional Clusters without involvement of the application. The library part of these Functional Clusters will use dedicated deployment information of Persistency, while Daemons of Func- tional Clusters can be modeled similarly to Adaptive Applications.

### Persistency in the Manifest

The Persistency usage of an Adaptive Application is modeled in the Exe- cution 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 Persis- tencyKeyValueStorageInterface and the PersistencyFileStorageInter- face, and the deployment information, aggregated by the PersistencyKeyVal- ueStorage and the PersistencyFileStorage. The latter is also used when the Persistency is accessed directly by another Functional Cluster.

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 either the identifier (the fully qualified shortName path) of a PortPrototype typed by a PersistencyInter- face, or the shortName path of a FunctionalClusterInteractsWithPersis- tencyDeploymentMapping. Both are then used as an ara::core::Instance- Specifier input parameter (see subsection 8.2.1 and subsection 8.3.1). Depend- ing on the nature of the PortPrototype or on the value of FunctionalCluster- InteractsWithPersistencyDeploymentMapping.persistencyAccess, the Key-Value Storage or File Storage will be accessible as:

**Read Only** if the PortPrototype is instantiated as RPortPrototype or Func- tionalClusterInteractsWithPersistencyDeploymentMapping. persistencyAccess is read, or

**Read/Write** if the PortPrototype is instantiated as PRPortPrototype or FunctionalClusterInteractsWithPersistencyDeploymentMapping. persistencyAccess is readWrite, or

**Write Only** if the PortPrototype is instantiated as PPortPrototype or Func- tionalClusterInteractsWithPersistencyDeploymentMapping. persistencyAccess is write.

In case of application access to Persistency, the manifest contains separate de- ployment data for each Process that references the Executable. The Process is bound to the deployment data by specialization of the class PersistencyPort- PrototypeToDeploymentMapping, 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 Stor- ages 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 Per- sistencyFileElement, and in PersistencyDeploymentElement for Persis- tencyKeyValuePair and PersistencyFile.

And the link between application design and deployment information, represented by PersistencyPortPrototypeToDeploymentMapping, is specialized as Persis- tencyPortPrototypeToKeyValueStorageMapping and PersistencyPort- PrototypeToFileStorageMapping.

### Key-Value Storages in the Manifest

Every Key-Value Storage is either represented by a PortPrototype typed by a PersistencyKeyValueStorageInterface in the application design for the re- spective AdaptiveApplicationSwComponentType, or by a FunctionalClus- terInteractsWithPersistencyDeploymentMapping, and in both cases 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 or Functional Cluster using the Persistency API (see subsubsection 8.2.5.8 and subsubsection 8.2.5.9).

A Key-Value Storage with predefined key-value pairs can be deployed with default data during installation or update of an Adaptive Application or Func- tional Cluster. This operation is (indirectly) triggered by the Update and Con- figuration Management [6] during installation or update using the deployment in- formation and data provided by the software package of the Adaptive Appli- cation or Functional Cluster. See [subsection](#_bookmark37) [7.2.6](#_bookmark37).

The link between application design and deployment information of a Key-

-Value Storage is represented by PersistencyPortPrototypeToKeyVal- ueStorageMapping, which refers to a PortPrototype typed by a Persis- tencyKeyValueStorageInterface, the corresponding PersistencyKeyVal- ueStorage, and a Process.

### File Storages in the Manifest

Every File Storage is represented by a PortPrototype typed by a Per- sistencyFileStorageInterface in the application design for the respective AdaptiveApplicationSwComponentType, or by a FunctionalClusterInter- actsWithPersistencyDeploymentMapping, and in both cases by a Persis- tencyFileStorage 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 Applica- tion or Functional Cluster 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 dur- ing installation or update. This operation is also (indirectly) triggered by the Up- date and Configuration Management [6]. All needed deployment information

and files come with the software package of the Adaptive Application or

Functional Cluster. See [subsection](#_bookmark37) [7.2.6](#_bookmark37).

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.

## General Features of Persistency

**[SWS\_PER\_00002]** *[*All specified classes within the Persistency shall reside within the C++ namespace ara::per.*♩(RS\_AP\_00115)*

### Functional Cluster Lifecycle

### Initialization and Shutdown of Persistency

Using ara::core::Initialize and ara::core::Deinitialize, the applica- tion 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 Persis- tency 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::UniqueHandles were destructed, and that not persisted values in all Key-Value Storages are dropped as though ara::per::KeyVal- ueStorage::DiscardPendingChanges was called. Afterwards, all access struc- tures shall be freed.*♩(RS\_PER\_00018)*

The application is expected not to call any API of Persistency (directly or indi- rectly 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 initializa- tion but before ara::core::Initialize was called or after ara::core::Deini- tialize was called.*♩(RS\_PER\_00018)*

### 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 er- ror message associated with the passed ara::core::ErrorCode.*♩(RS\_AP\_00128)*

The whole Persistency API has been designed to be exception-less. If an ap- plication 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)*

### Handling of General Errors

**[SWS\_PER\_00536]** *[*When a function or method of Persistency encounters a prob- lem with the hardware or the underlaying operating system services during the access to a storage or an element of a storage, Persistency shall return the error kPhysicalStorageFailure.*♩(RS\_AP\_00128)*

When kPhysicalStorageFailure occurs, the application cannot access the affected storage any more. Depending on the system, a reboot might restore the access.

**[SWS\_PER\_00537]** *[*When a method of Persistency would modify the underlying storage, but the storage is configured as read-only (the PortPrototype that is typed by the corresponding PersistencyInterface is instantiated as RPortPro- totype), Persistency shall return the error kIllegalWriteAccess.*♩(RS\_AP\_- 00128)*

**[SWS\_PER\_00538]** *[*When a function of Persistency is called with an ara::- core::InstanceSpecifier that is not available in the manifest, Persistency shall return the error kStorageNotFound.*♩(RS\_AP\_00128)*

The two previous errors (kIllegalWriteAccess and kStorageNotFound) occur when the configuration does not match the implemented access to a storage. An application might be implemented to be aware of this possibility and react accord- ingly to the error by avoiding (write) accesses to the storage. If the application depends on (write) access to the storage, these errors would hint at an incorrect configuration of the storage.

**[SWS\_PER\_00539]** *[*When a function or method of Persistency detects a funda- mental problem in the structure of the storage that prevents accessing the storage, Persistency shall return the error kIntegrityCorrupted.*♩(RS\_AP\_00128)*

The application can restore a corrupted storage by calling ara::per::Re- coverKeyValueStorage, ara::per::ResetKeyValueStorage, ara::per::- RecoverAllFiles, or ara::per::ResetAllFiles. When the kIntegrityCor- rupted is reported for an element of a storage, integrity can be restored by calling ara::per::KeyValueStorage::RecoverKey, ara::per::KeyVal- ueStorage::ResetKey, ara::per::FileStorage::RecoverFile, or ara::- per::FileStorage::ResetFile.

When Persistency detects insufficient or broken redundancy, it will report kVali- dationFailed as described in [[SWS\_PER\_00221](#_bookmark22)].

**[SWS\_PER\_00540]** *[*When encryption or decryption of persistent data fails within a function or method of Persistency, Persistency shall return the error kEncryptionFailed.*♩(RS\_AP\_00128)*

**[SWS\_PER\_00564]** *[*When the calculation or verification of the MAC of persistent data fails within a function or method of Persistency, Persistency shall return the error kAuthenticationFailed.*♩(RS\_AP\_00128)*

The two previous errors can occur when the configured cryptographic keys or algo- rithms are not available at run time, or when the storage or the element of a stor- age is corrupted. The latter case could be detected or even avoided by configuring redundancy.

**[SWS\_PER\_00541]** *[*When a method of Persistency tries to read data from a file, but the position is already at the end, Persistency shall return the error kIsEof.*♩ (RS\_AP\_00128)*

This error can typically be dealt with by the application, and can be avoided by using ara::per::ReadAccessor::IsEof.

**[SWS\_PER\_00542]** *[*When a function or method of Persistency would create a file, but the number of existing files already equals the configured Persistency- FileStorageInterface.maxNumberOfFiles of the corresponding File Stor- age, Persistency shall return the error kTooManyFiles.*♩(RS\_AP\_00128)*

This error might occur when a new file is opened for writing, or when a File Stor- age is updated or when it is recovered after an error. An application seeing this error might delete some files to be able to create the new file (or files), or reset the File Storage to the initial state using ara::per::ResetAllFiles or ara::per::ResetPersistency.

**[SWS\_PER\_00543]** *[*When a function or method of Persistency would increase the size of a storage or an element of a storage such that the total storage size would exceed the configured PersistencyDeployment.maximumAllowedSize of the storage, Persistency shall return the error kQuotaExceeded.*♩(RS\_AP\_00128)*

This error means that the application tried to store data that exceeds the configured quota of a storage. The application has to be able to deal with this situation, and either write less data to the affected storage or re- move outdated data from this storage, or reset the affected element/stor- age with a call to ara::per::ResetKeyValueStorage, ara::per::ResetAll- Files, ara::per::KeyValueStorage::ResetKey, ara::per::FileStor- age::ResetFile, or even ara::per::ResetPersistency.

**[SWS\_PER\_00544]** *[*When a function or method of Persistency cannot increase the size of a storage because the physical storage is not sufficient (e.g. file system quoata exceeded or partition full), Persistency shall return the error kOut- OfStorageSpace.*♩(RS\_AP\_00128)*

This error means that some other storage or even a completely independent process occupies so much physical storage that Persistency cannot store additional data. This can only happen if PersistencyDeployment.maximumAllowedSize is configured to a higher value than PersistencyDeployment.minimumSustained- Size. The application has to be able to deal with this situation, and either write less data to the affected storage or remove outdated data from this or another storage, or reset this or another element/storage with a call to ara::per::ResetKeyVal- ueStorage, ara::per::ResetAllFiles, ara::per::KeyValueStorage::- ResetKey, ara::per::FileStorage::ResetFile, or even ara::per::Re- setPersistency.

### Parallel Access to Persistent Data

According to [9], the persistent data is local to one Process. Therefore, Per- sistency 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 mul- tiple 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::Recov- erKeyValueStorage 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.

**[SWS\_PER\_00545]** *[*When a function of Persistency that modifies a storage is called while another function that modifies the same storage is being executed, Persistency shall return the error kResourceBusy.*♩(RS\_AP\_00128)*

This restriction also applies to global functions like ara::per::UpdatePersis- tency, which will only succeed if no storage is open, and no other thread is currently modifying a storage with functions like ara::per::RecoverKeyValueStorage or ara::per::ResetAllFiles.

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::Re- moveKey, ara::per::KeyValueStorage::RemoveAllKeys, ara::per::Key- ValueStorage::SyncToStorage, and ara::per::KeyValueStorage::Dis- cardPendingChanges.

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 an- other thread, and trying to open an already opened file will fail. Likewise, operations like ara::per::FileStorage::DeleteFile, ara::per::FileStorage::Re- coverFile, and ara::per::FileStorage::ResetFile will also not be possible on open files.

**[SWS\_PER\_00546]** *[*When a method of ara::per::FileStorage that opens or modifies a file is called while the same file is currently open or being modified by another method, Persistency shall return the error kResourceBusy.*♩(RS\_AP\_- 00128)*

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::SharedHandles 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.

### Security Concepts

The Persistency supports protection of the authenticity and confidentiality of data stored in a Key-Value Storage or File Storage. Which kind of protection is applied and the used algorithms are decided at deployment time. The application is not aware of this fact.

If confidentiality of data shall be protected, 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. Therefore, only encrypted data is stored persistently. In case of a read-only storage, encryption is done only once during installation, decryption is done every time the storage is opened.

If authenticity of data shall be protected, a message authentication code (MAC) is gen- erated after the creation of a storage and when the storage is saved, and verified when the storage is opened. Therefore, unauthorized modifications to the storage can be detected.

In case of a read-only storage, it is also possible to protect the authenticity of the storage(or an element therein) by calculating a hash value of the data to be pro- tected and comparing this calculated value to a hash value provided in the manifest. This assumes that the authenticity of the processed manifest is protected using other mechanisms (like secure boot).

If authenticity and confidentiality shall be protected, authenticated encryption schemes (like AES GCM) can be used.

**[SWS\_PER\_00210]** *[*If a PersistencyDeploymentToCryptoKeySlotMapping exists in the manifest, and PersistencyDeploymentToCryptoKeySlot- Mapping.keySlotUsage is set to encryption, the Persistency shall encrypt all data related to the storage using the algorithm specified by Persistency- DeploymentToCryptoKeySlotMapping.cryptoAlgorithmString before stor- ing it to the persistent memory.*♩(RS\_PER\_00005, RS\_PER\_00010)*

**[SWS\_PER\_00464]** *[*If a PersistencyDeploymentElementToCryptoKeySlot- Mapping exists in the manifest, and PersistencyDeploymentElementTo- CryptoKeySlotMapping.keySlotUsage is set to encryption, the Persistency shall encrypt the element data using the algorithm specified by Persistency- DeploymentElementToCryptoKeySlotMapping.cryptoAlgorithmString be- fore storing it to the persistent memory.*♩(RS\_PER\_00005, RS\_PER\_00010)*

**[SWS\_PER\_00211]** *[*If a PersistencyDeploymentToCryptoKeySlotMapping exists in the manifest, and PersistencyDeploymentToCryptoKeySlot- Mapping.keySlotUsage is set to encryption, the Persistency shall decrypt all data related to the storage using the algorithm specified by Persistency- DeploymentToCryptoKeySlotMapping.cryptoAlgorithmString after reading it from persistent memory.*♩(RS\_PER\_00005, RS\_PER\_00010)*

**[SWS\_PER\_00465]** *[*If a PersistencyDeploymentElementToCryptoKeySlot- Mapping exists in the manifest, and PersistencyDeploymentElementTo-

CryptoKeySlotMapping.keySlotUsage is set to encryption, the Persistency shall decrypt the element data using the algorithm specified by Persistency- DeploymentElementToCryptoKeySlotMapping.cryptoAlgorithmString af- ter reading it from persistent memory.*♩(RS\_PER\_00005, RS\_PER\_00010)*

**[SWS\_PER\_00449]** *[*If a PersistencyDeploymentToCryptoKeySlotMapping exists in the manifest, and PersistencyDeploymentToCryptoKeySlot- Mapping.keySlotUsage is set to verification, the Persistency shall create and store a message authentication code (MAC) for all data related to the storage using the algorithm specified by PersistencyDeploymentToCryptoKeySlot- Mapping.cryptoAlgorithmString when storing it to the persistent memory.*♩(RS\_- PER\_00005, RS\_PER\_00010)*

**[SWS\_PER\_00466]** *[*If a PersistencyDeploymentElementToCryptoKeySlot- Mapping exists in the manifest, and PersistencyDeploymentElementTo- CryptoKeySlotMapping.keySlotUsage is set to verification, the Persis- tency shall create and store a message authentication code (MAC) for the ele- ment data using the algorithm specified by PersistencyDeploymentElementTo- CryptoKeySlotMapping.cryptoAlgorithmString when storing it to the persis- tent memory.*♩(RS\_PER\_00005, RS\_PER\_00010)*

**[SWS\_PER\_00450]** *[*If a PersistencyDeploymentToCryptoKeySlotMapping exists in the manifest, and PersistencyDeploymentToCryptoKeySlot- Mapping.keySlotUsage is set to verification, the Persistency shall verify the MAC of all data related to the storage using the algorithm specified by Persistency- DeploymentToCryptoKeySlotMapping.cryptoAlgorithmString after reading it from persistent memory.*♩(RS\_PER\_00005, RS\_PER\_00010)*

**[SWS\_PER\_00467]** *[*If a PersistencyDeploymentElementToCryptoKeySlot- Mapping exists in the manifest, and PersistencyDeploymentElementTo- CryptoKeySlotMapping.keySlotUsage is set to verification, the Per- sistency shall verify the MAC of the element data using the algorithm spec- ified by PersistencyDeploymentElementToCryptoKeySlotMapping.cryp- toAlgorithmString after reading it from persistent memory.*♩(RS\_PER\_00005, RS\_PER\_00010)*

**[SWS\_PER\_00451]** *[*If PersistencyDeploymentToCryptoKeySlotMapping. verificationHash is available, the Persistency shall calculate a hash value over all data related to the storage using the hash algorithm specified by Persistency- DeploymentToCryptoKeySlotMapping.cryptoAlgorithmString and verify that the calculated hash value matches PersistencyDeploymentToCrypto- KeySlotMapping.verificationHash.*♩(RS\_PER\_00005, RS\_PER\_00010)*

**[SWS\_PER\_00468]** *[*If PersistencyDeploymentElementToCryptoKeySlot- Mapping.verificationHash is available, the Persistency shall calculate a hash value over the element data using the hash algorithm specified by Persistency- DeploymentElementToCryptoKeySlotMapping.cryptoAlgorithmString and verify that the calculated hash value matches PersistencyDeployment-

ElementToCryptoKeySlotMapping.verificationHash.*♩(RS\_PER\_00005, RS\_PER\_00010)*

The Persistency will use the services of the Cryptography [5] for all cryptographic operations.

### 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 ef- fectively 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 redundant- PerElement), or use PersistencyInterface.redundancyHandling to prese- lect the actual measures to be taken. During deployment, the integrator can define the actual measures taken to ensure data integrity using PersistencyDeployment.re- dundancyHandling. If PersistencyInterface.redundancyHandling is con- figured, 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 ev- ery storage represented by a PersistencyDeployment where Persisten- cyDeployment.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](#_bookmark34)], [[SWS\_PER\_00319](#_bookmark35)], and [[SWS\_PER\_00447](#_bookmark36)].

**[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 PersistencyRedundancyMOutOfN.

When the recovery failed, the application can choose to use ara::per::Recov- erKeyValueStorage, ara::per::KeyValueStorage::RecoverKey, ara::- per::RecoverAllFiles, or ara::per::FileStorage::RecoverFile to re- cover 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 re- stored, then Persistency shall try to recover all key-value pairs available in the Key-Value Storage as described in [[SWS\_PER\_00453](#_bookmark23)]. Depending on available information, the whole Key-Value Storage might be reset to the initial state as de- scribed in [[SWS\_PER\_00456](#_bookmark25)], losing all updated values of its key-value pairs, or may contain outdated key-value pairs after the operation.*♩(RS\_PER\_00009)*

**[SWS\_PER\_00547]** *[*When ara::per::KeyValueStorage::RecoverKey is called, Persistency shall first check whether the key-value pair is present in any instance of the Key-Value Storage, and otherwise return directly with kKeyNotFound.*♩(RS\_AP\_00128)*

**[SWS\_PER\_00453]** *[*When ara::per::KeyValueStorage::RecoverKey is called for an existing key-value pair, 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](#_bookmark26)], or might contain an outdated value after the operation.*♩ (RS\_PER\_00009)*

**[SWS\_PER\_00454]** *[*When ara::per::RecoverAllFiles is called, Persis- tency 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](#_bookmark27)], then Persistency shall try to recover all currently available files as described in [[SWS\_PER\_00455](#_bookmark24)]. 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\_00548]** *[*When ara::per::FileStorage::RecoverFile is called, Persistency shall first check whether the file is present in the File Storage, and otherwise return directly with kFileNotFound.*♩(RS\_AP\_00128)*

**[SWS\_PER\_00455]** *[*When ara::per::FileStorage::RecoverFile is called for an existing file, 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](#_bookmark28)], 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::ResetPersistency (see [[SWS\_PER\_00556](#_bookmark41)]), ara:-

:per::ResetKeyValueStorage, ara::per::KeyValueStorage::ResetKey, ara::per::ResetAllFiles, or ara::per::FileStorage::ResetFile to re- set the corrupted item to the initial state according to the current manifest.

**[SWS\_PER\_00456]** *[*When ara::per::ResetKeyValueStorage is called, Per- sistency shall reset the Key-Value Storage to the state it would have after in- stallation 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 ap- plication 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 installa- tion 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 applica- tions. This monitoring is supported by Persistency, which will trigger a callback function of the application in case of any problems with the storages. To acti- vate this monitoring, the application has to register that callback function using ara::per::RegisterRecoveryReportCallback.

**[SWS\_PER\_00480]** *[*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, inde- pendently 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, kKeyRecov- ered, 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, kFileStor- ageRecoveryFailed, or kFileRecoveryFailed if not. The callback also reports the affected storage, the affected elements, and how many copies of these ele- ments were affected (the latter only in case PersistencyRedundancyMOutOfN is configured).

**[SWS\_PER\_00481]** *[*When a Key-Value Storage is accessed, and a redun- dancy 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::Instance- Specifier of the Key-Value Storage, recoveryReportKind set to kKeyVal- ueStorageRecoveryFailed, an empty ara::core::Vector for reportedEle- ments, and an ara::core::Vector with the indices of the affected redundant in- stances of the Key-Value Storage in reportedInstances.*♩(RS\_PER\_00008)*

**[SWS\_PER\_00482]** *[*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]** *[*When a File Storage is accessed, and a redundancy prob- lem affecting the whole File Storage is detected that cannot be handled by Per- sistency (i.e. kValidationFailed is returned), Persistency shall call the reg- istered 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]** *[*When a File Storage is accessed, and a redundancy prob- lem affecting the whole File Storage is detected that can be handled by Persis- tency (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]** *[*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 han- dled by Persistency (i.e. kValidationFailed is returned), Persistency shall call the registered callback with storage set to the ara::core::InstanceSpeci- fier of the Key-Value Storage, recoveryReportKind set to kKeyRecovery- Failed, 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]** *[*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 regis- tered 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]** *[*When a redundancy problem of single keys is reported ac- cording to [[SWS\_PER\_00485](#_bookmark29)] or [[SWS\_PER\_00486](#_bookmark30)], Persistency shall in general ensure that each entry in reportedElements matches an entry in reportedIn- stances 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, report-

edElements may contain the affected key as single entry, related to all entries of

reportedInstances.*♩(RS\_PER\_00008)*

**[SWS\_PER\_00488]** *[*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 han- dled by Persistency (i.e. kValidationFailed is returned), Persistency shall call the registered callback with storage set to the ara::core::InstanceSpec- ifier of the File Storage, recoveryReportKind set to kFileRecovery- Failed, 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]** *[*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::Vec- tor with the affected files in reportedElements, and an ara::core::Vector with the indices of the affected redundant instances of the files in reportedIn- stances.*♩(RS\_PER\_00008)*

**[SWS\_PER\_00490]** *[*When a redundancy problem of single file is reported ac- cording to [[SWS\_PER\_00488](#_bookmark31)] or [[SWS\_PER\_00489](#_bookmark32)], Persistency shall in general ensure that each entry in reportedElements matches an entry in reportedIn- stances 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, report- edElements may contain the affected file as single entry, related to all entries of reportedInstances.*♩(RS\_PER\_00008)*

### Redundancy Types

The type of redundancy that is applied by the Persistency is defined by the set of PersistencyRedundancyHandling classes aggregated as PersistencyDe- ployment.redundancyHandling. The level to which redundancy is applied is de- fined by the possible values of the PersistencyRedundancyHandlingScopeEnum, which are persistencyRedundancyHandlingScopeStorage and persisten- cyRedundancyHandlingScopeElement 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 Persisten- cyRedundancyCrc, the Persistency shall calculate a CRC value when persisting the storage or an element of the storage (depending on PersistencyDeploy- ment.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 al- gorithm 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 Persisten- cyRedundancyMOutOfN, 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)*

It is possible to configure more than one PersistencyDeployment.deploymen- tUri for a storage that uses PersistencyRedundancyMOutOfN. In this case, the copies will be distributed over the different locations. The existence of multiple URIs for a single storage is limited to this case by [constr\_10366].

**[SWS\_PER\_00555]** *[*In case multiple PersistencyDeployment.deploymentUris exist, and PersistencyDeployment.redundancyHandling.scope is persis- tencyRedundancyHandlingScopeStorage, Persistency shall store the copies of the storage in the different locations as follows:

**2** The first location contains the main copy, the second location contains all other copies.

**n** (== PersistencyRedundancyMOutOfN.n) Each copy is placed in a separate lo- cation.

*♩(RS\_PER\_00008, RS\_PER\_00009, RS\_PER\_00010)*

**[SWS\_PER\_00447]**{DRAFT} *[*In case a PersistencyRedundancyHandling ag- gregated as PersistencyDeployment.redundancyHandling is derived as Per- sistencyRedundancyHash, the Persistency shall calculate a hash value when persisting the storage or an element of the storage (depending on Persis- tencyDeployment.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 Persisten- cyRedundancyHash.initializationVectorLength is configured, an initializa- tion 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 con- figured PersistencyRedundancyHash.length and PersistencyRedundancy-

Hash.initializationVectorLength are supported by the configured Persis- tencyRedundancyHash.algorithmFamily.

### Installation and Update of Persistent Data

The Update and Configuration Management [6] handles the life cycle of Adap- tive 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 appli- cation. The Adaptive Application may trigger this handling explicitly by calling ara::per::UpdatePersistency during the verification phase that follows the in- stallation or update, or rely on the Persistency to do this implicitly when per- sistent data is accessed (ara::per::OpenKeyValueStorage/ara::per::- OpenFileStorage). In both cases, the Persistency will compare the stored man- ifest version against the current manifest version, and perform the required action.

Persistency stores information about already installed storages together with ver- sion information in a central location.

**[SWS\_PER\_00463]**{DRAFT} *[*Persistency shall store information about the in- stalled Key-Value Storages and File Storages in the location denoted by ProcessToMachineMapping.persistencyCentralStorageURI of the Pro- cessToMachineMapping that refers to the Process that is referenced by Persis- tencyPortPrototypeToDeploymentMappings. 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 Per- sistency shall follow [[SWS\_PER\_00382](#_bookmark43)] (for installation), [[SWS\_PER\_00386](#_bookmark56)] and [[SWS\_PER\_00387](#_bookmark57)] (for update), or [[SWS\_PER\_00396](#_bookmark63)] (for roll-back) for each stor- age 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 applica- tion using ara::per::OpenKeyValueStorage, the Persistency shall follow [[SWS\_PER\_00382](#_bookmark43)] (for installation), [[SWS\_PER\_00386](#_bookmark56)] and [[SWS\_PER\_00387](#_bookmark57)] (for update), [[SWS\_PER\_00446](#_bookmark61)] (for finalization), or [[SWS\_PER\_00396](#_bookmark63)] (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 applica- tion using ara::per::OpenFileStorage, the Persistency shall follow [[SWS\_PER\_00382](#_bookmark43)] (for installation), [[SWS\_PER\_00386](#_bookmark56)] and [[SWS\_PER\_00387](#_bookmark57)] (for

update), [[SWS\_PER\_00446](#_bookmark61)] (for finalization), or [[SWS\_PER\_00396](#_bookmark63)] (for roll-back) for each File Storage configured as PersistencyFileStorage in the deploy- ment 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 persis- tently in the location denoted by ProcessToMachineMapping.persistencyCen- tralStorageURI.*♩(RS\_PER\_00010, RS\_PER\_00013, RS\_PER\_00014)*

The Executable.version is used by Persistency to detect a change of the ap- plication (see [[SWS\_PER\_00387](#_bookmark57)]), while the PersistencyDeployment.version is used to detect a change of the deployed persistent data (see [[SWS\_PER\_00386](#_bookmark56)] and [[SWS\_PER\_00396](#_bookmark63)]).

**[SWS\_PER\_CONSTR\_00001]**{DRAFT} *[*When the Executable.version is in- creased, the PersistencyDeployment.version needs to be increased, too.*♩(RS\_- PER\_00010, RS\_PER\_00012)*

The PersistencyDeployment.version and Executable.version are Stron- gRevisionLabelStrings. These strings consists of a MajorVersion, a Mi- norVersion, a PatchVersion, and additional labels for pre-release versions and build information. 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, MinorVer- sion, or PatchVersion have to be incremented.*♩(RS\_PER\_00010, RS\_PER\_- 00012)*

After installation of the Adaptive Application, the Persistency will install pre- defined 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 PersistencyFileStor- ageInterface.
* 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 Persisten- cyKeyValueStorage or PersistencyFileStorage.

**[SWS\_PER\_00379]** *[*Elements defined in the deployment data (PersistencyDe- ploymentElement) shall always be preferred over the elements defined in the ap- plication 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 Persis- tency 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 (Persistency- CollectionLevelUpdateStrategyEnum – keepExisting or delete) and for a single key-value pair or file (PersistencyElementLevelUpdateStrate- gyEnum – keepExisting, overwrite, or delete).

**[SWS\_PER\_00251]** *[*An update strategy defined in the deployment data (Persis- tencyDeploymentElement.updateStrategy) shall always be preferred over the update strategy defined in the application design (PersistencyInterfaceEle- ment.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 de- ployment and never used by Persistency.

**[SWS\_PER\_00380]** *[*An update strategy defined for a single element (Persisten- cyDeploymentElement.updateStrategy, PersistencyInterfaceElement. updateStrategy) shall always be preferred over the update strategy defined for the enclosing storage (PersistencyDeployment.updateStrategy, Persis- tencyInterface.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 up- date.

The application can always reset its storages or elements of the stor- ages to their initial state, using ara::per::ResetPersistency, ara:-

:per::ResetKeyValueStorage (see [[SWS\_PER\_00456](#_bookmark25)]), ara::per::Key- ValueStorage::ResetKey (see [[SWS\_PER\_00477](#_bookmark26)]), ara::per::ResetAll- Files (see [[SWS\_PER\_00478](#_bookmark27)]), or ara::per::FileStorage::ResetFile (see [[SWS\_PER\_00479](#_bookmark28)]).

**[SWS\_PER\_00556]** *[*When ara::per::ResetPersistency is called, Persis- tency shall reset all Key-Value Storages and File Storages to the state they would have after installation of the application using the current manifest infor- mation.*♩(RS\_PER\_00010, RS\_PER\_00012)*

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

### Installation of Persistent Data

**[SWS\_PER\_00382]** *[*When a storage is opened by the application, the Per- sistency 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](#_bookmark44) [7.2.6.1.1](#_bookmark44) and [paragraph](#_bookmark50) [7.2.6.1.2](#_bookmark50).

To be able to update Persistency with changed redundancy, Persistency stores the information about the used redundancy measures within the meta data of each storage. The same applies to the used keys and algorithms for encryption and au- thentication.

**[SWS\_PER\_00558]** *[*When a new storage is installed, Persistency shall store the information about the used redundancy in the meta data of the storage.*♩(RS\_- PER\_00008, RS\_PER\_00010, RS\_PER\_00012)*

**[SWS\_PER\_00559]** *[*When a new storage is installed, Persistency shall store the information about keys and algorithms used for encryption and authentication in the meta data of the storage.*♩(RS\_PER\_00005, RS\_PER\_00010, RS\_PER\_00012)*

### 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](#_bookmark45)] are identified at run- time by the shortName path of the PortPrototype, passed as ara::core::In- stanceSpecifier to ara::per::OpenKeyValueStorage.

**[SWS\_PER\_00252]** *[*Persistency shall create an entry in the Key-Value Stor- age 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](#_bookmark38)] applies. The update strategy is determined accord- ing to [[SWS\_PER\_00251](#_bookmark39)] and [[SWS\_PER\_00380](#_bookmark40)].

**[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 refer- enced 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 deter- mined, or if the determined data types are conflicting.*♩(RS\_PER\_00010, RS\_PER\_- 00012)*

Invalid manifests should be rejected by the tooling.

### Installation of File Storage

**[SWS\_PER\_00385]** *[*Persistency shall create a File Storage for each Port- Prototype 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](#_bookmark51)] are identified at run-time by the shortName path of the PortPrototype, passed as ara::core::InstanceSpec- ifier to ara::per::OpenFileStorage.

**[SWS\_PER\_00265]** *[*Persistency shall create a file in the File Storage for each PersistencyFileStorageInterface.fileElement and Persistency- FileStorage.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 PersistencyFileStorageIn- terface as fileElement and the PersistencyFileStorage as file, in which case [[SWS\_PER\_00379](#_bookmark38)] applies. The update strategy is determined according to [[SWS\_PER\_00251](#_bookmark39)] and [[SWS\_PER\_00380](#_bookmark40)].

**[SWS\_PER\_00266]** *[*Files in the File Storage shall use the file name iden- tified by PersistencyFileElement.fileName and/or PersistencyFile.file- Name.*♩(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 Persistency- FileElement.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 Per- sistencyFileElement and a PersistencyFile with the same file name dif- fers.*♩(RS\_PER\_00010, RS\_PER\_00012)*

Invalid manifests should be rejected by the tooling.

### Update of Persistent Data

**[SWS\_PER\_00386]** *[*When a storage is opened by the application, the Per- sistency 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](#_bookmark58) [7.2.6.2.1](#_bookmark58) and [paragraph](#_bookmark59) [7.2.6.2.2](#_bookmark59).

**[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](#_bookmark56)], 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 reg- istered function for each storage that was updated according to [[SWS\_PER\_00386](#_bookmark56)].*♩ (RS\_PER\_00010, RS\_PER\_00013)*

The function registered by the application using ara::per::RegisterAppli- cationDataUpdateCallback can be used by the application to update ele- ments of a storage manually. The storage is identified by the ara::core::In- stanceSpecifier 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.

In case the redundancy configuration or the configuration of encryption and authen- tication of the updated manifest differs from the old manifest, special care has to be taken to keep the data consistent and readable.

**[SWS\_PER\_00560]** *[*During the update, when the old storage is read, Persis- tency shall check the redundancy according to the information stored in the meta data of the storage.*♩(RS\_PER\_00008, RS\_PER\_00010, RS\_PER\_00013)*

**[SWS\_PER\_00561]** *[*When the storage is persisted after the update, Persistency shall use the redundancy configured in the manifest, and store the information about the used redundancy in the meta data of the storage.*♩(RS\_PER\_00008, RS\_PER\_00010, RS\_PER\_00013)*

Please note that this means that in some situations redundant information might be- come obsolete and can be removed, e.g. when the new manifest has a lower n for PersistencyRedundancyMOutOfN.

**[SWS\_PER\_00562]** *[*During the update, when the old storage is read, Persis- tency shall decrypt and verify the signature of the storage or the elements of the storage according to the information stored in the meta data of the storage.*♩ (RS\_PER\_00005, RS\_PER\_00010, RS\_PER\_00013)*

**[SWS\_PER\_00563]** *[*When the storage is persisted after the update, Persistency shall encrypt and sign the storage or the elements of the storage as configured in the manifest, and store the information about the used keys and algorithms in the meta data of the storage.*♩(RS\_PER\_00005, RS\_PER\_00010, RS\_PER\_00013)*

### Update of Key-Value Storage

**[SWS\_PER\_00388]** *[*When a new PortPrototype typed by a PersistencyKey- ValueStorageInterface is detected in an updated manifest, the Persistency shall create a Key-Value Storage as specified in [[SWS\_PER\_00383](#_bookmark45)].*♩(RS\_PER\_- 00010, RS\_PER\_00013)*

**[SWS\_PER\_00389]** *[*When a PortPrototype typed by a PersistencyKeyVal- ueStorageInterface 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 for which the update strategy is not delete, the Persistency shall create a new entry in the Key-Value Storage as specified in [[SWS\_PER\_00252](#_bookmark46)], [[SWS\_PER\_00253](#_bookmark47)], [[SWS\_PER\_00254](#_bookmark48)], and [[SWS\_PER\_00384](#_bookmark49)].*♩(RS\_PER\_00010, RS\_PER\_00013)*

**[SWS\_PER\_00391]** *[*When an existing key-value pair cannot be associated with any PersistencyKeyValueStorageInterface.dataElement or Persisten- cyKeyValueStorage.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](#_bookmark39)].

**[SWS\_PER\_00275]** *[*When an existing key-value pair can be associated with a PersistencyKeyValueStorageInterface.dataElement or Persis- tencyKeyValueStorage.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](#_bookmark48)] and [[SWS\_PER\_00384](#_bookmark49)].*♩ (RS\_PER\_00010, RS\_PER\_00013)*

An entry with identical key might be defined both in the PersistencyKey- ValueStorageInterface and the PersistencyKeyValueStorage, in which case [[SWS\_PER\_00379](#_bookmark38)] applies. The update strategy is determined according to [[SWS\_PER\_00251](#_bookmark39)] and [[SWS\_PER\_00380](#_bookmark40)].

**[SWS\_PER\_00277]** *[*When an existing key-value pair can be associated with a PersistencyKeyValueStorageInterface.dataElement or Persis- tencyKeyValueStorage.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.

To support the conversion from one CppImplementationDataType to another (or to a different version of the same type) in the function registered using ara::per::- RegisterApplicationDataUpdateCallback, Persistency provides the ability to read data types from a storage that are no longer used by the application. These types are configured in the manifest as previousDataType of a Persis- tencyKeyValueDataTypeMapping that references a currently used type as cur- rentDataType.

There are two scenarios in which such a conversion is necessary:

An existing data type has been modified for the new application. The data type still uses the same identifier.

A new data type was introduced that replaces a data type that is no longer used in the new application. The two data types have different identifiers.

**[SWS\_PER\_CONSTR\_00005]** *[*In case an existing data type is changed in a new ap- plication, Persistency expects PersistencyKeyValueDataTypeMappings referring to a copy of the old data type as previousDataType and the new data type as currentDataType. The name of the old data type shall be formed as follows:

<PersistencyKeyValueDataTypeMapping.currentDataType.shortName>\_< PersistencyKeyValueDataTypeMapping.previousExecutableVersion>.*♩ (RS\_PER\_00010, RS\_PER\_00013)*

### Update of File Storage

**[SWS\_PER\_00392]** *[*When a new PortPrototype typed by a Persistency- FileStorageInterface is detected in an updated manifest, the Persistency shall create a File Storage as specified in [[SWS\_PER\_00385](#_bookmark51)].*♩(RS\_PER\_00010, RS\_PER\_00013)*

**[SWS\_PER\_00393]** *[*When a PortPrototype typed by a PersistencyFileStor- ageInterface 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.fileEle- ment and/or PersistencyFileStorage.file with a new file name is detected in an updated manifest for which the update strategy is not delete, the Persistency shall create a new file in the File Storage as specified in [[SWS\_PER\_00265](#_bookmark52)], [[SWS\_PER\_00266](#_bookmark53)], and [[SWS\_PER\_00267](#_bookmark54)].*♩(RS\_PER\_00010, RS\_PER\_00013)*

**[SWS\_PER\_00395]** *[*When an existing file cannot be associated with any Per- sistencyFileStorageInterface.fileElement or PersistencyFileStor- age.file in an updated manifest, and the update strategy of the Persistency- FileStorage 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](#_bookmark39)].

**[SWS\_PER\_00281]** *[*When an existing file can be associated with a Persisten- cyFileStorageInterface.fileElement or PersistencyFileStorage.file in an updated manifest, and the update strategy is overwrite, the Persis- tency shall replace the content of that file with the new content as specified in [[SWS\_PER\_00267](#_bookmark54)].*♩(RS\_PER\_00010, RS\_PER\_00013)*

A file with the same file name might be defined both in the Persisten- cyFileStorageInterface and the PersistencyFileStorage, in which case [[SWS\_PER\_00379](#_bookmark38)] applies. The update strategy is determined according to [[SWS\_PER\_00251](#_bookmark39)] and [[SWS\_PER\_00380](#_bookmark40)].

**[SWS\_PER\_00283]** *[*When an existing file can be associated with a Persisten- cyFileStorageInterface.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.

### 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, Per- sistency 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 initial- ized, the Persistency shall compare the PersistencyDeployment.version in the manifest against the stored version. If the two versions are identical, the Per- sistency shall remove all backup data of the storage.*♩(RS\_PER\_00016)*

Update of persistent data is described in [subsubsection](#_bookmark55) [7.2.6.2](#_bookmark55).

### Roll-Back of Persistent Data after Failed Update

**[SWS\_PER\_00396]** *[*When a storage is opened by the application, the Per- sistency 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](#_bookmark42) [7.2.6.1](#_bookmark42).

### Removal of Persistent Data

Persistency is not able to remove its own data when the Update and Config- uration 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

and PersistencyDeployment.deploymentUri) to obtain the locations of per- sistent data, and, if it has access to the locations, remove it.

### 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 Persis- tencyInterface.minimumSustainedSize.

During deployment, the integrator may update the lower limit using PersistencyDe- ployment.minimumSustainedSize and add an upper limit using Persistency- Deployment.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 appro- priate measures.

**[SWS\_PER\_00321]** *[*The Persistency shall ensure that the space actually allocated by a storage never surpasses the amount configured by PersistencyDeploy- ment.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::GetCur- rentKeyValueStorageSize or ara::per::GetCurrentFileStorageSize, re- spectively. Naturally, the returned values will not drop below a configured minimum size (PersistencyDeployment.minimumSustainedSize) or rise above a config- ured maximum size (PersistencyDeployment.maximumAllowedSize).

**[SWS\_PER\_00491]** *[*ara::per::GetCurrentKeyValueStorageSize shall re- turn the total size of the storage space currently allocated to a Key-Value Storage, including administrative data (apart from data stored in ProcessToMachineMap- ping.persistencyCentralStorageURI), redundant data, and backup data. The reported size may be inaccurate if the Key-Value Storage is currently open, or if an- other operation is currently being executed on the same storage.*♩(RS\_PER\_00011)*

The inaccuracy is mainly due to the fact that meta data of the storage is only up- dated when the storage is fully synchronized, and predicting the meta data size is sometimes very difficult.

**[SWS\_PER\_00492]** *[*ara::per::GetCurrentFileStorageSize shall return the total size of the storage space currently allocated to a File Storage, including ad- ministrative data (apart from data stored in ProcessToMachineMapping.persis- tencyCentralStorageURI), all its files, redundant data, and backup data. The

reported size may be inaccurate if the File Storage is currently open, or if another operation is currently being executed on the same storage.*♩(RS\_PER\_00011)*

## 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 FunctionalClusterInteractsWithFunctionalClus- terMapping). 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](#_bookmark15)], Persis- tency 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](#_bookmark16)] applies.

All operations on a Key-Value Storage will be done in a temporary storage cre- ated during the call to ara::per::OpenKeyValueStorage, which the applica- tion can persist using ara::per::KeyValueStorage::SyncToStorage, or re- set to the last stored state with ara::per::KeyValueStorage::DiscardPend- ingChanges.

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 per- sisted with a call to ara::per::KeyValueStorage::SyncToStorage shall be dis- carded 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::KeyVal- ueStorage::DiscardPendingChanges, which resets the key-value pairs in all threads, and to ara::per::KeyValueStorage::SyncToStorage, which per- sists 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](#_bookmark21) [7.2.5](#_bookmark21).

**[SWS\_PER\_00495]** *[*When ara::per::KeyValueStorage::DiscardPend- ingChanges 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::Key- ValueStorage::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 cre- ate 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 tem- porary 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 tem- plated 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 tem- plated 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::Re- moveAllKeys 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]** *[*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 stor- age, 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, check the current size of a value using ara::per::KeyValueStorage::GetCurrentValueSize, 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\_00565]** *[*When ara::per::KeyValueStorage::GetCurrentVal- ueSize is called, Persistency shall first check whether the key-value pair is present in the temporary storage, and otherwise return directly with kKeyNot- Found.*♩(RS\_PER\_00003)*

**[SWS\_PER\_00566]** *[*When ara::per::KeyValueStorage::GetCurrentVal- ueSize is called for an existing key-value pair, Persistency shall return the current size of its value.*♩(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)*

### 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), ara::per::KeyVal- ueStorage::GetValue with out parameter (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. Persistency shall accept serialized binary data in the form of an ara::core::Vector of ara::core::Byte or an ara::core::Span of ara::core::Byte.*♩(RS\_PER\_00001)*

This allows the application to store custom data types.

Please note that an ara::core::Span of ara::core::Byte cannot be returned by ara::per::KeyValueStorage::GetValue. It can only be passed in to ara:-

:per::KeyValueStorage::SetValue and ara::per::KeyValueStorage::- GetValue with out parameter.

**[SWS\_PER\_00304]** *[*The Persistency shall be able to store all CppImple- mentationDataTypes referenced by PersistencyKeyValueStorageInter- face.dataTypeForSerialization or as PersistencyKeyValueStorageIn- terface.dataElement in the application design of a PersistencyKeyVal- ueStorage in the corresponding Key-Value Storage.*♩(RS\_PER\_00001, RS\_- PER\_00010)*

The definitions of these data types are generated as described in [11]. Typically,

Persistency will generate serializers and deserializers for these types.

## File Storage specific Features

To access a File Storage, the application has to call ara::per::Open- FileStorage with the ara::core::InstanceSpecifier derived from the man- ifest (a shortName path from the Executable to a PortPrototype or a mapping derived from FunctionalClusterInteractsWithFunctionalClus- terMapping). 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::Deini- tialize is called.

**[SWS\_PER\_00507]** *[*When ara::per::OpenFileStorage is called, and Persis- tency is properly initialized as described in [[SWS\_PER\_00408](#_bookmark15)], Persistency shall create the necessary structures to access the File Storage identified by the ara:-

:core::InstanceSpecifier, and create and return an ara::per::SharedHan- dle of an ara::per::FileStorage.*♩(RS\_PER\_00004)*

If ara::per::OpenFileStorage is called without proper initialization, [[SWS\_PER\_00410](#_bookmark16)] applies.

To check for the existence of a single file, the application may call ara:-

:per::FileStorage::FileExists, and ara::per::FileStorage::GetAll- FileNames 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 cre- ated 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]** *[*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::OpenFil- eReadOnly, 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\_00551]** *[*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 first check whether the file is present in the File Storage, and otherwise return di- rectly with kFileNotFound.*♩(RS\_AP\_00128)*

**[SWS\_PER\_00552]** *[*When ara::per::FileStorage::OpenFileReadOnly with ara::per::OpenMode (or the overloaded version ara::per::FileStorage::- OpenFileReadOnly with ara::per::OpenMode and separate buffer) is called, Persistency shall first check whether the provided mode consists only of either kAtTheBeginning or kAtTheEnd, and otherwise return directly with kInvali- dOpenMode.*♩(RS\_AP\_00128)*

**[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 for an existing file and with a valid ara::per::OpenMode, Persistency shall create the necessary struc- tures 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\_00553]** *[*When ara::per::FileStorage::OpenFileRead- Write with ara::per::OpenMode or ara::per::FileStorage::Open- FileWriteOnly with ara::per::OpenMode (or one of the overloaded versions ara::per::FileStorage::OpenFileReadWrite with ara::per::OpenMode and separate buffer or ara::per::FileStorage::OpenFileWriteOnly with ara::per::OpenMode and separate buffer) is called, Persistency shall first check whether the provided mode contains either kAtTheBeginning, possibly combined with kTruncate and kAppend, or kAtTheEnd, possibly combined with kAppend, or only kTruncate. Otherwise, the call shall return directly with kInvalidOpenMode.*♩ (RS\_AP\_00128)*

**[SWS\_PER\_00513]** *[*When ara::per::FileStorage::OpenFileReadWrite or ara::per::FileStorage::OpenFileWriteOnly (or one of their over- loaded versions ara::per::FileStorage::OpenFileReadWrite with ara:-

:per::OpenMode, ara::per::FileStorage::OpenFileReadWrite with ara::per::OpenMode and separate buffer, ara::per::FileStorage::Open- FileWriteOnly with ara::per::OpenMode, or ara::per::FileStorage::- OpenFileWriteOnly with ara::per::OpenMode and separate buffer) are called with a valid ara::per::OpenMode, Persistency shall create the necessary structures to access the file identified by the file name, and create and re- turn 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::Deinitial- ize is called.*♩(RS\_PER\_00004)*

Some of the overloads of the file opening functions receive an ara::per::Open- Mode 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]** *[*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]** *[*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]** *[*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 corre- sponds to the total file size, which can be obtained separately using ara::per::- ReadAccessor::GetSize.

**[SWS\_PER\_00519]** *[*ara::per::ReadAccessor::GetSize shall return the cur- rent 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 char- acter 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 posi- tion shall be set to the end of the file.*♩(RS\_PER\_00004)*

**[SWS\_PER\_00524]** *[*ara::per::ReadAccessor::ReadLine shall read all char- acters 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 char- acter following the delimiter or the end of the file.*♩(RS\_PER\_00004, RS\_AP\_00136)*

All these methods return characters with a size of eight bits, which are just so-called code units in case of UTF-8, not code points. Therefore, these methods might return incomplete 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]). It is also assumed that line endings are handled according to UNIX conventions, i.e. just LF ("\n").

**[SWS\_PER\_CONSTR\_00006]** *[*If a CppImplementationDataType with category equal to STRING is used as PersistencyDataElement, then the encoding of this string data type is expected to be UTF-8.*♩(RS\_PER\_00003, RS\_PER\_00004, RS\_- AP\_00136)*

This means that a CppImplementationDataType can only be mapped to an ApplicationDataType of category STRING where attribute swDataDefProps. swTextProps.baseType.baseTypeDefinition.baseTypeEncoding is set to the value UTF-8 as defined by [constr\_5035]. If a CppImplementationDataType with- out an ApplicationDataType is used there is no formal description about the UTF- 8 encoding in the ServiceInterface description.

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::ReadWriteAc- cessor::WriteText method or the ara::per::ReadWriteAccessor::opera- tor<< 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\_00557]** *[*If the file was opened with ara::per::OpenMode set to kAppend, Persistency shall always set the current position to the end of the file before writing data to the file according to [[SWS\_PER\_00529](#_bookmark71)], [[SWS\_PER\_00530](#_bookmark72)], or [[SWS\_PER\_00531](#_bookmark73)].*♩(RS\_PER\_00004)*

**[SWS\_PER\_00529]** *[*ara::per::ReadWriteAccessor::WriteText shall write the characters provided as ara::core::StringView to the file at the current position, possibly overwriting current content and advancing the end of the file if necessary. The position shall be set to the character following the last character that was written during this operation, or to the end of the file.*♩(RS\_PER\_00004)*

**[SWS\_PER\_00530]** *[*ara::per::ReadWriteAccessor::operator<< shall write the characters provided as ara::core::StringView to the file at the current po- sition, possibly overwriting current content and advancing the end of the file if nec- essary. The position shall be set to the 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. See also [[SWS\_PER\_00557](#_bookmark70)] for ara::per::OpenMode kAppend.

**[SWS\_PER\_00531]** *[*ara::per::ReadWriteAccessor::WriteBinary shall write the bytes provided as ara::core::Span of ara::core::Byte to the file at the current position, possibly overwriting current content and advancing the end of the file if necessary. The position shall be set to the 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::Set- FileSize.

**[SWS\_PER\_00532]** *[*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 per- sistent 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 stor- age. The actual update of the persistent storage may still be ongoing or may not even have started when this call returns.*♩(RS\_PER\_00004)*

### Access to Additional Information about Files

To gain information about stored files, the Persistency provides the methods ara::per::FileStorage::GetCurrentFileSize and ara::per::- FileStorage::GetFileInfo.

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\_00549]** *[*When ara::per::FileStorage::GetCurrentFileSize is called, Persistency shall first check whether the file is present in the File Stor- age, and otherwise return directly with kFileNotFound.*♩(RS\_AP\_00128)*

**[SWS\_PER\_00493]** *[*When ara::per::FileStorage::GetCurrentFileSize is called for an existing file, Persistency shall return the current size of the passed file. This size shall reflect only the data contained in the file. In case the file is currently open, Persistency shall return the current size of the file, which might differ from the size of the file in the storage if the last changes are not yet synchro- nized. Otherwise, if the file is not open, Persistency shall return the size of any existing instance of the file without checking the consistency/validity of the file.*♩ (RS\_PER\_00011)*

Please note that administrative and redundant information is not included in the file size reported by ara::per::FileStorage::GetCurrentFileSize, while it is in- cluded in the total size of the File Storage returned by ara::per::GetCurrent- FileStorageSize (see [[SWS\_PER\_00492](#_bookmark66)]).

Using ara::per::FileStorage::GetFileInfo, the application can acquire 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\_00550]** *[*When ara::per::FileStorage::GetFileInfo is called, Persistency shall first check whether the file is present in the File Storage, and otherwise return directly with kFileNotFound.*♩(RS\_AP\_00128)*

**[SWS\_PER\_00440]** *[*When ara::per::FileStorage::GetFileInfo is called for an existing file, Persistency 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 infor- mation (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]** *[*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”.

# 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::OpenFileRead- Write. 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).

## General Features of Persistency

### 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]).