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

Document Status	Final
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	17-10

Document Change History			
Date	**Release**	**Changed by**	**Description**
2017-10-27	17-10	AUTOSAR Release Management	• Introduction of AUTOSAR model • Security added • Redundancy added • Rework of FileProxy/Stream API
2017-03-31	17-03	AUTOSAR Release Management	• Initial release

Disclaimer

Table of Contents

1 Introduction and functional overview

This document is the software specification of the Persistency functional cluster within the Adaptive Platform.

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

2 Related documentation

2.1 Input documents & related standards and norms

[1] Requirements on Persistency
AUTOSAR_RS_Persistency

[2] General Requirements specific to Adaptive Platform
AUTOSAR_RS_General

[3] Specification of Manifest
AUTOSAR_TPS_ManifestSpecification

3 Constraints and assumptions

3.1 Limitations

- There is no possibility to model AUTOSAR data types for storage in `Persistency`. As a result with the methods offered by `Persistency` it is not possible to store Non-POD-types without a serialization algorithm implemented in the application.
- The interpretation of deployment related information in the AUTOSAR model is not yet covered in detail in this specification. Remarks are added to the relevant chapters (e.g. 5.1.1).

4 Requirements Tracing

The following table references the features specified in [1], [2] and links to the fulfillments of these.

Feature	Description	Satisfied by
[RS_AP_00115]	Standardized scope/namespace definition	[SWS_PER_00002]

[RS_AP_00116]	Header file name	[SWS_PER_00003] [SWS_PER_00100] [SWS_PER_00101]
[RS_AP_00117]	Class and structure names	[SWS_PER_00003] [SWS_PER_00100] [SWS_PER_00101]
[RS_AP_00118]	Exceptions	[SWS_PER_00060] [SWS_PER_00061] [SWS_PER_00076] [SWS_PER_00132] [SWS_PER_00133] [SWS_PER_00134]
[RS_PER_00001]	Persistency shall support storage of persistent data	[SWS_PER_00102] [SWS_PER_00103] [SWS_PER_00104] [SWS_PER_00105] [SWS_PER_00106] [SWS_PER_00107] [SWS_PER_00108] [SWS_PER_00109] [SWS_PER_00110] [SWS_PER_00111] [SWS_PER_00112] [SWS_PER_00113] [SWS_PER_00114] [SWS_PER_00115] [SWS_PER_00116] [SWS_PER_00117] [SWS_PER_00118] [SWS_PER_00119] [SWS_PER_00120] [SWS_PER_00121] [SWS_PER_00122] [SWS_PER_00123] [SWS_PER_00124] [SWS_PER_00125] [SWS_PER_00126] [SWS_PER_00127] [SWS_PER_00128] [SWS_PER_00140] [SWS_PER_00141] [SWS_PER_00142] [SWS_PER_00143] [SWS_PER_00144] [SWS_PER_00145] [SWS_PER_00160] [SWS_PER_00161]

[RS_PER_00002]	Persistency shall support to retrieve data that has been persistently stored on a platform instance	[SWS_PER_00012] [SWS_PER_00014] [SWS_PER_00015] [SWS_PER_00019] [SWS_PER_00042] [SWS_PER_00043] [SWS_PER_00044] [SWS_PER_00046] [SWS_PER_00047] [SWS_PER_00048] [SWS_PER_00049] [SWS_PER_00052] [SWS_PER_00500]
[RS_PER_00003]	Persistency shall support identification of data using a unique identifier	[SWS_PER_00003] [SWS_PER_00004] [SWS_PER_00005] [SWS_PER_00006] [SWS_PER_00007] [SWS_PER_00008] [SWS_PER_00010] [SWS_PER_00013] [SWS_PER_00016] [SWS_PER_00017] [SWS_PER_00018] [SWS_PER_00020] [SWS_PER_00041] [SWS_PER_00050] [SWS_PER_00051] [SWS_PER_00100] [SWS_PER_00101] [SWS_PER_00129] [SWS_PER_00130] [SWS_PER_00131] [SWS_PER_00150] [SWS_PER_00151] [SWS_PER_00152] [SWS_PER_00153] [SWS_PER_00154] [SWS_PER_00155] [SWS_PER_00156] [SWS_PER_00157]

[RS_PER_00004]	Persistency shall support access to file-like structures	[SWS_PER_00102] [SWS_PER_00103] [SWS_PER_00104] [SWS_PER_00105] [SWS_PER_00106] [SWS_PER_00107] [SWS_PER_00108] [SWS_PER_00109] [SWS_PER_00110] [SWS_PER_00111] [SWS_PER_00112] [SWS_PER_00113] [SWS_PER_00114] [SWS_PER_00115] [SWS_PER_00116] [SWS_PER_00117] [SWS_PER_00118] [SWS_PER_00119] [SWS_PER_00120] [SWS_PER_00121] [SWS_PER_00122] [SWS_PER_00123] [SWS_PER_00124] [SWS_PER_00125] [SWS_PER_00126] [SWS_PER_00127] [SWS_PER_00128] [SWS_PER_00140] [SWS_PER_00141] [SWS_PER_00142] [SWS_PER_00143] [SWS_PER_00144] [SWS_PER_00145] [SWS_PER_00160] [SWS_PER_00161]
[RS_PER_00005]	Persistency shall support encryption/decryption of persistent data	[SWS_PER_00210] [SWS_PER_00211]
[RS_PER_00007]	Persistency shall allow access to persistent data only to authorized Adaptive Applications	[SWS_PER_00200] [SWS_PER_00201]
[RS_PER_00008]	Persistency shall support detection of data corruption in persistent memory	[SWS_PER_00220] [SWS_PER_00221]
[RS_PER_00009]	Persistency shall support data recovery mechanisms if persistent data was corrupted	[SWS_PER_00220] [SWS_PER_00222]

5 Functional specification

5.1 General description

The functional cluster `Persistency` offers two different mechanisms to access persistent memory.

Key-Value-Storage offers access to one or multiple Key-Value-Databases for every AdaptiveApplicationSwComponentType. Every Key-Value-Database is represented by a PortPrototype typed by a PersistencyKeyValueDatabaseInterface in Application Design for the respective AdaptiveApplicationSwComponentType. Every Key-Value-Database can hold multiple Key-Value-Pairs. There is no mechanism to deploy a Key-Value Database that already contains Key-Value-Pairs defined in Application Design or Deployment. All Key-Value-Pairs need to be defined by the implementation of the Adaptive Application using the Persistency API (see 6.2.2.6).

The File-Proxy is represented by a PortPrototype typed by a PersistencyFileProxyInterface in Application Design for the respective AdaptiveApplicationSwComponentType. Every File-Proxy can hold multiple files as described in [3]. The deployment step of mapping existing PersistencyFiles to a PortPrototype typed by a PersistencyFileProxyInterface using a PersistencyFileProxyToFileMapping is currently not addressed in the Persistency specification. Similar to the Key-Value Pairs mentioned above the single files need to be defined by the implementation of the Adaptive Application using the Persistency API (see 6.2.3.1.4 and 6.2.3.1.6).

5.1.1 Architectural concepts

Persistent data is always private to one application instance. This design decision was taken to prevent an additional communication path for applications besides the mechanisms provided by the functional cluster Communication Management. If persistent data needs to be shared between multiple applications it is the duty of an application developer to provide Service Interfaces for communication.

[SWS_PER_00200] ⌈ The Persistency cluster shall reject any configuration in which multiple PersistencyPortPrototypeToKeyValueDatabaseMapping map to the same PersistencyKeyValueDatabase. ⌋*(RS_PER_00007)*

[SWS_PER_00201] ⌈ The Persistency cluster shall reject any configuration in which multiple PersistencyFileProxyToFileMapping map to the same PersistencyFile. ⌋*(RS_PER_00007)*

Since there is currently no deployment step including code generation, the responsibility to cover [SWS_PER_00200] and [SWS_PER_00201] lies with the integrator.

5.1.2 Design decisions

The API specification holds classes for Key-Value-Storage and File-Proxy access with a fixed API, taking the shortName of PortPrototype typed by a PersistencyKeyValueDatabaseInterface or PersistencyFileProxyInterface as an std::string input parameter (see 6.2.2.1 and 6.2.3.1.7).

5.2 Security concepts

Security requirements of the Key-Value-Storage and File-Proxy are modeled with the `CryptoNeed` element from [3]. For details which security algorithms etc. are supported refer to the `CryptoNeed` element.

[SWS_PER_00210] ⌈ The `Persistency` cluster shall encrypt data before storing it to the persistent memory according to the `CryptoNeedToCryptoJobMapping`. ⌋ *(RS_PER_00005)*

[SWS_PER_00211] ⌈ The `Persistency` cluster shall decrypt data after reading it from persistent memory according to the `CryptoNeedToCryptoJobMapping`. ⌋ *(RS_PER_00005)*

5.3 Redundancy concepts

The implementer of the AUTOSAR Adaptive `Persistency` cluster shall take care of the error detection and error correction mechanisms which are implementation specific. The need of a Key-Value-Storage for redundancy can be modeled with element `PersistencyRedundancyEnum` from [3].

Since this specification makes no assumptions about the used memory technology in an Adaptive Platform and available error detection and correction mechanisms the requirements on the implementation of the Redundancy concept is rather unspecific.

[SWS_PER_00220] ⌈ The `Persistency` cluster shall introduce redundant information for every Key-Value-Pair stored in a Key-Value-Database represented by a `PortPrototype` typed by a `PersistencyKeyValueDatabaseInterface` with a `PersistencyProvidedComSpec` with `PersistencyRedundancyEnum` equals to `redundant` ⌋*(RS_PER_00008, RS_PER_00009)*

[SWS_PER_00221] ⌈ The `Persistency` cluster shall use the redundant information to detect data corruption in the persistent memory. ⌋*(RS_PER_00008)*

[SWS_PER_00222] ⌈ The `Persistency` cluster shall use the redundant information to correct corrupted data if possible. ⌋*(RS_PER_00009)*

6 API specification

6.1 Type definitions

6.1.1 KvsType::Type

Name:	**[SWS_PER_00003]** ⌈`KvsType::Type`⌋(*RS_PER_00003*, *RS_AP_00117*, *RS_AP_00116*)		
Type:	`Scoped Enumeration of uint8_t`		
Range:	`kNotSupported`	`0`	`--`
	`kFloat`	`1`	`--`
	`kDouble`	`2`	`--`
	`kSInt8`	`3`	`--`
	`kSInt16`	`4`	`--`
	`kSInt32`	`5`	`--`
	`kSInt64`	`6`	`--`
	`kUInt8`	`7`	`--`
	`kUInt16`	`8`	`--`
	`kUInt32`	`9`	`--`
	`kUInt64`	`10`	`--`
	`kString`	`11`	`--`
	`kBinary`	`12`	`--`
	`kBoolean`	`13`	`--`
	`kObject`	`14`	`--`
	`kNotSet`	`15`	`--`
Syntax:	`enum class Type : uint8_t { kNotSupported, kFloat, kDouble, kSInt8, kSInt16, kSInt32, kSInt64, kUInt8, kUInt16, kUInt32, kUInt64, kString, kBinary, kBoolean, kObject, kNotSet };`		
Header file:	kvstype.h		
Description:	Defines the states of a `KvsType::Type` (see subsection 6.2.1).		

Table 6.1: KvsType::Type

6.1.2 KvsType::Status

Name:	**[SWS_PER_00004]** ⌈`KvsType::Status`⌋(*RS_PER_00003*)
Type:	`Scoped Enumeration of uint8_t`

Range:	kSuccess kNotFound kCheckSumError kGeneralError	0 1 2 3	-- -- -- --

Syntax:	enum class Type : uint8_t {
	kSuccess,
	kNotFound,
	kCheckSumError,
	kGeneralError,
	};		

Header file:	kvstype.h		
Description:	Defines the states of a KvsType::Status (see subsection 6.2.1). kSuccess - indicates that the value was successfully restored from the KVS-storage kNotFound - requested key was not found kCheckSumError - the key-value pair was found, but the checksum of it is incorrect kGeneralError - any other failure		

Table 6.2: KvsType::Status

6.1.3 BasicOperations::SeekDirection

Name:	**[SWS_PER_00100]** ⌈BasicOperations::SeekDirection⌋ *(RS_PER_00003, RS_AP_00117, RS_AP_00116)*		
Type:	Scoped enumeration		
Range:	beg end cur	0 1 2	-- -- --

Syntax:	enum class SeekDirection {
	beg,
	end,
	cur
	};		

Header file:	basicoperations.h		
Description:	Specification of seek direction. BasicOperations::SeekDirection (see subsubsection 6.2.3.2).		

Table 6.3: BasicOperations::Type

6.1.4 BasicOperations::OpenMode

Name:	**[SWS_PER_00101]** ⌈BasicOperations::OpenMode⌋*(RS_PER_00003, RS_AP_00117, RS_AP_00116)*		
Type:	Scoped enumeration uint8_t		
Range:	app binary in out trunc ate	1 << 0 1 << 1 1 << 2 1 << 3 1 << 4 1 << 5	-- -- -- -- -- --

Syntax:	`class OpenMode : uint8_t { app, binary, in, out, trunc, ate };`
Header file:	basicoperations.h
Description:	How the file shall be opened. `BasicOperations::OpenMode` (see subsubsection 6.2.3.2).

Table 6.4: BasicOperations::OpenMode

6.2 Class definitions

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

6.2.1 KvsType class

The `KvsType` class shall provide constructor methods for every possible datatype in the KvsType::Type enumeration.

6.2.1.1 KvsType::KvsType

Service name:	**[SWS_PER_00005]** ⌈`KvsType Constructor`⌋*(RS_PER_00003)*
Syntax:	`KvsType(<Type> value)`
Parameters (in):	`<Type>` | Value to be stored.
Parameters (inout):	None
Parameters (out):	None
Exceptions:	None
Return value:	None
Description:	Creates an instance of KvsType which can be used to store value(s). Supported types: - `bool` - `int8_t` - `int16_t` - `int32_t` - `int64_t` - `uint8_t` - `uint16_t` - `uint32_t` - `uint64_t` - `float` - `double` - `std::string` - `char*`

Table 6.5: KvsType Constructor

6.2.1.2 KvsType::KvsType (void)

Service name:	**[SWS_PER_00006]** ⌈`KvsType Constructor`⌋*(RS_PER_00003)*
Syntax:	`KvsType()`
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Exceptions:	None
Return value:	None
Description:	Creates an empty instance of KvsType with the status `kNotFound`.

Table 6.6: KvsType Constructor (void)

6.2.1.3 KvsType::KvsType (Plain-old-datatype)

Service name:	**[SWS_PER_00007]** ⌈`KvsType Constructor`⌋*(RS_PER_00003)*	
Syntax:	`KvsType(void* data, std::size_t len)`	
Parameters (in):	`void*`	Pointer to data to be stored
	`std::size_t`	Length of the data to be stored.
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	None	
Description:	Creates an instance of KvsType which can be used to store value(s).	

Table 6.7: KvsType Constructor (POD)

6.2.1.4 KvsType::~KvsType

Service name:	**[SWS_PER_00008]** ⌈`KvsType Destructor`⌋*(RS_PER_00003)*
Syntax:	`~KvsType()`
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Exceptions:	None
Return value:	None
Description:	~KvsType deletes the KvsType instance.

Table 6.8: KvsType Destructor

6.2.1.5 KvsType::GetType

Service name:	**[SWS_PER_00010]** ⌈`KvsType::GetType`⌋*(RS_PER_00003)*	
Syntax:	`KvsType::Type GetType()`	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	`KvsType::Type`	Type of the KvsType.
Description:	Returns the type of the stored value.	

Table 6.9: KvsType::GetType

6.2.1.6 KvsType::GetSInt

Service name:	**[SWS_PER_00150]** ⌈`KvsType::GetSInt`⌋*(RS_PER_00003)*
Syntax:	`int32_t GetSInt()`
Parameters (in):	None
Parameters (inout):	None

Parameters (out):	None	
Exceptions:	yes	Check exceptions chapter (subsection 6.2.4).
Return value:	`int32_t`	Stored value.
Description:	Returns the stored value.	

Table 6.10: KvsType::GetSInt

6.2.1.7 KvsType::GetUInt

Service name:	**[SWS_PER_00151]** ⌈`KvsType::GetUInt`⌋*(RS_PER_00003)*	
Syntax:	`uint32_t GetUInt()`	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	yes	Check exceptions chapter (subsection 6.2.4).
Return value:	`uint32_t`	Stored value.
Description:	Returns the stored value.	

Table 6.11: KvsType::GetUInt

6.2.1.8 KvsType::GetSInt64

Service name:	**[SWS_PER_00152]** ⌈`KvsType::GetSInt64`⌋*(RS_PER_00003)*	
Syntax:	`int64_t GetSInt64()`	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	yes	Check exceptions chapter (subsection 6.2.4).
Return value:	`int64_t`	Stored value.
Description:	Returns the stored value.	

Table 6.12: KvsType::GetSInt64

6.2.1.9 KvsType::GetUInt64

Service name:	**[SWS_PER_00153]** ⌈`KvsType::GetUInt64`⌋*(RS_PER_00003)*	
Syntax:	`uint64_t GetUInt64()`	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	yes	Check exceptions chapter (subsection 6.2.4).
Return value:	`uint64_t`	Stored value.
Description:	Returns the stored value.	

Table 6.13: KvsType::GetUInt64

6.2.1.10 KvsType::GetFloat

Service name:	**[SWS_PER_00154]** ⌈`KvsType::GetFloat`⌋*(RS_PER_00003)*

Syntax:	`float GetFloat()`	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	yes	Check exceptions chapter (subsection 6.2.4).
Return value:	`float`	Stored value.
Description:	Returns the stored value.	

Table 6.14: KvsType::GetFloat

6.2.1.11 KvsType::GetDouble

Service name:	**[SWS_PER_00155]** ⌈`KvsType::GetDouble`⌋*(RS_PER_00003)*	
Syntax:	`double GetDouble()`	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	yes	Check exceptions chapter (subsection 6.2.4).
Return value:	`double`	Stored value.
Description:	Returns the stored value.	

Table 6.15: KvsType::GetDouble

6.2.1.12 KvsType::GetString

Service name:	**[SWS_PER_00156]** ⌈`KvsType::GetString`⌋*(RS_PER_00003)*	
Syntax:	`std::string GetString()`	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	yes	Check exceptions chapter (subsection 6.2.4).
Return value:	`std::string`	Stored value.
Description:	Returns the stored value.	

Table 6.16: KvsType::GetString

6.2.1.13 KvsType::GetBool

Service name:	**[SWS_PER_00157]** ⌈`KvsType::GetBool`⌋*(RS_PER_00003)*	
Syntax:	`bool GetBool()`	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	yes	Check exceptions chapter (subsection 6.2.4).
Return value:	`bool`	Stored value.
Description:	Returns the stored value.	

Table 6.17: KvsType::GetBool

6.2.1.14 KvsType::GetBinary

Service name:	**[SWS_PER_00012]** ⌈`KvsType::GetBinary`⌋*(RS_PER_00002)*	
Syntax:	`void GetBinary(void *data, std::size_t len)`	
Parameters (in):	`std::size_t`	size of the data to be retrieved
Parameters (inout):	None	
Parameters (out):	`void*`	pointer where the data should be copied
Exceptions:	yes	Check exceptions chapter (subsection 6.2.4).
Return value:	None	
Description:	Returns the stored value.	

Table 6.18: KvsType::GetBinary

Usage example:

```
1 struct MyStructureWithPodData { .... };
2 KeyValueStorage db("databasename.json");
3
4 MyStructureWithPodData mystruct;
5
6 KvsType binary = db.getValue("my-binary-key");
7 binary.GetBinary(&mystruct, sizeof(mystruct));
```

6.2.1.15 KvsType::GetStatus

Service name:	**[SWS_PER_00013]** ⌈`KvsType::GetStatus`⌋*(RS_PER_00003)*	
Syntax:	`KvsType::Status GetStatus()`	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	`KvsType::Status`	Status of the KvsType.
Description:	The `KvsType` class shall provide a method to get the status of the KVS-Pair. The status shall be checked by the client before trying to use the stored KVS-pair	

Table 6.19: KvsType::GetStatus

6.2.1.16 KvsType::GetKey

Service name:	**[SWS_PER_00014]** ⌈`KvsType::GetKey`⌋*(RS_PER_00002)*	
Syntax:	`std::string GetKey()`	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	`std::string`	Key of the KvsType.
Description:	The `KvsType` class shall provide a method to get the key of the KVS-Pair key.	

Table 6.20: KvsType::GetKey

6.2.1.17 KvsType::SetKey

Service name:	**[SWS_PER_00015]** ⌈`KvsType::SetKey`⌋*(RS_PER_00002)*	
Syntax:	`void SetKey(const std::string& name)`	
Parameters (in):	`std::string&`	Value of the key to be stored.
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	None	
Description:	The `KvsType` class shall provide a method to set the key of the KVS-Pair.	

Table 6.21: KvsType::SetKey

6.2.1.18 KvsType::IsSignedInteger

Service name:	**[SWS_PER_00016] KvsType::IsSignedInteger** ⌈⌋*(RS_PER_00003)*	
Syntax:	`bool IsSignedInteger()`	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	`bool`	return value
Description:	The `KvsType` class shall provide a method to check if the internal type is any of the signed integer types (8,16,32,64bit)	

Table 6.22: KvsType::IsSignedInteger

6.2.1.19 KvsType::IsUnsignedInteger

Service name:	**[SWS_PER_00017] KvsType::IsUnsignedInteger** ⌈⌋*(RS_PER_00003)*	
Syntax:	`bool IsUnsignedInteger()`	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	`bool`	return value
Description:	The `KvsType` class shall provide a method to check if the internal type is any of the unsigned integer types (8,16,32,64bit)	

Table 6.23: KvsType::IsUnsignedInteger

6.2.1.20 KvsType::StoreArray

Service name:	**[SWS_PER_00018]** ⌈`KvsType::StoreArray`⌋*(RS_PER_00003)*	
Syntax:	`template <class Array> void StoreArray(const Array& array)`	
Parameters (in):	`const Array&`	Array to be stored
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	yes	Check exceptions chapter (subsection 6.2.4).
Return value:	None	

Description:	The `KvsType` class shall provide a template method to store an array of [integer, floating point, or KvsType]. The container shall be iterable and the elements are stored in the iteration order to the KVS internal data structure.

Table 6.24: KvsType::StoreArray

6.2.1.21 KvsType::GetArray

Service name:	**[SWS_PER_00019]** ⌈`KvsType::GetArray`⌋*(RS_PER_00002)*	
Syntax:	`template <class T> std::vector<T> GetArray()`	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	yes	Check exceptions chapter (subsection 6.2.4).
Return value:	`std::vector<T>`	Template vector of structure <T>.
Description:	The `KvsType` class shall provide a method to get an array. Returns a vector of requested type. Supported types are [integer, floating point and KVSType]. Restores the items in same order as they were saved with StoreArray.	

Table 6.25: KvsType::GetArray

6.2.1.22 KvsType::AddArrayItem

Service name:	**[SWS_PER_00020]** ⌈`KvsType::AddArrayItem`⌋*(RS_PER_00003)*	
Syntax:	`void AddArrayItem(const KvsType& kvs)`	
Parameters (in):	KvsType&	`KvsType` where the array values are to be taken from.
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	yes	Check exceptions chapter (subsection 6.2.4).
Return value:	None	
Description:	The `KvsType` class shall provide a method to add an element to the internal array container. Note: This function is used only if it's of type kObject and the underlying types are not relevant.	

Table 6.26: KvsType::AddArrayItem

6.2.2 KeyValueStorage class

[SWS_PER_00051] ⌈ The `KeyValueStorage` class defined in the `keyvaluestorage.h` header file shall define an interface to the common key-value store functions. ⌋*(RS_PER_00003)*

[SWS_PER_00041] ⌈ The `KeyValueStorage` class shall delete the following member functions as the class shall not be copyable or assignable.

```
KeyValueStorage(const KeyValueStorage&) = delete;
KeyValueStorage& operator=(const KeyValueStorage&) = delete;
```
⌋*(RS_PER_00003)*

6.2.2.1 KeyValueStorage::KeyValueStorage

Service name:	**[SWS_PER_00052]** ⌈KeyValueStorage Constructor⌋*(RS_PER_00002)*	
Syntax:	`KeyValueStorage( const std::string& database)`	
Parameters (in):	`const std::string&`	`shortName` of `PortPrototype` typed by a `PersistencyKeyValueDatabaseInterface`
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	yes	Check exceptions chapter (subsection 6.2.4).
Return value:	None	
Description:	Creates an instance of `KeyValueStorage` which configures the storage location.	

Table 6.27: KeyValueStorage Constructor

6.2.2.2 KeyValueStorage::~KeyValueStorage

Service name:	**[SWS_PER_00050]** ⌈KeyValueStorage Destructor⌋*(RS_PER_00003)*
Syntax:	`~KeyValueStorage()`
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Exceptions:	None
Return value:	None
Description:	~KeyValueStorage deletes the `KeyValueStorage` instance.

Table 6.28: KeyValueStorage Destructor

6.2.2.3 KeyValueStorage::GetAllKeys

Service name:	**[SWS_PER_00042]** ⌈KeyValueStorage::GetAllKeys⌋*(RS_PER_00002)*	
Syntax:	`std::vector<std::string> GetAllKeys()`	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	yes	Check exceptions chapter (subsection 6.2.4).
Return value:	std::vector<std::string>	Vector of type `std::string`.
Description:	The `KeyValueStorage` class shall provide a method to get a list of all keys explicitly set in the dataset. It shall return the list of available keys.	

Table 6.29: KeyValueStorage::GetAllKeys

6.2.2.4 KeyValueStorage::HasKey

Service name:	**[SWS_PER_00043]** ⌈KeyValueStorage::HasKey⌋*(RS_PER_00002)*	
Syntax:	`bool HasKey(const std::string& key)`	
Parameters (in):	const std::string&	Value of the key that shall be checked.
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	

Return value:	`bool`	Returns if the `key` could be located.
Description:	The `KeyValueStorage` class shall provide a method to determine whether the key exists in the dataset. It shall return true if the key exists in the dataset otherwise false.	

Table 6.30: KeyValueStorage::HasKey

6.2.2.5 KeyValueStorage::GetValue

Service name:	**[SWS_PER_00044]** ⌈ `KeyValueStorage::GetValue`⌋ *(RS_PER_00002)*	
Syntax:	`KvsType GetValue( const std::string& key)`	
Parameters (in):	const std::string&	Value of the key associated with the value.
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	`KvsType`	Returns and object of type `KvsType`. If the key could not have been located, the function shall return a `KvsType` with the status `kNotFound`.
Description:	The `KeyValueStorage` class shall provide a method to get the value assigned to the key.	

Table 6.31: KeyValueStorage::GetValue

6.2.2.6 KeyValueStorage::SetValue

Service name:	**[SWS_PER_00046]** ⌈`KeyValueStorage::SetValue`⌋*(RS_PER_00002)*	
Syntax:	`void SetValue( const std::string& key, const KvsType& value)`	
Parameters (in):	const std::string& const KvsType&	Value of the key associated with the value. `KvsType` containing the data to be stored.
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	yes	Check exceptions chapter (subsection 6.2.4).
Return value:	None	
Description:	The KeyValueStorage class shall provide a method to assign the value to the key. It will be explicitly stored it in the dataset.	

Table 6.32: KeyValueStorage::SetValue

6.2.2.7 KeyValueStorage::RemoveKey

Service name:	**[SWS_PER_00047]** ⌈`KeyValueStorage::RemoveKey`⌋*(RS_PER_00002)*	
Syntax:	`void RemoveKey(const std::string& key)`	
Parameters (in):	const std::string&	Value of the key associated with the value.
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	None	

Description:	The `KeyValueStorage` class shall provide a method that removes the key and associated value.

Table 6.33: KeyValueStorage::RemoveKey

6.2.2.8 KeyValueStorage::RemoveAllKeys

Service name:	**[SWS_PER_00048]** ⌈ `KeyValueStorage::RemoveAllKeys` ⌋ *(RS_PER_00002)*
Syntax:	`void RemoveAllKeys(void)`
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Exceptions:	None
Return value:	None
Description:	The `KeyValueStorage` class shall provide a method that removes all keys and associated values.

Table 6.34: KeyValueStorage::RemoveAllKeys

6.2.2.9 KeyValueStorage::SyncToStorage

Service name:	**[SWS_PER_00049]** ⌈`KeyValueStorage::SyncToStorage`⌋ *(RS_PER_00002)*	
Syntax:	`void SyncToStorage(void)`	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	yes	Check exceptions chapter (subsection 6.2.4).
Return value:	None	
Description:	The KeyValueStorage class shall provide a method to trigger flushing of key-value pairs to the physical storage.	

Table 6.35: KeyValueStorage::SyncToStorage

6.2.3 FileProxy

6.2.3.1 FileProxyAccessorFactory class

[SWS_PER_00109] ⌈ The `Persistency` cluster shall provide an abstract class FileProxyAccessorFactory which creates objects to read or write memory blocks. ⌋ *(RS_PER_00001, RS_PER_00004)*

6.2.3.1.1 FileProxyAccessorFactory::GetAllKeys

Service name:	**[SWS_PER_00110]** ⌈ `FileProxyAccessorFactory::GetAllKeys`⌋ *(RS_PER_00001, RS_PER_00004)*	
Syntax:	`std::vector<std::string> GetAllKeys()`	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	std::vector <std::string>	Returns a vector of identifiers.
Description:	The function returns a vector of available (file) identifiers within this proxy class.	

Table 6.36: FileProxyAccessorFactory GetAllKeys

6.2.3.1.2 FileProxyAccessorFactory::DeleteKey

Service name:	**[SWS_PER_00111]** ⌈ `FileProxyAccessorFactory::DeleteKey`⌋ *(RS_PER_00001, RS_PER_00004)*	
Syntax:	`void DeleteKey (const std::string& key)`	
Parameters (in):	const std::string& key	Identifier of the file.
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	None	
Description:	The function deletes a file with the given identifier from this accessor.	

Table 6.37: FileProxyAccessorFactory DeleteKey

6.2.3.1.3 FileProxyAccessorFactory::HasKey

Service name:	**[SWS_PER_00112]** ⌈ `FileProxyAccessorFactory::HasKey`⌋ *(RS_PER_00001, RS_PER_00004)*	
Syntax:	`bool HasKey (const std::string& key)`	
Parameters (in):	const std::string& key	Identifier of the file.
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	

Return value:	bool	Returns true if the file exists, false otherwise.
Description:	Query if a file with the given identifier is available in the proxy.	

Table 6.38: FileProxyAccessorFactory HasKey

6.2.3.1.4 FileProxyAccessorFactory::CreateRWAccess

Service name:	**[SWS_PER_00113]** ⌈`FileProxyAccessorFactory::CreateRWAccess`⌋(*RS_PER_00001*, *RS_PER_00004*)	
Syntax:	`std::unique_ptr<ReadWriteAccessor> CreateRWAccess (std::string const& key, BasicOperations::OpenMode const mode = BasicOperations::OpenMode::out | BasicOperations::OpenMode::in)`	
Parameters (in):	std::string const& key	Identifier of the file.
Parameters (in):	BasicOperations:: OpenMode const mode	Mode with which the file shall be opened.
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	std::unique_ptr <ReadWrite Accessor>	ReadWriteAccessor associated to the file.
Description:	The function creates an accessor for reading and writing. For keys that are not yet defined in the context of the File-Proxy the same error handling (setting of failbit) shall apply as defined in the C++ specification for std::fstream. If an accessor with a new key would be created the method shall check the attribute `PersistencyFileProxyInterface.maxNumberOfFiles` and set the failbit for the returned accessor if the defined number is already reached.	

Table 6.39: FileProxyAccessorFactory CreateRWAccess

6.2.3.1.5 FileProxyAccessorFactory::CreateReadAccess

Service name:	**[SWS_PER_00114]** ⌈`FileProxyAccessorFactory::CreateReadAccess`⌋(*RS_PER_00001*, *RS_PER_00004*)	
Syntax:	`std::unique_ptr<ReadAccessor> CreateReadAccess (std::string const& key, BasicOperations::OpenMode const mode = BasicOperations::OpenMode::in)`	
Parameters (in):	std::string const& key	Identifier of the file.
Parameters (in):	BasicOperations:: OpenMode const mode	Mode with which the file shall be opened.
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	

Return value:	std::unique_ptr <ReadAccessor>	ReadAccessor associated to the file.
Description:	The function creates an accessor for reading. For keys that are not yet defined in the context of the File-Proxy the same error handling (setting of failbit) shall apply as defined in the C++ specification for std::fstream.	

Table 6.40: FileProxyAccessorFactory CreateReadAccess

6.2.3.1.6 FileProxyAccessorFactory::CreateWriteAccess

Service name:	**[SWS_PER_00115]** ⌈ `FileProxyAccessorFactory ::CreateWriteAccess`⌋ *(RS_PER_00001, RS_PER_00004)*	
Syntax:	`std::unique_ptr<WriteAccessor> CreateWriteAccess (std::string const& key, BasicOperations::OpenMode const mode = BasicOperations::OpenMode::out)`	
Parameters (in):	std::string const& key	Identifier of the file.
Parameters (in):	BasicOperations:: OpenMode const mode	Mode with which the file shall be opened.
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	std::unique_ptr <WriteAccessor>	WriteAccessor associated to the file.
Description:	The function creates an accessor for writing. For keys that are not yet defined in the context of the File-Proxy the same error handling (setting of failbit) shall apply as defined in the C++ specification for std::fstream. If an accessor with a new key would be created the method shall check the attribute `PersistencyFileProxyInterface.maxNumberOfFiles` and set the failbit for the returned accessor if the defined number is already reached.	

Table 6.41: FileProxyAccessorFactory CreateWriteAccess

6.2.3.1.7 CreateFileAccessorFactory

Service name:	**[SWS_PER_00116]** ⌈ `CreateFileAccessorFactory`⌋ *(RS_PER_00001, RS_PER_00004)*	
Syntax:	`std::unique_ptr<FileProxyAccessorFactory> CreateFileAccessorFactory(std::string proxy)`	
Parameters (in):	std::string proxy	`shortName` of `PortPrototype` typed by a `PersistencyFileProxyInterface`
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	ara::per:: StorageLocation NotFound	In case the proxy is not modelled.
Return value:	std::unique_ptr <FileProxy AccessorFactory>	Creates an instance of FileProxyAccessorFactory.

Description:	The `FileProxyAccessorFactory` function shall be used as factory to create objects to read and write persistent memory. Please note: This operator is not a member of the FileProxyAccessor-Factory class.

Table 6.42: CreateFileAccessorFactory

6.2.3.2 BasicOperations class

[SWS_PER_00102] ⌈ The `Persistency` cluster shall provide an abstract class for basic file accessor operations. This class shall be called `BasicOperations` and shall be defined in the `basicOperations.h` header. ⌋*(RS_PER_00001, RS_PER_00004)*

6.2.3.2.1 BasicOperations::tell

Service name:	**[SWS_PER_00103]** ⌈ `BasicOperations::tell`⌋*(RS_PER_00001, RS_PER_00004)*	
Syntax:	`std::size_t tell()`	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	std::size_t	Get the current position in byte in the file from start.
Description:	The function returns the current position in byte in the file from start.	

Table 6.43: BasicOperations tell

6.2.3.2.2 BasicOperations::seek

Service name:	**[SWS_PER_00104]** ⌈ `BasicOperations::seek`⌋*(RS_PER_00001, RS_PER_00004)*	
Syntax:	`void seek(std::size_t const pos)`	
Parameters (in):	std::size_t const pos	Sets offset in bytes for seek start.
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	None	
Description:	The function sets the position in byte in the file from start.	

Table 6.44: BasicOperations seek

Service name:	**[SWS_PER_00105]** ⌈ `BasicOperations::seek`⌋*(RS_PER_00001, RS_PER_00004)*	
Syntax:	`void seek(std::size_t const off, SeekDirection const dir)`	
Parameters (in):	std::size_t const pos	Sets offset in bytes for seek start.
Parameters (in):	SeekDirection const dir	Seek direction.
Parameters (inout):	None	

Parameters (out):	None
Exceptions:	None
Return value:	None
Description:	The function sets the position in byte in the file in relation to the current position.

Table 6.45: BasicOperations seek

6.2.3.2.3 BasicOperations::good

Service name:	**[SWS_PER_00106]** ⌈ `BasicOperations::good`⌋(*RS_PER_00001*, *RS_PER_00004*)	
Syntax:	`bool good()`	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	bool	Returns true if no error occured.
Description:	Returns true if no error occured.	

Table 6.46: BasicOperations good

6.2.3.2.4 BasicOperations::eof

Service name:	**[SWS_PER_00107]** ⌈ `BasicOperations::eof`⌋(*RS_PER_00001*, *RS_PER_00004*)	
Syntax:	`bool eof()`	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	bool	Returns true if end of file was reached.
Description:	Returns true if end of file was reached.	

Table 6.47: BasicOperations eof

6.2.3.2.5 BasicOperations::fail

Service name:	**[SWS_PER_00108]** ⌈ `BasicOperations::fail`⌋(*RS_PER_00001*, *RS_PER_00004*)	
Syntax:	`bool fail()`	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	bool	Returns true if an error occurred during operation.
Description:	Returns true if an error occurred during operation.	

Table 6.48: BasicOperations fail

6.2.3.2.6 BasicOperations::bad

Service name:	**[SWS_PER_00140]** ⌈ `BasicOperations::bad`⌋*(RS_PER_00001, RS_PER_00004)*	
Syntax:	`bool bad()`	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	bool	Returns true if an error occurred during operation with a loss of integrity of the stream.
Description:	Returns true if an error occurred during operation with a loss of integrity of the stream.	

Table 6.49: BasicOperations bad

6.2.3.2.7 BasicOperations::operator!

Service name:	**[SWS_PER_00142]** ⌈ `BasicOperations::operator!`⌋ *(RS_PER_00001, RS_PER_00004)*	
Syntax:	`bool operator!()`	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	bool	Returns true if an error occurred during operation.
Description:	Synonym of fail(). Returns true if an error occurred during operation.	

Table 6.50: BasicOperations operator!

6.2.3.2.8 BasicOperations::operator bool

Service name:	**[SWS_PER_00143]** ⌈ `BasicOperations::operator bool`⌋ *(RS_PER_00001, RS_PER_00004)*	
Syntax:	`explicit operator bool()`	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	bool	Returns false if an error occurred during operation.
Description:	Synonym of !fail(). Returns false if an error occurred during operation.	

Table 6.51: BasicOperations operator bool

6.2.3.2.9 BasicOperations::clear

Service name:	**[SWS_PER_00141]** ⌈ `BasicOperations::clear`⌋*(RS_PER_00001, RS_PER_00004)*
Syntax:	`void clear()`
Parameters (in):	None
Parameters (inout):	None

Parameters (out):	None
Exceptions:	None
Return value:	None
Description:	Clears error flags.

Table 6.52: BasicOperations clear

6.2.3.2.10 operator| for BasicOperations::OpenMode

Service name:	**[SWS_PER_00144]** ⌈`operator|`⌋*(RS_PER_00001, RS_PER_00004)*	
Syntax:	`BasicOperations::OpenMode operator| (BasicOperations::OpenMode const& left, BasicOperations::OpenMode const& right)`	
Parameters (in):	BasicOperations:: OpenMode const& left	Left OpenMode modifiers.
Parameters (in):	BasicOperations:: OpenMode const& right	Right OpenMode modifiers.
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	BasicOperations:: OpenMode	Returns merged OpenMode modifiers.
Description:	Merge all OpenMode modifiers into one OpenMode object. Please note: This operator is not a member of the BasicOperations class.	

Table 6.53: operator| for BasicOperations::OpenMode

6.2.3.2.11 operator& for BasicOperations::OpenMode

Service name:	**[SWS_PER_00145]** ⌈`operator&`⌋*(RS_PER_00001, RS_PER_00004)*	
Syntax:	`BasicOperations::OpenMode operator& (BasicOperations::OpenMode const& left, BasicOperations::OpenMode const& right)`	
Parameters (in):	BasicOperations:: OpenMode const& left	Left OpenMode modifiers.
Parameters (in):	BasicOperations:: OpenMode const& right	Right OpenMode modifiers.
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	BasicOperations:: OpenMode	Returns intersection of OpenMode modifiers.
Description:	Intersect all OpenMode modifiers into one OpenMode object. Please note: This operator is not a member of the BasicOperations class.	

Table 6.54: operator& for BasicOperations::OpenMode

6.2.3.3 ReadAccessor class

[SWS_PER_00117] ⌈ The `Persistency` cluster shall provide an abstract class for read accessor operations. This class shall be called `ReadAccessor` and shall be defined in the `readaccessor.h` header.

⌋(*RS_PER_00001*, *RS_PER_00004*)

[SWS_PER_00131] ⌈ The `ReadAccessor` class shall inherit from class ara::per::BasicOperations. ⌋(*RS_PER_00003*)

6.2.3.3.1 ReadAccessor::read

Service name:	**[SWS_PER_00118]** ⌈ `ReadAccessor::read`⌋(*RS_PER_00001*, *RS_PER_00004*)	
Syntax:	`std::size_t read(char* const s, std::size_t const n)`	
Parameters (in):	char* const s	Destination pointer for read bytes.
Parameters (in):	std::size_t const n	Number of bytes that should be read.
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	std::size_t	Returns the actual number of read bytes.
Description:	The function reads n bytes into char pointer and returns actual number that were read.	

Table 6.55: ReadAccessor read

6.2.3.3.2 ReadAccessor::getline

Service name:	**[SWS_PER_00119]** ⌈ `ReadAccessor::getline`⌋(*RS_PER_00001*, *RS_PER_00004*)	
Syntax:	`ReadAccessor& getline(std::string& string, char const delim)`	
Parameters (in):	char const delim	Char that is used as delimiter.
Parameters (inout):	None	
Parameters (out):	std::string& string	Destination for next string that will be read.
Exceptions:	None	
Return value:	ReadAccessor&	Returns the ReadAccessor object.
Description:	The function reads the next string and stores it in string.	

Table 6.56: ReadAccessor getline

6.2.3.3.3 ReadAccessor::operator»

Service name:	**[SWS_PER_00160]** ⌈ `ReadAccessor::operator»`⌋ (*RS_PER_00001*, *RS_PER_00004*)
Syntax:	`ReadAccessor& operator»(std::string& string)`
Parameters (in):	None
Parameters (inout):	None

Parameters (out):	std::string& string	Destination for next string that will be read.
Exceptions:	None	
Return value:	ReadAccessor&	Returns the ReadAccessor object.
Description:	The operator reads the next string and stores it in string.	

Table 6.57: ReadAccessor operator»

6.2.3.3.4 getline

Service name:	**[SWS_PER_00161]** ⌈`getline`⌋*(RS_PER_00001, RS_PER_00004)*	
Syntax:	`ReadAccessor& getline(ReadAccessor& ifstream, std::string& string, char const delim)`	
Parameters (in):	char const delim	Char that is used as delimiter.
Parameters (inout):	ReadAccessor& ifstream	ReadAccessor to be read from.
Parameters (out):	std::string& string	Destination for next string that will be read.
Exceptions:	None	
Return value:	ReadAccessor&	Returns the ReadAccessor object.
Description:	The operator reads the next string from ifstream and stores it in string. Please note: This function is not a member of the ReadAccessor class.	

Table 6.58: getline

6.2.3.4 ReadWriteAccessor class

[SWS_PER_00120] ⌈ The `Persistency` cluster shall provide an abstract class for read/write accessor operations. This class shall be called `ReadWriteAccessor` and shall be defined in the `readwriteaccessor.h` header.

[SWS_PER_00129] ⌈ The `ReadWriteAccessor` class shall inherit from class ara::per::ReadAccessor and ara::per::WriteAccessor. ⌋*(RS_PER_00003)*

⌋*(RS_PER_00003)*

6.2.3.5 WriteAccessor class

[SWS_PER_00121] ⌈ The `Persistency` cluster shall provide an abstract class for write accessor operations. This class shall be called `WriteAccessor` and shall be defined in the `writeaccessor.h` header. ⌋*(RS_PER_00001, RS_PER_00004)*

[SWS_PER_00130] ⌈ The `WriteAccessor` class shall inherit from class ara::per::BasicOperations. ⌋*(RS_PER_00003)*

6.2.3.5.1 WriteAccessor::fsync

Service name:	**[SWS_PER_00122]** ⌈ `WriteAccessor::fsync`⌋*(RS_PER_00001, RS_PER_00004)*

Syntax:	`void fsync()`
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Exceptions:	None
Return value:	None
Description:	The function flushes and force the OS to write its data to persistent storage.

Table 6.59: WriteAccessor fsync

6.2.3.5.2 WriteAccessor::write

Service name:	**[SWS_PER_00123]** ⌈ `WriteAccessor::write`⌋*(RS_PER_00001, RS_PER_00004)*	
Syntax:	`std::size_t write(char const* const s, std::size_t const n)`	
Parameters (in):	char const* const s	Source of bytes to write.
Parameters (in):	std::size_t const n	Number of bytes to write.
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	std::size_t	Number of actual written bytes.
Description:	The function writes n bytes from s and returns actual number of written bytes.	

Table 6.60: WriteAccessor write

6.2.3.5.3 WriteAccessor::flush

Service name:	**[SWS_PER_00124]** ⌈ `WriteAccessor::flush`⌋*(RS_PER_00001, RS_PER_00004)*
Syntax:	`void flush()`
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Exceptions:	None
Return value:	None
Description:	The function passes write buffer to OS.

Table 6.61: WriteAccessor flush

6.2.3.5.4 WriteAccessor::operator«

Service name:	**[SWS_PER_00125]** ⌈ `WriteAccessor::operator«`⌋ *(RS_PER_00001, RS_PER_00004)*	
Syntax:	`WriteAccessor& operator«(std::string const& string)`	
Parameters (in):	std::string const& string	The string to be put into the WriteAccessor.
Parameters (inout):	None	
Parameters (out):	None	

Exceptions:	None	
Return value:	WriteAccessor&	Returns the WriteAccessor object.
Description:	The operator writes string to the WriteAccessor.	

Table 6.62: WriteAccessor operator«

Service name:	**[SWS_PER_00126]** ⌈ `WriteAccessor::operator«`⌋ *(RS_PER_00001, RS_PER_00004)*	
Syntax:	`WriteAccessor& operator«(WriteAccessor& (*op)(WriteAccessor&))`	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	WriteAccessor&	Returns the WriteAccessor object.
Description:	Operator overload for endl and flush (see below).	

Table 6.63: WriteAccessor operator«

6.2.3.5.5 endl

Service name:	**[SWS_PER_00127]** ⌈ `endl`⌋*(RS_PER_00001, RS_PER_00004)*	
Syntax:	`WriteAccessor& endl(WriteAccessor&)`	
Parameters (in):	None	
Parameters (inout):	WriteAccessor&	WriteAccessor object to be modified.
Parameters (out):	None	
Exceptions:	None	
Return value:	WriteAccessor&	Returns the WriteAccessor object.
Description:	The function writes newline to the file and calls flush(). Please note: This function is not a member of the WriteAccessor class.	

Table 6.64: endl

6.2.3.5.6 flush

Service name:	**[SWS_PER_00128]** ⌈ `flush`⌋*(RS_PER_00001, RS_PER_00004)*	
Syntax:	`WriteAccessor& flush(WriteAccessor&)`	
Parameters (in):	None	
Parameters (inout):	WriteAccessor&	WriteAccessor object to be modified.
Parameters (out):	None	
Exceptions:	None	
Return value:	WriteAccessor&	Returns the WriteAccessor object.
Description:	The functions flush calls flush(). Please note: This function is not a member of the WriteAccessor class.	

Table 6.65: flush

6.2.4 Exceptions

[SWS_PER_00500] Exceptions mapping table ⌈

Exceptions	`Exception Logic Error`	`Exception Storage Location NotFound`	`Exception Physical Storage Error`
KvsType::KvsType	no	no	no
KvsType::~KvsType	no	no	no
KvsType::GetSInt	yes	no	no
KvsType::GetUInt	yes	no	no
KvsType::GetSInt64	yes	no	no
KvsType::GetUInt64	yes	no	no
KvsType::GetFloat	yes	no	no
KvsType::GetString	yes	no	no
KvsType::GetBool	yes	no	no
KvsType::GetBinary	yes	no	no
KvsType::GetStatus	no	no	no
KvsType::GetKey	no	no	no
KvsType::SetKey	no	no	no
KvsType::IsSignedInteger	no	no	no
KvsType::IsUnsignedInteger	no	no	no
KvsType::StoreArray	no	yes	no
KvsType::GetArray	no	yes	no
KeyValueStorage::KeyValueStorage	no	yes	yes
KeyValueStorage:: KeyValueStorage	no	no	no
KeyValueStorage::GetAllKeys	yes	no	yes
KeyValueStorage::HasKey	no	no	no
KeyValueStorage::GetValue	no	no	no
KeyValueStorage::SetValue	yes	no	yes
KeyValueStorage::RemoveKey	no	no	no
KeyValueStorage::RemoveAllKeys	no	no	no
KeyValueStorage::SyncToStorage	yes	no	yes

⌋(*RS_PER_00002*)

6.2.4.1 ExceptionLogicError class

To check which API can throw the `ExceptionLogicError` exception check the `Exceptions mapping table` chapter (subsection 6.2.4).

6.2.4.1.1 ExceptionLogicError Constructor

Service name:	**[SWS_PER_00076]** ⌈`ExceptionLogicError Constructor`⌋ *(RS_AP_00118)*	
Syntax:	`ExceptionLogicError (const std::string& message);`	
Parameters (in):	const std::string&	Error message.
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	None	
Description:	The `ExceptionLogicError` class shall provide an exception which can be thrown by functions defined within the `ara::per` package. This exception is raised when logical errors occur during runtime. The class shall be defined in file `perexceptions.h`.	

Table 6.66: ExceptionLogicError Constructor

6.2.4.1.2 ExceptionLogicError::what

Service name:	**[SWS_PER_00132]** ⌈`ExceptionLogicError::what`⌋*(RS_AP_00118)*	
Syntax:	`const char* what();`	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	`const char*`	Reason of exception.
Description:	This method gives access to the error message contained in the exception.	

Table 6.67: ExceptionLogicError::what

6.2.4.2 ExceptionStorageLocationNotFound class

To check which API can throw the `ExceptionStorageLocationNotFound` exception check the `Exceptions mapping table` chapter (subsection 6.2.4).

6.2.4.2.1 ExceptionStorageLocationNotFound Constructor

Service name:	**[SWS_PER_00060]** ⌈`ExceptionStorageLocationNotFound Constructor`⌋*(RS_AP_00118)*	
Syntax:	`ExceptionStorageLocationNotFound (const std::string& location);`	
Parameters (in):	const std::string&	Error message.
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	None	

Description:	The `ExceptionStorageLocationNotFound` class shall provide an exception which will be thrown if the requested storage location is not found or not configured in the AUTOSAR model. The class shall be defined in file `perexceptions.h`.

Table 6.68: ExceptionStorageLocationNotFound Constructor

6.2.4.2.2 ExceptionStorageLocationNotFound::what

Service name:	**[SWS_PER_00133]** ⌈`ExceptionStorageLocationNotFound::what`⌋ *(RS_AP_00118)*	
Syntax:	`const char* what();`	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	`const char*`	Reason of exception.
Description:	This method gives access to the error message contained in the exception.	

Table 6.69: ExceptionStorageLocationNotFound::what

6.2.4.3 ExceptionPhysicalError class

To check which API can throw the `ExceptionPhysicalStorageError` exception check the `Exceptions mapping table` chapter (subsection 6.2.4).

6.2.4.3.1 ExceptionPhysicalStorageError Constructor

Service name:	**[SWS_PER_00061]** ⌈`ExceptionPhysicalStorageError Constructor`⌋*(RS_AP_00118)*	
Syntax:	`ExceptionPhysicalStorageError (const std::string& message);`	
Parameters (in):	const std::string&	Error message.
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	None	
Description:	The `ExceptionPhysicalStorageError` class shall provide an exception which is thrown if a severe error which might happen during the operation, such as out of memory or writing/reading to the storage return an error. The class shall be defined in file `perexceptions.h`.	

Table 6.70: ExceptionPhysicalStorageError Constructor

6.2.4.3.2 ExceptionPhysicalStorageError::what

Service name:	**[SWS_PER_00134]** ⌈`ExceptionPhysicalStorageError::what`⌋ *(RS_AP_00118)*	
Syntax:	`const char* what();`	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Exceptions:	None	
Return value:	`const char*`	Reason of exception.
Description:	This method gives access to the error message contained in the exception.	

Table 6.71: ExceptionPhysicalStorageError::what

A Mentioned Class Tables

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

Class	**AdaptiveApplicationSwComponentType**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::ApplicationStructure			
Note	This meta-class represents the ability to support the formal modeling of application software on the AUTOSAR adaptive platform. Consequently, it shall only be used on the AUTOSAR adaptive platform. **Tags:** atp.Status=draft; atp.recommendedPackage=AdaptiveApplicationSw ComponentTypes			
Base	ARElement, ARObject, AtpBlueprint, AtpBlueprintable, AtpClassifier, AtpType, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable, SwComponentType			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
internalBe havior	AdaptiveSwcInt ernalBehavior	0..1	aggr	This aggregation represents the internal behavior of the AdaptiveApplicationSwComponentType for the AUTOSAR adaptive platform. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=internalBehavior, variation Point.shortLabel; atp.Status=draft vh.latestBindingTime=preCompileTime

Table A.1: AdaptiveApplicationSwComponentType

Class	**CryptoNeed**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This meta-class represents a statement regarding the applicable crypto use case. **Tags:** atp.Status=draft; atp.recommendedPackage=CryptoNeeds			
Base	ARElement, ARObject, AtpBlueprint, AtpBlueprintable, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
primitiveFamily	String	1	attr	This attribute represents the ability to specify the algorithm family of the crypto need. **Tags:** atp.Status=draft

Table A.2: CryptoNeed

Class	**CryptoNeedToCryptoJobMapping**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::Crypto			
Note	This meta-class represents the ability to define a mapping from crypto need to crypto job. **Tags:** atp.Status=draft			
Base	ARObject, Identifiable, MultilanguageReferrable, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
cryptoJob	CryptoJob	1	ref	This represents the crypto job part of the mapping from crypto need to crypto job. **Tags:** atp.Status=draft
cryptoNeed	CryptoNeed	1	ref	This represents the crypto need part of the mapping from crypto need to crypto job. **Tags:** atp.Status=draft

Table A.3: CryptoNeedToCryptoJobMapping

Class	**PersistencyFile**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::Persistency			
Note	This meta-class represents the model of a file as part of the persistency on deployment level. **Tags:** atp.Status=draft; atp.recommendedPackage=PersistencyFiles			
Base	ARElement, ARObject, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable, UploadablePackageElement			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
path	String	0..1	attr	This attribute holds the absolute path to the represented file on the file system. **Tags:** atp.Status=draft

Table A.4: PersistencyFile

Class	**PersistencyFileProxyInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This meta-class provides the ability to implement a PortInterface for supporting persistency use cases for files. **Tags:** atp.Status=draft; atp.recommendedPackage=PersistencyFileProxyInterfaces			
Base	ARElement, ARObject, AtpBlueprint, AtpBlueprintable, AtpClassifier, AtpType, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, PersistencyInterface, PortInterface, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
encoding	BaseTypeEncodingString	0..1	attr	This attribute supports the definition of an encoding of the corresponding physical files. The possible values of this attribute may be partially standardized by AUTOSAR. But it is also possible to extend the set of values in a custom way (provided that the custom values use a notation that ensures the absence of clashes with further extensions of the standardized values, e.g. by using a company-specific prefix).
maxNumberOfFiles	PositiveInteger	0..1	attr	This attribute represents the definition of an upper bound for the handling of files at run-time in the context of the enclosing PersistencyFileProxyInterface.

Table A.5: PersistencyFileProxyInterface

Class	**PersistencyFileProxyToFileMapping**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::Persistency			
Note	This meta-class represents the ability to define a mapping between a file on deployment level to a given PortPrototype. **Tags:** atp.Status=draft; atp.recommendedPackage=PersistentFileProxyToFileMappings			
Base	ARElement, ARObject, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable, UploadablePackageElement			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
persistencyFileArray	PersistencyFileArray	0..1	ref	This reference represents the mapped array of files. **Tags:** atp.Status=draft
portPrototype	PortPrototype	0..1	iref	This reference represents the mapped PortPrototype. **Tags:** atp.Status=draft
process	Process	0..1	ref	This reference represents the process required as context for the mapping. **Tags:** atp.Status=draft

Table A.6: PersistencyFileProxyToFileMapping

Class	**PersistencyKeyValueDatabase**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::Persistency			
Note	This meta-class represents the ability to model a key/value data base on deployment level. **Tags:** atp.Status=draft; atp.recommendedPackage=PersistencyKeyValueDatabases			
Base	ARElement, ARObject, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable, UploadablePackageElement			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table A.7: PersistencyKeyValueDatabase

Class	**PersistencyKeyValueDatabaseInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This meta-class provides the ability to implement a PortInterface for supporting persistency use cases for data. **Tags:** atp.Status=draft; atp.recommendedPackage=PersistencyKeyValueDatabase Interfaces			
Base	ARElement, ARObject, AtpBlueprint, AtpBlueprintable, AtpClassifier, AtpType, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, PersistencyInterface, PortInterface, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table A.8: PersistencyKeyValueDatabaseInterface

Class	**PersistencyPortPrototypeToKeyValueDatabaseMapping**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::Persistency			
Note	This meta-class represents the ability to define a mapping between a PortPrototype and a key used in a persistent storage. **Tags:** atp.Status=draft; atp.recommendedPackage=PersistentPortPrototypeToKey ValueDatabaseMappings			
Base	ARElement, ARObject, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable, UploadablePackageElement			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
keyValueStorage	PersistencyKeyValueDatabase	0..1	ref	This reference represents the mapped key-value storage. **Tags:** atp.Status=draft
persistencyPortPrototype	PortPrototype	0..1	iref	This reference represents the affected Persistency PortPrototype **Tags:** atp.Status=draft
process	Process	0..1	ref	This reference represents the process required for context of the mapping. **Tags:** atp.Status=draft

Attribute	Type	Mul.	Kind	Note

Table A.9: PersistencyPortPrototypeToKeyValueDatabaseMapping

Class	PersistencyProvidedComSpec			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::ComSpec			
Note	This meta-class represents the ability to define port-specific attributes for supporting use cases of data persistency on the provided side. **Tags:** atp.Status=draft			
Base	ARObject, PPortComSpec			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
redundancy	PersistencyRedundancyEnum	0..1	attr	This attribute represents a requirement towards the redundancy of storage.

Table A.10: PersistencyProvidedComSpec

Enumeration	PersistencyRedundancyEnum
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::ComSpec
Note	This meta-class provides a way to specify the behavior of a given persistent data element with respect to redundancy. **Tags:** atp.Status=draft
Literal	**Description**
none	This value represents the requirement that a piece of data to be stored persistently shall not end up in a redundant persistent storage facility. **Tags:** atp.EnumerationValue=1
redundant	This value represents the requirement that a piece of data to be stored persistently shall end up in a redundant persistent storage facility. The nature of the redundant persistent storage is not further qualified and subject to integrator decisions. **Tags:** atp.EnumerationValue=0

Table A.11: PersistencyRedundancyEnum

Class	PortPrototype (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Base class for the ports of an AUTOSAR software component. The aggregation of PortPrototypes is subject to variability with the purpose to support the conditional existence of ports.			
Base	ARObject, AtpBlueprintable, AtpFeature, AtpPrototype, Identifiable, MultilanguageReferrable, Referrable			
Attribute	**Type**	**Mul.**	**Kind**	**Note**

Attribute	*Type*	*Mul.*	*Kind*	*Note*
clientServerAnnotation	ClientServerAnnotation	*	aggr	Annotation of this PortPrototype with respect to client/server communication.
delegatedPortAnnotation	DelegatedPortAnnotation	0..1	aggr	Annotations on this delegated port.
ioHwAbstractionServerAnnotation	IoHwAbstractionServerAnnotation	*	aggr	Annotations on this IO Hardware Abstraction port.
modePortAnnotation	ModePortAnnotation	*	aggr	Annotations on this mode port.
nvDataPortAnnotation	NvDataPortAnnotation	*	aggr	Annotations on this non voilatile data port.
parameterPortAnnotation	ParameterPortAnnotation	*	aggr	Annotations on this parameter port.
portPrototypeProps	PortPrototypeProps	0..1	aggr	This attribute allows for the definition of further qualification of the semantics of a PortPrototype. **Tags:** atp.Status=draft
senderReceiverAnnotation	SenderReceiverAnnotation	*	aggr	Collection of annotations of this ports sender/receiver communication.
triggerPortAnnotation	TriggerPortAnnotation	*	aggr	Annotations on this trigger port.

Table A.12: PortPrototype

Class	**Referrable (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (while adhering to namespace borders).			
Base	ARObject			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
shortName	Identifier	1	attr	This specifies an identifying shortName for the object. It needs to be unique within its context and is intended for humans but even more for technical reference. **Tags:** xml.enforceMinMultiplicity=true; xml.sequenceOffset=-100
shortNameFragment	ShortNameFragment	*	aggr	This specifies how the Referrable.shortName is composed of several shortNameFragments. **Tags:** xml.sequenceOffset=-90

Table A.13: Referrable