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	17-03

Document Change History			
Date	Release	Changed by	Description
2017-03-31	17-03	AUTOSAR Release Management	• Initial release

Table of Contents

1 Introduction and functional overview

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

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.

Persistency offers a library-based approach to access the non-volatile memory.

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 Constraints and assumptions

3.1 Limitations

- Specification is currently very close to a particular implementententation, users can expect that the level of abstraction will increase
- APIs are not modeled and are expected to be formulated in a more abstract form in upcoming releases
- The persistency API is not able to handle concurrent access to one persistent storage location. Data can only be shared between multiple applications in read only mode where no applications writes to that persistent storage location.

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]
[RS_AP_00117]	Class and structure names	[SWS_PER_00003]

[RS_AP_00118]	Exceptions	[SWS_PER_00060] [SWS_PER_00061] [SWS_PER_00062] [SWS_PER_00066] [SWS_PER_00069] [SWS_PER_00070] [SWS_PER_00071] [SWS_PER_00072] [SWS_PER_00073] [SWS_PER_00074] [SWS_PER_00075] [SWS_PER_00076]
[RS_PER_00001]	Adaptive Applications shall be able to store data on a platform instance persistently over boot and ignition cycles.	[SWS_PER_00021] [SWS_PER_00022] [SWS_PER_00023] [SWS_PER_00024] [SWS_PER_00025] [SWS_PER_00026] [SWS_PER_00027] [SWS_PER_00028] [SWS_PER_00029] [SWS_PER_00053] [SWS_PER_00054] [SWS_PER_00055] [SWS_PER_00056] [SWS_PER_00057] [SWS_PER_00058]
[RS_PER_00002]	Adaptive Applications shall be able to retrieve data persistently stored on a platform instance.	[SWS_PER_00011] [SWS_PER_00012] [SWS_PER_00014] [SWS_PER_00019] [SWS_PER_00042] [SWS_PER_00043] [SWS_PER_00044] [SWS_PER_00045] [SWS_PER_00052]

[RS_PER_00003]	Adaptive Applications shall be able to access data identified by a unique identifier	[SWS_PER_00003] [SWS_PER_00004] [SWS_PER_00005] [SWS_PER_00006] [SWS_PER_00007] [SWS_PER_00010] [SWS_PER_00013] [SWS_PER_00015] [SWS_PER_00016] [SWS_PER_00017] [SWS_PER_00018] [SWS_PER_00020] [SWS_PER_00040] [SWS_PER_00041] [SWS_PER_00046] [SWS_PER_00047] [SWS_PER_00048] [SWS_PER_00049] [SWS_PER_00050] [SWS_PER_00051] [SWS_PER_00059] [SWS_PER_00077] [SWS_PER_00078]
[RS_PER_00004]	Adaptive Applications shall be able to access file-like structures.	[SWS_PER_00021] [SWS_PER_00022] [SWS_PER_00023] [SWS_PER_00024] [SWS_PER_00025] [SWS_PER_00026] [SWS_PER_00027] [SWS_PER_00028] [SWS_PER_00029] [SWS_PER_00053] [SWS_PER_00054] [SWS_PER_00055] [SWS_PER_00056] [SWS_PER_00057] [SWS_PER_00058]

5 API specification

5.1 Class definitions

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

5.1.1 KvsType class

[SWS_PER_00003] ⌈ The `KvsType` class defined in the `kvstype.h` header file shall define a container class for storing different kinds of data inside. The type of the data itself is defined by the `Type`-Enumeration member.

```
class KvsType {
        enum class Type : uint8_t {
                                         kNotSupported = 0,
                                         kFloat,
                                         kDouble,
                                         kSInt8,
                                         kSInt16,
                                         kSInt32,
                                         kSInt64,
                                         kUInt8,
                                         kUInt16,
                                         kUInt32,
                                         kUInt64,
                                         kString,
                                         kBinary,
                                         kBoolean,
                                         kObject,
                                         kNotSet
        };
};
```

⌋*(RS_PER_00003, RS_AP_00117, RS_AP_00116)*

[SWS_PER_00004] ⌈ The `KvsType` class shall provide an enumeration to get the Status of an access to the KVS.

```
enum class Status : uint8_t {
    kSuccess = 0,
    kSuccessDefaultValue,
    kNotFound,
    kCheckSumError,
    kGeneralError
};
```

- `kSuccess` indicates that the value was successfully restored from the KVS-storage.
- `kSuccessDefaultValue` indicates that the requested value wasn't found, but a default value could be restored.
- `kNotFound` requested key was not found.
- `kCheckSumError` the key-value pair was found, but the checksum of it is incorrect.
- `kGeneralError` any other failure.

⌋*(RS_PER_00003)*

[SWS_PER_00005] ⌈ The `KvsType` class shall provide constructor methods for every possible datatype in the `Type`-Enumeration, taking one argument:

- `Value`: The value that is of type `Type` to construct a new `KvsType` object.

For example:

```
    KvsType(bool value);
    KvsType(int8_t value);
```

⌋*(RS_PER_00003)*

[SWS_PER_00006] ⌈ The `KvsType` class shall provide a default constructor method, that creates a "not found"-state object, taking no argument.

```
    KvsType();
```

⌋*(RS_PER_00003)*

[SWS_PER_00007] ⌈ The `KvsType` class shall provide a generic constructor method, for any plain-old-datatype (POD)-type.

```
    KvsType(void* data, std::size_t len);
```

⌋*(RS_PER_00003)*

[SWS_PER_00078] ⌈ The `KvsType` class shall be movable. ⌋*(RS_PER_00003)*

[SWS_PER_00010] ⌈ The `KvsType` shall provide a method to get the type of the value stored in the KVS-Pair.

```
Type GetType() const noexcept;
```

⌋*(RS_PER_00003)*

[SWS_PER_00011] ⌈ The `KvsType` class shall provide methods to retrieve the Data of the value stored in the KVS-Pair. The return types shall match the types within the `Type`-Enumeration. The name is constructed of "Get`<Type>`". For integers there is a simplification. Only GetSInt, GetUInt, GetSInt64 and GetUInt64 shall be implemented.

```
int32_t GetSInt() const noexcept(false);
uint32_t GetUInt() const noexcept(false);
int64_t GetSInt64() const noexcept(false);
uint64_t GetUInt64() const noexcept(false);
float GetFloat() const noexcept(false);
double GetDouble() const noexcept(false);
std::string GetString() const noexcept(false);
bool GetBool() const noexcept(false);
```

⌋*(RS_PER_00002)*

[SWS_PER_00062] ⌈ The following functions shall throw the exception of type `ara::per::exceptions::logic_error` for example if a wrong "Get`<Type>`" function is used to retrieve data from the persistent storage.

```
int32_t GetSInt() const noexcept(false);
uint32_t GetUInt() const noexcept(false);
int64_t GetSInt64() const noexcept(false);
uint64_t GetUInt64() const noexcept(false);
float GetFloat() const noexcept(false);
double GetDouble() const noexcept(false);
std::string GetString() const noexcept(false);
bool GetBool() const noexcept(false);
```

⌋*(RS_AP_00118)*

[SWS_PER_00012] ⌈ The `KvsType` class shall provide a method to retrieve the binary Data of the value stored in the KVS-Pair identified by the `kBinary` value of the `Type`-Enumeration.

```
void GetBinary(void* data, std::size_t len) noexcept(false);
```

Restores the stored value to the given memory address.

- param data: Pointer to the memory, where the data is to be restored
- param len: Length the data

Usage example:

```
struct MyStructureWithPodData { .... };
KeyValueStorage db("databasename.json");

MyStructureWithPodData mystruct;

KvsType binary = db.getValue("my-binary-key");
binary.GetBinary(&mystruct, sizeof(mystruct));
```

⌋*(RS_PER_00002)*

[SWS_PER_00071] ⌈ The function `GetBinary` can throw the exception of type `ara::per::exceptions::logic_error`. This happens if one of the following errors occur:

- Given memory is nullptr
- Number of bytes to read is 0
- Number of stored bytes does not match to the given memory

⌋*(RS_AP_00118)*

[SWS_PER_00013] ⌈ 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.

```
Status GetStatus() const noexcept;;
```

⌋*(RS_PER_00003)*

[SWS_PER_00014] ⌈ The `KvsType` class shall provide a method to get the key of the KVS-Pair.

```
std::string GetKey() const noexcept;
```

⌋*(RS_PER_00002)*

[SWS_PER_00015] ⌈ The `KvsType` class shall provide a method to set the key of the KVS-Pair.

```
void SetKey(const std::string& name) noexcept;
```

⌋*(RS_PER_00003)*

[SWS_PER_00016] ⌈ The `KvsType` class shall provide a method to check if the internal type is any of the signed integer types (8,16,32,64bit)

```
bool IsSignedInteger() const noexcept;
```

⌋*(RS_PER_00003)*

[SWS_PER_00017] ⌈ The `KvsType` class shall provide a method to check if the internal type is any of the unsigned integer types (8,16,32,64bit)

```
bool IsUnsignedInteger() const noexcept;
```

⌋*(RS_PER_00003)*

[SWS_PER_00018] ⌈ The `KvsType` class shall provide a template method to store an array of [integer, floating point, or KvsType]. The container must be iterable and the elements are stored in the iteration order to the KVS internal data structure.

```
template <class Array> void StoreArray(const Array& array)
                                       noexcept(false);
```

⌋*(RS_PER_00003)*

[SWS_PER_00074] ⌈ The function `StoreArray` can throw the exception of type `ara::per::exceptions::logic_error` and `ara::per::exceptions::physical_storage_error`. ⌋*(RS_AP_00118)*

[SWS_PER_00019] ⌈ 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`.

```
template <class T> std::vector<T> GetArray() noexcept(false);
```

⌋*(RS_PER_00002)*

[SWS_PER_00075] ⌈ The function `GetArray` can throw the exception of type `ara::per::exceptions::logic_error` and `ara::per::exceptions::physical_storage_error`. ⌋*(RS_AP_00118)*

[SWS_PER_00020] ⌈ The `KvsType` class shall provide a method to add an element to the internal array container.

```
void AddArrayItem(const KvsType& kvs) noexcept(false);
```

⌋*(RS_PER_00003)*

[SWS_PER_00072] ⌈ The function `AddArrayItem` can throw the exception of type `ara::per::exceptions::logic_error`. ⌋*(RS_AP_00118)*

5.1.2 KeyValueStorageBase class

[SWS_PER_00040] ⌈ The `KeyValueStorageBase` class defined in the `keyvaluestorage.h` header file shall define an interface to the common key-value store functions.

```
class KeyValueStorageBase {
...
};
```

⌋*(RS_PER_00003)*

[SWS_PER_00041] ⌈ The `KeyValueStorageBase` class shall delete the following member functions:

```
KeyValueStorageBase(const KeyValueStorageBase&) = delete;
KeyValueStorageBase& operator=(const KeyValueStorageBase&) = delete;
```

The class KeyValueStorageBase shall neither be copiable nor assignable. ⌋*(RS_PER_00003)*

[SWS_PER_00059] ⌈ The `KeyValueStorageBase` class shall be movable. ⌋*(RS_PER_00003)*

[SWS_PER_00077] ⌈ The constructor of `KeyValueStorageBase` class shall be protected. ⌋*(RS_PER_00003)*

[SWS_PER_00042] ⌈ The `KeyValueStorageBase` 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. Default values are not considered here.

```
std::vector<std::string> GetAllKeys() const noexcept(false);
```

⌋*(RS_PER_00002)*

[SWS_PER_00070] ⌈ The function `GetAllKeys` can throw the exception of type `ara::per::exceptions::logic_error` and `ara::per::exceptions::physical_storage_error`. ⌋*(RS_AP_00118)*

[SWS_PER_00043] ⌈ The `KeyValueStorageBase` 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.

```
bool HasKey(const std::string& key) const noexcept;
```

⌋*(RS_PER_00002)*

[SWS_PER_00044] ⌈ The `KeyValueStorageBase` class shall provide a method to get the value assigned to the key. It shall return `KvsType` object with status as specified.

```
KvsType GetValue(const std::string& key) const noexcept;
```

⌋*(RS_PER_00002)*

[SWS_PER_00045] ⌈ The `KeyValueStorageBase` class shall provide a method to get the default value associated with the key.

```
KvsType GetDefaultValue(const std::string& key) const noexcept;
```

⌋*(RS_PER_00002)*

[SWS_PER_00046] ⌈ The `KeyValueStorageBase` class shall provide a method to assign the value to the key. Even if the value is equal to the default value (see getDefaultValue), it will still be explicitly stored it in the dataset.

```
void SetValue(const std::string& key, const KvsType& value)
              noexcept(false);
```

⌋*(RS_PER_00003)*

[SWS_PER_00066] ⌈ The function `SetValue` can throw the exception of type `ara::per::exceptions::logic_error` and `ara::per::exceptions::physical_storage_error`. ⌋*(RS_AP_00118)*

[SWS_PER_00047] ⌈ The `KeyValueStorageBase` class shall provide a method that removes the key and associated value.

```
void RemoveKey(const std::string& key) noexcept;
```

⌋*(RS_PER_00003)*

[SWS_PER_00048] ⌈ The `KeyValueStorageBase` class shall provide a method that removes all keys and associated values.

```
void RemoveAllKeys() noexcept;
```

⌋*(RS_PER_00003)*

[SWS_PER_00049] ⌈ The `KeyValueStorageBase` class shall provide a method to trigger flushing of key-value pairs to the physical storage.

```
void SyncToStorage() const noexcept(false);
```

⌋*(RS_PER_00003)*

[SWS_PER_00069] ⌈ The function `SyncToStorage` can throw the exception of type `ara::per::exceptions::logic_error` and `ara::per::exceptions::physical_storage_error`. ⌋*(RS_AP_00118)*

[SWS_PER_00050] ⌈ The `KeyValueStorageBase` class shall provide a destructor to trigger flushing of key-value pairs to the physical storage.

```
virtual ~KeyValueStorageBase() noexcept;
```

⌋(*RS_PER_00003*)

5.1.3 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.

```
class KeyValueStorage final : public KeyValueStorageBase {
...
};
```

⌋(*RS_PER_00003*)

[SWS_PER_00052] ⌈ The `KeyValueStorage` class shall provide a constructor to open a database identified by a unique name.

```
KeyValueStorage(const std::string& database) noexcept(false);
```

⌋(*RS_PER_00002*)

[SWS_PER_00073] ⌈ The constructor `KeyValueStorage` shall throw the exception of type `ara::per::exceptions::storage_location_not_found` if the given location for the persistent storage is not valid. The exception `ara::per::exceptions::physical_storage_error` shall be thrown if any error during reading or parsing of the persistent storage occurs. ⌋(*RS_AP_00118*)

5.1.4 arafstream class

[SWS_PER_00021] ⌈ The `Persistency` cluster shall provide a class for plain file access fulfilling the PSE51 requirements of not creating or deleting new files. This class shall be called `arafstream` and shall be defined in the `arafstream.h` header and shall be derived from std::fstream class.

```
class arafstream : public std::fstream
{
...
};
```

⌋(*RS_PER_00001*, *RS_PER_00004*)

[SWS_PER_00053] ⌈ The `arafstream` class shall provide a constructor to open a file identified by a unique name. The call of this constructor must not create a new file if the accessed file does not exist. If the file is not present, the failbit flag of the stream is set.

```
explicit arafstream (const char* filename,
                     ios_base::openmode mode =
                     ios_base::in | ios_base::out) noexcept;
```

⌋*(RS_PER_00001, RS_PER_00004)*

[SWS_PER_00054] ⌈ The `arafstream` class shall provide a constructor to open a file identified by a unique name. The call of this constructor must not create a new file if the accessed file does not exist. If the file is not present the exception handling shall be the same as in the underlying standard library of the programming language.

```
explicit arafstream (const std::string& filename,
                     ios_base::openmode mode =
                     ios_base::in | ios_base::out) noexcept;
```

⌋*(RS_PER_00001, RS_PER_00004)*

[SWS_PER_00022] ⌈ The `arafstream` class shall provide a method `open`. The call of this method must not create a new file if the accessed file does not exist. If the file is not present the exception handling shall be the same as in the underlying standard library of the programming language.

```
void open (const char* filename,
           ios_base::openmode mode =
           ios_base::in | ios_base::out) noexcept(false);
```

⌋*(RS_PER_00001, RS_PER_00004)*

[SWS_PER_00023] ⌈ The `arafstream` class shall provide a method `open`. The call of this method must not create a new file if the accessed file does not exist. If the file is not present the exception handling shall be the same as in the underlying standard library of the programming language.

```
void open (const std::string& filename,
           ios_base::openmode mode =
           ios_base::in | ios_base::out) noexcept(false);
```

⌋*(RS_PER_00001, RS_PER_00004)*

5.1.5 araifstream class

[SWS_PER_00024] ⌈ The `Persistency` cluster shall provide a class for plain file access fulfilling the PSE51 requirements of not creating or deleting new files. This class shall be called `araifstream` and shall be defined in the `araifstream.h` header and shall be derived from std::ifstream class.

```
class araifstream : public std::ifstream {
...
};
```

⌋*(RS_PER_00001, RS_PER_00004)*

[SWS_PER_00055] ⌈ The `araifstream` class shall provide a constructor to open a file identified by a unique name. The call of this constructor must not create a new file if the accessed file does not exist. If the file is not present the exception handling shall be the same as in the underlying standard library of the programming language.

```
explicit araifstream (const char* filename,
                      ios_base::openmode mode = ios_base::in)
                      noexcept;
```

⌋*(RS_PER_00001, RS_PER_00004)*

[SWS_PER_00056] ⌈ The `araifstream` class shall provide a constructor to open a file identified by a unique name. The call of this constructor must not create a new file if the accessed file does not exist. If the file is not present the exception handling shall be the same as in the underlying standard library of the programming language.

```
explicit araifstream (const std::string& filename,
                      ios_base::openmode mode = ios_base::in)
                      noexcept;
```

⌋*(RS_PER_00001, RS_PER_00004)*

[SWS_PER_00025] ⌈ The `araifstream` class shall provide a method `open`. The call of this method must not create a new file if the accessed file does not exist. If the file is not present the exception handling shall be the same as in the underlying standard library of the programming language.

```
void open (const char* filename,
           ios_base::openmode mode = ios_base::in) noexcept(false);
```

⌋*(RS_PER_00001, RS_PER_00004)*

[SWS_PER_00026] ⌈ The `araifstream` class shall provide a method `open`. The call of this method must not create a new file if the accessed file does not exist. If the file is not present the exception handling shall be the same as in the underlying standard library of the programming language.

```
void open (const std::string& filename,
           ios_base::openmode mode = ios_base::in) noexcept(false);
```

⌋*(RS_PER_00001, RS_PER_00004)*

5.1.6 araofstream class

[SWS_PER_00027] ⌈ The `Persistency` cluster shall provide a class for plain file access fulfilling the PSE51 requirements of not creating or deleting new files. This class shall be called `araofstream` and shall be defined in the `araofstream.h` header and shall be derived from std::ofstream class.

```
class araofstream : public std::ofstream {
...
};
```

⌋(RS_PER_00001, RS_PER_00004)

[SWS_PER_00057] ⌈ The `araofstream` class shall provide a constructor to open a file identified by a unique name. The call of this constructor must not create a new file if the accessed file does not exist. If the file is not present the exception handling shall be the same as in the underlying standard library of the programming language.

```
explicit araofstream (const char* filename,
                      ios_base::openmode mode = ios_base::out)
                      noexcept;
```

⌋(RS_PER_00001, RS_PER_00004)

[SWS_PER_00058] ⌈ The `araofstream` class shall provide a constructor to open a file identified by a unique name. The call of this constructor must not create a new file if the accessed file does not exist. If the file is not present the exception handling shall be the same as in the underlying standard library of the programming language.

```
explicit araofstream (const std::string& filename,
                      ios_base::openmode mode = ios_base::out)
                      noexcept;
```

⌋(RS_PER_00001, RS_PER_00004)

[SWS_PER_00028] ⌈ The `araofstream` class shall provide a method `open`. The call of this method must not create a new file if the accessed file does not exist. If the file is not present the exception handling shall be the same as in the underlying standard library of the programming language.

```
void open (const char* filename,
           ios_base::openmode mode = ios_base::out) noexcept(false);
```

⌋(RS_PER_00001, RS_PER_00004)

[SWS_PER_00029] ⌈ The `araofstream` class shall provide a method `open`. The call of this method must not create a new file if the accessed file does not exist. If the file is not present the exception handling shall be the same as in the underlying standard library of the programming language.

```
void open (const std::string& filename,
           ios_base::openmode mode = ios_base::out) noexcept(false);
```

⌋(RS_PER_00001, RS_PER_00004)

5.1.7 logic error

[SWS_PER_00076] ⌈ The `ara::per::exceptions::logic_error` 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. For example requesting an Integer if a String is stored in the persistent storage. The class shall be defined in file `perexceptions.h`.

The `ara::per::exceptions::logic_error` class shall provide a constructor method with a String that gives more information for the exception reason.

```
    logic_error(const std::string& message);
```

⌋*(RS_AP_00118)*

5.1.8 storage location not found

[SWS_PER_00060] ⌈The `ara::per::exceptions::storage_location_not_found` class shall provide an exception which will be thrown if the requested storage, typically a file, but depending on the implementation, could also be a service, is not found. The class shall be defined in file `perexceptions.h`.

The `ara::per::exceptions::storage_location_not_found` class shall provide a constructor method with a String that gives more information for the exception reason.

```
    storage_location_not_found(const std::string& message);
```

⌋*(RS_AP_00118)*

5.1.9 physical storage error

[SWS_PER_00061] ⌈The `ara::per::exceptions::physical_storage_error` 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`.

The `ara::per::exceptions::physical_storage_error` class shall provide a constructor method with a String that gives more information for the exception reason.

```
    physical_storage_error(const std::string& message);
```

⌋*(RS_AP_00118)*