Disclaimer

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.
## Table of Contents

1. Introduction and functional overview........................................... 4
2. Related documentation.............................................................. 4
   2.1 Input documents & related standards and norms.......................... 4
3. Constraints and assumptions...................................................... 4
   3.1 Limitations........................................................................... 4
4. Requirements Tracing................................................................. 4
5. API specification.......................................................................... 6
   5.1 Class definitions.................................................................... 6
      5.1.1 KvsType class................................................................. 7
      5.1.2 KeyValueStorageBase class............................................ 11
      5.1.3 KeyValueStorage class.................................................... 13
      5.1.4 arafstream class............................................................. 13
      5.1.5 arafstream class............................................................. 14
      5.1.6 arafostream class............................................................ 15
      5.1.7 logic error..................................................................... 16
      5.1.8 storage location not found............................................. 17
      5.1.9 physical storage error..................................................... 17
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

AUTOSAR_RS_General

3 Constraints and assumptions

3.1 Limitations

- Specification is currently very close to a particular implementation, 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.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
<th>Satisfied by</th>
</tr>
</thead>
<tbody>
<tr>
<td>[RS_AP_00115]</td>
<td>Standardized scope/namespace definition</td>
<td>[SWS_PER_00002]</td>
</tr>
<tr>
<td>[RS_AP_00116]</td>
<td>Header file name</td>
<td>[SWS_PER_00003]</td>
</tr>
<tr>
<td>[RS_AP_00117]</td>
<td>Class and structure names</td>
<td>[SWS_PER_00003]</td>
</tr>
<tr>
<td>[RS_AP_00118]</td>
<td>Exceptions</td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------</td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00060]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00061]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00062]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00066]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00069]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00070]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00071]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00072]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00073]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00074]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00075]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00076]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[RS.PER.00001]</th>
<th>Adaptive Applications shall be able to store data on a platform instance persistently over boot and ignition cycles.</th>
</tr>
</thead>
<tbody>
<tr>
<td>[SWS.PER.00021]</td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00022]</td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00023]</td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00024]</td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00025]</td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00026]</td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00027]</td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00028]</td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00029]</td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00053]</td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00054]</td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00055]</td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00056]</td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00057]</td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00058]</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[RS.PER.00002]</th>
<th>Adaptive Applications shall be able to retrieve data persistently stored on a platform instance.</th>
</tr>
</thead>
<tbody>
<tr>
<td>[SWS.PER.00011]</td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00012]</td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00014]</td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00019]</td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00042]</td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00043]</td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00044]</td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00045]</td>
<td></td>
</tr>
<tr>
<td>[SWS.PER.00052]</td>
<td></td>
</tr>
<tr>
<td>[RS_PER_00003]</td>
<td>Adaptive Applications shall be able to access data identified by a unique identifier</td>
</tr>
<tr>
<td>----------------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>[RS_PER_00004]</td>
<td>Adaptive Applications shall be able to access file-like structures.</td>
</tr>
</tbody>
</table>

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.
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:

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

The **KvsType** class shall provide a default constructor method, that creates a "not found"-state object, taking no argument.

```cpp
KvsType();
```

The **KvsType** class shall provide a generic constructor method, for any plain-old-datatype (POD)-type.

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

The **KvsType** class shall be movable.

The **KvsType** class shall provide a method to get the type of the value stored in the KVS-Pair.

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

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.

```cpp
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);
```

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:

```cpp
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;
The KvsType class shall provide a method to get the key of the KVS-Pair.

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

The KvsType class shall provide a method to set the key of the KVS-Pair.

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

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

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

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

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

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.

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

The function StoreArray can throw the exception of type `ara::per::exceptions::logic_error` and `ara::per::exceptions::physical_storage_error`

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.

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

The function GetArray can throw the exception of type `ara::per::exceptions::logic_error` and `ara::per::exceptions::physical_storage_error`
void AddArrayItem(const KvsType& kvs) noexcept(false);

[SWS_PER_00003] [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.

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

[SWS_PER_00003]

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

```cpp
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.

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

[SWS_PER_00002] [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.

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

[SWS_PER_00002]
The KeyValueStorageBase class shall provide a method to get the value assigned to the key. It shall return KvsType object with status as specified.

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

The KeyValueStorageBase class shall provide a method to get the default value associated with the key.

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

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.

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

The function SetValue can throw the exception of type ara::per::exceptions::logic_error and ara::per::exceptions::physical_storage_error.

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

The KeyValueStorageBase class shall provide a method that removes all keys and associated values.

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

The KeyValueStorageBase class shall provide a method to trigger flushing of key-value pairs to the physical storage.

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

The function SyncToStorage can throw the exception of type ara::per::exceptions::logic_error and ara::per::exceptions::physical_storage_error.

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

```cpp
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.

```cpp
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.

```cpp
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.

```cpp
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.

```cpp
```
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);
```

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);
```

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);
```