Document Title	General Requirements specific to Adaptive Platform
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	714

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

Document Change History

Date	Release	Changed by	Description
2021-11-25	R21-11	AUTOSAR Release Management	• Guidance on error handling added • More design guidelines added • the sub-namespace ::internal is reserved for vendor-specific use
2020-11-30	R20-11	AUTOSAR Release Management	• More design guidelines for special member functions added • Support of C++ 14 added
2019-11-28	R19-11	AUTOSAR Release Management	• More design guidelines added • Changed Document Status from Final to published
2019-03-29	19-03	AUTOSAR Release Management	• No content changes.

2018-10-31	18-10	AUTOSAR Release Management	• More details to clause 1 Scope of document given • Former chapter 4.3 on Design requirements putted below chapter 4.2 Non-functional requirements • Following requirements have been revised: [RS_AP_00111], [RS_AP_00113], [RS_AP_00114], [RS_AP_00115], [RS_AP_00122], [RS_AP_00120], [RS_AP_00121], [RS_AP_00124], [RS_AP_00125] • Following requirements have been deleted: [RS_AP_00117], [RS_AP_00118] • Following requirements have been added: [RS_AP_00127], [RS_AP_00128], [RS_AP_00129], [RS_AP_00130], [RS_AP_00131], [RS_AP_00132], [RS_AP_00134]
2018-03-29	18-03	AUTOSAR Release Management	• Text entry for Supporting Material for [RS_AP_00111] • Text entry for Supporting Material for [RS_AP_00114] only refers now to ISO/IEC 14882 • Description of [RS_AP_00115] revised • Description of [RS_AP_00116], [RS_AP_00117], [RS_AP_00118], [RS_AP_00120], [RS_AP_00121], [RS_AP_00124], [RS_AP_00125] revised (in general "all ara libraries" changed to "all functional clusters").
2017-10-27	17-10	AUTOSAR Release Management	• Minor fixes
2017-03-31	17-03	AUTOSAR Release Management	• Initial release

Disclaimer

Table of Contents

1 Scope of this document

The goal of this document is to define a common set of basic requirements that apply to all **SWS documents** of the Adaptive Platform. Adaptive applications and functional cluster internals does not need to comply to these requirements.

2 Conventions to be used

The representation of requirements in AUTOSAR documents follows the table specified in [TPS_STDT_00078], see Standardization Template, chapter Support for Traceability ([1]).

The verbal forms for the expression of obligation specified in [TPS_STDT_00053] shall be used to indicate requirements, see Standardization Template, chapter Support for Traceability ([1]).

3 Acronyms and Abbreviations

There are no acronyms and abbreviations relevant within this document that are not included in the [2, AUTOSAR glossary].

4 Requirements Specification

4.1 Functional overview

4.2 Non-functional Requirements

[RS_AP_00111] The AUTOSAR Adaptive Platform shall support source code portability for AUTOSAR Adaptive applications. ⌈

Description:	The AUTOSAR Adaptive platform shall support source code portability.
Rationale:	Ensure reuse of existing IPs.
Dependencies:	–
Use Case:	Integration of Adaptive Applications developed on different implementations of Adaptive Platform.
Supporting Material:	Any implementation of the Adaptive Platform shall allow successful compilation and linking of an Adaptive Application that uses ARA only as specified in the standard. No changes to the source code, and no conditional compilation constructs shall be necessary for this, if the application only uses constructs from the designated minimum C++ language version. The implementation may provide proprietary, non-ARA interfaces, as long as they are not contradicting with the AP standard. However, an implementation shall not add declarations or definitions that are not specified in an SWS to the namespace ara or any of its direct sub-namespaces.

⌋*(RS_Main_00150)*

[RS_AP_00130] AUTOSAR Adaptive Platform shall represent a rich and modern programming environment. ⌈

Description:	AUTOSAR Adaptive Platform shall represent a rich and modern programming environment
Rationale:	Programmer productivity is an important aspect of any software framework. By providing and using advanced types and APIs, productivity is improved, and the platform's attractiveness increases.
Dependencies:	–
Use Case:	Some of these advanced types and APIs might be originally designed by AUTOSAR, whereas others might be back-ported from more recent C++ standards than defined by [RS_AP_00114].
Supporting Material:	–

⌋*(RS_Main_00420)*

4.2.1 Design Requirements

[RS_AP_00114] C++ interface shall be compatible with C++14. ⌈

Description:	The interface of AUTOSAR Adaptive Platform shall be compatible with C++14.
Rationale:	The interface of AUTOSAR Adaptive platform is designed to be compatible with C++14 due to high availability of C++14 compiler for embedded devices. Nevertheless projects are free to use newer C++ version like C++17. Adaptive Platform vendors may restrict their package to a newer C++ version (e.g. to support newer build systems).
Dependencies:	RS_Main_00513
Use Case:	To manage the complexity of the application development, the Adaptive platform shall support object-oriented programming. C++ is the programming language which supports object-oriented programming and is best suited for performance-critical and real-time applications.
Supporting Material:	ISO/IEC 14882

⌋(*RS_Main_00513*)

[RS_AP_00151] C++ Core Guidelines. ⌈

Description:	AUTOSAR C++ APIs should follow the "C++ Core Guidelines". The exceptions for hard-real-time systems shall apply. The AUTOSAR guidelines defined in RS-General shall overrule the "C++ Core Guidelines" in case of conflict. If a part of the AUTOSAR C++ API cannot follow the "C++ Core Guidelines" for some other reason, its specification shall state the rationale (how this is done in detail, shall be aligned with the Architecture group)
Rationale:	These guidelines are well accepted in the market. Their aim is to help C++ programmers writing simpler, more efficient, more maintainable code. Specific guidelines for the automotive domain for C++ 14 are not available. When the upcoming version of the MISRA C++ standard is published, this decision/requirement may be replaced by a decision/requirement to follow MISRA C++.
Dependencies:	–
Use Case:	–
Supporting Material:	"C++ Core Guidelines" [3]: https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines

⌋(*RS_Main_00011*, *RS_Main_00350*)

[RS_AP_00150] Provide only interfaces that are intended to be used by AUTOSAR applications and other Functional Clusters. ⌈

Description:	It is explicitly prohibited to standardize implementation details, like: • Classes, base-classes, functions etc. that are not used on the application level or in platform extension APIs • Implementation inheritance in the public APIs • C++ SFINAE techniques of any kind • Private members of classes
Rationale:	Provide only narrow interfaces to avoid coupling to implementation details. Hide implementation details because by AUTOSAR definition the implementation details are on the platform vendor.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋*(RS_Main_00011, RS_Main_00350)*

[RS_AP_00115] Public namespaces. ⌈

Description:	The namespace of Adaptive Platform in global scope shall be "ara". Within "ara" namespace each Functional Cluster shall have exactly one own namespace with its shortname (defined in [4]). No other namespaces directly below "ara" are allowed. All names shall use lower-case only. Underscores may be used.
Rationale:	Harmonized look and feel.
Dependencies:	–
Use Case:	–

⌋*(RS_Main_00500, RS_Main_00150)*

[RS_AP_00154] Internal namespaces. ⌈

Description:	Within each Functional Cluster's namespace, the sub-namespace ::internal shall be reserved for vendor-specific use.
Rationale:	–
Dependencies:	–
Use Case:	–

⌋*(RS_Main_00500, RS_Main_00150)*

[RS_AP_00116] Header file name. ⌈

Description:	All Functional Clusters shall provide a self-contained header file for each public class (except scoped enum and exceptions). The header file name shall be derived from the class name. All header file names shall have the extensions .h.
Rationale:	Harmonized look and feel.
Dependencies:	–
Use Case:	–
Supporting Material:	Google C++ Style Guide: `https://google.github.io/styleguide/cppguide.html`

⌋(*RS_Main_00500*, *RS_Main_00150*)

[RS_AP_00122] Type names. ⌈

Description:	For all Functional Clusters the name of their public types - classes, structs, type aliases, and type template parameters • shall be standardized in upper camel case. • underscores shall not be used. Except for fixed width integer types, postfix _t shall not be used. • capitalized acronyms shall be used as single words. Further the following exception is given: **exception:** all requirements and expectations that the C++ language standard or the C++ standard library place on the naming of certain symbols shall be heeded for all types and functions. Examples: nested type definitions that help with template metaprogramming such as value_type, size_type etc.
Rationale:	–
Dependencies:	–
Use Case:	Harmonized look and feel.
Supporting Material:	**CamelCase:** see [5] **STL:** see [6] **Google C++ Style Guide:** see [7]

⌋(*RS_Main_00500*, *RS_Main_00150*)

[RS_AP_00120] Method and Function names. ⌈

Description:	For all Functional Clusters, the name of their public methods and functions shall use upper camel case. Further underscores shall not be used. Capitalized acronyms shall be used as single words. Further the following exceptions are given: **exception 1:** any function that fundamentally replicates a function which has been defined by an external standard (including, but not limited to, the C++ standard) shall keep that external standard's naming rules for that function, and for all symbols associated with it, including any external functions that are highly integrated with it. **exception 2:** all requirements and expectations that the C++ language standard or the C++ standard library place on the naming of certain symbols shall be heeded for all functions.
Rationale:	For the exceptions mentioned above the following rationals are given: **Rational for exception 2:** Certain special member functions and types cannot adopt the principal AUTOSAR naming rules, because their naming is defined by the C++ standard. Amongst these are: all operator functions, begin()/end() and all their variations, and virtual functions inherited from base classes of the C++ standard library.
Dependencies:	–
Use Case:	–
Supporting Material:	**CamelCase:** see [5] **STL:** see [6] **Google C++ Style Guide:** see [7]

⌋*(RS_Main_00500, RS_Main_00150)*

[RS_AP_00121] Parameter names. ⌈

Description:	For all Functional Clusters, the name of parameters in public methods shall use lower camel case. Further underscores shall not be used. Capitalized acronyms shall be used as single words.
Rationale:	Harmonized look and feel.
Dependencies:	–
Use Case:	–
Supporting Material:	CamelCase: see [5]

⌋*(RS_Main_00500, RS_Main_00150)*

[RS_AP_00124] Variable names. ⌈

Description:	For all Functional Clusters, the name of their public variables (like Common Variable names, Class Data Members and Struct Data Members) shall use lower camel case. Further underscores shall not be used. Capitalized acronyms shall be used as single words.
Rationale:	Harmonized look and feel.
Dependencies:	–
Use Case:	–
Supporting Material:	CamelCase: see [5]

⌋*(RS_Main_00500, RS_Main_00150)*

[RS_AP_00125] Enumerator and constant names. ⌈

Description:	For all Functional Clusters, the name of public enumerations shall use upper camel case. The individual enumerators and constants shall be written with a leading "k" followed by upper camel case. Further underscores shall not be used. Capitalized acronyms shall be used as single words.
Rationale:	Harmonized look and feel.
Dependencies:	–
Use Case:	–
Supporting Material:	CamelCase: see [5]

⌋*(RS_Main_00500, RS_Main_00150)*

[RS_AP_00141] Usage of out parameters. ⌈

Description:	Out parameters can be used for inplace modifications but shall not be used for returning values.
Rationale:	Harmonized look and feel.
Dependencies:	–
Use Case:	–
Supporting Material:	• See ArcDecision in AP EXP SWArchitecturalDecision *Usage of out parameters*. • C++ Core Guidelines [3]: F.20: For "out" output values, prefer return values to output parameters.

⌋*(RS_Main_00150)*

[RS_AP_00119] Return values / application errors. ⌈

Description:	All API function specifications shall give the exact list of errors (linked to the ErrorDomains which define them) that can originate from them, and which situations can cause which of those errors. Furthermore, for return values (especially integral, floating-point, enumeration, and string), the exact range of possible values shall be specified.
Rationale:	Harmonized look and feel.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00150*)

[RS_AP_00138] Return type of asynchronous function calls. ⌈

Description:	Asynchronous function calls that need to return a value, or that can potentially fail should use ara::core::Future as return type.
Rationale:	Harmonized look and feel.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00150*)

[RS_AP_00139] Return type of synchronous function calls. ⌈

Description:	Synchronous function calls that can potentially fail should use ara::core::Result as return type and use it for returning both values and errors.
Rationale:	Harmonized look and feel.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00150*)

[RS_AP_00142] Handling of unsuccessful operations. ⌈

Description:	Functional Clusters shall differentiate recoverable unsuccessful operations from non-recoverable ones.
Rationale:	–
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋*(RS_Main_00010, RS_Main_00011)*

[RS_AP_00132] noexcept behavior of API functions ⌈

Description:	Each library function having a wide contract that cannot throw or shall never throw should be marked as unconditionally noexcept.
Rationale:	–
Dependencies:	–
Use Case:	Safety-related projects
Supporting Material:	A function has a “wide contract” if it does not specify any undefined behavior. It therefore does not put any additional runtime constraints on its arguments, any object state, or any global state. The opposite of a “wide contract” is called a “narrow contract”. An example of a function with a wide contract would be `ara::core::Vector<T>::size()`. An example of a function with a narrow contract would be `ara::core::Vector<T>::front()`, because it has the precondition that the container must not be empty. This requirement is based on the “Adopted Guidelines” from the document N3279: see [8]

⌋*(RS_Main_00010, RS_Main_00012, RS_Main_00350)*

[RS_AP_00134] noexcept behavior of class destructors ⌈

Description:	No class destructor should throw. They should use an explicitly supplied “noexcept” specifier.
Rationale:	–
Dependencies:	–
Use Case:	Safety-related projects
Supporting Material:	N3279: see [8]

⌋*(RS_Main_00010, RS_Main_00012, RS_Main_00350)*

[RS_AP_00133] noexcept behavior of move and swap operations ⌈

Description:	If a library swap function, move-constructor, or move-assignment operator is conditionally-wide (i.e. can be proven to not throw by applying the noexcept operator) then it should be marked as conditionally noexcept. No other function should use a conditional noexcept specification.
Rationale:	–
Dependencies:	–
Use Case:	Safety-related projects
Supporting Material:	N3279: see [8]

⌋(*RS_Main_00010, RS_Main_00012, RS_Main_00350*)

[RS_AP_00153] Assignment operators should restrict "this" to lvalues ⌈

Description:	All specifications of assignment operators should be declared with the ref-qualifier &.
Rationale:	Assigning to temporaries is rarely, if ever, useful, and is more likely the result of a programming mistake. Adding the "&" ref-qualifier lets the compiler detect and reject such code.
Dependencies:	–
Use Case:	Safety-related projects
Supporting Material:	HIC++ v4.0, see [9]

⌋(*RS_Main_00420*)

[RS_AP_00144] Availability of a named constructor. ⌈

Description:	If the construction of an object can fail in a way that is recoverable by the caller, the class shall have named constructors returning a Result in addition to its regular constructors. Unless other considerations apply, the name of a named constructor should be *Create*, and its arguments shall be the same as those of the corresponding regular constructor.
Rationale:	All objects should be valid after their construction.
Dependencies:	–
Use Case:	–
Supporting Material:	• See ArcDecision in AP EXP SWArchitecturalDecision *Describe chosen approaches to object creation* • C++ Core Guidelines [3]: C.42: If a constructor cannot construct a valid object, throw an exception.

⌋(*RS_Main_00010, RS_Main_00012, RS_Main_00350*)

[RS_AP_00152] Faults inside constructor. ⌈

Description:	Calling a constructor that may throw exceptions as part of its defined behavior shall result in a compilation error if the compiler toolchain does not support exceptions. The compilation error shall result from a static_assert with the error message "This constructor requires exception support.".
Rationale:	Unintended calls to constructors that may throw exceptions are detected at compile time. static_assert is the only viable option. Declaring the constructor protected or private is more complicated. Moreover, static_assert supports a customized error message which explicitly states the cause.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋*(RS_Main_00010, RS_Main_00012, RS_Main_00350)*

[RS_AP_00145] Availability of special member functions. ⌈

Description:	The rule of five shall apply. If it is necessary to define or =delete any copy, move, or destructor function, define or =delete them all. It is necessary to define own constructors, if the default (or implicit created) constructors will not create valid and fully initialized object.
Rationale:	Consistency.
Dependencies:	–
Use Case:	–
Supporting Material:	• See ArcDecision in AP EXP SWArchitecturalDecision *Describe chosen approaches to object creation* • C++ Core Guidelines [3]: – C.21: If you define or =delete any copy, move, or destructor function, define or =delete them all – C.41: A constructor should create a fully initialized object

⌋*(RS_Main_00011, RS_Main_00160)*

[RS_AP_00146] Classes whose construction requires interaction by the ARA framework. ⌈

Description:	A class which is not intended to be constructible by application shall delete the default constructor. Rationale: To show the intent that this class is not intended to be constructible by the application.
Rationale:	–
Dependencies:	–
Use Case:	–
Supporting Material:	See ArcDecision in AP EXP SWArchitecturalDecision *Describe chosen approaches to object creation*

⌋(*RS_Main_00011*, *RS_Main_00160*)

[RS_AP_00147] Classes which are created by an InstanceSpecifer shall not be copyable, but at most movable. ⌈

Description:	Classes which are created by an InstanceSpecifer shall not be copyable, but may be non-throwing movable (noexcept).
Rationale:	To only have one way to construct the object and register the internals.
Dependencies:	–
Use Case:	–
Supporting Material:	See ArcDecision in AP EXP SWArchitecturalDecision *Describe chosen approaches to object creation*

⌋(*RS_Main_00011*, *RS_Main_00160*)

[RS_AP_00127] Usage of ara::core types. ⌈

Description:	ARA interface shall use ara::core types instead of C++ standard types if ara::core provides the equivalent types.
Rationale:	–
Dependencies:	–
Use Case:	The ara::core types shall define common types in AP. Furthermore, it allows platform vendors to e.g. make use of own allocators for safety related projects.
Supporting Material:	–

⌋*(RS_Main_00010, RS_Main_00012, RS_Main_00350)*

[RS_AP_00143] Use 32-bit integral types by default. ⌈

Description:	Type aliases to integral types, and scoped enum base types should prefer 32-bit types over 16-bit or 8-bit ones.
Rationale:	Many CPUs lack instructions to handle such types efficiently.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋*(RS_Main_00002, RS_Main_00513)*

[RS_AP_00129] Public types defined by functional clusters shall be designed to allow implementation without dynamic memory allocation. ⌈

Description:	Public types defined by functional clusters shall be designed to allow implementation without dynamic memory allocation after the init-phase (i.e. after reaching Execution State Running of Execution Management).
Rationale:	Memory allocator used in the project needs to guarantee that memory allocation and deallocation are executed within defined time constraints that are appropriate for the response time constraints defined for the real-time system and its programs.
Dependencies:	–
Use Case:	Safety related projects
Supporting Material:	See ArcDecision in AP EXP SWArchitecturalDecision *Dynamic memory allocation*.

⌋*(RS_Main_00010, RS_Main_00012, RS_Main_00350)*

[RS_AP_00135] Avoidance of shared ownership. ⌈

Description:	APIs shall be designed in a way that the ownership of each data is unique. This is achieved either by transferring ownership between caller and callee (e.g. by means of std::move) or by creating a copy of data at the receiver. In case of ownership transfer usage of unique_ptr instead of shared_ptr shall be used. In case of asynchronous operations the type ara::core::Future shall be used to avoid introduction of own shared states.
Rationale:	Unique ownership is conceptually simpler and more predictable (responsibility for destruction) to manage.
Dependencies:	–
Use Case:	–
Supporting Material:	See ArcDecision in AP EXP SWArchitecturalDecision *Using local proxy object*.

⌋*(RS_Main_00010, RS_Main_00012, RS_Main_00350)*

[RS_AP_00136] Usage of string types. ⌈

Description:	The default encoding of any string type (like ara::core::String or ara::core::StringView) in the ARA interfaces shall be UTF-8. In case the encoding is deviating from UTF-8, it shall be documented in the API definition (including the rationale as a note).
Rationale:	Harmonized usage
Dependencies:	–
Use Case:	Compatibility of strings in the platform
Supporting Material:	UTF-8: ISO/IEC 10646

⌋*(RS_Main_00010, RS_Main_00012, RS_Main_00350)*

[RS_AP_00137] Connecting run-time interface with model. ⌈

Description:	Any reference of an API on application level to another element in the model shall refer to the other element using an ara::core::InstanceSpecifier. Modeling shall be done with PortPrototypes. No alternative methods of creating references to other elements in the model, such as FC-defined IDs are allowed.
Rationale:	Decoupling of interfaces and harmonized look and feel.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋*(RS_Main_00160, RS_Main_00150, RS_Main_00513)*

[RS_AP_00140] Usage of "final specifier" in ara types. ⌈

Description:	ARA types shall use the "final specifier", unless they are meant to be used as a base class.
Rationale:	Clear expression of the design (class hierarchy). Avoid problems that arise when deriving of a type which is not prepared for sub-classing.
Dependencies:	–
Use Case:	–
Supporting Material:	See ArcDecision in AP EXP SWArchitecturalDecision *Making Adaptive Runtime classes final*.

⌋*(RS_Main_00010, RS_Main_00012)*

[RS_AP_00148] Default arguments are not allowed in virtual functions. ⌈

Description:	Default arguments shall not be used at all in virtual functions.
Rationale:	The according RQ of the "C++ core guidelines" are too weak .. (they state, that it needs to be made sure that a default argument is always the same) ... this would lead to code duplication with dependencies and high risks of inconsistencies, which can easily lead to unexpected behavior.
Dependencies:	–
Use Case:	–
Supporting Material:	C++ Core Guidelines [3]: C.140: Do not provide different default arguments for a virtual function and an overrider

⌋*(RS_Main_00010, RS_Main_00012)*

[RS_AP_00155] Avoidance of cluster-specific initialization functions. ⌈

Description:	If a cluster needs an explicit (de)initialization, it shall use ara::core::(De)/Initialize.
Rationale:	Avoidance of cluster-specific initialization functions.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋*(RS_Main_00011)*

4.2.2 Error handling

[RS_AP_00128] Error reporting. ⌈

Description:	Interfaces shall be designed to report recoverable errors via a suitable return type, such as ara::core::Result or ara::core::Future.
Rationale:	Few compilers in the market allows to use exceptions in safety related projects.
Dependencies:	–
Use Case:	Safety-related projects
Supporting Material:	–

⌋(*RS_Main_00010, RS_Main_00012, RS_Main_00350*)

Guidelines on recoverable errors:

- avoid "general error"-kinds of errors, e.g. *kGeneralError, kGenericError, kInternalError* - always strive to describe concrete error conditions.

  Note: It is recommended that error codes are recoverable and not too generic, but also not too specific. It is recommended to use the same error code for the same error reaction. e.g. lost daemon connection, link down, IPC corrupt, TCP/IP driver error, buffer overflow should be merged to the general communication error in ara::com.

- for error codes originating from a 3rd-party standard (e.g. ISO), prefer to take over those error code names as close to the original definitions as possible, even if that violates other of these guidelines (prefer to follow AR formatting, though, e.g. follow the *kCamelCase* formatting)

- avoid to define error codes for non-recoverable errors

[RS_AP_00149] Guidance on error handling. ⌈

Description:	Error codes for non-initialized Functional Cluster (i.e. when ara::core::Initialize() has not been called) shall be avoided.
Rationale:	A call to a non-initialized API is treated as a violation because it is a systematic error.
Dependencies:	–
Use Case:	Safety-related projects
Supporting Material:	–

⌋(*RS_Main_00010, RS_Main_00012, RS_Main_00350*)

Guidelines on error naming:

- avoid "...Error" suffixes, e.g. *kBadSomethingError* - all these enum values are errors, there is no need to mention "Error" again

- prefer singular to plural form, e.g. prefer *kInvalidArgument* over *kInvalidArguments*, even if multiple arguments may be affected

- prefer to omit predicates, e.g. prefer *kSomethingNotValid* over *kSomething*Is*NotValid*
- prefer to omit verbs, e.g. prefer *kFileNotFound* over *kFile*Was*NotFound*
- American English has to be used: e.g. modeled(AE) over modelled(BE), canceled(AE) over cancelled(BE)
- get rid of redundant suffixes like "error" (e.g. by more fine-grained errors), or "is". Example: Communication*Is*Lost*Error* vs CommunicationLost (last should be used)
- verbs should be avoided. If it is unavoidable past is preferred, e.g. ...Failed and not ...Fails
- Prefer to phrase "failed-effort"-kind of error codes as "<Something>Failed", as opposed to e.g. "CouldNot<something>" or "FailedTo<something>"
- Prefer <Subject><Adverb> over <Adjective><Subject>, e.g. "ResourceBusy" rather than "BusyResource"

5 Requirements Tracing

The following table references the requirements specified in [10] and links to the fulfillments of these.

Requirement	Description	Satisfied by
[RS_Main_00002]	AUTOSAR shall provide a software platform for high performance computing platforms	[RS_AP_00143]
[RS_Main_00010]	Safety Mechanisms	[RS_AP_00127] [RS_AP_00128] [RS_AP_00129] [RS_AP_00132] [RS_AP_00133] [RS_AP_00134] [RS_AP_00135] [RS_AP_00136] [RS_AP_00140] [RS_AP_00142] [RS_AP_00144] [RS_AP_00148] [RS_AP_00149] [RS_AP_00152]
[RS_Main_00011]	Mechanisms for Reliable Systems	[RS_AP_00142] [RS_AP_00145] [RS_AP_00146] [RS_AP_00147] [RS_AP_00150] [RS_AP_00151] [RS_AP_00155]
[RS_Main_00012]	Highly Available Systems Support	[RS_AP_00127] [RS_AP_00128] [RS_AP_00129] [RS_AP_00132] [RS_AP_00133] [RS_AP_00134] [RS_AP_00135] [RS_AP_00136] [RS_AP_00140] [RS_AP_00144] [RS_AP_00148] [RS_AP_00149] [RS_AP_00152]

[RS_Main_00150]	AUTOSAR shall support the deployment and reallocation of AUTOSAR Application Software	[RS_AP_00111] [RS_AP_00115] [RS_AP_00116] [RS_AP_00119] [RS_AP_00120] [RS_AP_00121] [RS_AP_00122] [RS_AP_00124] [RS_AP_00125] [RS_AP_00137] [RS_AP_00138] [RS_AP_00139] [RS_AP_00141] [RS_AP_00154]
[RS_Main_00160]	Interface Modeling	[RS_AP_00137] [RS_AP_00145] [RS_AP_00146] [RS_AP_00147]
[RS_Main_00350]	Documented Software Architecture	[RS_AP_00127] [RS_AP_00128] [RS_AP_00129] [RS_AP_00132] [RS_AP_00133] [RS_AP_00134] [RS_AP_00135] [RS_AP_00136] [RS_AP_00144] [RS_AP_00149] [RS_AP_00150] [RS_AP_00151] [RS_AP_00152]
[RS_Main_00420]	AUTOSAR shall use established software standards and consolidate de-facto standards for basic software functionality	[RS_AP_00130] [RS_AP_00153]
[RS_Main_00500]	AUTOSAR shall provide naming conventions	[RS_AP_00115] [RS_AP_00116] [RS_AP_00120] [RS_AP_00121] [RS_AP_00122] [RS_AP_00124] [RS_AP_00125] [RS_AP_00154]
[RS_Main_00513]	AUTOSAR shall support language bindings for different programming languages	[RS_AP_00114] [RS_AP_00137] [RS_AP_00143]

6 References

[1] Standardization Template
AUTOSAR_TPS_StandardizationTemplate

[2] Glossary
AUTOSAR_TR_Glossary

[3] C++ Core Guidelines
https://github.com/isocpp/CppCoreGuidelines/blob/master/CppCoreGuidelines.md

[4] Functional Cluster Shortnames
AUTOSAR_TR_FunctionalClusterShortnames

[5] Camel case
https://en.wikipedia.org/wiki/CamelCase

[6] Standard Template Library
https://en.wikipedia.org/wiki/Standard_Template_Library

[7] Cpp Styleguide
https://google.github.io/styleguide/cppguide.html#Type_Names

[8] Conservative use of noexcept in the Library
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2011/n3279.pdf

[9] High Integrity C++ (HIC++) v4.0
https://www.perforce.com/resources/qac/high-integrity-cpp-coding-standard

[10] Main Requirements
AUTOSAR_RS_Main

7 Change History of this Document

7.1 Change History of this document according to AUTOSAR Release 19-11

7.1.1 Added Traceables in 19-11

Number	Heading
[RS_AP_00133]	noexcept behavior of move and swap operations
[RS_AP_00135]	Avoidance of shared ownership.
[RS_AP_00136]	Usage of string types.
[RS_AP_00137]	Connecting run-time interface with model.
[RS_AP_00138]	Return type of asynchronous function calls.
[RS_AP_00139]	Return type of synchronous function calls.
[RS_AP_00140]	Usage of "final specifier" in ara types.
[RS_AP_00141]	Usage of out parameters.
[RS_AP_00142]	Handling of unsuccessful operations.

Table 7.1: Added Traceables in 19-11

7.1.2 Changed Traceables in 19-11

Number	Heading
[RS_AP_00115]	Namespaces.
[RS_AP_00116]	Header file name.
[RS_AP_00119]	Return values / application errors.
[RS_AP_00122]	Type names.
[RS_AP_00127]	Usage of ara::core types.
[RS_AP_00128]	Error reporting.
[RS_AP_00129]	Public types defined by functional clusters shall be designed to allow implementation without dynamic memory allocation.
[RS_AP_00132]	noexcept behavior of API functions
[RS_AP_00134]	noexcept behavior of class destructors

Table 7.2: Changed Traceables in 19-11

7.1.3 Deleted Traceables in 19-11

Number	Heading
[RS_AP_00113]	API specification shall comply with selected coding guidelines.
[RS_AP_00131]	Use of verbal forms to express requirement levels.

Table 7.3: Deleted Traceables in 19-11

7.2 Change History of this document according to AUTOSAR Release 20-11

7.2.1 Added Traceables in R20-11

Number	Heading
[RS_AP_00143]	Use 32-bit integral types by default.
[RS_AP_00144]	Availability of a named constructor.
[RS_AP_00145]	Availability of special member functions.
[RS_AP_00146]	Classes whose construction requires interaction by the ARA framework.
[RS_AP_00147]	Classes which are created by an InstanceSpecifer shall not be copyable, but at most movable.

Table 7.4: Added Traceables in R20-11

7.2.2 Changed Traceables in R20-11

Number	Heading
[RS_AP_00114]	C++ interface shall be compatible with C++14.
[RS_AP_00129]	Public types defined by functional clusters shall be designed to allow implementation without dynamic memory allocation.

Table 7.5: Changed Traceables in R20-11

7.2.3 Deleted Traceables in R20-11

none

7.3 Change History of this document according to AUTOSAR Release 21-11

7.3.1 Added Traceables in R21-11

Number	Heading
[RS_AP_00148]	Default arguments are not allowed in virtual functions.
[RS_AP_00149]	Guidance on error handling.
[RS_AP_00150]	Provide only interfaces that are intended to be used by AUTOSAR applications and other Functional Clusters.
[RS_AP_00151]	C++ Core Guidelines.
[RS_AP_00152]	Faults inside constructor.
[RS_AP_00153]	Assignment operators should restrict "this" to lvalues
[RS_AP_00154]	Internal namespaces.
[RS_AP_00155]	Avoidance of cluster-specific initialization functions.

Table 7.6: Added Traceables in R21-11

7.3.2 Changed Traceables in R21-11

Number	Heading
[RS_AP_00111]	The AUTOSAR Adaptive Platform shall support source code portability for AUTOSAR Adaptive applications.
[RS_AP_00115]	Public namespaces.
[RS_AP_00129]	Public types defined by functional clusters shall be designed to allow implementation without dynamic memory allocation.
[RS_AP_00133]	noexcept behavior of move and swap operations
[RS_AP_00135]	Avoidance of shared ownership.
[RS_AP_00140]	Usage of "final specifier" in ara types.
[RS_AP_00141]	Usage of out parameters.
[RS_AP_00144]	Availability of a named constructor.
[RS_AP_00145]	Availability of special member functions.
[RS_AP_00146]	Classes whose construction requires interaction by the ARA framework.
[RS_AP_00147]	Classes which are created by an InstanceSpecifer shall not be copyable, but at most movable.

Table 7.7: Changed Traceables in R21-11

7.3.3 Deleted Traceables in R21-11

none