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

Document Status	Final
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	19-03

Document Change History

Date	Release	Changed by	Description
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. ⌈

Type:	draft
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 sub-namespaces.

⌋*(RS_Main_00150)*

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

Type:	draft
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)*

[RS_AP_00131] Use of verbal forms to express requirement levels. ⌈

Type:	draft
Description:	The following verbal forms for the expression of obligation shall be used to indicate requirements. The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as follows, based on [3]. Note that the requirement level of the document in which they are used modifies the force of these words. • MUST: This word, or the adjective "LEGALLY REQUIRED", means that the definition is an absolute requirement of the specification due to legal issues. • MUST NOT: This phrase, or the phrase "MUST NOT", means that the definition is an absolute prohibition of the specification due to legal issues. • SHALL: This phrase, or the adjective "REQUIRED", means that the definition is an absolute requirement of the specification. • SHALL NOT: This phrase means that the definition is an absolute prohibition of the specification. • SHOULD: This word, or the adjective "RECOMMENDED", means that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course. • SHOULD NOT: This phrase, or the phrase "NOT RECOMMENDED", means that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label. • MAY: This word, or the adjective "OPTIONAL", means that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation, which does not include a particular option, SHALL be prepared to interoperate with another implementation, which does include the option, though perhaps with reduced functionality. In the same vein an implementation, which does include a particular option, SHALL be prepared to interoperate with another implementation, which does not include the option (except, of course, for the feature the option provides.)
Rationale:	
Dependencies:	–
Use Case:	–
Supporting Material:	TPS_STDT_00053

⌋(*RS_STDT_00014*)

4.2.1 Design Requirements

[RS_AP_00113] API specification shall comply with selected coding guidelines. ⌈

Type:	draft
Description:	API specification shall comply with coding guideline "Guidelines for the use of the C++14 language in critical and safety-related systems" in order to adopt best practices for the safe and secure implementations.
Rationale:	Use of appropriate coding guidelines are essential and also critical in safety/security related systems.
Dependencies:	–
Use Case:	–
Supporting Material:	"Guidelines for the use of the C++14 language in critical and safety-related systems" [4]

⌋*(RS_Main_00500, RS_Main_00010)*

[RS_AP_00114] C++ interface shall be compatible with C++11. ⌈

Type:	draft
Description:	The interface of AUTOSAR Adaptive Platform shall be compatible with C++11.
Rationale:	The interface of AUTOSAR Adaptive platform is designed to be compatible with C++11 due to high availability of C++11 compiler for embedded devices. Nevertheless projects are free to use newer C++ version like C++14. 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_00115] Namespaces. ⌈

Type:	draft
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 [5]). No other namespaces below "ara" are allowed. All names shall use lower-case only. Underscores may be used.
Rationale:	Harmonized look and feel.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋(*RS_Main_00500*, *RS_Main_00150*)

[RS_AP_00116] Header file name. ⌈

Type:	draft
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 as recommended by Google C++ Style Guide. 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. ⌈

Type:	draft
Description:	For all Functional Clusters the name of their public types - classes, structs, type aliases, enums, 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 [6] **STL:** see [7] **Google C++ Style Guide:** see [8]

⌋(*RS_Main_00500*, *RS_Main_00150*)

[RS_AP_00120] Method and Function names. ⌈

Type:	draft
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 [6] **STL:** see [7] **Google C++ Style Guide:** see [8]

⌋*(RS_Main_00500, RS_Main_00150)*

[RS_AP_00121] Parameter names. ⌈

Type:	draft
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 [6]

⌋*(RS_Main_00500, RS_Main_00150)*

[RS_AP_00124] Variable names. ⌈

Type:	draft
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 [6]

⌋*(RS_Main_00500, RS_Main_00150)*

[RS_AP_00125] Enumerator and constant names. ⌈

Type:	draft
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 [6]

⌋*(RS_Main_00500, RS_Main_00150)*

[RS_AP_00119] Return values / application errors. ⌈

Type:	draft
Description:	For return values and application errors, the condition(s) when they are returned shall be specified. Furthermore for each return value the list of valid output parameter(s) shall be specified.
Rationale:	Harmonized look and feel.
Dependencies:	–
Use Case:	–
Supporting Material:	–

⌋*(RS_Main_00150)*

[RS_AP_00128] Use of exceptions in API. ⌈

Type:	draft
Description:	Interfaces in AP shall be designed to use ara::result for checked exceptions or any other recoverable error. Unchecked exceptions should only be used for non-recoverable errors.
Rationale:	Few compilers in the market allows to use exceptions in safety related projects.
Dependencies:	–
Use Case:	Saftey-related projects
Supporting Material:	–

⌋()

[RS_AP_00132] Usage of noexcept keyword. ⌈

Type:	draft
Description:	Each library function having a wide contract (e.g. logging APIs), should be standardized as unconditionally noexcept. In general the noexcept keyword shall be handled with care (see also document "Conservative use of noexcept in the Library" N3279).
Rationale:	–
Dependencies:	–
Use Case:	Saftey-related projects
Supporting Material:	N3279: see [9]

⌋()

[RS_AP_00134] Library destructors shall be tagged with noexcept. ⌈

Type:	draft
Description:	No library destructor should throw. They shall use the implicitly supplied (nonthrowing) exception specification.
Rationale:	–
Dependencies:	–
Use Case:	Common sense
Supporting Material:	N3279: see [9]

⌋()

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

Type:	draft
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_AP_00129] Public types defined by functional clusters shall be designed to allow implementation without dynamic memory allocation. ⌈

Type:	draft
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:	–
Dependencies:	–
Use Case:	Safety related projects
Supporting Material:	Rule A18-5-7 of "Guidelines for the use of the C++14 language in critical and safety-related systems" [4]

⌋()

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_00010]	AUTOSAR shall support the development of safety related systems	[RS_AP_00113]
[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_Main_00420]	AUTOSAR shall use established software standards and consolidate de-facto standards for basic software functionality	[RS_AP_00130]
[RS_Main_00500]	AUTOSAR shall provide naming conventions	[RS_AP_00113] [RS_AP_00115] [RS_AP_00116] [RS_AP_00120] [RS_AP_00121] [RS_AP_00122] [RS_AP_00124] [RS_AP_00125]
[RS_Main_00513]	AUTOSAR shall support language bindings for different programming languages	[RS_AP_00114]
[RS_STDT_00014]	No description	[RS_AP_00131]

6 References

[1] Standardization Template
AUTOSAR_TPS_StandardizationTemplate

[2] Glossary
AUTOSAR_TR_Glossary

[3] Key words for use in RFCs to Indicate Requirement Levels
http://www.ietf.org/rfc/rfc2119.txt

[4] Guidelines for the use of the C++14 language in critical and safety-related systems
AUTOSAR_RS_CPP14Guidelines

[5] Functional Cluster Shortnames
AUTOSAR_TR_FunctionalClusterShortnames

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

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

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

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

[10] Main Requirements
AUTOSAR_RS_Main