Document Title	Specification of Adaptive Platform Core
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	903

Document Status	published
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	R24-11

Document Change History

Date	**Release**	**Changed by**	**Description**
2024-11-27	R24-11	AUTOSAR Release Management	• Extend specification of ara::core::MemoryResource and derived classes • Add full specification of ara::core::Optional, ara::core::Variant, and ara::core::StringView • Add T& specializations of ara::core::Optional and ara::core::Result • Specify exception safety and thread safety of ARA APIs • Mandate ara::core::ErrorCode as ErrorType of Result and Future/Promise • Various extensions and fixes to the C++ data types • Adapt the document to the new template with a generated Chapter 8 • Refine specification about platform initialization • Enable ara::core::Initialize() take command line arguments • Refine Violation specification of ARA APIs with standardized Violation messages

▽

△

2023-11-23	R23-11	AUTOSAR Release Management	• Add specification of ara::core::MemoryResource • Remove specification of ara::core::ScaleLinearAndTexttable • Refine specification about platform initialization • Refine specification of Future, and Promise with regards to error handling • Extend Array specification with accessor functions performing checked access • Make undefined behavior explicit by mandating Violations across various C++ data types • Rework of chapter 5 with dependencies to other modules
2022-11-24	R22-11	AUTOSAR Release Management	• Extend ara::core::Abort to allow multiple arguments • Add support for registering multiple AbortHandlers • Merge header files of ara::core::Future and ara::core:Promise into a single one • Add full specification of ara::core::String and ara::core::BasicString • Forbid user extensions of standardized AUTOSAR namespaces
2021-11-25	R21-11	AUTOSAR Release Management	• Add spec items for error handling definitions • Add specifications for ScaleLinearAndTexttable, taken over from SWS_CommunicationManagement • Refine scope of ara::core::Initialize • Adapt some APIs to C++14's enhanced capabilities • Align Span with std::span from the C++20 standard ▽

▽

△

			△ • Reduce requirements imposed on handling Violations • Rename document into “Adaptive Platform Core”
2020-11-30	R20-11	AUTOSAR Release Management	• Add specifications about “Explicit Operation Abortion” • Add specification about reserved symbol prefixes • Add specification of class SteadyClock • Add section about async signal safety of ARA APIs • Extend error domain scope with vendor-defined error domains • Add specifications about defining own error domains • Various extensions and fixes to the C++ data types • Incorporate contents of SWS_General • Rename document into “Adaptive Core”
2019-11-28	R19-11	AUTOSAR Release Management	• Rework error handling definitions • Add specifications of BasicString and Byte, and add overloads and template specializations for ErrorCode, Result, Future, and Promise • Add bits about validity of InstanceSpecifier arguments, and rework the specification of its construction mechanism • Rework ErrorCode to get rid of “User Message” and make “SupportDataType” implementation-defined • Replace PosixErrorDomain with CoreErrorDomain • Rename FutureErrorDomain accessor function ▽

▽

△

			△ • Changed Document Status from Final to published
2019-03-29	19-03	AUTOSAR Release Management	• Add specification of the template specialization Result<void, E>
2018-10-31	18-10	AUTOSAR Release Management	• Add chapter 2 with acronyms • Add chapter 4 with limitations of the current specifications • Add chapter 5 with dependencies to other modules • Add chapter 7 • Add classes representing the approach to error handling to chapter 8 • Adapt classes Future and Promise to the error handling approach • Add global functions for initialization and shutdown of the framework • Add class InstanceSpecifier to chapter 8 • Add more types and functions from the C++ standard
2018-03-29	18-03	AUTOSAR Release Management	• Initial Release

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.

Contents

1 Introduction and functional overview

This specification describes the functionality, API and the configuration for the functional cluster Core. It defines basic requirements that apply to all Functional Clusters of the Adaptive Platform.

To aid in this, it also defines functionality that applies to the entire framework, including a set of common data types used by multiple Functional Clusters as part of their public interfaces.

2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations that are only relevant to Core. A general list of acronyms and abbreviations is available in the [1, AUTOSAR glossary].

Abbreviation / Acronym:	Description:
Explicitly aborting an Operation	Immediately aborting an API call, which is initiated by calling ara::core::Abort, usually as a consequence of the detection of a Violation
UUID	*Universally Unique Identifier*, a 128-bit number used to identify information in computer systems
Violation	The semantics of a Violation are defined in [SWS_CORE_00021]
Standardized Violation	A Violation that is standardized in the AUTOSAR AP standard
Non-Standardized Violation	A Violation that is not standardized in the AUTOSAR AP standard

Table 2.1: Acronyms and abbreviations used in the scope of this Document

3 Related documentation

3.1 Input documents & related standards and norms

[1] Glossary
AUTOSAR_FO_TR_Glossary

[2] Explanation of Adaptive Platform Software Architecture
AUTOSAR_AP_EXP_SWArchitecture

[3] Main Requirements
AUTOSAR_FO_RS_Main

[4] General Requirements specific to Adaptive Platform
AUTOSAR_AP_RS_General

[5] ISO/IEC 14882:2014, Information technology – Programming languages – C++
https://www.iso.org

[6] Explanation of Adaptive and Classic Platform Software Architectural Decisions
AUTOSAR_FO_EXP_SWArchitecturalDecisions

[7] Explanation of Adaptive Platform Design
AUTOSAR_AP_EXP_PlatformDesign

[8] Specification of Log and Trace
AUTOSAR_AP_SWS_LogAndTrace

[9] AUTOSAR Vendor ID List
https://www.autosar.org/vendor-id

[10] ValueOrError and ValueOrNone types
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p0786r1.pdf

[11] ISO/IEC 14882:2017, Programming languages – C++
https://www.iso.org

[12] N3857: Improvements to std::future<T> and Related APIs
https://isocpp.org/files/papers/N3857.pdf

[13] Standard for Information Technology–Portable Operating System Interface (POSIX(R)) Base Specifications, Issue 7
http://pubs.opengroup.org/onlinepubs/9699919799/

[14] Specification of Execution Management
AUTOSAR_AP_SWS_ExecutionManagement

[15] Explanation of ara::com API
AUTOSAR_AP_EXP_ARAComAPI

[16] ISO/IEC 14882:2020, Programming languages – C++
https://www.iso.org

[17] N4950: Working Draft, Standard for Programming Language C++
https://open-std.org/JTC1/SC22/WG21/docs/papers/2023/n4950.pdf

[18] Specification of Manifest
AUTOSAR_AP_TPS_ManifestSpecification

4 Constraints and assumptions

4.1 Known limitations

- The specification of some data types (Array, Map, Optional, String, StringView, Variant) mentions “supporting constructs”, but lacks a precise scope definition of this term.
- The specification of some data types (Map, Vector, String) is lacking a comprehensive definition of memory allocation behavior; it currently only describes it as “implementation-defined”. The interacton with ara::core::MemoryResource and derived types is not defined.
- Chapter 7 (“Functional specification”) describes some behavior informally that should rather be given as specification items.
- Some parts of the chapter 8 (“API specification”) (SynchronizedPoolResource, UnsynchronizedPoolResource, Executor, String, some constructors) have not been assigned a thread safety category as specified in the chapter 7.3 (“Thread safety”).

5 Dependencies to other Functional Clusters

AUTOSAR decided not to standardize interfaces which are exclusively used between Functional Clusters (on platform-level only), to allow efficient implementations, which might depend e.g. on the used Operating System.

This chapter provides an informative guideline of the interaction of Core with other Functional Clusters in the AUTOSAR Adaptive Platform. Section 5.1 "Provided Interfaces" lists the public interfaces provided by Core to other Functional Clusters. Section 5.2 "Required Interfaces" lists the public interfaces required by Core.

The goal is to provide a clear understanding of Functional Cluster boundaries and interaction, without specifying syntactical details. This ensures compatibility between documents specifying different Functional Clusters and supports parallel implementation of different Functional Clusters. Details of internal interfaces are up to the platform provider. Additional internal interfaces, parameters and return values can be added.

A detailed technical architecture documentation of the overall AUTOSAR Adaptive Platform is provided in [2].

5.1 Provided Interfaces

Table 5.1 provides a complete list of interfaces provided to other Functional Clusters within the AUTOSAR Adaptive Platform.

Interface	*Functional Cluster*	*Purpose*
No provided interfaces		

Table 5.1: Interfaces provided to other Functional Clusters

5.2 Required Interfaces

Table 5.2 provides a complete list of required interfaces from other Functional Clusters within the AUTOSAR Adaptive Platform.

Functional Cluster	*Interface*	*Purpose*
No required interfaces		

Table 5.2: Interfaces required from other Functional Clusters

5.3 Functional Cluster initialization

`ara::core::Initialize` and `ara::core::Deinitialize` initialize and de-initialize other Functional Clusters as necessary for the particular implementation. All Functional Clusters where this is necessary thus need to provide internal interfaces for their initialization and de-initialization.

6 Requirements Tracing

The following tables reference the requirements specified in AUTOSAR RS Main [3] and AUTOSAR RS General [4], and links to the fulfillment of these.

Requirement	Description	Satisfied by
[RS_AP_00111]	Source Code Portability Support	[SWS_CORE_00055] [SWS_CORE_15005] [SWS_CORE_90001] [SWS_CORE_90002] [SWS_CORE_90003] [SWS_CORE_90004] [SWS_CORE_90005] [SWS_CORE_90007] [SWS_CORE_90021] [SWS_CORE_90022]
[RS_AP_00115]	Public namespaces	[SWS_CORE_90025]
[RS_AP_00116]	Header file name	[SWS_CORE_90001]
[RS_AP_00119]	Return values / application errors	[SWS_CORE_10301] [SWS_CORE_10302] [SWS_CORE_10303] [SWS_CORE_10304] [SWS_CORE_10401] [SWS_CORE_10600]
[RS_AP_00127]	Usage of ara::core types	[SWS_CORE_00052]
[RS_AP_00128]	Error reporting	[SWS_CORE_00002] [SWS_CORE_10600] [SWS_CORE_10800]
[RS_AP_00130]	AUTOSAR Adaptive Platform shall represent a rich and modern programming environment	[SWS_CORE_00008] [SWS_CORE_00009] [SWS_CORE_00010] [SWS_CORE_00011] [SWS_CORE_00013] [SWS_CORE_00014] [SWS_CORE_00015] [SWS_CORE_00016] [SWS_CORE_00017] [SWS_CORE_00018] [SWS_CORE_00019] [SWS_CORE_00024] [SWS_CORE_00025] [SWS_CORE_00026] [SWS_CORE_00027] [SWS_CORE_00028] [SWS_CORE_00029] [SWS_CORE_00030] [SWS_CORE_00031] [SWS_CORE_00032] [SWS_CORE_00033] [SWS_CORE_00034] [SWS_CORE_00035] [SWS_CORE_00036] [SWS_CORE_00037] [SWS_CORE_00038] [SWS_CORE_00039] [SWS_CORE_00040] [SWS_CORE_00041] [SWS_CORE_00042] [SWS_CORE_00043] [SWS_CORE_00044] [SWS_CORE_00045] [SWS_CORE_00046] [SWS_CORE_00047] [SWS_CORE_00048] [SWS_CORE_00049] [SWS_CORE_00056] [SWS_CORE_00057] [SWS_CORE_00058] [SWS_CORE_00059] [SWS_CORE_00060] [SWS_CORE_00061] [SWS_CORE_00062] [SWS_CORE_00063] [SWS_CORE_00064] [SWS_CORE_00065] [SWS_CORE_00066] [SWS_CORE_00067] [SWS_CORE_00068] [SWS_CORE_00069] [SWS_CORE_00070] [SWS_CORE_00071] [SWS_CORE_00072] [SWS_CORE_00073] [SWS_CORE_00093] [SWS_CORE_00110] [SWS_CORE_00121] [SWS_CORE_00122] [SWS_CORE_00123] [SWS_CORE_00131] [SWS_CORE_00132] [SWS_CORE_00133] [SWS_CORE_00134] [SWS_CORE_00135] [SWS_CORE_00136] [SWS_CORE_00137] [SWS_CORE_00138] [SWS_CORE_00151] [SWS_CORE_00152] [SWS_CORE_00153] [SWS_CORE_00154] [SWS_CORE_00321] [SWS_CORE_00322] [SWS_CORE_00323] [SWS_CORE_00325] [SWS_CORE_00326] [SWS_CORE_00327] [SWS_CORE_00328] [SWS_CORE_00329] [SWS_CORE_00330] [SWS_CORE_00331] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SWS_CORE_00332] [SWS_CORE_00333] [SWS_CORE_00334] [SWS_CORE_00335] [SWS_CORE_00336] [SWS_CORE_00337] [SWS_CORE_00340] [SWS_CORE_00341] [SWS_CORE_00342] [SWS_CORE_00343] [SWS_CORE_00344] [SWS_CORE_00345] [SWS_CORE_00346] [SWS_CORE_00349] [SWS_CORE_00350] [SWS_CORE_00351] [SWS_CORE_00352] [SWS_CORE_00353] [SWS_CORE_00354] [SWS_CORE_00355] [SWS_CORE_00356] [SWS_CORE_00361] [SWS_CORE_00400] [SWS_CORE_00411] [SWS_CORE_00412] [SWS_CORE_00421] [SWS_CORE_00431] [SWS_CORE_00432] [SWS_CORE_00441] [SWS_CORE_00442] [SWS_CORE_00443] [SWS_CORE_00444] [SWS_CORE_00480] [SWS_CORE_00490] [SWS_CORE_00501] [SWS_CORE_00512] [SWS_CORE_00513] [SWS_CORE_00514] [SWS_CORE_00515] [SWS_CORE_00516] [SWS_CORE_00518] [SWS_CORE_00519] [SWS_CORE_00571] [SWS_CORE_00572] [SWS_CORE_00601] [SWS_CORE_00611] [SWS_CORE_00612] [SWS_CORE_00613] [SWS_CORE_00614] [SWS_CORE_00615] [SWS_CORE_00616] [SWS_CORE_00617] [SWS_CORE_00618] [SWS_CORE_00701] [SWS_CORE_00711] [SWS_CORE_00712] [SWS_CORE_00721] [SWS_CORE_00722] [SWS_CORE_00723] [SWS_CORE_00724] [SWS_CORE_00725] [SWS_CORE_00726] [SWS_CORE_00727] [SWS_CORE_00731] [SWS_CORE_00732] [SWS_CORE_00733] [SWS_CORE_00734] [SWS_CORE_00735] [SWS_CORE_00736] [SWS_CORE_00741] [SWS_CORE_00742] [SWS_CORE_00743] [SWS_CORE_00744] [SWS_CORE_00745] [SWS_CORE_00751] [SWS_CORE_00752] [SWS_CORE_00753] [SWS_CORE_00754] [SWS_CORE_00755] [SWS_CORE_00756] [SWS_CORE_00757] [SWS_CORE_00758] [SWS_CORE_00759] [SWS_CORE_00761] [SWS_CORE_00762] [SWS_CORE_00763] [SWS_CORE_00764] [SWS_CORE_00765] [SWS_CORE_00766] [SWS_CORE_00767] [SWS_CORE_00768] [SWS_CORE_00769] [SWS_CORE_00770] [SWS_CORE_00771] [SWS_CORE_00772] [SWS_CORE_00773] [SWS_CORE_00774] [SWS_CORE_00775] [SWS_CORE_00776] [SWS_CORE_00777] [SWS_CORE_00780] [SWS_CORE_00781] [SWS_CORE_00782] [SWS_CORE_00783] [SWS_CORE_00784] [SWS_CORE_00785] [SWS_CORE_00786] [SWS_CORE_00787] [SWS_CORE_00788] [SWS_CORE_00789] [SWS_CORE_00796] [SWS_CORE_00801] [SWS_CORE_00811] [SWS_CORE_00812] [SWS_CORE_00821] [SWS_CORE_00823] [SWS_CORE_00824] [SWS_CORE_00825] [SWS_CORE_00826] [SWS_CORE_00827] [SWS_CORE_00831] [SWS_CORE_00834] [SWS_CORE_00835] [SWS_CORE_00836] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SWS_CORE_00841] [SWS_CORE_00842] [SWS_CORE_00843] [SWS_CORE_00844] [SWS_CORE_00845] [SWS_CORE_00851] [SWS_CORE_00852] [SWS_CORE_00853] [SWS_CORE_00855] [SWS_CORE_00857] [SWS_CORE_00858] [SWS_CORE_00861] [SWS_CORE_00863] [SWS_CORE_00864] [SWS_CORE_00865] [SWS_CORE_00866] [SWS_CORE_00867] [SWS_CORE_00868] [SWS_CORE_00869] [SWS_CORE_00870] [SWS_CORE_00876] [SWS_CORE_00901] [SWS_CORE_00902] [SWS_CORE_00903] [SWS_CORE_00904] [SWS_CORE_00905] [SWS_CORE_00906] [SWS_CORE_00907] [SWS_CORE_00908] [SWS_CORE_00909] [SWS_CORE_00910] [SWS_CORE_00911] [SWS_CORE_00912] [SWS_CORE_00913] [SWS_CORE_00914] [SWS_CORE_00915] [SWS_CORE_00916] [SWS_CORE_00917] [SWS_CORE_00918] [SWS_CORE_00919] [SWS_CORE_00920] [SWS_CORE_00921] [SWS_CORE_00922] [SWS_CORE_00923] [SWS_CORE_00924] [SWS_CORE_00925] [SWS_CORE_00926] [SWS_CORE_00927] [SWS_CORE_00928] [SWS_CORE_00929] [SWS_CORE_00930] [SWS_CORE_00931] [SWS_CORE_00932] [SWS_CORE_00933] [SWS_CORE_00934] [SWS_CORE_00935] [SWS_CORE_00936] [SWS_CORE_00937] [SWS_CORE_00938] [SWS_CORE_00939] [SWS_CORE_00940] [SWS_CORE_00941] [SWS_CORE_01031] [SWS_CORE_01032] [SWS_CORE_01033] [SWS_CORE_01034] [SWS_CORE_01096] [SWS_CORE_01100] [SWS_CORE_01101] [SWS_CORE_01102] [SWS_CORE_01103] [SWS_CORE_01104] [SWS_CORE_01105] [SWS_CORE_01106] [SWS_CORE_01107] [SWS_CORE_01108] [SWS_CORE_01109] [SWS_CORE_01110] [SWS_CORE_01111] [SWS_CORE_01112] [SWS_CORE_01113] [SWS_CORE_01114] [SWS_CORE_01115] [SWS_CORE_01116] [SWS_CORE_01117] [SWS_CORE_01118] [SWS_CORE_01119] [SWS_CORE_01120] [SWS_CORE_01121] [SWS_CORE_01122] [SWS_CORE_01123] [SWS_CORE_01124] [SWS_CORE_01125] [SWS_CORE_01126] [SWS_CORE_01127] [SWS_CORE_01128] [SWS_CORE_01129] [SWS_CORE_01130] [SWS_CORE_01131] [SWS_CORE_01132] [SWS_CORE_01133] [SWS_CORE_01134] [SWS_CORE_01135] [SWS_CORE_01136] [SWS_CORE_01138] [SWS_CORE_01139] [SWS_CORE_01140] [SWS_CORE_01150] [SWS_CORE_01151] [SWS_CORE_01152] [SWS_CORE_01153] [SWS_CORE_01154] [SWS_CORE_01155] [SWS_CORE_01156] [SWS_CORE_01157] [SWS_CORE_01158] [SWS_CORE_01159] [SWS_CORE_01160] [SWS_CORE_01161] [SWS_CORE_01162] [SWS_CORE_01163] [SWS_CORE_01164] [SWS_CORE_01165] [SWS_CORE_01166] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SWS_CORE_01167] [SWS_CORE_01168] [SWS_CORE_01169] [SWS_CORE_01170] [SWS_CORE_01171] [SWS_CORE_01172] [SWS_CORE_01173] [SWS_CORE_01174] [SWS_CORE_01175] [SWS_CORE_01201] [SWS_CORE_01210] [SWS_CORE_01211] [SWS_CORE_01212] [SWS_CORE_01213] [SWS_CORE_01214] [SWS_CORE_01215] [SWS_CORE_01216] [SWS_CORE_01217] [SWS_CORE_01218] [SWS_CORE_01219] [SWS_CORE_01220] [SWS_CORE_01241] [SWS_CORE_01242] [SWS_CORE_01250] [SWS_CORE_01251] [SWS_CORE_01252] [SWS_CORE_01253] [SWS_CORE_01254] [SWS_CORE_01255] [SWS_CORE_01256] [SWS_CORE_01257] [SWS_CORE_01258] [SWS_CORE_01259] [SWS_CORE_01260] [SWS_CORE_01261] [SWS_CORE_01262] [SWS_CORE_01263] [SWS_CORE_01264] [SWS_CORE_01265] [SWS_CORE_01266] [SWS_CORE_01267] [SWS_CORE_01268] [SWS_CORE_01269] [SWS_CORE_01270] [SWS_CORE_01271] [SWS_CORE_01272] [SWS_CORE_01273] [SWS_CORE_01274] [SWS_CORE_01280] [SWS_CORE_01281] [SWS_CORE_01282] [SWS_CORE_01283] [SWS_CORE_01284] [SWS_CORE_01285] [SWS_CORE_01290] [SWS_CORE_01291] [SWS_CORE_01292] [SWS_CORE_01293] [SWS_CORE_01294] [SWS_CORE_01295] [SWS_CORE_01296] [SWS_CORE_01301] [SWS_CORE_01390] [SWS_CORE_01391] [SWS_CORE_01392] [SWS_CORE_01393] [SWS_CORE_01394] [SWS_CORE_01395] [SWS_CORE_01396] [SWS_CORE_01400] [SWS_CORE_01496] [SWS_CORE_01600] [SWS_CORE_01601] [SWS_CORE_01602] [SWS_CORE_01603] [SWS_CORE_01604] [SWS_CORE_01605] [SWS_CORE_01606] [SWS_CORE_01607] [SWS_CORE_01608] [SWS_CORE_01609] [SWS_CORE_01610] [SWS_CORE_01611] [SWS_CORE_01612] [SWS_CORE_01613] [SWS_CORE_01614] [SWS_CORE_01615] [SWS_CORE_01616] [SWS_CORE_01617] [SWS_CORE_01618] [SWS_CORE_01619] [SWS_CORE_01620] [SWS_CORE_01621] [SWS_CORE_01622] [SWS_CORE_01623] [SWS_CORE_01624] [SWS_CORE_01626] [SWS_CORE_01627] [SWS_CORE_01628] [SWS_CORE_01629] [SWS_CORE_01630] [SWS_CORE_01631] [SWS_CORE_01632] [SWS_CORE_01633] [SWS_CORE_01634] [SWS_CORE_01640] [SWS_CORE_01641] [SWS_CORE_01642] [SWS_CORE_01643] [SWS_CORE_01644] [SWS_CORE_01645] [SWS_CORE_01646] [SWS_CORE_01649] [SWS_CORE_01650] [SWS_CORE_01651] [SWS_CORE_01652] [SWS_CORE_01653] [SWS_CORE_01654] [SWS_CORE_01655] [SWS_CORE_01656] [SWS_CORE_01657] [SWS_CORE_01658] [SWS_CORE_01659] [SWS_CORE_01660] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SWS_CORE_01661] [SWS_CORE_01662] [SWS_CORE_01663] [SWS_CORE_01664] [SWS_CORE_01665] [SWS_CORE_01666] [SWS_CORE_01667] [SWS_CORE_01668] [SWS_CORE_01669] [SWS_CORE_01670] [SWS_CORE_01671] [SWS_CORE_01696] [SWS_CORE_01900] [SWS_CORE_01901] [SWS_CORE_01911] [SWS_CORE_01912] [SWS_CORE_01914] [SWS_CORE_01915] [SWS_CORE_01916] [SWS_CORE_01917] [SWS_CORE_01918] [SWS_CORE_01919] [SWS_CORE_01920] [SWS_CORE_01921] [SWS_CORE_01922] [SWS_CORE_01923] [SWS_CORE_01931] [SWS_CORE_01941] [SWS_CORE_01942] [SWS_CORE_01943] [SWS_CORE_01944] [SWS_CORE_01945] [SWS_CORE_01946] [SWS_CORE_01947] [SWS_CORE_01948] [SWS_CORE_01949] [SWS_CORE_01950] [SWS_CORE_01951] [SWS_CORE_01952] [SWS_CORE_01953] [SWS_CORE_01954] [SWS_CORE_01959] [SWS_CORE_01960] [SWS_CORE_01961] [SWS_CORE_01962] [SWS_CORE_01963] [SWS_CORE_01964] [SWS_CORE_01965] [SWS_CORE_01966] [SWS_CORE_01967] [SWS_CORE_01968] [SWS_CORE_01969] [SWS_CORE_01970] [SWS_CORE_01971] [SWS_CORE_01972] [SWS_CORE_01973] [SWS_CORE_01974] [SWS_CORE_01975] [SWS_CORE_01976] [SWS_CORE_01977] [SWS_CORE_01978] [SWS_CORE_01979] [SWS_CORE_01980] [SWS_CORE_01981] [SWS_CORE_01990] [SWS_CORE_01991] [SWS_CORE_01992] [SWS_CORE_01993] [SWS_CORE_01994] [SWS_CORE_02001] [SWS_CORE_02100] [SWS_CORE_02101] [SWS_CORE_02102] [SWS_CORE_02103] [SWS_CORE_02104] [SWS_CORE_02105] [SWS_CORE_02106] [SWS_CORE_02107] [SWS_CORE_02108] [SWS_CORE_02109] [SWS_CORE_02110] [SWS_CORE_02111] [SWS_CORE_02112] [SWS_CORE_02113] [SWS_CORE_02114] [SWS_CORE_02115] [SWS_CORE_02116] [SWS_CORE_02117] [SWS_CORE_02118] [SWS_CORE_02119] [SWS_CORE_02120] [SWS_CORE_02121] [SWS_CORE_02122] [SWS_CORE_02123] [SWS_CORE_02124] [SWS_CORE_02125] [SWS_CORE_02126] [SWS_CORE_02127] [SWS_CORE_02128] [SWS_CORE_02129] [SWS_CORE_02130] [SWS_CORE_02131] [SWS_CORE_02132] [SWS_CORE_02133] [SWS_CORE_02134] [SWS_CORE_02135] [SWS_CORE_02136] [SWS_CORE_02137] [SWS_CORE_02138] [SWS_CORE_02139] [SWS_CORE_02140] [SWS_CORE_02141] [SWS_CORE_02142] [SWS_CORE_02143] [SWS_CORE_02144] [SWS_CORE_02145] [SWS_CORE_02146] [SWS_CORE_02147] [SWS_CORE_02148] [SWS_CORE_02149] [SWS_CORE_02150] [SWS_CORE_02151] [SWS_CORE_02152] [SWS_CORE_02153] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SWS_CORE_02154] [SWS_CORE_02155] [SWS_CORE_02156] [SWS_CORE_02157] [SWS_CORE_02158] [SWS_CORE_02159] [SWS_CORE_02160] [SWS_CORE_02161] [SWS_CORE_02162] [SWS_CORE_02163] [SWS_CORE_02164] [SWS_CORE_02165] [SWS_CORE_02166] [SWS_CORE_02167] [SWS_CORE_02168] [SWS_CORE_02169] [SWS_CORE_02170] [SWS_CORE_02171] [SWS_CORE_02172] [SWS_CORE_02173] [SWS_CORE_02174] [SWS_CORE_02175] [SWS_CORE_02176] [SWS_CORE_02177] [SWS_CORE_02178] [SWS_CORE_02179] [SWS_CORE_02180] [SWS_CORE_02181] [SWS_CORE_02182] [SWS_CORE_02183] [SWS_CORE_02184] [SWS_CORE_02185] [SWS_CORE_02186] [SWS_CORE_02187] [SWS_CORE_02188] [SWS_CORE_02189] [SWS_CORE_02190] [SWS_CORE_03000] [SWS_CORE_03001] [SWS_CORE_03012] [SWS_CORE_03296] [SWS_CORE_03301] [SWS_CORE_03302] [SWS_CORE_03303] [SWS_CORE_03304] [SWS_CORE_03305] [SWS_CORE_03306] [SWS_CORE_03307] [SWS_CORE_03308] [SWS_CORE_03309] [SWS_CORE_03310] [SWS_CORE_03311] [SWS_CORE_03312] [SWS_CORE_03313] [SWS_CORE_03314] [SWS_CORE_03315] [SWS_CORE_03316] [SWS_CORE_03317] [SWS_CORE_03318] [SWS_CORE_03319] [SWS_CORE_03320] [SWS_CORE_03321] [SWS_CORE_03322] [SWS_CORE_03323] [SWS_CORE_04011] [SWS_CORE_04012] [SWS_CORE_04013] [SWS_CORE_04021] [SWS_CORE_04022] [SWS_CORE_04023] [SWS_CORE_04031] [SWS_CORE_04032] [SWS_CORE_04033] [SWS_CORE_04110] [SWS_CORE_04111] [SWS_CORE_04112] [SWS_CORE_04113] [SWS_CORE_04120] [SWS_CORE_04121] [SWS_CORE_04130] [SWS_CORE_04131] [SWS_CORE_04132] [SWS_CORE_04200] [SWS_CORE_05200] [SWS_CORE_05211] [SWS_CORE_05212] [SWS_CORE_05221] [SWS_CORE_05231] [SWS_CORE_05232] [SWS_CORE_05241] [SWS_CORE_05242] [SWS_CORE_05243] [SWS_CORE_05244] [SWS_CORE_05280] [SWS_CORE_05290] [SWS_CORE_06221] [SWS_CORE_06222] [SWS_CORE_06223] [SWS_CORE_06225] [SWS_CORE_06226] [SWS_CORE_06227] [SWS_CORE_06228] [SWS_CORE_06229] [SWS_CORE_06230] [SWS_CORE_06231] [SWS_CORE_06232] [SWS_CORE_06233] [SWS_CORE_06234] [SWS_CORE_06235] [SWS_CORE_06236] [SWS_CORE_06237] [SWS_CORE_06340] [SWS_CORE_06341] [SWS_CORE_06342] [SWS_CORE_06343] [SWS_CORE_06344] [SWS_CORE_06345] [SWS_CORE_06349] [SWS_CORE_06350] [SWS_CORE_06351] [SWS_CORE_06352] [SWS_CORE_06353] [SWS_CORE_06354] [SWS_CORE_06355] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SWS_CORE_06356] [SWS_CORE_06401] [SWS_CORE_06411] [SWS_CORE_06412] [SWS_CORE_06413] [SWS_CORE_06414] [SWS_CORE_06431] [SWS_CORE_06432] [SWS_CORE_06500] [SWS_CORE_06501] [SWS_CORE_06502] [SWS_CORE_06503] [SWS_CORE_06504] [SWS_CORE_06505] [SWS_CORE_06506] [SWS_CORE_06507] [SWS_CORE_06520] [SWS_CORE_06521] [SWS_CORE_06522] [SWS_CORE_06523] [SWS_CORE_06524] [SWS_CORE_06525] [SWS_CORE_06526] [SWS_CORE_06527] [SWS_CORE_06528] [SWS_CORE_06529] [SWS_CORE_06530] [SWS_CORE_06531] [SWS_CORE_06540] [SWS_CORE_06541] [SWS_CORE_06542] [SWS_CORE_06543] [SWS_CORE_06544] [SWS_CORE_06545] [SWS_CORE_06546] [SWS_CORE_06547] [SWS_CORE_06548] [SWS_CORE_06549] [SWS_CORE_06550] [SWS_CORE_06551] [SWS_CORE_06552] [SWS_CORE_06553] [SWS_CORE_06554] [SWS_CORE_06555] [SWS_CORE_06556] [SWS_CORE_06557] [SWS_CORE_06560] [SWS_CORE_06561] [SWS_CORE_06562] [SWS_CORE_06563] [SWS_CORE_06564] [SWS_CORE_06565] [SWS_CORE_06566] [SWS_CORE_06567] [SWS_CORE_06568] [SWS_CORE_06569] [SWS_CORE_06570] [SWS_CORE_06571] [SWS_CORE_06572] [SWS_CORE_06573] [SWS_CORE_06574] [SWS_CORE_06575] [SWS_CORE_06576] [SWS_CORE_06577] [SWS_CORE_10100] [SWS_CORE_10101] [SWS_CORE_10102] [SWS_CORE_10103] [SWS_CORE_10104] [SWS_CORE_10105] [SWS_CORE_10106] [SWS_CORE_10107] [SWS_CORE_10108] [SWS_CORE_10109] [SWS_CORE_10110] [SWS_CORE_10200] [SWS_CORE_10201] [SWS_CORE_10202] [SWS_CORE_10203] [SWS_CORE_10300] [SWS_CORE_10400] [SWS_CORE_10900] [SWS_CORE_10901] [SWS_CORE_10902] [SWS_CORE_10903] [SWS_CORE_10910] [SWS_CORE_10911] [SWS_CORE_10912] [SWS_CORE_10930] [SWS_CORE_10931] [SWS_CORE_10932] [SWS_CORE_10933] [SWS_CORE_10934] [SWS_CORE_10950] [SWS_CORE_10951] [SWS_CORE_10952] [SWS_CORE_10953] [SWS_CORE_10980] [SWS_CORE_10981] [SWS_CORE_10982] [SWS_CORE_10990] [SWS_CORE_10991] [SWS_CORE_10999] [SWS_CORE_11000] [SWS_CORE_11200] [SWS_CORE_11300] [SWS_CORE_11400] [SWS_CORE_11600] [SWS_CORE_11800] [SWS_CORE_11801] [SWS_CORE_11900] [SWS_CORE_11952] [SWS_CORE_12000] [SWS_CORE_12200] [SWS_CORE_12403] [SWS_CORE_12404] [SWS_CORE_12405] [SWS_CORE_12406] [SWS_CORE_12407] [SWS_CORE_12408] [SWS_CORE_12409] [SWS_CORE_13206] [SWS_CORE_90023] [SWS_CORE_90024]

▽

△

Requirement	Description	Satisfied by
[RS_AP_00134]	noexcept behavior of class destructors	[SWS_CORE_08029]
[RS_AP_00136]	Usage of string types	[SWS_CORE_00052] [SWS_CORE_08032]
[RS_AP_00137]	Connecting run-time interface with model	[SWS_CORE_08032]
[RS_AP_00138]	Return type of asynchronous function calls	[SWS_CORE_10800]
[RS_AP_00139]	Return type of synchronous function calls	[SWS_CORE_00002]
[RS_AP_00140]	Usage of "final specifier"	[SWS_CORE_00501] [SWS_CORE_08001] [SWS_CORE_10932]
[RS_AP_00142]	Handling of unsuccessful operations	[SWS_CORE_00002] [SWS_CORE_00003] [SWS_CORE_00004] [SWS_CORE_00005] [SWS_CORE_00006] [SWS_CORE_00007] [SWS_CORE_00020] [SWS_CORE_00021] [SWS_CORE_00022] [SWS_CORE_00023] [SWS_CORE_00090] [SWS_CORE_00091] [SWS_CORE_00092] [SWS_CORE_10600] [SWS_CORE_13018] [SWS_CORE_13019] [SWS_CORE_15002] [SWS_CORE_90021] [SWS_CORE_90026] [SWS_CORE_90027] [SWS_CORE_90028] [SWS_CORE_90029] [SWS_CORE_90030]
[RS_AP_00145]	Availability of special member functions	[SWS_CORE_00617]
[RS_AP_00149]	Error handling for non-initialized Functional Cluster	[SWS_CORE_90021]
[RS_AP_00159]	usage of "noexcept" specifier	[SWS_CORE_00050] [SWS_CORE_00051] [SWS_CORE_00052] [SWS_CORE_00053] [SWS_CORE_00054]
[RS_AP_00163]	Reentrancy of Functions	[SWS_CORE_13210]
[RS_AP_00164]	Thread-safety of Functions	[SWS_CORE_13200] [SWS_CORE_13201] [SWS_CORE_13202] [SWS_CORE_13203] [SWS_CORE_13204] [SWS_CORE_13207] [SWS_CORE_13208] [SWS_CORE_13209]
[RS_AP_00165]	Concurrent Use of Different Objects	[SWS_CORE_13205]
[RS_Main_00011]	Mechanisms for Reliable Systems	[SWS_CORE_10001] [SWS_CORE_10002] [SWS_CORE_10003] [SWS_CORE_15003] [SWS_CORE_15004] [SWS_CORE_15006]
[RS_Main_00150]	AUTOSAR shall support the deployment and reallocation of AUTOSAR Application Software	[SWS_CORE_08032]
[RS_Main_00320]	AUTOSAR shall provide formats to specify system development	[SWS_CORE_08001] [SWS_CORE_08021] [SWS_CORE_08022] [SWS_CORE_08023] [SWS_CORE_08024] [SWS_CORE_08025] [SWS_CORE_08029] [SWS_CORE_08041] [SWS_CORE_08042] [SWS_CORE_08043] [SWS_CORE_08044] [SWS_CORE_08045] [SWS_CORE_08046] [SWS_CORE_08081] [SWS_CORE_08082]

Table 6.1: Requirements Tracing

7 Functional specification

Section 7.1 (“General requirements for all Functional Clusters”) defines a common set of basic requirements that apply to all Functional Clusters of the Adaptive Platform. It adds a common part to the specifications and it needs to be respected by platform vendors.

The remaining sections in this chapter describe the concepts that are introduced with this Functional Cluster. Particular emphasis is put on error handling.

7.1 General requirements for all Functional Clusters

7.1.1 Functional Cluster Names

AUTOSAR Functional Clusters are assigned a standardized "name" e.g. com together with a reserved Functional Cluster namespace ([RS_AP_00115]) below the top-level namespace. Table [SWS_CORE_90025] shows the reserved Functional Cluster settings for each.

[SWS_CORE_90025] AUTOSAR-standardized Functional Cluster names

Status: DRAFT
Upstream requirements: RS_AP_00115

⌈

Name	Description	C++ namespace
aag	Automotive API Gateway	n/a
com	Communication Management	ara::com
core	Core	ara::core
crypto	Cryptography	ara::crypto
diag	Diagnostics	ara::diag
exec	Execution Management	ara::exec
fw	Firewall	ara::fw
idsm	Intrusion Detection System Manager	ara::idsm
log	Log and Trace	ara::log
nm	Network Management	ara::nm
osi	Operating System Interface	n/a
per	Persistency	ara::per
phm	Platform Health Management	ara::phm
rds	Raw Data Stream	ara::rds
sm	State Management	ara::sm
tsync	Time Synchronization	ara::tsync
ucm	Update and Configuration Management	ara::ucm
vucm	Vehicle Update and Configuration Management	ara::vucm

⌋

7.1.2 Coding Guidelines

[SWS_CORE_90001] Include folder structure

Upstream requirements: RS_AP_00116, RS_AP_00111

⌈All #include directives in header files that refer to ARA libraries shall be written in the form: #include "ara/fc/header.h" where:

- ara: reserved; as per [RS_AP_00115]
- fc: reserved; is being the remaining directory path of the implementation's *installed* header file, starting with the Functional Cluster name as per [SWS_CORE_90025] e.g. com
- header.h: the filename of the header file as per [RS_AP_00116]

⌋

Example: for class ara::exec::ExecutionClient present in #include "ara/exec/execution_client.h"

The "..." form of #include statements shall be used, due to the recommendation given in [5, C++14] [cpp.include] Par. 7.

[SWS_CORE_90002] Prevent multiple inclusion of header file

Upstream requirements: RS_AP_00111

⌈All public header files shall prevent multiple inclusion by using #include guards that are likely to be system-wide unique.⌋

While uniqueness can generally not be guaranteed, the likelihood of collisions can be decreased with a naming scheme that is regular and results in long symbol names. The following #include guard naming scheme should be used by implementations for all header files that cover symbols within the ara:: namespace or a sub-namespace therein: ARA_<PATH>_H_

Where <PATH> is the relative path name of the header file within the location of the implementation's *installed* header files, starting with the Functional Cluster name (and omitting the file extension), and with all components of <PATH> separated by underscore ("_") characters and containing only upper-case characters of the ASCII character set.

Example: The header file included with #include "ara/log/logger.h" should use the #include guard symbol ARA_LOG_LOGGER_H_.

[SWS_CORE_90003] Reservation of ARA preprocessor macros

Upstream requirements: RS_AP_00111

⌈C/C++ preprocessor symbols that start with `ARA` are reserved for use by AUTOSAR.⌋

The Adaptive Platform tries to avoid the use of C/C++ preprocessor macros. Such macros start with the prefix `ARA`. Platform vendors should thus not define any symbols (both macros and C/C++ ones) with this prefix, lest they conflict with such future additions to the standard.

[SWS_CORE_90004] Implementation-defined declaration classifiers

Upstream requirements: RS_AP_00111

⌈All APIs shall be implemented with the exact same declaration classifiers that are specified, except for `inline` and `friend`, which may be added as necessary.⌋

Note: The order of declarations may be freely chosen.

[5, C++14] `[dcl.spec]` defines the specifiers that can be used in a declaration; these include, for instance, `static`, `virtual`, `constexpr`, `inline` and `friend`. An implementation that uses a different set of specifiers in its declaration of a specified API may be incompatible to the standard, or may allow non-standardized usage of that API, leading to portability concerns.

[SWS_CORE_90005] Custom declarations and definitions

Upstream requirements: RS_AP_00111

⌈Implementation shall not add public declarations or definitions that are not specified in an SWS to the namespace `ARA` or any of its direct sub-namespaces, except in the sub-namespace `::internal` which is reserved for vendor-specific use.⌋

The Adaptive Platform is designed for source code portability. Wherefore any conformant implementation of the Adaptive Platform allows a 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 will 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 the AP standard.

[SWS_CORE_90007] Potentially throwing constructors

Upstream requirements: RS_AP_00111

⌈Constructors that may throw exceptions shall not participate in overload resolution when C++ exceptions are disabled in the compiler toolchain.⌋

See ArcDecision in FO_EXP_SWArchitecturalDecisions [6] *Potentially throwing constructors*.

7.1.3 Initialization/De-initialization

7.1.3.1 Common aspects

This section describes the global initialization and shutdown of the ARA framework. Before the framework is initialized, and after it is de-initialized, not all ARA functionality may be available.

While it is usually possible for a framework implementation to initialize all parts of the framework in an “initialize on first use” fashion, this might not always be desirable, as it introduces potentially noticeable delays during runtime.

For this reason, initialization and shutdown functions exist that may be used by the framework vendor to initialize/shutdown the framework to an extent that no lazy initialization during runtime is necessary.

On the other hand, another framework implementation might well have empty implementations of these functions, e.g. if this framework chooses to fully adopt the “initialize on first use” idiom.

[SWS_CORE_90021] Pre-conditions for `ara::core::InstanceSpecifier`

Upstream requirements: RS_AP_00111, RS_AP_00142, RS_AP_00149

⌈If a constructor or function takes an `ara::core::InstanceSpecifier` as an argument it shall check for an initialized platform. That is: `ara::core::Initialize` has been called successfully and `ara::core::Deinitialize` has not (yet) been executed. If such a constructor or function is called while the platform is not initialized it shall be treated as a `PlatformNotInitializedViolation` ([SWS_CORE_13007]).⌋

Note: Member functions of the constructed objects do not need to check for an initialized platform afterwards.

Rationale for [SWS_CORE_90021]: These constructors or functions are usually costly operations (connection to daemon established, etc.) and are called infrequently. Therefore, the performance impact of this check is considered insignificant. The rationale to treat this as a `Violation` is that such occurrences cannot be handled by the caller of the API at the point in time the error is detected. Aborting execution is the only way to signal this kind of systematic error and prevent later failures.

[SWS_CORE_15002] Special ara::core types to be used independently of initialization

Upstream requirements: RS_AP_00142

⌈A small subset of ara::core types and functions shall be usable independently of initialization with ara::core::Initialize and de-initialization with ara::core::Deinitialize. These are:

- ara::core::ErrorCode and all its member functions and supporting constructs (including non-member operators)
- ara::core::StringView and all its member functions and supporting constructs (including non-member operators)
- ara::core::Result and all its member functions and supporting constructs, except for ara::core::Result::ValueOrThrow
- ara::core::ErrorDomain and all its member functions and its sub-classes, as long as they adhere to [SWS_CORE_10400], but excluding <Prefix>ErrorDomain::ThrowAsException
- ara::core::Initialize
- ara::core::Abort
- ara::core::SetAbortHandler
- ara::core::AddAbortHandler

⌋

The rationale for [SWS_CORE_15002] is the intended use of these types and functions before initialization and after de-initialization. As well as that these types and functions are used as part of the initialization/de-initialization (ara::core::Result, ara::core::ErrorCode, ara::core::ErrorDomain). ara::core::Abort is intended to be used if ara::core::Initialize or ara::core::Deinitialize fails.

7.1.3.2 Initialization

ara::core::Initialize allows a central initialization of all included shared libraries of the ARA framework. This could include initialization of internal data or the setup of daemon links (details are up to the platform vendor).

The general advice for application developers is to call ara::core::Initialize right at the entry point of the application.

[SWS_CORE_15003] Startup and initialization of ARA

Upstream requirements: RS_Main_00011

⌈The ara::core::Initialize function shall initiate the start-up of the ARA framework, which might include (but is not limited to):

- initialization of ARA framework specific data structures
- initialization of system resources
- spawning of background threads

⌋

[SWS_CORE_15006] Command line argument injection in ara::core::Initialize

Upstream requirements: RS_Main_00011

⌈If command line argument injection is used in ara::core::Initialize, the relevant command line arguments shall be removed transparently by this function to not influence Application's command line parsing.⌋

[SWS_CORE_15005] ARA behavior before ara::core::Initialize

Upstream requirements: RS_AP_00111

⌈The behavior before initialization of the Adaptive Platform with ara::core::Initialize of functions that are not explicitly supported according to [SWS_CORE_15002] or explicitly not supported according to [SWS_CORE_90022] is implementation-defined.⌋

7.1.3.3 De-initialization

[SWS_CORE_15004] Shutdown and de-initialization of ARA

Upstream requirements: RS_Main_00011

⌈The ara::core::Deinitialize function shall initiate the shutdown of the ARA framework, which might include (but is not limited to):

- orderly shutdown of spawned background threads
- de-allocation of dynamically allocated memory
- de-allocation of other system resources

⌋

An error returned by ara::core::Deinitialize is the only way for the ARA to report an error that is guaranteed to be available, e.g. in case ara::log has already been de-initialized. The user is not expected to be able to recover from such an error. However, the user may have a project-specific way of recording errors during de-initialization without ara::log. A typical error case to be reported here is that the user is still holding some resource from the ARA.

Calling ara::core::Deinitialize while ARA APIs are still being called concurrently results in undefined behavior of the application and the framework.

For a proper shutdown, it is also expected that ara::core::Deinitialize is called before the statically initialized data is destructed.

[SWS_CORE_90022] ARA behavior after ara::core::Deinitialize

Upstream requirements: RS_AP_00111

⌈If a functionality (other than the ones mentioned in [SWS_CORE_15002]) is called after ara::core::Deinitialize has been called, the behavior is implementation-defined.⌋

Rationale: A check for de-initialization would require runtime checks and semaphores to verify the platform state in each API call. Making this check mandatory would have a significant negative performance impact.

7.1.3.4 Sequence of Initialization and De-initialization

When ara::core::Initialize has returned successfully, calling it again without a prior call to ara::core::Deinitialize will return an error.

When ara::core::Initialize returns with an error, the application may identify the error cause and then try the call again. Calling ara::core::Deinitialize in this case is not necessary, as any necessary cleanup work has already been performed by ara::core::Initialize before returning to the caller.

When ara::core::Deinitialize returns with an error other than kNotInitialized, it is generally unsafe to call either ara::core::Deinitialize or ara::core::Initialize again.

[SWS_CORE_90026] Framework initialization

Upstream requirements: RS_AP_00142

⌈A successful call of ara::core::Initialize shall bring the framework into a state that makes all its configured components ready for immediate servicing of further API calls.⌋

[SWS_CORE_90027] Clean deinitialization

Upstream requirements: RS_AP_00142

⌈A successful call of `ara::core::Deinitialize` shall bring the framework into a state that is equivalent to the state before the first call to `ara::core::Initialize`.⌋

[SWS_CORE_90028] Re-try of initialization

Upstream requirements: RS_AP_00142

⌈When `ara::core::Initialize` returns with an error, the implementation shall perform all necessary cleanup work so that `ara::core::Initialize` can be called again by the application.⌋

[SWS_CORE_90029] Double initialization

Upstream requirements: RS_AP_00142

⌈When `ara::core::Initialize` has returned successfully, a subsequent call to `ara::core::Initialize` (without a prior call to `ara::core::Deinitialize`) shall return `kAlreadyInitialized` without impeding the currently running framework instance.⌋

[SWS_CORE_90030] Double de-initialization

Upstream requirements: RS_AP_00142

⌈When the framework instance has not been successfully initialized, or has already been de-initialized (successful or not), any call to `ara::core::Deinitialize` shall return `kNotInitialized`.⌋

7.1.4 Logging and Tracing

The Adaptive Platform Foundation provides libraries that are integrated into adaptive application processes (see [7] "Figure: ’Overview Log and Trace’" and [8] "Figure: ’Architecture overview’"). Each function cluster has its own "Context ID" to distinguish these ARA libraries from the application logging and from each other. Further information on Context ID can be found in [8] in chapter: “Context ID”.

[SWS_CORE_00055] Functional Cluster Log and Trace messages

Upstream requirements: RS_AP_00111

⌈Each ARA Functional Cluster shall use standardized Logging and Tracing settings from [SWS_CORE_90024] when logging their messages via `ara::log`:

- "Log and Trace Context Identifier": for use with DLT (see [PRS_Dlt_01054])
- "Log and Trace Message Identifier range": (see [PRS_Dlt_01062])

⌋

[SWS_CORE_90024] AUTOSAR-standardized Functional Cluster Log and Trace settings

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Name	***Log & Trace Context Identifier***	***Log & Trace Message Identifier range (lower)***	***Log & Trace Message Identifier range (upper)***
aag	#AAG	0x8001’1000	0x8001’1fff
com	#COM	0x8000’0000	0x8000’0fff
core	#COR	0x8000’1000	0x8000’1fff
crypto	#CRY	0x8000’2000	0x8000’2fff
diag	#DIA	0x8000’3000	0x8000’3fff
exec	#EXE	0x8000’4000	0x8000’4fff
fw	#FWX	0x8001’0000	0x8001’0fff
idsm	#IDS	0x8000’5000	0x8000’5fff
log	#LOG	0x8000’6000	0x8000’6fff
nm	#NMX	0x8000’7000	0x8000’7fff
osi	#OSI	0x8000’e000	0x8000’efff
per	#PER	0x8000’8000	0x8000’8fff
phm	#PHM	0x8000’9000	0x8000’9fff
rds	#RDS	0x8000’f000	0x8000’ffff
sm	#SMX	0x8000’a000	0x8000’afff
tsync	#TSY	0x8000’b000	0x8000’bfff
ucm	#UCM	0x8000’c000	0x8000’cfff
vucm	#VUM	0x8000’d000	0x8000’dfff

⌋

7.2 Error handling

During execution of an implementation of Adaptive Platform APIs, different abnormal conditions might be detected and need to be handled and/or reported. Based on their nature, in the following sub-chapters, the types of unsuccessful operations are distinguished within the Adaptive Platform and shall be treated in different ways.

7.2.1 Traditional error handling in C and C++

The C language largely relies on error codes for any kind of error handling. While it also has the setjmp/longjmp facility for performing “non-local gotos”, its use for error handling is not widespread, mostly due to the difficulty of reliably avoiding resource leaks.

Error codes in C come in several flavors:

- return values
- out parameters
- error singletons (e.g. errno)

Typically, these error codes in C are plain int variables, making them a very low-level facility without any type safety.

C++ inherited these approaches to error handling from C (not least due to the inheritance of the C standard library as part of the C++ standard), but it also introduced exceptions as an alternative means of error propagation. There are many advantages of using exceptions for error propagation, which is why the C++ standard library generally relies on them for error propagation.

Notwithstanding the advantages of exceptions, error codes are still in widespread use in C++, even within the standard library. Some of that can be explained with concerns about binary compatibility with C, but many new libraries still prefer error codes to exceptions. Reasons for that include:

- with exceptions, it can be difficult to reason about a program’s control flow
- exceptions have much higher runtime cost than error codes (either in general, or only in the exception-thrown case)

The first of these reasons concerns both humans and code analysis tools. Because exceptions are, in effect, a kind of hidden control flow, a C++ function that seems to contain only a single return statement might in fact have many additional function returns due to exceptions. That can make such a function hard to review for humans, but also hard to analyze for static code analysis tools.

The second one is even more critical in the context of developing safety-critical software. The specification of C++ exceptions pose significant problems for C++ compiler vendors that want their products be certified for development of safety-critical software.

In fact, ASIL-certified C++ compilers generally do not support exceptions at all. One particular problem with exceptions is that exception handling, as specified for C++, implies the use of dynamic memory allocation, which generally has non-predictable or even unbounded execution time. This makes exceptions currently unsuitable for development of certain safety-critical software in the automotive industry.

7.2.2 Violation

[SWS_CORE_00021] Semantics of a Violation

Upstream requirements: RS_AP_00142

⌈A `Violation` is the consequence of failed pre- or post-conditions of an internal state of the application framework. They are the Adaptive Platform's analog to a failed assertion. A `Violation` is non-recoverable.⌋

Note: systematic errors can be considered as a failed pre-condition.

[SWS_CORE_00091] Messages for `Violations`

Upstream requirements: RS_AP_00142

⌈For every `Violation` a DLT Message shall be defined. It shall have the mandatory string parameters *processIdentifier* and *location* as well as the `messageTypeInfo` DLT_LOG_FATAL. Its description shall explain how a message string shall be created from the DLT message.

The stringified DLT message shall start with: "Violation detected in {processIdentifier} at {location}: "⌋

[SWS_CORE_00090] Handling of `Standardized Violations`

Upstream requirements: RS_AP_00142

⌈If a `Violation` is detected that is standardized in the AUTOSAR AP, then the operation shall be terminated by:

- explicitly terminating the process abnormally via a call to ara::core::Abort

In this case a message to support debugging (see [SWS_CORE_00091]) shall be created according to the DLT message defined for the concrete `Violation`. It shall be output in its string format according to its description to the process' standard error stream. Additionally, it should be logged into the log sinks with the context id of the functional cluster that detected the `Violation` as defined by ara::log for the affected process or the `Execution Management`. Which one is chosen is implementation defined.⌋

Note: The standard does not mandate the direct use of ara::log to allow for optimized solutions, in particular for a fast shutdown of the process without losing the message.

[SWS_CORE_00003] Handling of Non-Standardized Violations

Upstream requirements: RS_AP_00142

⌈If a Violation is detected that is not standardized in the AUTOSAR AP, then the operation shall be terminated by either:

- throwing an exception that is not a sub-class of ara::core::Exception
- explicitly terminating the process abnormally via a call to ara::core::Abort

In both cases a message to support debugging (see [SWS_CORE_00091]) should be created according to the DLT message defined for the concrete Violation. It should be output in its string format according to its description to the process' standard error stream. Additionally, it should be be logged into the log sinks with the context id of the functional cluster that detected the Violation as defined by ara::log for the affected process or the Execution Management. Which one is chosen is implementation defined.⌋

See also: [6] Messages for unrecoverable errors.

Note: There can be situations in which it might not be possible to generate the message. E.g., when the process is terminated through std::terminate in code that is not changeable by the implementer.

Note: ara::core::Abort is ideal for implementing the two previous requirements.

[SWS_CORE_00006] Handling of exception-based Violations

Upstream requirements: RS_AP_00142

⌈If a Violation is realized using an Exception (that is not a sub-class of ara::core::Exception), and the noexcept specifier condition ([5] [except.spec]) of the function in which it is raised, evaluates to true: the C++ runtime will invoke std::terminate() automatically.⌋

Note: This will be the case more often than not as AUTOSAR API functions typically evaluate to noexcept(true).

7.2.3 Corruption

[SWS_CORE_00022] Semantics of a Corruption

Upstream requirements: RS_AP_00142

⌈A Corruption is the consequence of the corruption of a system resource, e.g. stack or heap overflow, or a hardware memory flaw (including even, for instance, a detected bit flip). A Corruption is non-recoverable.⌋

[SWS_CORE_00004] Handling of Corruptions

Upstream requirements: RS_AP_00142

⌈If a Corruption is detected, it shall result in unsuccessful process termination, in an implementation-defined way.⌋

Note: It can either be abnormal or normal unsuccessful termination, depending on the implementation's ability to detect the Corruption and to react to it by cleaning up resources.

It is expected that a Violation or Corruption might occur during development of the framework, when new features are just coming together, but will not be experienced by a user (i.e. an application developer), unless there is something seriously wrong in the system's environment (e.g. faulty hardware: Corruption), or basic assumptions about resource requirements are violated (Violation), or possibly the user runs the framework in a configuration that is not supported by its vendor (Violation).

7.2.4 Failed default allocation

[SWS_CORE_00023] Semantics of a Failed Default Allocation

Upstream requirements: RS_AP_00142

⌈A Failed Default Allocation is the inability of the framework's default memory allocation mechanism to satisfy an allocation request. A Failed Default Allocation is non-recoverable.⌋

[SWS_CORE_00005] Handling of failed default allocations

Upstream requirements: RS_AP_00142

⌈A Failed Default Allocation shall be treated the same as a Non-Standardized Violation (see [SWS_CORE_00003]), except that the DLT message is FailedDefaultAllocation ([SWS_CORE_13018]).⌋

Note: An error of a custom allocator is not subject to this definition.

[SWS_CORE_00007] Handling of exception-based Failed Default Allocations

Upstream requirements: RS_AP_00142

⌈If a Failed Default Allocation is realized using an Exception (that is not a sub-class of ara::core::Exception), and the noexcept specifier condition ([5] [except.spec]) of the function in which it is raised, evaluates to true: the C++ runtime will invoke std::terminate() automatically.⌋

Note: This will be the case more often than not as AUTOSAR API functions typically evaluate to noexcept(true).

7.2.5 Error

[SWS_CORE_00020] Semantics of an Error

Upstream requirements: RS_AP_00142

⌈An Error is the inability of an assumed-bug-free API function to fulfill its specified purpose; it is often a consequence of invalid and/or unexpected (i.e. possibly valid, but received in unexpected circumstances) input data. An Error is recoverable.⌋

[SWS_CORE_00002] Handling of Errors

Upstream requirements: RS_AP_00142, RS_AP_00139, RS_AP_00128

⌈An Error shall be returned from the function as an instance of ara::core::Result or ara::core::Future.⌋

For handling Errors, there are a number of data types defined that help in dealing with them. These are described in the following sub-chapters.

7.2.5.1 ErrorCode

As its name implies, ara::core::ErrorCode is a form of error code; however, it is a class type, loosely modeled on std::error_code, and thus allows much more sophisticated handling of errors than the simple error codes as used in typical C APIs. It always contains a low-level error code value and a reference to an error domain.

The error code value is an enumeration, typically a scoped one. When stored into a ara::core::ErrorCode, it is type-erased into an integral type and thus handled similarly to a C-style error code. The error domain reference defines the context for which the error code value is applicable and thus provides some measure of type safety.

An ara::core::ErrorCode also contains a support data value, which *can* be defined by an implementation of the Adaptive Platform to give a vendor-specific additional piece of data about the error.

[SWS_CORE_10302] Semantics of ErrorCode

Upstream requirements: RS_AP_00119

⌈The type ara::core::ErrorCode provides a class interface for storing an error condition. It shall contain these properties:

- error code value: an integral representation of a low-level error code
- error domain: reference to the context for which the *error code value* is applicable
- support data value: an optional vendor-specific additional piece of data about the error

⌋

ara::core::ErrorCode instances are usually not created directly, but only via the forwarding form of the function ara::core::Result::FromError.

An ara::core::ErrorCode is not restricted to any known set of error domains. Its internal type erasure of the enumeration makes sure that it is a simple (i.e., non-templated) type which can contain arbitrary errors from arbitrary domains.

However, comparison of two ara::core::ErrorCode instances only considers the error code value and the error domain reference; the support data value member is not considered for checking equality. This is due to the way ara::core::ErrorCode instances are usually compared against a known set of errors for which to check:

```
ErrorCode ec = ...
if (ec == MyEnum::some_error)
  // ...
else if (ec == AnotherEnum::another_error)
  // ...
```

Each of these comparisons will create a temporary ara::core::ErrorCode object for the right-hand side of the comparison, and then compare ec against that. Such automatically created instances naturally do not contain any meaningful support data value.

[SWS_CORE_10301] Comparison of ara::core::ErrorCode instances

Upstream requirements: RS_AP_00119

⌈Any comparison of two ara::core::ErrorCode instances shall consider only the following members:

- error code value

- error domain

⌋

This frequent creation of temporary ara::core::ErrorCode instances is expected to be so fast as to induce no noticeable runtime cost. This is usually ensured by ara::core::ErrorCode being a *literal type*.

[SWS_CORE_10300] ErrorCode type properties

Upstream requirements: RS_AP_00130

⌈Class ara::core::ErrorCode shall be a *literal type*, as defined in [5, C++14] [basic.types] Par. 10.⌋

7.2.5.2 ErrorDomain

ara::core::ErrorDomain is the abstract base class for concrete error domains that are defined within Functional Clusters or even Adaptive Applications. This class is loosely based on std::error_category, but differs significantly from it.

An error domain has an associated error code enumeration and an associated base exception type. Both these are usually defined in the same namespace as the ara::core::ErrorDomain sub-class. For normalized access to these associated types, type aliases with standardized names are defined within the ara::core::ErrorDomain sub-class. This makes the ErrorDomain sub-class the root of all data about errors.

[SWS_CORE_10303] Semantics of ErrorDomain

Upstream requirements: RS_AP_00119

⌈The type ara::core::ErrorDomain defines a context for a set of error conditions.⌋

Identity of error domains is defined in terms of unique identifiers. AUTOSAR-defined error domains are given standardized identifiers; user-defined error domains are also required to define unique identifiers.

The ara::core::ErrorDomain class definition requires this unique identifier to be of unsigned 64 bit integer type (std::uint64_t). The range of possible values is large enough to apply UUID-like generation patterns (for UID-64) even if typical UUIDs have 128 bits and are thus larger than that. When a new error domain is created (either an AUTOSAR defined or an user defined one) an according Id shall be randomly generated, which represents this error domain. The uniqueness and standardization of such an Id per error domain is mandatory, since the exchange of information on occurred errors between callee and caller (potentially located at different ECUs) is based on this Id.

[SWS_CORE_10401] Identity of ErrorDomains

Upstream requirements: RS_AP_00119

⌈Two instances of ara::core::ErrorDomain shall compare equal if and only if their unique identifiers are the same.⌋

Given this definition of identity of error domains, it usually makes sense to have only one single instance of each ara::core::ErrorDomain sub-class. While new instances of these sub-classes can be created by calling their constructors, the recommended way to gain access to these sub-classes is to call their non-member accessor functions. For instance, the error domain class ara::core::FutureErrorDomain is referenced by calling ara::core::GetFutureErrorDomain; within any process space, this will always return a reference to the same global instance of this class.

For error domains that are modeled in ARXML (as ApApplicationErrorDomain), the C++ language binding will create a C++ class for each such ApApplicationErrorDomain. This C++ class will be a sub-class of ara::core::ErrorDomain, and its name will follow a standard scheme.

ara::core has two pre-defined error domains, called ara::core::CoreErrorDomain (containing the set of errors returned by non-Future/Promise facilities from the ara::core Functional Cluster) and ara::core::FutureErrorDomain (containing errors equivalent to those defined by std::future_errc).

Application programmers usually do not interact with class ara::core::ErrorDomain or its sub-classes directly; most access is done via ara::core::ErrorCode.

As ara::core::ErrorDomain sub-classes are expected to be implicitly referred to from within constant (i.e. compile-time) expressions (typically involving ara::core::ErrorCode), they are expected to be *literal types*.

[SWS_CORE_10304] Availability of ara::core::ErrorDomain::ThrowAsException and overriding functions

Upstream requirements: RS_AP_00119

⌈ara::core::ErrorDomain::ThrowAsException and overriding functions shall not participate in overload resolution when C++ exceptions are disabled in the compiler toolchain.⌋

[SWS_CORE_10400] ErrorDomain type properties

Upstream requirements: RS_AP_00130

⌈Class ara::core::ErrorDomain and all its sub-classes shall be *literal types*, as defined in [5, C++14] [basic.types] Par. 10.⌋

7.2.5.2.1 Error domain Identifiers

The full range of unique error domain identifiers is partitioned into a range of:

- AUTOSAR-standardized Ids: MSB=1; reserved by AUTOSAR; shown in [SWS_CORE_90023]
- Vendor-specific Ids: MSB=1; reserved by an AUTOSAR stack vendor
- User-defined Ids: MSB=0; defined via ApApplicationErrorDomain.value

Regardless of the origin of an error domain identifier, it shall be unique in the platform.

[SWS_CORE_90023] AUTOSAR-standardized Functional Cluster Error Domain Identifiers

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Name	C++ namespace	Error Domain Identifier
core	ara::core::FutureErrorDomain	0x8000’0000’0000’0013
core	ara::core::CoreErrorDomain	0x8000’0000’0000’0014
per	ara::per::PerErrorDomain	0x8000’0000’0000’0101
exec	ara::exec::ExecErrorDomain	0x8000’0000’0000’0202
sm	ara::sm::SmErrorDomain	0x8000’0000’0000’0301
diag	ara::diag::DiagErrorDomain	0x8000’0000’0000’0401
diag	ara::diag::DiagReportingErrorDomain	0x8000’0000’0000’0402
diag	ara::diag::DiagOfferErrorDomain	0x8000’0000’0000’0403
diag	ara::diag::DiagUdsNrcErrorDomain	0x8000’0000’0000’0411
diag	ara::diag::DiagSovdErrorDomain	0x8000’0000’0000’0412
fw	ara::fw::FwErrorDomain	0x8000’0000’0000’0501
phm	ara::phm::PhmErrorDomain	0x8000’0000’0000’0602
ucm	ara::ucm::UcmErrorDomain	0x8000’0000’0000’0701
crypto	ara::crypto::CryptoErrorDomain	0x8000’0000’0000’0801
tsync	ara::tsync::TsyncErrorDomain	0x8000’0000’0000’0901
nm	ara::nm::NmErrorDomain	0x8000’0000’0000’1001
idsm	ara::idsm::IdsmErrorDomain	0x8000’0000’0000’1101
com	ara::com::ComErrorDomain	0x8000’0000’0000’1267
com	ara::com::e2e::ComE2eErrorDomain	0x8000’0000’0000’1268
com	apext::com::secoc::ComSecOcFvmErrorDomain	0x8000’0000’0000’1271
rds	ara::rds::RawErrorDomain	0x8000’0000’0000’1280
vucm	ara::vucm::VucmErrorDomain	0x8000’0000’0000’1301

⌋

[SWS_CORE_00010] Error domain identifier

Upstream requirements: RS_AP_00130

⌈All error domains shall have a system-wide unique identifier that is represented as a 64-bit unsigned integer value.⌋

[SWS_CORE_00011] AUTOSAR error domain range

Upstream requirements: RS_AP_00130

⌈Error domain identifiers where bit #63 is set to 1 and bit #62 is set to 0 are reserved for AUTOSAR-defined error domains.⌋

[SWS_CORE_00016] Vendor-specific error domain range

Upstream requirements: RS_AP_00130

⌈Error domain identifiers where the top 32 bits (i.e. bit #63..#32) are equal to 0xc000’0000 are reserved for vendor-specific error domains. Bits #31..#16 hold the vendor’s numerical identifier [9], and bits #15..#0 can be used by each vendor for error domain identifiers.⌋

[SWS_CORE_00013] The Future error domain

Upstream requirements: RS_AP_00130

⌈There shall be an error domain `ara::core::FutureErrorDomain` for all errors originating from the interaction of the classes `ara::core::Future` and `ara::core::Promise`. It shall have the shortname `Future` and the identifier 0x8000’0000’0000’0013.⌋

[SWS_CORE_00014] The Core error domain

Upstream requirements: RS_AP_00130

⌈There shall be an error domain `ara::core::CoreErrorDomain` for errors originating from non-`Future`/`Promise` facilities of `ara::core`. It shall have the shortname `Core` and the identifier 0x8000’0000’0000’0014.⌋

7.2.5.2.2 Vendor error domains

Each API that specifies a return type capable of holding an instance of `ara::core::ErrorCode` typically specifies a number of specified errors. These are error conditions that all or at least most implementations of that API are expected to generate.

In addition to that, however, an implementation may also return additional, non-specified errors. These are called vendor-specific errors.

Note: This applies in particular to APIs returning `ara::core::Result` or `ara::core::Future`.

[SWS_CORE_00092] Vendor-specific errors

Upstream requirements: RS_AP_00142

⌈A vendor-specific error shall use a vendor-specific error domain.⌋

7.2.5.2.3 Vendor Header File

To still make portable applications possible, vendor-specific errors could be enclosed in a pre-processor check of the vendor.

[SWS_CORE_00093] Vendor header file

Upstream requirements: RS_AP_00130

⌈The preprocessor symbol `ARA_VENDOR` defined in `ara/core/vendor.h` shall unconditionally contain the Vendor ID of the AUTOSAR framework implementation:

`#define ARA_VENDOR /* Vendor ID */` [9]⌋

7.2.5.2.4 Creating new error domains

Any new software module with significant logical separation from all existing modules of the Adaptive Platform should define one or more own error domains.

An error domain consists of:

- an error condition enumeration
- an exception base class
- an `ara::core::ErrorDomain` sub-class
- a non-member `ErrorDomain` sub-class accessor function
- a non-member `MakeErrorCode` function overload

All these are to reside not in the `ara::core` namespace, but in the “target” one.

[SWS_CORE_10999] Custom error domain scope

Upstream requirements: RS_AP_00130

⌈The `ara::core::ErrorDomain` sub-class and the corresponding enumeration, exception base class, non-member accessor function, and the `MakeErrorCode` overload shall be defined in the same namespace as the software module for which they are being specified.⌋

Note: This is to help making sure that the C++ ADL mechanism works as expected by other parts of this standard.

An error domain defined in the way specified in this section is suitable to be used for the ApApplicationErrorDomain model element.

Throughout this section, the character sequence <SN> is a placeholder for the *short-name* of the ApApplicationErrorDomain.

7.2.5.2.4.1 ErrorDomain sub-class

Then, a new class is created that derives from ara::core::ErrorDomain and overrides all the pure virtual member functions. In addition to that, it also needs to define in its scope a type alias called Errc for the error condition enumeration, as well as another type alias called Exception for the exception base class for this new error domain.

[SWS_CORE_10930] ErrorDomain sub-class type

Upstream requirements: RS_AP_00130

⌈Each error domain shall define a class type that derives publicly from ara::core::ErrorDomain.⌋

[SWS_CORE_10931] ErrorDomain sub-class naming

Upstream requirements: RS_AP_00130

⌈All sub-classes of ara::core::ErrorDomain shall follow the naming scheme <SN>ErrorDomain, where <SN> is the shortname of the ApApplicationErrorDomain.⌋

[SWS_CORE_10932] ErrorDomain sub-class non-extensibility

Upstream requirements: RS_AP_00130, RS_AP_00140

⌈All sub-classes of ara::core::ErrorDomain shall be final.⌋

[SWS_CORE_10933] ErrorDomain sub-class Errc symbol

Upstream requirements: RS_AP_00130

⌈All sub-classes of ara::core::ErrorDomain shall contain in their scope a type alias called Errc that refers to the error condition enumeration defined by [SWS_CORE_10900].⌋

[SWS_CORE_10934] ErrorDomain sub-class Exception symbol

Upstream requirements: RS_AP_00130

⌈All sub-classes of ara::core::ErrorDomain shall contain in their scope a type alias called Exception that refers to the exception base type defined by [SWS_CORE_10910].⌋

All ErrorDomain sub-classes are usable from within constant expressions, see [SWS_CORE_10400]. In particular, this includes that ErrorDomain sub-classes can be defined as constexpr global variables.

In order to further ease working with error domains, all member functions of the ErrorDomain sub-class are required to be noexcept, with the obvious exception of ara::core::ErrorDomain::ThrowAsException.

[SWS_CORE_10950] ErrorDomain sub-class member function property

Upstream requirements: RS_AP_00130

⌈With the exception of ara::core::ErrorDomain::ThrowAsException, all public member functions of all ErrorDomain sub-classes shall be noexcept.⌋

The virtual member function ara::core::ErrorDomain::Name returns the shortname of the ApApplicationErrorDomain, mostly for logging purposes.

[SWS_CORE_10951] ErrorDomain sub-class shortname retrieval

Upstream requirements: RS_AP_00130

⌈The return value of an error domain's ara::core::ErrorDomain::Name member function shall be equal to the shortname of the ApApplicationErrorDomain.⌋

Each error domain has an identifier that is used to determine equality of error domains. The error domains that are pre-defined by the Adaptive Platform have standardized identifiers. Application-specific error domains should make sure their identifiers are system-wide unique.

[SWS_CORE_10952] ErrorDomain sub-class unique identifier retrieval

Upstream requirements: RS_AP_00130

⌈The return value of an error domain's ara::core::ErrorDomain::Id member function shall be a unique identifier that follows the rules defined by [SWS_CORE_00010].⌋

An ErrorDomain can "transform" an ErrorCode into an exception.

[SWS_CORE_10953] Throwing ErrorCodes as exceptions

Upstream requirements: RS_AP_00130

⌈The type of an exception thrown by the ErrorDomain sub-class's implementation of ara::core::ErrorDomain::ThrowAsException shall derive from that ErrorDomain sub-class's Exception type alias defined by [SWS_CORE_10934].⌋

7.2.5.2.4.2 Non-member ErrorDomain sub-class accessor function

A non-member accessor function for the new error domain class is to be defined. For an error domain class MyErrorDomain, the accessor function is named GetMyErrorDomain. This accessor function returns a reference to a single global instance of that class. This accessor function shall be fully constexpr-capable; this in turn implies that the ErrorDomain sub-class also shall be constexpr-constructible (see [SWS_CORE_10400]).

[SWS_CORE_10980] ErrorDomain sub-class accessor function

Upstream requirements: RS_AP_00130

⌈For all sub-classes of ara::core::ErrorDomain, there shall be a non-member constexpr function that returns a reference-to-const to a singleton instance of it.⌋

[SWS_CORE_10981] ErrorDomain sub-class accessor function naming

Upstream requirements: RS_AP_00130

⌈All ara::core::ErrorDomain sub-class accessor functions shall follow the naming scheme Get<SN>ErrorDomain, where <SN> is the shortname of the ApApplicationErrorDomain.⌋

[SWS_CORE_10982] ErrorDomain sub-class accessor function

Upstream requirements: RS_AP_00130

⌈All ara::core::ErrorDomain sub-class accessor functions shall have a return type of const ErrorDomain&.⌋

7.2.5.2.4.3 Non-member MakeErrorCode overload

A non-member factory function MakeErrorCode needs to be defined, which is implicitly used by the convenience constructors of class ara::core::ErrorCode. This factory function will make use of the non-member accessor function for the error domain sub-class, and call the type-erased constructor of class ara::core::ErrorCode.

[SWS_CORE_10990] MakeErrorCode overload for new error domains

Upstream requirements: RS_AP_00130

⌈For all sub-classes of `ara::core::ErrorDomain`, there shall be a `constexpr` overload of the non-member function `MakeErrorCode` that creates an `ara::core::ErrorCode` instance for a given error condition value within the `ara::core::ErrorDomain` sub-class's error condition range.⌋

[SWS_CORE_10991] MakeErrorCode overload signature

Upstream requirements: RS_AP_00130

⌈All overloads of the non-member function `MakeErrorCode` shall have the following signature:

```
    constexpr ErrorCode MakeErrorCode(<SN>Errc code, ErrorDomain::
        SupportDataType data) noexcept;
```

where `<SN>` is the shortname of the `ApApplicationErrorDomain`.⌋

7.2.5.2.4.4 Error condition enumeration

The error condition enumeration describes all known error conditions of the new software module. It should be reasonably fine-grained to allow users to differentiate error conditions that they might want to handle in different ways.

[SWS_CORE_10900] Error condition enumeration type

Upstream requirements: RS_AP_00130

⌈Each error domain shall define an error condition enum class with the base type `ara::core::ErrorDomain::CodeType` that holds all error conditions of that error domain.⌋

[SWS_CORE_10901] Error condition enumeration naming

Upstream requirements: RS_AP_00130

⌈Error domain error condition enumerations shall follow the naming scheme `<SN>Errc`, where `<SN>` is the shortname of the `ApApplicationErrorDomain`.⌋

[SWS_CORE_10902] Error condition enumeration contents

Upstream requirements: RS_AP_00130

⌈Error domain error condition enumerations shall not contain any values that indicate success.⌋

[SWS_CORE_10903] Error condition enumeration numbers

Upstream requirements: RS_AP_00130

⌈Error domain error condition enumerations shall keep the number 0 unassigned.⌋

7.2.5.2.4.5 Exception base class

As a complement to the error condition enumeration, an exception base class for this error domain also needs to be defined. This exception base class is used for the "transformation" of an `ara::core::ErrorCode` object into an exception.

Additional exception types can be defined by the software module, but all these then derive from this base type.

[SWS_CORE_10910] ErrorDomain exception base type

Upstream requirements: RS_AP_00130

⌈Each error domain shall define an exception base type that is a sub-class of `ara::core::Exception`.⌋

[SWS_CORE_10911] ErrorDomain exception base type naming

Upstream requirements: RS_AP_00130

⌈All error domain exception base types specified by [SWS_CORE_10910] shall follow the naming scheme `<SN>Exception`, where `<SN>` is the shortname of the `ApApplicationErrorDomain`.⌋

[SWS_CORE_10912] ErrorDomain exception type hierarchy

Upstream requirements: RS_AP_00130

⌈All additional exception types defined by a software module shall have the exception base type specified by [SWS_CORE_10910] as a base class.⌋

7.2.5.2.4.6 Error Domain C++ pseudo code example

The following C++ pseudo code illustrates how these definitions come together:

```
namespace my
{

enum class <SN>Errc : ara::core::ErrorDomain::CodeType
{
    // ...
};

class <SN>Exception : public ara::core::Exception
{
public:
    <SN>Exception(ara::core::ErrorCode err) noexcept;
};

class <SN>ErrorDomain final : public ara::core::ErrorDomain
{
public:
    using Errc = <SN>Errc;
    using Exception = <SN>Exception;

    constexpr <SN>ErrorDomain() noexcept;

    const char* Name() const noexcept override;
    const char* Message(ara::core::ErrorDomain::CodeType errorCode)
        const noexcept override;
    void ThrowAsException(const ara::core::ErrorCode& errorCode) const
        noexcept(false) override;
};

constexpr const ara::core::ErrorDomain& Get<SN>ErrorDomain() noexcept;

constexpr ara::core::ErrorCode MakeErrorCode(<SN>Errc code, ara::core::
    ErrorDomain::SupportDataType data) noexcept;

}  // namespace my
```

7.2.5.3 Result

The `ara::core::Result` type follows the `ValueOrError` concept from the C++ proposal p0786 [10]. It either contains a value (of type `ValueType`), or an error (of type `ErrorType`). Both `ValueType` and `ErrorType` are template parameters of `ara::core::Result`, and due to their templated nature, both value and error can be of any type. `ErrorType` is defaulted to `ara::core::ErrorCode`, and it is enforced that only this type is allowed as the `ErrorType` template parameter for `ara::core::Result`.

`ara::core::Result` acts as a “wrapper type” that connects the exception-less API approach using `ara::core::ErrorCode` with C++ exceptions. As there is a direct mapping between `ara::core::ErrorCode` and a domain-specific exception type, `ara::core::Result` allows to “transform” its embedded `ara::core::ErrorCode` into the appropriate exception type, by calling `ara::core::Result::ValueOrThrow`.

[SWS_CORE_10600] Semantics of `ara::core::Result`

Upstream requirements: RS_AP_00119, RS_AP_00142, RS_AP_00128

⌈The type `ara::core::Result` shall provide a means to handle both return values and errors from synchronous function calls in an exception-less way, by providing an encapsulated return type which may be either:

- a ara::core::Result::value_type=T: where T may be any C++ type; or
- an ara::core::Result::error_type=E: where E is ara::core::ErrorCode.

⌋

[SWS_CORE_00069] ara::core::Result<T&> may hold lvalue reference types

Upstream requirements: RS_AP_00130

⌈The type ara::core::Result<T&> shall support containerization of lvalue reference types (see [basic.type.qualifier], [basic.lval] in [11]) as value_type.⌋

[SWS_CORE_00070] Assigning to ara::core::Result<T&>

Upstream requirements: RS_AP_00130

⌈Assignment to ara::core::Result<T&> shall "rebind" to the contained value⌋

See also the "rebind" approach in 7.6.4.1.1

The class ara::core::Result<void,E> provides a further specialization of ara::core::Result where the value_type=void. This specialization omits these member functions that are defined in the generic template:

- operator->
- Bind

In addition, a number of function overloads collapse to a single, no-argument one.

7.2.5.4 Future and Promise

ara::core::Future and its companion class ara::core::Promise are closely modeled on std::future and std::promise, but have been adapted to interoperate with ara::core::Result. Similar to ara::core::Result described in section 7.2.5.3, the class ara::core::Future either contains a value, or an error (the Future first has to be in “ready” state, though).

Class ara::core::Promise has been adapted in two aspects:

- std::promise::set_exception has been removed and ara::core::Promise::SetError has been introduced in its stead.
- For ara::core::Future, there is a new member function ara::core::Future::GetResult that is similar to ara::core::Future::get, but never throws an exception and returns a ara::core::Result instead.

Thus, ara::core::Future as return type allows the same dual approach to error handling as ara::core::Result, in that it either works exception-based (with ara::core::Future::get), or exception-free (with ara::core::Future::GetResult).

ara::core::Result is a type used for returning values or errors from a *synchronous* function call, whereas ara::core::Future is a type used for returning values or errors from an *asynchronous* function call.

Whenever there is a mention of a standard C++14 item (class, class template, enum or function) such as std::future or std::promise, the implied source material is [5]. Whenever there is a mention of an experimental C++ item such as std::experimental::future::is_ready, the implied source material is [12].

Futures are technically referred to as "asynchronous return objects", and Promises are referred to as "asynchronous providers". Their interaction is made possible by a shared state. The shared state concept is described in [5], section 30.6.4. The description also applies to the shared state behind ara::core::Future and ara::core::Promise, with the following changes:

- The text *", as used by async when policy is launch::deferred"* is removed from paragraph 2.
- Paragraph 10, referring to *"promise::set_value_at_thread_exit"*, is removed.
- Each mention of "exception" is replaced with "error"
- In paragraph 7 "stores an exception object of type future_error with an error condition of broken_promise within its shared state; and then" is replaced with "stores the ErrorCode kBrokenPromise defined in [SWS_CORE_00400] in its shared state; and then"

Class ara::core::Future and ara::core::Promise are closely modeled on std::future and std::promise. Consequently, the behavior of ara::core::Future and ara::core::Promise is expected to be same as that of std::future and std::promise from [5, the C++14 standard] and the corresponding std::experimental:: classes from [12], except for the deviations from the std:: classes that result from the integration with ara::core::Result.

[SWS_CORE_10800] Semantics of ara::core::Future and ara::core::Promise

Upstream requirements: RS_AP_00138, RS_AP_00128

⌈The types ara::core::Future and ara::core::Promise shall provide a means to handle both return values and errors from asynchronous function calls in an exception-less way. Together, they provide a means to store a value type *T* or an ara::core::ErrorCode which may be asynchronously retrieved in a thread-safe manner at a later point in time.⌋

7.2.5.5 Exceptions

The Adaptive Platform defines a base exception type `ara::core::Exception` for all exceptions defined in the standard. This exception takes a `ara::core::ErrorCode` object as mandatory constructor argument, similar to the way `std::system_error` takes a `std::error_code` argument for construction.

Below this exception base type, there is an additional layer of exception base types, one for each error domain.

For error domains that are modeled in ARXML, the C++ language binding will generate an exception class in addition to the ErrorDomain sub-class (which is described in section 7.2.5.2). This exception class also conforms to a standard naming scheme: `shortName` of `ApApplicationErrorDomain` plus `Exception` suffix (this makes it distinguishable from the `ara::core::ErrorDomain` sub-class itself). It is located in the same namespace as the corresponding `ara::core::ErrorDomain` sub-class.

7.2.5.6 Duality of ErrorCode and Exception

By using the classes listed above, all APIs of the Adaptive Platform can be used with either an exception-based or an exception-less error handling workflow. However, no API function will ever treat an `Error` by throwing an exception directly; it will always return an error code in the form of a `ara::core::Result` or `ara::core::Future` return value instead. It is then possible for the caller to “transform” the `Error` into an exception, typically via the member function `ara::core::Result::ValueOrThrow`.

When working with a C++ compiler that does not support exceptions at all (or one that has been configured to disable them with an option such as g++’s `-fno-exceptions`), all API functions still show the same behavior. What *does* differ then is that `ara::core::Result::ValueOrThrow` is not defined – this member function is only defined when the compiler does support exceptions.

7.3 Thread safety

The [5, the C++14 standard] does not provide explicit criteria for defining thread-safety in functions, it is therefore necessary to provide an AUTOSAR definition to set the criteria during evaluation of whether an AUTOSAR function shall be thread-safe. The definitions are deliberately scoped to AUTOSAR APIs, and do not apply to non-AUTOSAR APIs where no guarantees are given.

[SWS_CORE_13200] AUTOSAR definition of a thread-safe member function

Upstream requirements: RS_AP_00164

⌈An AUTOSAR API member function is thread-safe if the function can be called on the same object concurrently by multiple threads without the need for the caller to synchronize between the calls.⌋

Note: This definition also applies to destructors and assignment operators. They are usually not thread safe. That means concurrent assignments to the same object need to be synchronized by the user of the API.

[SWS_CORE_13201] AUTOSAR definition of multiple thread-safe member functions

Upstream requirements: RS_AP_00164

⌈For all AUTOSAR API member functions of an object that are marked thread-safe it shall be possible to call them on the same object concurrently by multiple threads without the need for the caller to synchronize between the calls.⌋

[SWS_CORE_13202] AUTOSAR definition of a thread-safe non-member function

Upstream requirements: RS_AP_00164

⌈An AUTOSAR API non-member function is thread-safe if the function can be called concurrently by multiple threads without the need for the caller to synchronize between the calls.⌋

[SWS_CORE_13203] Behavior of a concurrently executed non-thread-safe member function

Upstream requirements: RS_AP_00164

⌈If a non-thread-safe member function is called concurrently on the same object to itself or other functions of that object, the behavior is undefined.⌋

Note: This applies even if the other function(s) are thread-safe.

[SWS_CORE_13204] Behavior of a concurrently executed non-thread-safe non-member function

Upstream requirements: RS_AP_00164

⌈If a non-thread-safe non-member function is called concurrently, the behavior is undefined.⌋

Note: Adaptive Applications should not call non-thread-safe functions from different threads concurrently.

[SWS_CORE_13205] Behavior of concurrently executed functions on different objects

Upstream requirements: RS_AP_00165

⌈AUTOSAR API member functions can be called concurrently on different objects by multiple threads without the need for the caller to synchronize between the calls.⌋

[SWS_CORE_13206] AUTOSAR `callable` definition

Upstream requirements: RS_AP_00130

⌈The `callable` related definitions in `[func.def]` in [5] shall apply to AUTOSAR APIs when used.⌋

[SWS_CORE_13207] Behavior of thread-safe `callable`

Upstream requirements: RS_AP_00164

⌈The thread-safe `callable` may be called concurrently (i.e. in the context of multiple threads or interleaved with multiple `ara::core::Future`/`ara::core::Promise` objects.⌋

[SWS_CORE_13208] Behavior of non-thread-safe `callable`

Upstream requirements: RS_AP_00164

⌈The non-thread-safe `callable` is always called non-concurrently.⌋

[SWS_CORE_13209] Behavior of conditionally thread-safe `callable`

Upstream requirements: RS_AP_00164

⌈The conditionally thread-safe `callable` is called as defined by the "Thread Safety" attribute of the function in the respective table.⌋

7.4 Async signal safety

An *async-signal-safe* function is one that can be safely called from within a POSIX signal handler.

[13, The POSIX standard] defines a set of functions that are guaranteed to be async-signal-safe; all functions not on that list need to be assumed unsuitable to be called within a signal handler. This includes all ARA APIs, as it is not specified (and in general not possible to determine) which other functions (whether from POSIX or from other standards or implementations) are called within them.

[SWS_CORE_13210] ARA API usage within a signal handler

Upstream requirements: RS_AP_00163

⌈Usage of any ARA API within a signal handler will result in undefined behavior of the application, unless otherwise specified.⌋

7.5 Explicitly aborting an Operation

If a Violation has been detected by the implementation of an API function, [SWS_CORE_00003] mandates to abort this operation immediately. It allows two ways to do this; either by throwing certain kinds of exceptions (if the implementation supports C++ exceptions), or by calling ara::core::Abort.

Calling ara::core::Abort will result in an Explicitly aborting an Operation, which usually leads to an Unexpected Termination as defined by [14]. This section defines the behavior of this mechanism.

Like std::abort, calling ara::core::Abort is meant to terminate the current process abnormally and immediately, without performing stack unwinding and without calling destructors of static objects.

[SWS_CORE_12403] Standard Error Stream Logging of Explicitly aborting an Operation

Upstream requirements: RS_AP_00130

⌈Calling ara::core::Abort shall result in a log message being output to the process' standard error stream. The log message shall contain the string representation of the parameters that have been passed to the function as arguments, in the same order that they have been passed.⌋

[SWS_CORE_12408] DLT Logging of Explicitly aborting an Operation

Upstream requirements: RS_AP_00130

⌈Calling ara::core::Abort should result in a DLT message being logged into the log sinks as a fatal log as defined by ara::log for the affected process or the Execution Management.
For that ara::core::Abort needs to distinguish between calls that resulted from a Violation or Failed Default Allocation and calls from application code.

- For calls that resulted from a Violation or Failed Default Allocation the logging behavior as defined in [SWS_CORE_00090], [SWS_CORE_00003], and [SWS_CORE_00005] applies.
- For calls from application code an AbortMessage ([SWS_CORE_13019]) with the *message* argument equal to the log message that was output to the standard

error stream according to [SWS_CORE_12403] and the context id of ara::core shall be used.

⌋

Note: To distinguish between calls that resulted from a `Violation` or `Failed Default Allocation` and calls from application code, the caller can, in the first case, pass special implementation-defined types that represent the desired DLT message as an argument.

[SWS_CORE_12407] Thread-safety when `Explicitly aborting an Operation`

Upstream requirements: RS_AP_00130

⌈While a call to `ara::core::Abort` is in progress, other calls to this function shall block the calling threads.⌋

Note: The blocked thread will never continue, due to the forced termination by the initial `ara::core::Abort` call. There will be no logging provided for this incident.

`ara::core::Abort` provides a means to add a “hook” into the system, by calling `ara::core::SetAbortHandler`, similar to the way `std::atexit` allows to install a callback for the `std::exit` mechanism.

[SWS_CORE_12404] AbortHandler invocation

Upstream requirements: RS_AP_00130

⌈Calling `ara::core::Abort` shall invoke the `AbortHandlers` after the log message as per [SWS_CORE_12403] has been output, in the reverse order of installation.⌋

7.5.1 AbortHandler

This handler can be installed with `ara::core::SetAbortHandler` or `ara::core::AddAbortHandler`. It is invoked in turn when `ara::core::Abort` is called, and it may perform arbitrary operations and then has these four principal choices for its final statements: it can either

- terminate the process, or
- return from the function call, or
- defer function return by entering an infinite loop, or
- perform a non-local goto operation such as `std::longjmp`.

The use of non-local goto operations, including std::longjmp, is strongly discouraged and also expressively prohibited by MISRA, and most other coding guidelines as well.

Similarly, deferring function return by entering an infinite loop is discouraged as well; while this still leads to the desired outcome that the *operation* which caused a Violation has been aborted, it will do so at the cost of "defunct'ing" the calling thread and risking the destabilization of the software, which already has encountered a Violation.

An AbortHandler that terminates the process is strongly advised to do so by calling std::abort. This will make sure that the Unexpected Termination is properly seen by Execution Management as an Abnormal Termination as well.

If all AbortHandlers return, or if no AbortHandler is defined at all, then the final action of ara::core::Abort is to call std::abort.

[SWS_CORE_12405] Final action without AbortHandler

Upstream requirements: RS_AP_00130

⌈If there is no custom ara::core::AbortHandler that has been installed with ara::core::SetAbortHandler or ara::core::AddAbortHandler, then the implementation of ara::core::Abort shall call std::abort().⌋

[SWS_CORE_12406] Final action with returning AbortHandlers

Upstream requirements: RS_AP_00130

⌈If there are custom ara::core::AbortHandlers that have been installed with ara::core::SetAbortHandler or ara::core::AddAbortHandler and all of them return, then the implementation of ara::core::Abort shall call std::abort().⌋

[SWS_CORE_12409] Usage of ARA API within AbortHandlers

Upstream requirements: RS_AP_00130

⌈Usage of any ARA API within an ara::core::AbortHandler may result in undefined behavior of the application, unless otherwise specified.⌋

7.5.2 SIGABRT handler

In addition to the ara::core::AbortHandler, or alternatively to it, the application can also influence this mechanism by installing a signal handler for SIGABRT.

The signal handler for SIGABRT has the same choices of actions as the ara::core::AbortHandler: it can terminate the process, return from the function call, defer function return by entering an infinite loop, or perform a non-local goto operation. The same caveats as for the ara::core::AbortHandler apply here: non-local goto operations and infinite loops should be avoided.

If the SIGABRT handler does not return, it should in general terminate abnormally with SIGABRT. To do this without entering an infinite loop, it should restore the default disposition of SIGABRT with std::signal(SIGABRT, SIG_DFL) and then re-raise SIGABRT with e.g. std::raise(SIGABRT).

This "second step" of influence that the SIGABRT handler provides allows applications that are already handling other synchronous signals such as SIGSEGV or SIGFPE to treat SIGABRT the same way.

7.6 Core data types

7.6.1 InstanceSpecifier

Instances of ara::core::InstanceSpecifier are used to identify service port prototype instances within the AUTOSAR meta-model and are therefore used in the ara::com API and elsewhere. A detailed description and background can be found in [15] chapter ("Instance Identifiers") and chapter ("Usage of meta-model identifiers within ara::com based application code").

ara::core::InstanceSpecifier can conceptually be understood to be a wrapper for a string representation of a valid meta-model path. It is designed to be either constructed from a string representation via a factory method ara::core::InstanceSpecifier::Create, which provides an exception-free solution, or directly by using the constructor, which might throw an exception if the string representation is invalid.

[SWS_CORE_10200] Valid InstanceSpecifier representations - application interaction

Upstream requirements: RS_AP_00130

⌈In case of application interaction and thus in the presence of PortPrototypes, the string representation of a valid ara::core::InstanceSpecifier starts with a leading "/" and defines the model path with a "/"-separated list of model element shortNames: starting from an Executable via the RootSwComponentPrototype and optionally several further SwComponentPrototypes (if CompositionSwComponentTypes are used) to the respective PortPrototype to which the ara::core::InstanceSpecifier shall apply.⌋

Thus, in case of application interaction the content of a valid ara::core::InstanceSpecifier adheres to the following pattern:
/Executable.shortName/RootSwComponentPrototype.shortName
/SwComponentPrototype.shortName/.../PortPrototype.shortName

[SWS_CORE_10203] Valid InstanceSpecifier representations - functional cluster interaction

Upstream requirements: RS_AP_00130

⌈In case of functional cluster interaction and thus in the absence of `PortPrototype`s the string representation of a valid `ara::core::InstanceSpecifier` starts with a leading `"/"` and defines the model path starting and ending with the `FunctionalClusterInteractsWithFunctionalClusterMapping.shortName` (see [TPS_MANI_03268] for further details).⌋

Thus, in case of functional cluster interaction the content of a valid `ara::core::InstanceSpecifier` adheres to the following pattern: `/FunctionalClusterInteractsWithFunctionalClusterMapping.shortName`

[SWS_CORE_10201] Validation of meta-model paths

Upstream requirements: RS_AP_00130

⌈The construction mechanisms of class `ara::core::InstanceSpecifier` shall reject meta-model paths that are syntactically invalid according to the syntax rules defined in [SWS_CORE_10200].⌋

[SWS_CORE_10202] Construction of InstanceSpecifier objects

Upstream requirements: RS_AP_00130

⌈APIs for construction of `ara::core::InstanceSpecifier` objects shall be available in both potentially-throwing and non-throwing form.⌋

7.6.2 Executor

The `ara::core::Executor` provides a standardized interface for AUTOSAR APIs to execute callable objects asynchronously in a thread-safe context. For callable objects which have a defined return type, the execution result/error is wrapped in a `ara::core::Future` object and returned to the caller. For callable objects with "fire-and-forget" execution semantics, this is omitted.

7.6.3 Types derived from the base C++ standard

In addition to AUTOSAR-devised data types, which are mentioned in the previous chapters, the Adaptive Platform also contains a number of generic data types and helper functions.

Some types are already contained in [5, C++14]; however, types with almost identical behavior are re-defined within the `ara::core` namespace. The reason for this is that the memory allocation behavior of the `std::` types is often unsuitable for automotive purposes. Thus, the ara::core ones define their own memory allocation behavior, and perform some other necessary adaptions as well, including about the throwing of exceptions.

[SWS_CORE_00040] Errors originating from C++ standard classes

Upstream requirements: RS_AP_00130

⌈For the classes in ara::core specified below by means of the corresponding classes of the C++ standard, all functions that are specified by [5, C++14], [11, C++17], or [16, C++20] to throw any exceptions, are instead specified to be the cause of a Violation when they do so.⌋

Examples for such data types are: ara::core::Array, ara::core::Vector, ara::core::Map, and ara::core::String.

7.6.3.1 Array

This section describes the ara::core::Array type that represents a container which encapsulates fixed size arrays.

ara::core::Array is an almost-equivalent of std::array, and most type properties of std::array apply to ara::core::Array as well.

These differences to std::array are intended:

- ara::core::Array::at uses Violations instead of exceptions as the error mechanism

The overloads of std::get, contained in the ara::core namespace, are available. They can either be called explicitly (i.e. namespace-qualified), or be invoked via ADL. For ADL lookup to work in C++14, get needs to be called without namespace qualification, similar to the way that swap is recommended to be called, e.g.:

```
using std::get;

ara::core::Array<int, 4> array = {1, 2, 3, 4};
int& e = get<0>(array);
```

[SWS_CORE_11200] Array base behavior

Upstream requirements: RS_AP_00130

⌈ara::core::Array and all its member functions and supporting constructs shall behave identical to those of std::array in header <array> from [5, the C++14 standard], except for the differences specified in this document.⌋

7.6.3.2 Vector

This section describes the ara::core::Vector type that represents a container of variable size.

[SWS_CORE_11300] Vector base behavior

Upstream requirements: RS_AP_00130

⌈ara::core::Vector and all its member functions and supporting constructs shall behave identical to those of std::vector in header <vector> from [5, the C++14 standard], except for the differences specified in this document.⌋

7.6.3.3 Map

This section describes the ara::core::Map type that represents an associative container of variable size.

[SWS_CORE_11400] Map base behavior

Upstream requirements: RS_AP_00130

⌈ara::core::Map and all its member functions and supporting constructs shall behave identical to those of std::map in header <map> from [5, the C++14 standard], except for the differences specified in this document.⌋

7.6.3.4 String and BasicString

This section describes the ara::core::String and ara::core::BasicString types.

These types are closely modeled on std::string and std::basic_string respectively from [5, the C++14 standard], with a number of additions coming from [11, the C++17 standard].

As the UTF-8 encoding is used throughout the Adaptive Platform, only the char type is supported for ara::core::BasicString.

[SWS_CORE_12000] String base behavior

Upstream requirements: RS_AP_00130

⌈ara::core::String, ara::core::BasicString and all their member functions and supporting constructs shall behave identical to those of std::string and std::basic_string in header <string> from [5, C++14], except for the differences specified in this document.⌋

7.6.3.5 SteadyClock

7.6.3.5.1 Definitions of terms

The C++ `std::chrono` library defines a number of concepts and types for handling time and durations. One of these concepts is that of a “clock” which is able to create snapshots of specific “time points”. When talking about clocks and time points, the three qualities *resolution*, *precision*, and *accuracy* are distinguished within this document as follows:

- The `resolution` relates to the smallest increment that can be expressed with the clock’s measurement data type.

  For clocks of the POSIX `clock_gettime` API, the `resolution` is implicitly defined as nanoseconds by the API’s usage of `struct timespec` with its `timespec::tv_nsec` field.

  For C++ clocks of the `std::chrono` APIs, the `resolution` is variable.

- The `precision` of a clock is the smallest time interval that its timer is able to measure. The `precision` is implementation-defined and depends on the properties and capabilities of the physical machine as well as the operating system.

- The `accuracy` of a clock is the relation between the reported value and the truth.

In addition to that, the `epoch` is an important property of a clock as well, as it defines the base of the time range that can originate from a clock. Clocks that measure calendar time often use “Unix time”, which is given as number of seconds (without leap seconds) since the “Unix Epoch”, which is 1970-01-01, 00:00:00 UTC.

Clocks that place more emphasis on high `precision` often do not relate to calendar time at all, but generate timestamps as offsets from something like the power-up time of the system.

7.6.3.5.2 Clocks in the Adaptive Platform

The C++ `std::chrono` library defines a number of standard clocks. Amongst these is `std::chrono::steady_clock`, which represents a monotonic clock whose time points are strictly increasing with a fixed interval.

However, the C++ standard does not place any requirements on the `resolution`, `precision`, and `accuracy` of this clock. The undefined-ness of its `resolution` can pose some difficulties for application programmers, but these can usually be solved by agreeing on a common – or minimum – `resolution`. The `precision` and `accuracy` are always dependent on the physical properties of the machine and of the operating system.

The Adaptive Platform defines `ara::core::SteadyClock` as a `std::chrono`-compatible clock with nanosecond `resolution` and a `std::int64_t` datatype. Its precision and accuracy are still implementation-defined and can be given as characteristic values of a concrete platform. Its epoch is the power-up time of the ECU. With these properties, timestamps generated by ara::core::SteadyClock will not overflow until 292 years after its epoch.

It is the standard clock of the Adaptive Platform and should be used for most timekeeping purposes.

The properties of ara::core::SteadyClock imply that a type alias to std::chrono::steady_clock is a conforming implementation of ara::core::SteadyClock, if std::chrono::steady_clock::period is equivalent to std::nano, and std::chrono::steady_clock::rep is a 64-bit signed integer type such as std::int64_t.

[SWS_CORE_11800] SteadyClock type requirements

Upstream requirements: RS_AP_00130

⌈Class ara::core::SteadyClock shall meet the requirements of *TrivialClock* from [5, the C++14 standard].⌋

[SWS_CORE_11801] Epoch of SteadyClock

Upstream requirements: RS_AP_00130

⌈The epoch of ara::core::SteadyClock shall be the system start-up.⌋

7.6.4 Types derived from newer C++ standards

These types have been defined in or proposed for a newer C++ standard, and the Adaptive Platform includes them into the ara::core namespace, usually because they are necessary for certain constructs of the Manifest.

Examples for such data types are: Optional, StringView, Span, and Variant.

7.6.4.1 Optional

This section describes the class template ara::core::Optional. The type ara::core::Optional manages optional values, i.e. values that may or may not be present in a container. The existence can be evaluated during both compile-time (via constexpr specifiers) and runtime. Whenever there is a mention of std::optional, the implied source material is [11, C++17].

There are two usages:

- to provide access to optional record elements of a StdCppImplementationDataType.category==STRUCTURE (see [TPS_MANI_03181], [SWS_LBAP_00010]). Mandatory record elements are declared directly with the corresponding CppImplementationDataType without using ara::core::Optional.
- to provide a return value of a function that may or may not exist

[SWS_CORE_11000] Optional base behavior

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈ara::core::Optional and all its member functions and supporting constructs shall behave identical to those of std::optional in header <optional> from [11, C++17], except for the differences specified in this document.⌋

Note: The term “supporting constructs” is meant to include all non-member functions, types and objects specified in header <optional> from [11, the C++17 standard]. The bad_optional_access exception defined in the C++ standard library is left out of this specification to provide an API that does not make use of exceptions. Accessor functions e.g., operator* use Violations instead of exceptions as the error mechanism. Use either has_value or operator bool() to check if the ara::core::Optional contains a value before accessing the value. Alternatively, use the value_or functions to access the value and provide a default value in case the ara::core::Optional contains no value.

[SWS_CORE_01031] class bad_optional_access

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈Contrary to the definitions in [11], no class named bad_optional_access shall be defined in the ara::core namespace.⌋

7.6.4.1.1 Optional references

[SWS_CORE_01032] Optional references

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈The type ara::core::Optional shall support lvalue references.⌋

For Optional references, a major question is how assignments should behave. For sequence of statements such as:

```
int x = 42;
Optional<int&> opt = x;  // "bind" opt to x
```

```
int y = 43;
opt = y;  // "assign-through" or "rebind"?
```

the resulting program state could be:

- opt still refers to x, and x now equals 43 (“assign-through”)
- opt now refers to y, and x is unmodified (“rebind”)

The behavior of ara::core::Optional<T&> follows the "rebind" approach.

[SWS_CORE_01034] Assignment behavior of Optional references

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈Assignment to an ara::core::Optional<T&> shall "rebind" to the new object.⌋

7.6.4.2 Variant

This section describes the ara::core::Variant type that represents a type-safe union.

[SWS_CORE_11600] Variant base behavior

Upstream requirements: RS_AP_00130

⌈ara::core::Variant and all its member functions and supporting constructs shall behave identical to those of std::variant in header <variant> from [11, C++17], except for the differences specified in this document.⌋

7.6.4.3 StringView

This section describes the ara::core::StringView type that represents a non-owning view over a contiguous sequence of characters. ara::core::StringView is an almost-equivalent of std::string_view from [11, the C++17 standard], and most type properties of std::string_view apply to ara::core::StringView as well.

A number of member functions have been added that originate from [16, C++20 standard]:

- starts_with
- ends_with

and from [17, the C++23 standard draft]:

- contains

It is the programmer's responsibility to ensure that ara::core::StringView does not outlive the pointed-to character array.

As the UTF-8 encoding is used throughout the Adaptive Platform, only the char type is supported for ara::core::StringView.

[SWS_CORE_12200] StringView base behavior

Upstream requirements: RS_AP_00130

⌈ara::core::StringView and all its member functions and supporting constructs shall behave identical to those of std::string_view in header <string_view> from [11, C++17], except for the differences specified in this document.⌋

7.6.4.4 Span

ara::core::Span is a type that represents an abstraction over a linear sequence of values of a certain type. It is closely modeled on std::span from [16, C++20], with deviations mostly coming from the lack of [16, C++20]'s "ranges" feature.

[SWS_CORE_11900] Span base behavior

Upstream requirements: RS_AP_00130

⌈ara::core::Span and all its member functions and supporting constructs shall behave identical to those of std::span in header <span> from [16, C++20], except for the differences specified in this document.⌋

7.6.4.5 Byte

ara::core::Byte is a type that is able to hold a "byte" of the machine. It is an own type distinct from any other type.

The definitions of this section have been carefully set up in a way to make std::byte from [11, C++17] a conforming implementation, but also allow a class-based implementation with only C++14 means.

Unlike std::byte from [11, C++17], it is implementation-defined whether ara::core::Byte can be used for type aliasing without triggering Undefined Behavior.

[SWS_CORE_10100] Type property of `ara::core::Byte`

Upstream requirements: RS_AP_00130

⌈The type `ara::core::Byte` shall not be an integral type. In particular, the value `std::is_integral<ara::core::Byte>::value` shall be 0.⌋

[SWS_CORE_10101] Size of type `ara::core::Byte`

Upstream requirements: RS_AP_00130

⌈The size (in bytes) of an instance of type `ara::core::Byte` (determined with `sizeof(ara::core::Byte)`) shall be 1.⌋

[SWS_CORE_10102] Value range of type `ara::core::Byte`

Upstream requirements: RS_AP_00130

⌈The value of an instance of type `ara::core::Byte` shall be constrained to the range `[0..std::numeric_limits<unsigned char>::max()]`.⌋

[SWS_CORE_10103] Creation of `ara::core::Byte` instances

Upstream requirements: RS_AP_00130

⌈An instance of type `ara::core::Byte` shall be creatable from an integral type with brace-initialization syntax. This initialization shall also be possible when called in a constant expression. If the initializer value is outside the value range of type `ara::core::Byte` (see [SWS_CORE_10102]), the behavior is undefined.⌋

[SWS_CORE_10104] Default-constructed `ara::core::Byte` instances

Upstream requirements: RS_AP_00130

⌈An instance of type `ara::core::Byte` shall be constructible without giving an initializer value. Such a variable definition shall incur no runtime cost, and the value of the instance shall have indeterminate content.⌋

[SWS_CORE_10105] Destructor of type `ara::core::Byte`

Upstream requirements: RS_AP_00130

⌈The destructor of type `ara::core::Byte` shall be trivial.⌋

[SWS_CORE_10106] Implicit conversion from other types

Upstream requirements: RS_AP_00130

⌈The type `ara::core::Byte` shall not be implicitly convertible from any other type.⌋

[SWS_CORE_10107] Implicit conversion to other types

Upstream requirements: RS_AP_00130

⌈The type `ara::core::Byte` shall allow no implicit conversion to any other type, including `bool`.⌋

[SWS_CORE_10108] Conversion to unsigned char

Upstream requirements: RS_AP_00130

⌈The type `ara::core::Byte` shall allow conversion to `unsigned char` with a `static_cast<>` expression. This conversion shall also be possible when called in a constant expression.⌋

[SWS_CORE_10109] Equality comparison for `ara::core::Byte`

Upstream requirements: RS_AP_00130

⌈The type `ara::core::Byte` shall be comparable for equality with other instances of type `ara::core::Byte`. This comparison shall also be possible when called in a constant expression.⌋

[SWS_CORE_10110] Non-equality comparison for `ara::core::Byte`

Upstream requirements: RS_AP_00130

⌈The type `ara::core::Byte` shall be comparable for non-equality with other instances of type `ara::core::Byte`. This comparison shall also be possible when called in a constant expression.⌋

7.6.4.6 MemoryResource

AUTOSAR provides a set of interface APIs encapsulating memory resources for managing dynamic memory allocation. These are based on [16, C++20] standardized APIs and in general follow their specification:

- `ara::core::MemoryResource` (implements `std::pmr::memory_resource`)
- `ara::core::MonotonicBufferResource` (implements `std::pmr::monotonic_buffer_resource`)
- `ara::core::PolymorphicAllocator` (implements `std::pmr::polymorphic_allocator`)
- `ara::core::SynchronizedPoolResource` (implements `std::pmr::synchronized_pool_resource`)
- `ara::core::UnsynchronizedPoolResource` (implements `std::pmr::unsynchronized_pool_resource`)

Deviations in the AUTOSAR APIs to the ISO C++ APIs (if any) are explicitly documented in the respective API member.

7.6.4.7 Generic helpers

7.6.4.7.1 In-place disambiguation tags

The data types ara::core::in_place_t, ara::core::in_place_type_t, and ara::core::in_place_index_t are disambiguation tags that can be passed to certain constructors of ara::core::Optional and ara::core::Variant to indicate that the contained type shall be constructed in-place, i.e. without any copy operation taking place.

They are equivalent to std::in_place_t, std::in_place_type_t, and std::in_place_index_t from [11, C++17]. All these symbols are provided here in order to give the necessary support for implementing ara::core::Optional and ara::core::Variant in a way that is highly compatible with the corresponding classes from [11, C++17].

7.6.4.7.2 Non-member container access

ara::core::data, ara::core::size, ara::core::empty non-member functions allow uniform access to the data and size properties of contiguous containers.

They are equivalent to std::data, std::size, and std::empty from [11, C++17].

7.6.5 C++ attributes from newer C++ standards

Some compilers running with [5, ISO C++14] support a backport of [[nodiscard]] from [11, ISO C++17] but this is compiler specific.

[SWS_CORE_11952] Resolution of macro ARA_COMPILER_DEFINED_NODISCARD

Upstream requirements: RS_AP_00130

⌈The macro ARA_COMPILER_DEFINED_NODISCARD shall conditionally resolve to the C++ attribute [[nodiscard]], depending on whether this is supported by the compiler.⌋

7.7 Functional cluster life-cycle

Please note that there is a general behavior for Initialize and De-initialize defined in [2] by [SWS_CORE_90021] and [SWS_CORE_90022].

7.7.1 Startup

This functional cluster does not define any requirements related to its startup.

7.7.2 Shutdown

This functional cluster does not define any requirements related to its shutdown.

7.8 Reporting

7.8.1 Security Events

This functional cluster does not define any security events.

7.8.2 Log Messages

This chapter contains all Log Messages (i.e. DLT messages) of this Functional Cluster.

[SWS_CORE_13019] LogMessage AbortMessage

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

Dlt-Message	AbortMessage		
Description	Sent in case the affected process was terminated because of a call to ara::core::Abort. String format: "Process aborted via ara::core::Abort: {message}".		
MessageId	0x80001001		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
message	The message generated from the arguments given to ara::core::Abort.	uint8 [encoding UTF-8]	NoUnit

⌋

[SWS_CORE_13018] LogMessage FailedDefaultAllocation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

Dlt-Message	FailedDefaultAllocation		
Description	Sent in case the affected process was terminated because of a failed default allocation. String format: "Failed Default Allocation detected in {processIdentifier} at {location}: {message}".		
MessageId	0x80001000		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
processIdentifier	Identifier of the process that caused the violation.	uint8 [encoding UTF-8]	NoUnit
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	NoUnit
message	Additional message that can describe the cause of the failed default allocation in more detail.	uint8 [encoding UTF-8]	NoUnit

⌋

7.8.3 Violation Messages

This chapter contains all Violation Messages (i.e. DLT messages logged for Violations according to [SWS_CORE_00021]) defined by this Functional Cluster.

[SWS_CORE_13000] Violation Message InsufficientPermissionsViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

Dlt-Message	InsufficientPermissionsViolation		
Description	Sent in case the caller had insufficient permissions for the requested operation. String format: "Violation detected in {processIdentifier} at {location} due to insufficient permissions: {message}"		
MessageId	0x80001fff		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
processIdentifier	Identifier of the process that caused the violation.	uint8 [encoding UTF-8]	NoUnit
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	NoUnit
message	Additional message that describes the cause of the access violation.	uint8 [encoding UTF-8]	NoUnit

⌋

[SWS_CORE_13001] ViolationMessage BadVariantAccessViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

Dlt-Message	BadVariantAccessViolation		
Description	Violation message that is sent in case of a bad variant access String format: "Violation detected in {processIdentifier} at {location}: Bad variant access"		
MessageId	0x80001ffe		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
processIdentifier	Identifier of the process that caused the violation.	uint8 [encoding UTF-8]	NoUnit
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	NoUnit

⌋

[SWS_CORE_13002] ViolationMessage StringViewOutOfRangeViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

Dlt-Message	StringViewOutOfRangeViolation		
Description	Violation message that is sent in case of an out of range StringView access String format: "Violation detected in {processIdentifier} at {location}: StringView access out of range: Tried to access {pos} character of StringView of the length {N}""		
MessageId	0x80001ffd		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
processIdentifier	Identifier of the process that caused the violation.	uint8 [encoding UTF-8]	NoUnit
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	NoUnit
position	Position index value passed as input parameter.	uint8 [encoding UTF-8]	NoUnit
stringSize	Size of StringView	uint8 [encoding UTF-8]	NoUnit

⌋

[SWS_CORE_13003] ViolationMessage InstanceSpecifierMappingIntegrityViolation

Status: DRAFT

Upstream requirements: RS_AP_00142

⌈

Dlt-Message	InstanceSpecifierMappingIntegrityViolation		
Description	InstanceSpecifier either cannot be resolved in the model in the context of your executable, or it refers to a model element other than a PortPrototype. String format: "Violation detected in {processIdentifier} at {location}: Invalid InstanceSpecifer {instanceSpecifier} in a constructor of class: {className}"		
MessageId	0x80001ffc		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
processIdentifier	Identifier of the process that caused the violation.	uint8 [encoding UTF-8]	NoUnit
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	NoUnit
instanceSpecifier	InstanceSpecifier used to try to create the object.	uint8 [encoding UTF-8]	NoUnit
className	Name of the class that was instantiated.	uint8 [encoding UTF-8]	NoUnit

⌋

[SWS_CORE_13004] ViolationMessage PortInterfaceMappingViolation

Status: DRAFT

Upstream requirements: RS_AP_00142

⌈

Dlt-Message	PortInterfaceMappingViolation		
Description	The type of mapping does not match the expected type of PortInterface: {portInterfaceTypeName} referenced by a {mappingTypeName}. String format: "Violation detected in {processIdentifier} at {location}: Invalid InstanceSpecifer {instanceSpecifier} in a constructor of class: {className}"		
MessageId	0x80001ffb		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
processIdentifier	Identifier of the process that caused the violation.	uint8 [encoding UTF-8]	NoUnit
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	NoUnit
instanceSpecifier	InstanceSpecifier used to try to create the object.	uint8 [encoding UTF-8]	NoUnit
className	Name of the class that was instantiated.	uint8 [encoding UTF-8]	NoUnit

⌋

[SWS_CORE_13005] ViolationMessage ProcessMappingViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

Dlt-Message	ProcessMappingViolation		
Description	Matching InstanceRef exists, but no matching (modelled) Process found that matches the (runtime) process. String format: "Violation detected in {processIdentifier} at {location}: Invalid InstanceSpecifer {instanceSpecifier} in a constructor of class: {className}"		
MessageId	0x80001ffa		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
processIdentifier	Identifier of the process that caused the violation.	uint8 [encoding UTF-8]	NoUnit
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	NoUnit
instanceSpecifier	InstanceSpecifier used to try to create the object.	uint8 [encoding UTF-8]	NoUnit
className	Name of the class that was instantiated.	uint8 [encoding UTF-8]	NoUnit

⌋

[SWS_CORE_13006] ViolationMessage InstanceSpecifierAlreadyInUseViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

Dlt-Message	InstanceSpecifierAlreadyInUseViolation		
Description	Violation message that is sent in case a constructor in the ara framework was called with an Instance Specifier already in use in this process. String format: "Violation detected in {processIdentifier} at {location}: InstanceSpecifer {instanceSpecifier} in constructor of class {className} already in use in this process"		
MessageId	0x80001ff9		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
processIdentifier	Identifier of the process that caused the violation.	uint8 [encoding UTF-8]	NoUnit
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	NoUnit
instanceSpecifier	InstanceSpecifier used to try to create the object.	uint8 [encoding UTF-8]	NoUnit
className	Name of the class that was instantiated.	uint8 [encoding UTF-8]	NoUnit

⌋

Note: Functional Cluster implementation should synchronize calls to constructors or named constructors to assure correct detection of an instance specifier that is already in use.

[SWS_CORE_13007] ViolationMessage PlatformNotInitializedViolation

Status: DRAFT

Upstream requirements: RS_AP_00142

⌈

Dlt-Message	PlatformNotInitializedViolation		
Description	Violation message that is sent in case a constructor or function that takes an ara::core::InstanceSpecifier as an argument is called while the platform is not initialized. String format: "Violation detected in {processIdentifier} at {location}: Platform not initialized. The platform needs to be initialized before the execution of {functionName}."		
MessageId	0x80001ff8		
MessageTypeInfo	DLT_LOG_FATAL		
Dlt-Argument	**ArgumentDescription**	**ArgumentType**	**ArgumentUnit**
processIdentifier	Identifier of the process that caused the violation.	uint8 [encoding UTF-8]	NoUnit
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	NoUnit
instanceSpecifier	InstanceSpecifier used to try to create the object.	uint8 [encoding UTF-8]	NoUnit
functionName	Name of the function that was called to create the object.	uint8 [encoding UTF-8]	NoUnit

⌋

[SWS_CORE_13008] ViolationMessage WrongDomainOfAnErrorCodeViolation

Status: DRAFT

Upstream requirements: RS_AP_00142

⌈

Dlt-Message	WrongDomainOfAnErrorCodeViolation		
Description	Violation message that is sent in case an error code from a different error domain was passed to an error domain object for the error code's message retrieval. String format: "Violation detected in {processIdentifier} at {location}: The errorCode {errorCodeValue} from {errorCodeDomainName} did not originate from {errorDomainName}."		
MessageId	0x80001ff7		
MessageTypeInfo	DLT_LOG_FATAL		
Dlt-Argument	**ArgumentDescription**	**ArgumentType**	**ArgumentUnit**
processIdentifier	Identifier of the process that caused the violation.	uint8 [encoding UTF-8]	NoUnit
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	NoUnit
errorCodeValue	Domain-specific error code value of the passed error code.	uint8 [encoding UTF-8]	NoUnit
errorCodeDomainName	ErrorDomain associated with the passed error code.	uint8 [encoding UTF-8]	NoUnit
errorDomainName	The function calls ErrorDomain name.	uint8 [encoding UTF-8]	NoUnit

⌋

[SWS_CORE_13009] ViolationMessage ResultWithNoValueViolation

Status: DRAFT

Upstream requirements: RS_AP_00142

⌈

Dlt-Message	ResultWithNoValueViolation		
Description	Violation message that is sent in case value of a Result is accessed while there is no value contained in this Result object. String format: "Violation detected in {processIdentifier} at {location}: No value contained in this Result."		
MessageId	0x80001ff6		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	**ArgumentDescription**	**ArgumentType**	**ArgumentUnit**
processIdentifier	Identifier of the process that caused the violation.	uint8 [encoding UTF-8]	NoUnit
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	NoUnit

⌋

[SWS_CORE_13010] ViolationMessage ResultWithNoErrorViolation

Status: DRAFT

Upstream requirements: RS_AP_00142

⌈

Dlt-Message	ResultWithNoErrorViolation		
Description	Violation message that is sent in case error of a Result is accessed while there is no error contained in this Result object. String format: "Violation detected in {processIdentifier} at {location}: No error contained in this Result."		
MessageId	0x80001ff5		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	**ArgumentDescription**	**ArgumentType**	**ArgumentUnit**
processIdentifier	Identifier of the process that caused the violation.	uint8 [encoding UTF-8]	NoUnit
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	NoUnit

⌋

[SWS_CORE_13011] ViolationMessage FutureInvalidViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

Dlt-Message	FutureInvalidViolation		
Description	Violation message that is sent in case a prohibited function call was performed on an invalid Future object. String format: "Violation detected in {processIdentifier} at {location}: Calling {methodName} on an invalid Future is not allowed."		
MessageId	0x80001ff4		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
processIdentifier	Identifier of the process that caused the violation.	uint8 [encoding UTF-8]	NoUnit
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	NoUnit
methodName	Future class method name where the violation was detected.	uint8 [encoding UTF-8]	NoUnit

⌋

[SWS_CORE_13012] ViolationMessage FutureContinuationHasThrownViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

Dlt-Message	FutureContinuationHasThrownViolation		
Description	Violation message that is sent in case the continuation given to Future::then threw an exception. String format: "Violation detected in {processIdentifier} at {location}: The continuation given to Future::then threw an exception with the explanation: {exceptionExplanatoryString}."		
MessageId	0x80001ff3		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
processIdentifier	Identifier of the process that caused the violation.	uint8 [encoding UTF-8]	NoUnit
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	NoUnit
exception Explanatory String	Explanatory string of the exception if available.	uint8 [encoding UTF-8]	NoUnit

⌋

[SWS_CORE_13013] ViolationMessage FutureAlreadyRetrievedViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

Dlt-Message	FutureAlreadyRetrievedViolation		
Description	Violation message that is sent in case ara::core::Promise::get_future was called more than once on the same shared state. String format: "Violation detected in {processIdentifier} at {location}: The Future was already retrieved. The Future cannot be retrieved again."		
MessageId	0x80001ff2		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
processIdentifier	Identifier of the process that caused the violation.	uint8 [encoding UTF-8]	NoUnit
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	NoUnit

⌋

[SWS_CORE_13014] ViolationMessage PromiseWithNoSharedStateViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

Dlt-Message	PromiseWithNoSharedStateViolation		
Description	Violation message that is sent in case the Promise had no shared state when get_future was called to retrieve an associated Future. String format: "Violation detected in {processIdentifier} at {location}: The Future associated with this Promise cannot be retrieved, since it has no shared state."		
MessageId	0x80001ff1		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
processIdentifier	Identifier of the process that caused the violation.	uint8 [encoding UTF-8]	NoUnit
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	NoUnit

⌋

[SWS_CORE_13015] ViolationMessage PromiseAlreadySatisfiedViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

Dlt-Message	PromiseAlreadySatisfiedViolation		
Description	Violation message that is sent in case the shared state in a Promise already had a stored value or error and a function was called to set it again. String format: "Violation detected in {processIdentifier} at {location}: The Promise is already satisfied. The {targetField} cannot be set again."		
MessageId	0x80001ff0		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
processIdentifier	Identifier of the process that caused the violation.	uint8 [encoding UTF-8]	NoUnit
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	NoUnit
targetField	Target field to set in the Promise setter method call where the violation was detected. Allowed values are: "value", "error", "result".	uint8 [encoding UTF-8]	NoUnit

⌋

[SWS_CORE_13016] ViolationMessage PromiseNoSharedStateToSetViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

Dlt-Message	PromiseNoSharedStateToSetViolation		
Description	Violation message that is sent in case the Promise had no shared state when a function was called that expected the shared state to exist. String format: "Violation detected in {processIdentifier} at {location}: The {targetField} of this Promise cannot be set, since it has no shared state."		
MessageId	0x80001fef		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
processIdentifier	Identifier of the process that caused the violation.	uint8 [encoding UTF-8]	NoUnit
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	NoUnit
targetField	Target field to set in the Promise setter method call where the violation was detected. Allowed values are: "value", "error", "result".	uint8 [encoding UTF-8]	NoUnit

⌋

[SWS_CORE_13017] ViolationMessage ArrayAccessOutOfRangeViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

Dlt-Message	ArrayAccessOutOfRangeViolation		
Description	Violation message that is sent in case an index in an ara::core::Array::at call was not within the range of the array. String format: "Violation detected in {processIdentifier} at {location}: Array access out of range: Tried to access {indexValue} in array of size {arraySize}".		
MessageId	0x80001fee		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
processIdentifier	Identifier of the process that caused the violation.	uint8 [encoding UTF-8]	NoUnit
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	NoUnit
indexValue	Index value passed as input parameter	uint8 [encoding UTF-8]	NoUnit
arraySize	Array size	uint8 [encoding UTF-8]	NoUnit

⌋

[SWS_CORE_13020] ViolationMessage NoValueInOptionalViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

Dlt-Message	NoValueInOptionalViolation		
Description	Violation message that is sent in case if an accessor method was called on an Optional that does not contain a value. String format: "Violation detected in {processIdentifier} at {location}: No value contained in this Optional."		
MessageId	0x80001fed		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
processIdentifier	Identifier of the process that caused the violation.	uint8 [encoding UTF-8]	NoUnit
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	NoUnit

⌋

7.8.4 Production Errors

This functional cluster does not define any production errors (i.e., Diagnostic Events).

8 API specification

This chapter provides a reference of the APIs defined by this functional cluster. The API is described in the following chapters in tables. Table 8.1 explains the content that is described in such an API table.

Kind:	Defines the kind of the declaration that this API table describes. The following values are supported: • class (Declaration of a class) • function (Declaration of a member or non-member function) • struct (Declaration of a structure) • type alias (Declaration of a type alias) • enumeration (Declaration of an enumeration) • variable (Declaration of a variable)	
Header File:	Defines the header file to be included according to [SWS_CORE_90001]	
Forwarding Header File:	Defines the forwarding header file to be included according to [SWS_CORE_90001]	
Scope:	Defines the scope that may be a namespace (in case of a class or non-member function) or a class declaration (in case of a member)	
Symbol:	Entity name	
Thread Safety:	Defines whether a function is thread-safe, not thread-safe, or conditional according to [SWS_CORE_13200] and [SWS_CORE_13202]	
Syntax:	Description of C++ syntax	
Template Param:	Template parameter (0..*)	Template parameter(s) used to parametrize the template
Parameters (in):	Parameter declaration (0..*)	Parameter(s) that are passed to the function
Parameters (out):	Parameter declaration (0..*)	Parameter(s) that are returned to the caller
Return Value:	Return type	Type of the value that the function returns
Exception Safety:	Defines whether a function is exception-safe, not exception safe or conditionally exception safe	
Exceptions:	List of exceptions that may be thrown from the function	
Violations:	List of violations that may occur in the function	
Errors:	Error type (0..*)	List of defined error codes that may be returned by the function with their recoverability class defined in [RS_AP_00160]. APIs can be extended with vendor-specific error codes. These are not part of the AUTOSAR SWS specifications
Description:	Brief description of the function	

Table 8.1: Explanation of an API table

All symbols described in this chapter have `public` visibility unless otherwise noted.

8.1 Header: ara/core/error_domain.h

8.1.1 Class: ErrorDomain

[SWS_CORE_00110] Definition of API class ara::core::ErrorDomain

Upstream requirements: RS_AP_00130

⌈

Kind:	class
Header file:	#include "ara/core/error_domain.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	`namespace ara::core`
Symbol:	ErrorDomain
Syntax:	`class ErrorDomain {...};`
Description:	Encapsulation of an error domain. An error domain is the controlling entity for ErrorCode's error code values, and defines the mapping of such error code values to textual representations. This type constitutes a base class for error domain implementations. This class is a literal type, and subclasses are strongly advised to be literal types as well.

⌋

8.1.1.1 Public Member Types

8.1.1.1.1 Type Alias: CodeType

[SWS_CORE_00122] Definition of API type ara::core::ErrorDomain::CodeType

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/error_domain.h"
Scope:	`class ara::core::ErrorDomain`
Symbol:	CodeType
Syntax:	`using CodeType = std::int32_t;`
Description:	Alias type for a domain-specific error code value .

⌋

8.1.1.1.2 Type Alias: IdType

[SWS_CORE_00121] Definition of API type ara::core::ErrorDomain::IdType

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/error_domain.h"
Scope:	`class ara::core::ErrorDomain`
Symbol:	IdType
Syntax:	`using IdType = std::uint64_t;`
Description:	Alias type for a unique ErrorDomain identifier type .

⌋

8.1.1.1.3 Type Alias: SupportDataType

[SWS_CORE_00123] Definition of API type ara::core::ErrorDomain::SupportDataType

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/error_domain.h"
Scope:	`class ara::core::ErrorDomain`
Symbol:	SupportDataType
Syntax:	`using SupportDataType = <implementation-defined>;`
Description:	Alias type for vendor-specific supplementary data .

⌋

8.1.1.2 Public Member Functions

8.1.1.2.1 Special Member Functions

8.1.1.2.1.1 Copy Constructor

[SWS_CORE_00131] Definition of API function ara::core::ErrorDomain::ErrorDomain

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/error_domain.h"
Scope:	`class ara::core::ErrorDomain`
Syntax:	`ErrorDomain (const ErrorDomain &)=delete;`
Description:	Copy construction shall be disabled.

⌋

8.1.1.2.1.2 Move Constructor

[SWS_CORE_00132] Definition of API function ara::core::ErrorDomain::ErrorDomain

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/error_domain.h"
Scope:	`class ara::core::ErrorDomain`
Syntax:	`ErrorDomain (ErrorDomain &&)=delete;`
Description:	Move construction shall be disabled.

⌋

8.1.1.2.1.3 Move Assignment Operator

[SWS_CORE_00134] Definition of API function ara::core::ErrorDomain::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/error_domain.h"
Scope:	`class ara::core::ErrorDomain`
Syntax:	`ErrorDomain & operator= (ErrorDomain &&)=delete;`
Description:	Move assignment shall be disabled.

⌋

8.1.1.2.1.4 Copy Assignment Operator

[SWS_CORE_00133] Definition of API function ara::core::ErrorDomain::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/error_domain.h"
Scope:	`class ara::core::ErrorDomain`
Syntax:	`ErrorDomain & operator= (const ErrorDomain &)=delete;`
Description:	Copy assignment shall be disabled.

⌋

8.1.1.2.2 Member Functions

8.1.1.2.2.1 Id

[SWS_CORE_00151] Definition of API function ara::core::ErrorDomain::Id

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/error_domain.h"	
Scope:	`class ara::core::ErrorDomain`	
Syntax:	`constexpr IdType Id () const noexcept;`	
Return value:	IdType	the identifier
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return the unique domain identifier.	

⌋

8.1.1.2.2.2 Message

[SWS_CORE_00153] Definition of API function ara::core::ErrorDomain::Message

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/error_domain.h"	
Scope:	`class ara::core::ErrorDomain`	
Syntax:	`virtual const char * Message (CodeType errorCode) const noexcept=0;`	
Parameters (in):	errorCode	the domain-specific error code
Return value:	const char *	the text as a null-terminated string, never nullptr
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`WrongDomainOfAnErrorCodeViolation`	If the errorCode did not originate from this error domain.
Description:	Return a textual representation of the given error code. The returned pointer remains owned by the ErrorDomain subclass and shall not be freed by clients.	

⌋

8.1.1.2.2.3 Name

[SWS_CORE_00152] Definition of API function ara::core::ErrorDomain::Name

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/error_domain.h"	
Scope:	`class ara::core::ErrorDomain`	
Syntax:	`virtual const char * Name () const noexcept=0;`	
Return value:	const char *	the name as a null-terminated string, never nullptr
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return the name of this error domain. The returned pointer remains owned by class ErrorDomain and shall not be freed by clients.	

⌋

8.1.1.2.2.4 ThrowAsException

[SWS_CORE_00154] Definition of API function ara::core::ErrorDomain::ThrowAsException

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/error_domain.h"	
Scope:	`class ara::core::ErrorDomain`	
Syntax:	`virtual void ThrowAsException (const ErrorCode &errorCode) const noexcept(false)=0;`	
Parameters (in):	errorCode	the ErrorCode
Return value:	None	
Exceptions:	<TYPE>	an exception of the type as defined in [SWS_CORE_10953] containing the given `ErrorCode`
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Throw the given error as exception. This function will determine the appropriate exception type for the given `ErrorCode` according to [SWS_CORE_10953] and throw it. The thrown exception will contain the given `ErrorCode`. As per [SWS_CORE_10304], this function does not participate in overload resolution when C++ exceptions are disabled in the compiler toolchain.	

⌋

8.1.1.2.2.5 operator!=

[SWS_CORE_00138] Definition of API function ara::core::ErrorDomain::operator!=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/error_domain.h"	
Scope:	`class ara::core::ErrorDomain`	
Syntax:	`constexpr bool operator!= (const ErrorDomain &other) const noexcept;`	
Parameters (in):	other	the other instance
Return value:	bool	true if other is not equal to *this, false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Compare for non-equality with another ErrorDomain instance.	

⌋

8.1.1.2.2.6 operator==

[SWS_CORE_00137] Definition of API function ara::core::ErrorDomain::operator==

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/error_domain.h"	
Scope:	`class ara::core::ErrorDomain`	
Syntax:	`constexpr bool operator== (const ErrorDomain &other) const noexcept;`	
Parameters (in):	other	the other instance
Return value:	bool	true if other is equal to *this, false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Compare for equality with another ErrorDomain instance. Two ErrorDomain instances compare equal when their identifiers (returned by Id()) are equal.	

⌋

8.1.1.3 Protected Member Functions

8.1.1.3.1 Special Member Functions

8.1.1.3.1.1 Destructor

[SWS_CORE_00136] Definition of API function ara::core::ErrorDomain::~ErrorDomain

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/error_domain.h"
Scope:	`class ara::core::ErrorDomain`
Syntax:	`~ErrorDomain () noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	Destructor. This dtor is non-virtual (and trivial) so that this class can be a literal type. While this class has virtual functions, no polymorphic destruction is needed.
Visibility:	protected

⌋

8.1.1.3.2 Constructors

8.1.1.3.2.1 ErrorDomain

[SWS_CORE_00135] Definition of API function ara::core::ErrorDomain::ErrorDomain

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/error_domain.h"	
Scope:	`class ara::core::ErrorDomain`	
Syntax:	`explicit constexpr ErrorDomain (IdType id) noexcept;`	
Parameters (in):	id	the unique identifier
Exception Safety:	exception safe	
Thread Safety:	thread-safe	

▽

△

Description:	Construct a new instance with the given identifier. Identifiers are expected to be system-wide unique.
Visibility:	protected

⌋

8.2 Header: ara/core/error_code.h

8.2.1 Non-Member Functions

8.2.1.1 Other

8.2.1.1.1 operator!=

[SWS_CORE_00572] Definition of API function ara::core::operator!=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/error_code.h"	
Scope:	`namespace ara::core`	
Syntax:	`constexpr bool operator!= (const ErrorCode &lhs, const ErrorCode &rhs) noexcept;`	
Parameters (in):	lhs	the left hand side of the comparison
	rhs	the right hand side of the comparison
Return value:	bool	true if the two instances compare not equal, false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Non-member operator!= for ErrorCode. Two ErrorCode instances compare equal if the results of their Value() and Domain() functions are equal. The result of SupportData() is not considered for equality.	

⌋

8.2.1.1.2 operator==

[SWS_CORE_00571] Definition of API function ara::core::operator==

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/error_code.h"	
Scope:	`namespace ara::core`	
Syntax:	`constexpr bool operator== (const ErrorCode &lhs, const ErrorCode &rhs) noexcept;`	
Parameters (in):	lhs	the left hand side of the comparison
	rhs	the right hand side of the comparison
Return value:	bool	true if the two instances compare equal, false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Non-member operator== for ErrorCode. Two ErrorCode instances compare equal if the results of their Value() and Domain() functions are equal. The result of SupportData() is not considered for equality.	

⌋

8.2.2 Class: ErrorCode

[SWS_CORE_00501] Definition of API class ara::core::ErrorCode

Upstream requirements: RS_AP_00130, RS_AP_00140

⌈

Kind:	class
Header file:	#include "ara/core/error_code.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	`namespace ara::core`
Symbol:	ErrorCode
Syntax:	`class ErrorCode final {...};`
Description:	Encapsulation of an error code. An ErrorCode contains a raw error code value and an error domain. The raw error code value is specific to this error domain.

⌋

8.2.2.1 Public Member Functions

8.2.2.1.1 Constructors

8.2.2.1.1.1 ErrorCode

[SWS_CORE_00512] Definition of API function ara::core::ErrorCode::ErrorCode

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/error_code.h"	
Scope:	`class ara::core::ErrorCode`	

Syntax:	`template <typename EnumT>`
	`constexpr ErrorCode (EnumT e, ErrorDomain::SupportDataType data=ErrorDomain::SupportDataType()) noexcept;`	

Template param:	EnumT	an enum type that contains error code values
Parameters (in):	e	a domain-specific error code value
	data	optional vendor-specific supplementary error context data
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Construct a new ErrorCode instance with parameters. This constructor does not participate in overload resolution unless EnumT is an enum type.	

⌋

8.2.2.1.1.2 ErrorCode

[SWS_CORE_00513] Definition of API function ara::core::ErrorCode::ErrorCode

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/error_code.h"	
Scope:	`class ara::core::ErrorCode`	
Syntax:	`constexpr ErrorCode (ErrorDomain::CodeType value, const ErrorDomain &domain, ErrorDomain::SupportDataType data=ErrorDomain::SupportDataType()) noexcept;`	
Parameters (in):	value	a domain-specific error code value
	domain	the ErrorDomain associated with value
	data	optional vendor-specific supplementary error context data
Exception Safety:	exception safe	
Thread Safety:	implementation defined	

▽

△

Description:	Construct a new ErrorCode instance with parameters.

⌋

8.2.2.1.2 Member Functions

8.2.2.1.2.1 Domain

[SWS_CORE_00515] Definition of API function ara::core::ErrorCode::Domain
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/error_code.h"	
Scope:	`class ara::core::ErrorCode`	
Syntax:	`constexpr const ErrorDomain & Domain () const noexcept;`	
Return value:	const ErrorDomain &	the ErrorDomain
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return the domain with which this ErrorCode is associated.	

⌋

8.2.2.1.2.2 Message

[SWS_CORE_00518] Definition of API function ara::core::ErrorCode::Message
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/error_code.h"	
Scope:	`class ara::core::ErrorCode`	
Syntax:	`StringView Message () const noexcept;`	
Return value:	StringView	the error message text
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a textual representation of this ErrorCode.	

⌋

8.2.2.1.2.3 SupportData

[SWS_CORE_00516] Definition of API function ara::core::ErrorCode::SupportData

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/error_code.h"	
Scope:	`class ara::core::ErrorCode`	
Syntax:	`constexpr ErrorDomain::SupportDataType SupportData () const noexcept;`	
Return value:	ErrorDomain::SupportDataType	the supplementary error context data
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return the supplementary error context data. The underlying type and the meaning of the returned value are implementation-defined.	

⌋

8.2.2.1.2.4 ThrowAsException

[SWS_CORE_00519] Definition of API function ara::core::ErrorCode::ThrowAsException

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/error_code.h"	
Scope:	`class ara::core::ErrorCode`	
Syntax:	`void ThrowAsException () const noexcept(false);`	
Return value:	None	
Exceptions:	<TYPE>	an exception of the type determined by the associated `ErrorDomain` as defined in [SWS_CORE_10953]
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Throw this error as exception. This function will determine the appropriate exception type for this `ErrorCode` and throw it. The thrown exception will contain this `ErrorCode`. Behaves as if `this->Domain().ThrowAsException(*this)`. This function shall not participate in overload resolution when C++ exceptions are disabled in the compiler toolchain.	

⌋

8.2.2.1.2.5 Value

[SWS_CORE_00514] Definition of API function ara::core::ErrorCode::Value

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/error_code.h"	
Scope:	`class ara::core::ErrorCode`	
Syntax:	`constexpr ErrorDomain::CodeType Value () const noexcept;`	
Return value:	ErrorDomain::CodeType	the raw error code value
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return the raw error code value.	

⌋

8.3 Header: ara/core/exception.h

8.3.1 Class: Exception

[SWS_CORE_00601] Definition of API class ara::core::Exception

Upstream requirements: RS_AP_00130

⌈

Kind:	class
Header file:	#include "ara/core/exception.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	`namespace ara::core`
Symbol:	Exception
Base class:	std::exception
Syntax:	`class Exception :  public exception {...};`
Description:	Base type for all exception types defined by the Adaptive Platform.

⌋

8.3.1.1 Public Member Functions

8.3.1.1.1 Special Member Functions

8.3.1.1.1.1 Move Constructor

[SWS_CORE_00615] Definition of API function ara::core::Exception::Exception

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/exception.h"	
Scope:	`class ara::core::Exception`	
Syntax:	`Exception (Exception &&other) noexcept=default;`	
Parameters (in):	other	the other instance
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Move constructor from another instance.	

⌋

8.3.1.1.1.2 Move Assignment Operator

[SWS_CORE_00616] Definition of API function ara::core::Exception::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/exception.h"	
Scope:	`class ara::core::Exception`	
Syntax:	`Exception & operator= (Exception &&other) & noexcept=default;`	
Parameters (in):	other	the other instance
Return value:	Exception &	*this
Exception Safety:	exception safe	
Thread Safety:	non_threadsafe	
Description:	Move assignment operator from another instance.	

⌋

8.3.1.1.1.3 Destructor

[SWS_CORE_00617] Definition of API function ara::core::Exception::~Exception

Upstream requirements: RS_AP_00130, RS_AP_00145

⌈

Kind:	function	
Header file:	#include "ara/core/exception.h"	
Scope:	`class ara::core::Exception`	
Syntax:	`virtual ~Exception () noexcept override=default;`	
Exception Safety:	exception safe	
Thread Safety:	non_threadsafe	
Description:	Destructs the Exception object.	

⌋

8.3.1.1.2 Constructors

8.3.1.1.2.1 Exception

[SWS_CORE_00611] Definition of API function ara::core::Exception::Exception

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/exception.h"	
Scope:	`class ara::core::Exception`	
Syntax:	`explicit Exception (ErrorCode err) noexcept;`	
Parameters (in):	err	the ErrorCode
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Construct a new Exception object with a specific ErrorCode.	

⌋

8.3.1.1.3 Member Functions

8.3.1.1.3.1 Error

[SWS_CORE_00613] Definition of API function ara::core::Exception::Error

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/exception.h"	
Scope:	`class ara::core::Exception`	
Syntax:	`const ErrorCode & Error () const noexcept;`	
Return value:	const ErrorCode &	reference to the embedded ErrorCode
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return the embedded ErrorCode that was given to the constructor.	

⌋

8.3.1.1.3.2 what

[SWS_CORE_00612] Definition of API function ara::core::Exception::what

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/exception.h"	
Scope:	`class ara::core::Exception`	
Syntax:	`const char * what () const noexcept override;`	
Return value:	const char *	a null-terminated string
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return the explanatory string. This function overrides the virtual function std::exception::what. All guarantees about the lifetime of the returned pointer that are given for std::exception::what are preserved.	

⌋

8.3.1.2 Protected Member Functions

8.3.1.2.1 Special Member Functions

8.3.1.2.1.1 Copy Constructor

[SWS_CORE_00618] Definition of API function ara::core::Exception::Exception

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/exception.h"	
Scope:	`class ara::core::Exception`	
Syntax:	`Exception (const Exception &other) noexcept=default;`	
Parameters (in):	other	the other instance
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Copy constructor from another instance. This function is "protected" in order to prevent some opportunities for accidental object slicing.	
Visibility:	protected	

⌋

8.3.1.2.1.2 Copy Assignment Operator

[SWS_CORE_00614] Definition of API function ara::core::Exception::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/exception.h"	
Scope:	`class ara::core::Exception`	
Syntax:	`Exception & operator= (const Exception &other) noexcept=default;`	
Parameters (in):	other	the other instance
Return value:	Exception &	*this
Exception Safety:	exception safe	
Thread Safety:	non_threadsafe	
Description:	Copy assignment operator from another instance. This function is "protected" in order to prevent some opportunities for accidental object slicing.	
Visibility:	protected	

⌋

8.4 Header: ara/core/result.h

8.4.1 Non-Member Functions

8.4.1.1 Other

8.4.1.1.1 operator!=

[SWS_CORE_00788] Definition of API function ara::core::operator!=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T, typename E>`
	`bool operator!= (const Result< T, E > &lhs, const E &rhs);`	

Parameters (in):	lhs	the Result instance
	rhs	the error to compare with
Return value:	bool	true if the Result's error compares unequal to the rhs error, false otherwise
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Compare a Result instance for inequality to an error. A Result that contains no error is unequal to every error. A Result is equal to an error only if the Result contains an error of the same type, and the errors compare equal.	

⌋

8.4.1.1.2 operator!=

[SWS_CORE_00781] Definition of API function ara::core::operator!=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T, typename E>`
	`bool operator!= (const Result< T, E > &lhs, const Result< T, E > &rhs);`	

Parameters (in):	lhs	the left hand side of the comparison
	rhs	the right hand side of the comparison

▽

△

Return value:	bool	true if the two instances compare unequal, false otherwise
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Compare two Result instances for inequality. A Result that contains a value is unequal to every Result containing an error. A Result containing a value is equal to another Result only if both contain the same type, and either that type is `void` or the value of that type compares equal. A Result containing an error is equal to another Result only if both contain the same error type, and the contained errors compare equal.	
See also:	[SWS_CORE_00780]	

⌋

8.4.1.1.3 operator!=

[SWS_CORE_00785] Definition of API function ara::core::operator!=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T, typename E>`
	`bool operator!= (const T &lhs, const Result< T, E > &rhs);`	

Parameters (in):	lhs	the value to compare with
	rhs	the Result instance
Return value:	bool	true if the Result's value compares unequal to the lhs value, false otherwise
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Compare a Result instance for inequality to a value. A Result that contains no value is unequal to every value. A Result is equal to a value only if the Result contains a value of the same type, and the values compare equal.	

⌋

8.4.1.1.4 operator!=

[SWS_CORE_00789] Definition of API function ara::core::operator!=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T, typename E>`
	`bool operator!= (const E &lhs, const Result< T, E > &rhs);`	

Parameters (in):	lhs	the error to compare with
	rhs	the Result instance
Return value:	bool	true if the Result's error compares unequal to the lhs error, false otherwise
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Compare a Result instance for inequality to an error. A Result that contains no error is unequal to every error. A Result is equal to an error only if the Result contains an error of the same type, and the errors compare equal.	

⌋

8.4.1.1.5 operator!=

[SWS_CORE_00784] Definition of API function ara::core::operator!=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T, typename E>`
	`bool operator!= (const Result< T, E > &lhs, const T &rhs);`	

Parameters (in):	lhs	the Result instance
	rhs	the value to compare with
Return value:	bool	true if the Result's value compares unequal to the rhs value, false otherwise
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Compare a Result instance for inequality to a value. A Result that contains no value is unequal to every value. A Result is equal to a value only if the Result contains a value of the same type, and the values compare equal.	

⌋

8.4.1.1.6 operator==

[SWS_CORE_00782] Definition of API function ara::core::operator==

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T, typename E>`
	`bool operator== (const Result< T, E > &lhs, const T &rhs);`	

Parameters (in):	lhs	the Result instance
	rhs	the value to compare with
Return value:	bool	true if the Result's value compares equal to the rhs value, false otherwise
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Compare a Result instance for equality to a value. A Result that contains no value is unequal to every value. A Result is equal to a value only if the Result contains a value of the same type, and the values compare equal.	

⌋

8.4.1.1.7 operator==

[SWS_CORE_00786] Definition of API function ara::core::operator==

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T, typename E>`
	`bool operator== (const Result< T, E > &lhs, const E &rhs);`	

Parameters (in):	lhs	the Result instance
	rhs	the error to compare with
Return value:	bool	true if the Result's error compares equal to the rhs error, false otherwise
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Compare a Result instance for equality to an error. A Result that contains no error is unequal to every error. A Result is equal to an error only if the Result contains an error of the same type, and the errors compare equal.	

⌋

8.4.1.1.8 operator==

[SWS_CORE_00787] Definition of API function ara::core::operator==

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T, typename E>`
	`bool operator== (const E &lhs, const Result< T, E > &rhs);`	

Parameters (in):	lhs	the error to compare with
	rhs	the Result instance
Return value:	bool	true if the Result's error compares equal to the lhs error, false otherwise
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Compare a Result instance for equality to an error. A Result that contains no error is unequal to every error. A Result is equal to an error only if the Result contains an error of the same type, and the errors compare equal.	

⌋

8.4.1.1.9 operator==

[SWS_CORE_00780] Definition of API function ara::core::operator==

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T, typename E>`
	`bool operator== (const Result< T, E > &lhs, const Result< T, E > &rhs);`	

Parameters (in):	lhs	the left hand side of the comparison
	rhs	the right hand side of the comparison
Return value:	bool	true if the two instances compare equal, false otherwise
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	

▽

△

Description:	Compare two Result instances for equality. A Result that contains a value is unequal to every Result containing an error. A Result containing a value is equal to another Result only if both contain the same type, and either that type is `void` or the value of that type compares equal. A Result containing an error is equal to another Result only if both contain the same error type, and the contained errors compare equal.

⌋

8.4.1.1.10 operator==

[SWS_CORE_00783] Definition of API function ara::core::operator==

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T, typename E>`
	`bool operator== (const T &lhs, const Result< T, E > &rhs);`	

Parameters (in):	lhs	the value to compare with
	rhs	the Result instance
Return value:	bool	true if the Result's value compares equal to the lhs value, false otherwise
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Compare a Result instance for equality to a value. A Result that contains no value is unequal to every value. A Result is equal to a value only if the Result contains a value of the same type, and the values compare equal.	

⌋

8.4.1.1.11 swap

[SWS_CORE_00796] Definition of API function ara::core::swap

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/result.h"
Scope:	`namespace ara::core`

▽

△

Syntax:	`template <typename T, typename E>`
	`void swap (Result< T, E > &lhs, Result< T, E > &rhs)`
	`noexcept(noexcept(lhs.Swap(rhs)));`	

Parameters (in):	lhs	one instance
	rhs	another instance
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	Swap the contents of the two given arguments	

⌋

8.4.2 Class: Result

[SWS_CORE_00701] Definition of API class ara::core::Result

Upstream requirements: RS_AP_00130

⌈

Kind:	class	
Header file:	#include "ara/core/result.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace ara::core`	
Symbol:	Result	

Syntax:	`template <typename T, typename E = ErrorCode>`
	`class Result final {...};`	

Template param:	typename T	the type of value
	typename E = ErrorCode	the type of error
Description:	This class is a type that contains either a value or an error. The implementation shall flag the condition `E != ara::core::ErrorCode` as a compile error.	

⌋

8.4.2.1 Public Member Types

8.4.2.1.1 Type Alias: error_type

[SWS_CORE_00712] Definition of API type ara::core::Result::error_type

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/result.h"
Scope:	`class ara::core::Result`
Symbol:	error_type
Syntax:	`using error_type = E;`
Description:	Type alias for the type E of errors .

⌋

8.4.2.1.2 Type Alias: value_type

[SWS_CORE_00711] Definition of API type ara::core::Result::value_type

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/result.h"
Scope:	`class ara::core::Result`
Symbol:	value_type
Syntax:	`using value_type = T;`
Description:	Type alias for the type T of values .

⌋

8.4.2.2 Public Member Functions

8.4.2.2.1 Special Member Functions

8.4.2.2.1.1 Move Constructor

[SWS_CORE_00726] Definition of API function ara::core::Result::Result

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`Result (Result &&other) noexcept(std::is_nothrow_move_constructible< T >::value &&std::is_nothrow_move_constructible< E >::value);`	
Parameters (in):	other	the other instance
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	Move-construct a new Result from another instance.	

⌋

8.4.2.2.1.2 Copy Constructor

[SWS_CORE_00725] Definition of API function ara::core::Result::Result

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`Result (const Result &other) noexcept(std::is_nothrow_copy_constructible< T >::value &&std::is_nothrow_copy_constructible< E >::value);`	
Parameters (in):	other	the other instance
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	Copy-construct a new Result from another instance.	

⌋

8.4.2.2.1.3 Move Assignment Operator

[SWS_CORE_00742] Definition of API function ara::core::Result::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`Result & operator= (Result &&other) noexcept(std::is_nothrow_move_constructible< T >::value &&std::is_nothrow_move_assignable< T >::value &&std::is_nothrow_move_constructible< E >::value &&std::is_nothrow_move_assignable< E >::value);`	
Parameters (in):	other	the other instance
Return value:	Result &	*this, containing the contents of other
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	Move-assign another Result to this instance.	

⌋

8.4.2.2.1.4 Copy Assignment Operator

[SWS_CORE_00741] Definition of API function ara::core::Result::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`Result & operator= (const Result &other) noexcept(std::is_nothrow_copy_constructible< T >::value &&std::is_nothrow_copy_assignable< T >::value &&std::is_nothrow_copy_constructible< E >::value &&std::is_nothrow_copy_assignable< E >::value);`	
Parameters (in):	other	the other instance
Return value:	Result &	*this, containing the contents of other
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	Copy-assign another Result to this instance.	

⌋

8.4.2.2.1.5 Destructor

[SWS_CORE_00727] Definition of API function ara::core::Result::~Result

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/result.h"
Scope:	`class ara::core::Result`
Syntax:	`~Result () noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	Destructor. This destructor is trivial if std::is_trivially_destructible<T>::value && std::is_trivially_destructible<E>::value is true.

⌋

8.4.2.2.2 Constructors

8.4.2.2.2.1 Result

[SWS_CORE_00724] Definition of API function ara::core::Result::Result

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`explicit Result (E &&e) noexcept(std::is_nothrow_move_constructible< E >::value);`	
Parameters (in):	e	the error to put into the Result
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	Construct a new Result from the specified error (given as rvalue).	

⌋

8.4.2.2.2.2 Result

[SWS_CORE_00722] Definition of API function ara::core::Result::Result

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`Result (T &&t) noexcept(std::is_nothrow_move_constructible< T >::value);`	
Parameters (in):	t	the value to put into the Result
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	Construct a new Result from the specified value (given as rvalue).	

⌋

8.4.2.2.2.3 Result

[SWS_CORE_00721] Definition of API function ara::core::Result::Result

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`Result (const T &t) noexcept(std::is_nothrow_copy_constructible< T >::value);`	
Parameters (in):	t	the value to put into the Result
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	Construct a new Result from the specified value (given as lvalue).	

⌋

8.4.2.2.2.4 Result

[SWS_CORE_00723] Definition of API function ara::core::Result::Result

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`explicit Result (const E &e) noexcept(std::is_nothrow_copy_constructible< E >::value);`	
Parameters (in):	e	the error to put into the Result
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	Construct a new Result from the specified error (given as lvalue).	

⌋

8.4.2.2.3 Member Functions

8.4.2.2.3.1 Bind

[SWS_CORE_00768] Definition of API function ara::core::Result::Bind

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	

Syntax:	`template <typename F>`
	`auto Bind (F &&f) const -> <see below>;`	

Template param:	F	the type of the Callable f
Parameters (in):	f	the Callable
Return value:	*<see below>*	a new Result instance of the possibly transformed type
Exception Safety:	not exception safe	
Thread Safety:	conditional,	depends on thread safety of f

▽

△

Description:	Apply the given Callable to the value of this instance, and return a new Result with the result of the call. The Callable is expected to be compatible to one of these two interfaces: • `Result<XXX, E> f(const T&);` • `XXX f(const T&);` meaning that the Callable either returns a Result<XXX> or a XXX directly, where XXX can be any type that is suitable for use by class Result. The return type of this function is `decltype(f(Value()))` for a template argument F that returns a Result type, and it is `Result<decltype(f(Value())), E>` for a template argument F that does not return a Result type. If this instance does not contain a value, a new Result<XXX, E> is still created and returned, with the original error contents of this instance being copied into the new instance.

⌋

8.4.2.2.3.2 CheckError

[SWS_CORE_00765] Definition of API function ara::core::Result::CheckError

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	

Syntax:	`template <typename G>`
	`bool CheckError (G &&error) const noexcept;`	

Template param:	G	the type of the error argument error
Parameters (in):	error	the error to check
Return value:	bool	true if *this contains an error that is equivalent to the given error, false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return whether this instance contains the given error. This call compares the argument error, static_cast'd to E, with the return value from Error().	

⌋

8.4.2.2.3.3 EmplaceError

[SWS_CORE_00744] Definition of API function ara::core::Result::EmplaceError

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	

Syntax:	`template <typename...  Args>`
	`void EmplaceError (Args &&...  args) noexcept(std::is_nothrow_`
	`constructible< E, Args...  >::value);`	

Template param:	Args...	the types of arguments given to this function
Parameters (in):	args	the arguments used for constructing the error
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	Put a new error into this instance, constructed in-place from the given arguments.	

⌋

8.4.2.2.3.4 EmplaceValue

[SWS_CORE_00743] Definition of API function ara::core::Result::EmplaceValue

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	

Syntax:	`template <typename...  Args>`
	`void EmplaceValue (Args &&...  args) noexcept(std::is_nothrow_`
	`constructible< T, Args...  >::value);`	

Template param:	Args...	the types of arguments given to this function
Parameters (in):	args	the arguments used for constructing the value
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	Put a new value into this instance, constructed in-place from the given arguments.	

⌋

8.4.2.2.3.5 Err

[SWS_CORE_00773] Definition of API function ara::core::Result::Err

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`Optional< E > Err () &&noexcept(std::is_nothrow_constructible< Optional< E >, E && >::value);`	
Return value:	Optional< E >	an Optional with the error, if present
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return the contained error as an Optional.	

⌋

8.4.2.2.3.6 Err

[SWS_CORE_00772] Definition of API function ara::core::Result::Err

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`Optional< E > Err () const &noexcept(std::is_nothrow_constructible< Optional< E >, const E &>::value);`	
Return value:	Optional< E >	an Optional with the error, if present
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return the contained error as an Optional.	

⌋

8.4.2.2.3.7 Error

[SWS_CORE_00758] Definition of API function ara::core::Result::Error

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`E && Error () &&noexcept;`	
Return value:	E &&	an rvalue reference to the contained error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`ResultWithNoErrorViolation`	If *this does not contain an error.
Description:	Access the contained error.	

⌋

8.4.2.2.3.8 Error

[SWS_CORE_00776] Definition of API function ara::core::Result::Error

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`E & Error () & noexcept;`	
Return value:	E &	a reference to the contained error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`ResultWithNoErrorViolation`	If *this does not contain an error.
Description:	Access the contained error.	

⌋

8.4.2.2.3.9 Error

[SWS_CORE_00757] Definition of API function ara::core::Result::Error

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`const E & Error () const & noexcept;`	
Return value:	const E &	a const reference to the contained error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`ResultWithNoErrorViolation`	If *this does not contain an error.
Description:	Access the contained error.	

⌋

8.4.2.2.3.10 ErrorOr

[SWS_CORE_00764] Definition of API function ara::core::Result::ErrorOr

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	

Syntax:	`template <typename G>`
	`E ErrorOr (G &&defaultError) &&noexcept(std::is_nothrow_move_constructible< E >::value &&std::is_nothrow_constructible< E, G && >::value);`	

Template param:	G	the type of defaultError
Parameters (in):	defaultError	the error to use if *this does not contain an error
Return value:	E	the error
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return the contained error or the given default error. If *this contains an error, it is std::move'd into the return value. Otherwise, the specified default error is returned, static_cast'd to E.	

⌋

8.4.2.2.3.11 ErrorOr

[SWS_CORE_00763] Definition of API function ara::core::Result::ErrorOr

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	

Syntax:	`template <typename G>`
	`E ErrorOr (G &&defaultError) const &noexcept(std::is_nothrow_copy_constructible< E >::value &&std::is_nothrow_constructible< E, G &&>::value);`	

Template param:	G	the type of defaultError
Parameters (in):	defaultError	the error to use if *this does not contain an error
Return value:	E	the error
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return the contained error or the given default error. If *this contains an error, it is returned. Otherwise, the specified default error is returned, static_cast'd to E.	

⌋

8.4.2.2.3.12 HasValue

[SWS_CORE_00751] Definition of API function ara::core::Result::HasValue

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`bool HasValue () const noexcept;`	
Return value:	bool	true if *this contains a value, false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Check whether *this contains a value.	

⌋

8.4.2.2.3.13 Ok

[SWS_CORE_00771] Definition of API function ara::core::Result::Ok

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`Optional< T > Ok () &&noexcept(std::is_nothrow_constructible< Optional< T >, T && >::value);`	
Return value:	Optional< T >	an Optional with the value, if present
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return the contained value as an Optional.	

⌋

8.4.2.2.3.14 Ok

[SWS_CORE_00770] Definition of API function ara::core::Result::Ok

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`Optional< T > Ok () const &noexcept(std::is_nothrow_constructible< Optional< T >, const T &>::value);`	
Return value:	Optional< T >	an Optional with the value, if present
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return the contained value as an Optional.	

⌋

8.4.2.2.3.15 Resolve

[SWS_CORE_00767] Definition of API function ara::core::Result::Resolve

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	

Syntax:	`template <typename F>`
	`T Resolve (F &&f) const;`	

Template param:	F	the type of the Callable f
Parameters (in):	f	the Callable
Return value:	T	the value
Exception Safety:	not exception safe	
Thread Safety:	conditional,	depends on thread safety of f
Description:	Return the contained value or return the result of a function call. If *this contains a value, it is returned. Otherwise, the specified callable is invoked and its return value which is to be compatible to type T is returned from this function. The Callable is expected to be compatible to this interface: `T f(const E&);`	

⌋

8.4.2.2.3.16 Swap

[SWS_CORE_00745] Definition of API function ara::core::Result::Swap

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`void Swap (Result &other) noexcept(std::is_nothrow_move_constructible< T >::value &&std::is_nothrow_move_assignable< T >::value &&std::is_nothrow_move_constructible< E >::value &&std::is_nothrow_move_assignable< E >::value);`	
Parameters (inout):	other	the other instance
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	Exchange the contents of this instance with those of other.	

⌋

8.4.2.2.3.17 Value

[SWS_CORE_00756] Definition of API function ara::core::Result::Value

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`T && Value () &&noexcept;`	
Return value:	T &&	an rvalue reference to the contained value
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`ResultWithNoValueViolation`	If *this does not contain a value.
Description:	Access the contained value.	

⌋

8.4.2.2.3.18 Value

[SWS_CORE_00775] Definition of API function ara::core::Result::Value

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`T & Value () & noexcept;`	
Return value:	T &	a reference to the contained value
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`ResultWithNoValueViolation`	If *this does not contain a value.
Description:	Access the contained value.	

⌋

8.4.2.2.3.19 Value

[SWS_CORE_00755] Definition of API function ara::core::Result::Value

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`const T & Value () const & noexcept;`	
Return value:	const T &	a const reference to the contained value
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`ResultWithNoValueViolation`	If *this does not contain a value.
Description:	Access the contained value.	

⌋

8.4.2.2.3.20 ValueOr

[SWS_CORE_00761] Definition of API function ara::core::Result::ValueOr

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	

Syntax:	`template <typename U>`
	`T ValueOr (U &&defaultValue) const &noexcept(std::is_nothrow_copy_constructible< T >::value &&std::is_nothrow_constructible< T, U &&>::value);`	

Template param:	U	the type of defaultValue
Parameters (in):	defaultValue	the value to use if *this does not contain a value
Return value:	T	the value
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return the contained value or the given default value. If *this contains a value, it is returned. Otherwise, the specified default value is returned, static_cast'd to T.	

⌋

8.4.2.2.3.21 ValueOr

[SWS_CORE_00762] Definition of API function ara::core::Result::ValueOr

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	

Syntax:	`template <typename U>`
	`T ValueOr (U &&defaultValue) &&noexcept(std::is_nothrow_move_constructible< T >::value &&std::is_nothrow_constructible< T, U && >::value);`	

Template param:	U	the type of defaultValue
Parameters (in):	defaultValue	the value to use if *this does not contain a value
Return value:	T	the value
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return the contained value or the given default value. If *this contains a value, it is returned. Otherwise, the specified default value is returned, static_cast'd to T.	

⌋

8.4.2.2.3.22 ValueOrThrow

[SWS_CORE_00769] Definition of API function ara::core::Result::ValueOrThrow

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`T && ValueOrThrow () &&noexcept(false);`	
Return value:	T &&	an rvalue reference to the contained value
Exceptions:	<TYPE>	the exception type associated with the contained error
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Return the contained value or throw an exception. This function shall not participate in overload resolution when C++ exceptions are disabled in the compiler toolchain.	

⌋

8.4.2.2.3.23 ValueOrThrow

[SWS_CORE_00766] Definition of API function ara::core::Result::ValueOrThrow

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`const T & ValueOrThrow () const &noexcept(false);`	
Return value:	const T &	a const reference to the contained value
Exceptions:	<TYPE>	the exception type associated with the contained error
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Return the contained value or throw an exception. This function shall not participate in overload resolution when C++ exceptions are disabled in the compiler toolchain.	

⌋

8.4.2.2.3.24 operator bool

[SWS_CORE_00752] Definition of API function ara::core::Result::operator bool

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`explicit operator bool () const noexcept;`	
Return value:	bool	true if *this contains a value, false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Check whether *this contains a value. This function shall be disabled if `std::is_same<std::decay_t<T>, bool>::value` is true.	

⌋

8.4.2.2.3.25 operator*

[SWS_CORE_00753] Definition of API function ara::core::Result::operator*

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`const T & operator* () const & noexcept;`	
Return value:	const T &	a const_reference to the contained value
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`ResultWithNoValueViolation`	If *this does not contain a value.
Description:	Access the contained value.	

⌋

8.4.2.2.3.26 operator*

[SWS_CORE_00759] Definition of API function ara::core::Result::operator*

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`T && operator* () &&noexcept;`	
Return value:	T &&	an rvalue reference to the contained value
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`ResultWithNoValueViolation`	If *this does not contain a value.
Description:	Access the contained value.	

⌋

8.4.2.2.3.27 operator*

[SWS_CORE_00774] Definition of API function ara::core::Result::operator*

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`T & operator* () & noexcept;`	
Return value:	T &	a reference to the contained value
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`ResultWithNoValueViolation`	If *this does not contain a value.
Description:	Access the contained value.	

⌋

8.4.2.2.3.28 operator->

[SWS_CORE_00754] Definition of API function ara::core::Result::operator->

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`const T * operator-> () const noexcept;`	
Return value:	const T *	a pointer to the contained value
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`ResultWithNoValueViolation`	If *this does not contain a value.
Description:	Access the contained value.	

⌋

8.4.2.2.3.29 operator->

[SWS_CORE_00777] Definition of API function ara::core::Result::operator->

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`T * operator-> () noexcept;`	
Return value:	T *	a pointer to the contained value
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`ResultWithNoValueViolation`	If *this does not contain a value.
Description:	Access the contained value.	

⌋

8.4.2.2.4 Named Constructors

8.4.2.2.4.1 FromError

[SWS_CORE_00736] Definition of API function ara::core::Result::FromError

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	

Syntax:	`template <typename... Args>`
	`static Result FromError (Args &&... args) noexcept(std::is_nothrow_constructible< E, Args... >::value);`	

Template param:	Args...	the types of arguments given to this function
Parameters (in):	args	the arguments used for constructing the error
Return value:	Result	a Result that contains an error
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	

▽

△

Description:	Build a new Result from an error that is constructed in-place from the given arguments. This function shall not participate in overload resolution unless: std::is_constructible<E, Args&&...>::value is true, and • the first type of the expanded parameter pack is not E, and • the first type of the expanded parameter pack is not a specialization of Result

⌋

8.4.2.2.4.2 FromError

[SWS_CORE_00734] Definition of API function ara::core::Result::FromError

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`static Result FromError (const E &e) noexcept(std::is_nothrow_copy_constructible< E >::value);`	
Parameters (in):	e	the error to put into the Result
Return value:	Result	a Result that contains the error e
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	Build a new Result from the specified error (given as lvalue).	

⌋

8.4.2.2.4.3 FromError

[SWS_CORE_00735] Definition of API function ara::core::Result::FromError

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`static Result FromError (E &&e) noexcept(std::is_nothrow_move_constructible< E >::value);`	
Parameters (in):	e	the error to put into the Result

▽

△

Return value:	Result	a Result that contains the error e
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	Build a new Result from the specified error (given as rvalue).	

⌋

8.4.2.2.4.4 FromValue

[SWS_CORE_00733] Definition of API function ara::core::Result::FromValue

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	

Syntax:	`template <typename... Args>`
	`static Result FromValue (Args &&... args) noexcept(std::is_nothrow_constructible< T, Args... >::value);`	

Template param:	Args...	the types of arguments given to this function
Parameters (in):	args	the arguments used for constructing the value
Return value:	Result	a Result that contains a value
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	Build a new Result from a value that is constructed in-place from the given arguments. This function shall not participate in overload resolution unless: std::is_constructible<T, Args&&...>::value is true, and ● the first type of the expanded parameter pack is not T, and ● the first type of the expanded parameter pack is not a specialization of Result	

⌋

8.4.2.2.4.5 FromValue

[SWS_CORE_00732] Definition of API function ara::core::Result::FromValue

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`static Result FromValue (T &&t) noexcept(std::is_nothrow_move_constructible< T >::value);`	
Parameters (in):	t	the value to put into the Result
Return value:	Result	a Result that contains the value t
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	Build a new Result from the specified value (given as rvalue).	

⌋

8.4.2.2.4.6 FromValue

[SWS_CORE_00731] Definition of API function ara::core::Result::FromValue

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`static Result FromValue (const T &t) noexcept(std::is_nothrow_copy_constructible< T >::value);`	
Parameters (in):	t	the value to put into the Result
Return value:	Result	a Result that contains the value t
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	Build a new Result from the specified value (given as lvalue).	

⌋

8.4.3 Class: Result

[SWS_CORE_00901] Definition of API class ara::core::Result< T &, E >

Upstream requirements: RS_AP_00130

⌈

Kind:	class	
Header file:	#include "ara/core/result.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace ara::core`	
Symbol:	Result< T &, E >	

Syntax:	`template <typename T, typename E>`
	`class Result< T &, E > final {...};`	

Template param:	typename T	the type of value
	typename E	the type of error
Description:	Specialization of class Result for reference to value types. The implementation shall flag the condition `E != ara::core::ErrorCode` as a compile error.	

⌋

8.4.3.1 Public Member Types

8.4.3.1.1 Type Alias: error_type

[SWS_CORE_00903] Definition of API type ara::core::Result< T &, E >::error_type

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/result.h"
Scope:	`class ara::core::Result< T &, E >`
Symbol:	error_type
Syntax:	`using error_type = E;`
Description:	Type alias for the type E of errors

⌋

8.4.3.1.2 Type Alias: value_type

[SWS_CORE_00902] Definition of API type ara::core::Result< T &, E >::value_type

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/result.h"
Scope:	`class ara::core::Result< T &, E >`
Symbol:	value_type
Syntax:	`using value_type = T&;`
Description:	Type alias for the type T& of values

⌋

8.4.3.2 Public Member Functions

8.4.3.2.1 Special Member Functions

8.4.3.2.1.1 Move Constructor

[SWS_CORE_00912] Definition of API function ara::core::Result< T &, E >::Result

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`Result (Result &&other) noexcept(std::is_nothrow_move_constructible< E >::value);`	
Parameters (in):	other	the other instance
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	Move-construct a new Result from another instance.	

⌋

8.4.3.2.1.2 Copy Constructor

[SWS_CORE_00911] Definition of API function ara::core::Result< T &, E >::Result

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`Result (const Result &other) noexcept(std::is_nothrow_copy_constructible< E >::value);`	
Parameters (in):	other	the other instance
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	Copy-construct a new Result from another instance.	

⌋

8.4.3.2.1.3 Copy Assignment Operator

[SWS_CORE_00914] Definition of API function ara::core::Result< T &, E >::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`Result & operator= (const Result &other) noexcept(std::is_nothrow_copy_constructible< E >::value &&std::is_nothrow_copy_assignable< E >::value);`	
Parameters (in):	other	the other instance
Return value:	Result &	*this, containing the contents of other
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	Copy-assignment operator If other contains a value, the contained reference is rebound to the referee of other. If other contains an error, the contained error is copy-constructed or copy-assigned from the one contained by other.	

⌋

8.4.3.2.1.4 Move Assignment Operator

[SWS_CORE_00915] Definition of API function ara::core::Result< T &, E >::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`Result & operator= (Result &&other) noexcept(std::is_nothrow_move_constructible< E >::value &&std::is_nothrow_move_assignable< E >::value);`	
Parameters (in):	other	the other instance
Return value:	Result &	*this, containing the contents of other
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	Move-assignment operator If other contains a value, the contained reference is rebound to the referee of other. If other contains an error, the contained error is move-constructed or move-assigned from the one contained by other.	

⌋

8.4.3.2.1.5 Destructor

[SWS_CORE_00913] Definition of API function ara::core::Result< T &, E >::~Result

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/result.h"
Scope:	`class ara::core::Result< T &, E >`
Syntax:	`~Result () noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	Destructor. This destructor is trivial if std::is_trivially_destructible<E>::value is true.

⌋

8.4.3.2.2 Constructors

8.4.3.2.2.1 Result

[SWS_CORE_00909] Definition of API function ara::core::Result< T &, E >::Result

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`explicit Result (const E &e) noexcept(std::is_nothrow_copy_constructible< E >::value);`	
Parameters (in):	e	the error to put into the Result
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	Construct a new Result from the specified error (given as lvalue).	

⌋

8.4.3.2.2.2 Result

[SWS_CORE_00910] Definition of API function ara::core::Result< T &, E >::Result

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`explicit Result (E &&e) noexcept(std::is_nothrow_move_constructible< E >::value);`	
Parameters (in):	e	the error to put into the Result
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	Construct a new Result from the specified error (given as rvalue).	

⌋

8.4.3.2.2.3 Result

[SWS_CORE_00908] Definition of API function ara::core::Result< T &, E >::Result

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	

Syntax:	`template <typename U = T>`
	`Result (const U &&t) noexcept;`	

Template param:	U	the type of t
Parameters (in):	t	the value to bind to
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Construct a new Result containing a reference to value from the specified reference to value. If U is not an lvalue, the program is ill-formed.	

⌋

8.4.3.2.3 Member Functions

8.4.3.2.3.1 Bind

[SWS_CORE_00941] Definition of API function ara::core::Result< T &, E >::Bind

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	

Syntax:	`template <typename F>`
	`auto Bind (F &&f) const -> <see below>;`	

Template param:	F	the type of the Callable f
Parameters (in):	f	the Callable
Return value:	*<see below>*	a new Result instance of the possibly transformed type
Exception Safety:	not exception safe	
Thread Safety:	conditional,	depends on thread-safety of f

▽

△

Description:	Apply the given Callable to the value of this instance, and return a new Result with the result of the call. The Callable is expected to be compatible to one of these two interfaces: • `Result<XXX, E> f(const T&);` • `XXX f(const T&);` meaning that the Callable either returns a Result<XXX> or a XXX directly, where XXX can be any type that is suitable for use by class Result. The return type of this function is `decltype(f(Value()))` for a template argument F that returns a Result type, and it is `Result<decltype(f(Value())), E>` for a template argument F that does not return a Result type. If this instance does not contain a value, a new Result<XXX, E> is still created and returned, with the original error contents of this instance being copied into the new instance.

⌋

8.4.3.2.3.2 CheckError

[SWS_CORE_00937] Definition of API function ara::core::Result< T &, E >::CheckError

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	

Syntax:	`template <typename G>`
	`bool CheckError (G &&error) const;`	

Template param:	G	the type of the error argument error
Parameters (in):	error	the error to check
Return value:	bool	true if *this contains an error that is equivalent to the given error, false otherwise
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Return whether this instance contains the given error. This call compares the argument error, static_cast'd to E, with the return value from Error().	

⌋

8.4.3.2.3.3 EmplaceError

[SWS_CORE_00916] Definition of API function ara::core::Result< T &, E >::EmplaceError

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	

Syntax:	`template <typename... Args>`
	`void EmplaceError (Args &&... args) noexcept(std::is_nothrow_constructible< E, Args... >::value);`	

Template param:	Args...	the types of arguments given to this function
Parameters (in):	args	the arguments used for constructing the error
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	Put a new error into this instance, constructed in-place from the given arguments.	

⌋

8.4.3.2.3.4 Err

[SWS_CORE_00931] Definition of API function ara::core::Result< T &, E >::Err

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`Optional< E > Err () const &noexcept(std::is_nothrow_constructible< Optional< E >, const E &>::value);`	
Return value:	Optional< E >	an Optional with the error, if present
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return the contained error as an Optional.	

⌋

8.4.3.2.3.5 Err

[SWS_CORE_00932] Definition of API function ara::core::Result< T &, E >::Err

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result< T &, E >	
Syntax:	Optional< E > Err () &&noexcept(std::is_nothrow_constructible< Optional< E >, E && >::value);	
Return value:	Optional< E >	an Optional with the error, if present
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return the contained error as an Optional.	

⌋

8.4.3.2.3.6 Error

[SWS_CORE_00929] Definition of API function ara::core::Result< T &, E >::Error

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result< T &, E >	
Syntax:	E & Error () & noexcept;	
Return value:	E &	a reference to the contained error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	ResultWithNoErrorViolation	If *this does not contain an error.
Description:	Access the contained error.	

⌋

8.4.3.2.3.7 Error

[SWS_CORE_00930] Definition of API function ara::core::Result< T &, E >::Error

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`E && Error () &&noexcept;`	
Return value:	E &&	an rvalue reference to the contained error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`ResultWithNoErrorViolation`	If *this does not contain an error.
Description:	Access the contained error.	

⌋

8.4.3.2.3.8 Error

[SWS_CORE_00928] Definition of API function ara::core::Result< T &, E >::Error

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`const E & Error () const & noexcept;`	
Return value:	const E &	a const reference to the contained error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`ResultWithNoErrorViolation`	If *this does not contain an error.
Description:	Access the contained error.	

⌋

8.4.3.2.3.9 ErrorOr

[SWS_CORE_00936] Definition of API function ara::core::Result< T &, E >::ErrorOr

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	

Syntax:	`template <typename G>`
	`E ErrorOr (G &&defaultError) &&noexcept(std::is_nothrow_move_constructible< E >::value &&std::is_nothrow_constructible< E, G && >::value);`	

Template param:	G	the type of defaultError
Parameters (in):	defaultError	the error to use if *this does not contain an error
Return value:	E	the error
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return the contained error or the given default error. If *this contains an error, it is std::move'd into the return value. Otherwise, the specified default error is returned, static_cast'd to E.	

⌋

8.4.3.2.3.10 ErrorOr

[SWS_CORE_00935] Definition of API function ara::core::Result< T &, E >::ErrorOr

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	

Syntax:	`template <typename G>`
	`E ErrorOr (G &&defaultError) const &noexcept(std::is_nothrow_copy_constructible< E >::value &&std::is_nothrow_constructible< E, G &&>::value);`	

Template param:	G	the type of defaultError
Parameters (in):	defaultError	the error to use if *this does not contain an error
Return value:	E	the error
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	

▽

△

Description:	Return the contained error or the given default error. If *this contains an error, it is returned. Otherwise, the specified default error is returned, static_cast'd to E.	

⌋

8.4.3.2.3.11 HasValue

[SWS_CORE_00918] Definition of API function ara::core::Result< T &, E >::HasValue

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`bool HasValue () const noexcept;`	
Return value:	bool	true if *this contains a value, false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Check whether *this contains a value.	

⌋

8.4.3.2.3.12 Ok

[SWS_CORE_00926] Definition of API function ara::core::Result< T &, E >::Ok

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`Optional< const T & > Ok () const noexcept;`	
Return value:	Optional< const T & >	an Optional with the const-reference to value, if present
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return the referred-to value as an Optional	

⌋

8.4.3.2.3.13 Ok

[SWS_CORE_00927] Definition of API function ara::core::Result< T &, E >::Ok

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`Optional< T & > Ok () noexcept;`	
Return value:	Optional< T & >	an Optional with the reference to value, if present
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return the referred-to value as an Optional.	

⌋

8.4.3.2.3.14 Resolve

[SWS_CORE_00940] Definition of API function ara::core::Result< T &, E >::Resolve

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	

Syntax:	`template <typename F>`
	`T & Resolve (F &&f) const;`	

Template param:	F	the type of the Callable f
Parameters (in):	f	the Callable
Return value:	T &	the referred-to value
Exception Safety:	not exception safe	
Thread Safety:	conditional,	depends on thread safety of f
Description:	Return the referred-to value or return the result of a function call. If *this contains a value, it is returned. Otherwise, the specified callable is invoked and its return value which is to be compatible to type T& is returned from this function. The Callable is expected to be compatible to this interface: `T& f(const E&);`	

⌋

8.4.3.2.3.15 Resolve

[SWS_CORE_00939] Definition of API function ara::core::Result< T &, E >::Resolve

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	

Syntax:	`template <typename F>`
	`const T & Resolve (F &&f) const;`	

Template param:	F	the type of the Callable f
Parameters (in):	f	the Callable
Return value:	const T &	the referred-to value
Exception Safety:	not exception safe	
Thread Safety:	conditional,	depends on thread safety of f
Description:	Return the referred-to value or return the result of a function call. If *this contains a value, it is returned. Otherwise, the specified callable is invoked and its return value which is to be compatible to type T& is returned from this function. The Callable is expected to be compatible to this interface: `const T& f(const E&);`	

⌋

8.4.3.2.3.16 Swap

[SWS_CORE_00917] Definition of API function ara::core::Result< T &, E >::Swap

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`void Swap (Result &other) noexcept(std::is_nothrow_move_constructible< E >::value &&std::is_nothrow_move_assignable< E >::value);`	
Parameters (inout):	other	the other instance
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	Exchange the contents of this instance with those of other.	

⌋

8.4.3.2.3.17 Value

[SWS_CORE_00925] Definition of API function ara::core::Result< T &, E >::Value

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`T & Value () & noexcept;`	
Return value:	T &	a reference to the referred-to value
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`ResultWithNoValueViolation`	If *this does not contain a value.
Description:	Access the referred-to value.	

⌋

8.4.3.2.3.18 Value

[SWS_CORE_00924] Definition of API function ara::core::Result< T &, E >::Value

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`const T & Value () const & noexcept;`	
Return value:	const T &	a const reference to the referred-to value
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`ResultWithNoValueViolation`	If *this does not contain a value.
Description:	Access the referred-to value.	

⌋

8.4.3.2.3.19 ValueOr

[SWS_CORE_00934] Definition of API function ara::core::Result< T &, E >::ValueOr

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	

Syntax:	`template <typename U>`
	`T ValueOr (U &&defaultValue) &&noexcept(std::is_nothrow_move_constructible< T >::value &&std::is_nothrow_constructible< T, U && >::value);`	

Template param:	U	the type of defaultValue
Parameters (in):	defaultValue	the value to use if *this does not contain a value
Return value:	T	the value
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return a copy of the referred-to value or the given default value. If *this contains a value, it is returned. Otherwise, the specified default value is returned, static_cast'd to T.	

⌋

8.4.3.2.3.20 ValueOr

[SWS_CORE_00933] Definition of API function ara::core::Result< T &, E >::ValueOr

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	

Syntax:	`template <typename U>`
	`T ValueOr (U &&defaultValue) const &noexcept(std::is_nothrow_copy_constructible< T >::value &&std::is_nothrow_constructible< T, U &&>::value);`	

Template param:	U	the type of defaultValue
Parameters (in):	defaultValue	the value to use if *this does not contain a value
Return value:	T	the value
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	

▽

△

Description:	Return a copy of the referred-to value or the given default value. If *this contains a value, it is returned. Otherwise, the specified default value is returned, static_cast'd to T.

⌋

8.4.3.2.3.21 ValueOrThrow

[SWS_CORE_00938] Definition of API function ara::core::Result< T &, E >::ValueOrThrow

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`const T & ValueOrThrow () const noexcept(false);`	
Return value:	const T &	a const reference to the contained value
Exceptions:	<TYPE>	the exception type associated with the contained error
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Return the contained value or throw an exception. This function shall not participate in overload resolution when C++ exceptions are disabled in the compiler toolchain.	

⌋

8.4.3.2.3.22 operator bool

[SWS_CORE_00919] Definition of API function ara::core::Result< T &, E >::operator bool

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`explicit operator bool () const noexcept;`	
Return value:	bool	true if *this contains a value, false otherwise

▽

△

Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	Check whether *this contains a value.

⌋

8.4.3.2.3.23 operator*

[SWS_CORE_00921] Definition of API function ara::core::Result< T &, E >::operator*

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`T & operator* () & noexcept;`	
Return value:	T &	a pointer to the referred-to value
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`ResultWithNoValueViolation`	If *this does not contain a value.
Description:	Access the referred-to value.	

⌋

8.4.3.2.3.24 operator*

[SWS_CORE_00920] Definition of API function ara::core::Result< T &, E >::operator*

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`const T & operator* () const & noexcept;`	
Return value:	const T &	a const_reference to the referred-to value

▽

△

Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`ResultWithNoValueViolation`	If *this does not contain a value.
Description:	Access the referred-to value.	

⌋

8.4.3.2.3.25 operator->

[SWS_CORE_00922] Definition of API function ara::core::Result< T &, E >::operator->

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`const T * operator-> () const noexcept;`	
Return value:	const T *	a const pointer to the referred-to value
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`ResultWithNoValueViolation`	If *this does not contain a value.
Description:	Access the referred-to value.	

⌋

8.4.3.2.3.26 operator->

[SWS_CORE_00923] Definition of API function ara::core::Result< T &, E >::operator->

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/result.h"
Scope:	`class ara::core::Result< T &, E >`
Syntax:	`T * operator-> () noexcept;`

▽

△

Return value:	T *	a pointer to the contained value
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`ResultWithNoValueViolation`	If *this does not contain a value.
Description:	Access the referred-to value.	

⌋

8.4.3.2.4 Named Constructors

8.4.3.2.4.1 FromError

[SWS_CORE_00907] Definition of API function ara::core::Result< T &, E >::FromError

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	

Syntax:	`template <typename... Args>`
	`static Result FromError (Args &&... args) noexcept(std::is_nothrow_constructible< E, Args... >::value);`	

Template param:	Args...	the types of arguments given to this function
Parameters (in):	args	the parameter pack used for constructing the error
Return value:	Result	a Result that contains an error
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	Build a new Result from an error that is constructed in-place from the given arguments. This function shall not participate in overload resolution unless: std::is_constructible<E, Args&&...>::value is true, and ● the first type of the expanded parameter pack is not E, and ● the first type of the expanded parameter pack is not a specialization of Result	

⌋

8.4.3.2.4.2 FromError

[SWS_CORE_00905] Definition of API function ara::core::Result< T &, E >::FromError

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`static Result FromError (const E &e) noexcept(std::is_nothrow_copy_constructible< E >::value);`	
Parameters (in):	e	the error to put into the Result
Return value:	Result	a Result that contains the error e
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	Build a new Result from the specified error (given as lvalue).	

⌋

8.4.3.2.4.3 FromError

[SWS_CORE_00906] Definition of API function ara::core::Result< T &, E >::FromError

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`static Result FromError (E &&e) noexcept(std::is_nothrow_move_constructible< E >::value);`	
Parameters (in):	e	the error to put into the Result
Return value:	Result	a Result that contains the error e
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	Build a new Result from the specified error (given as rvalue).	

⌋

8.4.3.2.4.4 FromValue

[SWS_CORE_00904] Definition of API function ara::core::Result< T &, E >::FromValue

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	

Syntax:	`template <typename U = T>`
	`static Result FromValue (U &&u) noexcept;`	

Template param:	U	the type of u
Parameters (in):	u	the value to bind to
Return value:	Result	a Result that contains a reference to value
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Build a new Result from the reference to a specified value (given as lvalue). If U is not an lvalue, the program is ill-formed.	

⌋

8.4.4 Class: Result

[SWS_CORE_00801] Definition of API class ara::core::Result< void, E >

Upstream requirements: RS_AP_00130

⌈

Kind:	class	
Header file:	#include "ara/core/result.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace ara::core`	
Symbol:	Result< void, E >	

Syntax:	`template <typename E>`
	`class Result< void, E > final {...};`	

Template param:	typename E	the type of error
Description:	Specialization of class Result for "void" values. The implementation shall flag the condition `E != ara::core::ErrorCode` as a compile error.	

⌋

8.4.4.1 Public Member Types

8.4.4.1.1 Type Alias: error_type

[SWS_CORE_00812] Definition of API type ara::core::Result< void, E >::error_type

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/result.h"
Scope:	`class ara::core::Result< void, E >`
Symbol:	error_type
Syntax:	`using error_type = E;`
Description:	Type alias for the type E of errors .

⌋

8.4.4.1.2 Type Alias: value_type

[SWS_CORE_00811] Definition of API type ara::core::Result< void, E >::value_type

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/result.h"
Scope:	`class ara::core::Result< void, E >`
Symbol:	value_type
Syntax:	`using value_type = void;`
Description:	Type alias for the type T of values, always "void" for this specialization .

⌋

8.4.4.2 Public Member Functions

8.4.4.2.1 Special Member Functions

8.4.4.2.1.1 Copy Constructor

[SWS_CORE_00825] Definition of API function ara::core::Result< void, E >::Result

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	
Syntax:	`Result (const Result &other) noexcept(std::is_nothrow_copy_constructible< E >::value);`	
Parameters (in):	other	the other instance
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	Copy-construct a new Result from another instance.	

⌋

8.4.4.2.1.2 Move Constructor

[SWS_CORE_00826] Definition of API function ara::core::Result< void, E >::Result

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	
Syntax:	`Result (Result &&other) noexcept(std::is_nothrow_move_constructible< E >::value);`	
Parameters (in):	other	the other instance
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	Move-construct a new Result from another instance.	

⌋

8.4.4.2.1.3 Default Constructor

[SWS_CORE_00821] Definition of API function ara::core::Result< void, E >::Result

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/result.h"
Scope:	`class ara::core::Result< void, E >`
Syntax:	`Result () noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	Construct a new Result with a "void" value.

⌋

8.4.4.2.1.4 Move Assignment Operator

[SWS_CORE_00842] Definition of API function ara::core::Result< void, E >::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	
Syntax:	`Result & operator= (Result &&other) noexcept(std::is_nothrow_move_constructible< E >::value &&std::is_nothrow_move_assignable< E >::value);`	
Parameters (in):	other	the other instance
Return value:	Result &	*this, containing the contents of other
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	Move-assign another Result to this instance.	

⌋

8.4.4.2.1.5 Copy Assignment Operator

[SWS_CORE_00841] Definition of API function ara::core::Result< void, E >::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	
Syntax:	`Result & operator= (const Result &other) noexcept(std::is_nothrow_copy_constructible< E >::value &&std::is_nothrow_copy_assignable< E >::value);`	
Parameters (in):	other	the other instance
Return value:	Result &	*this, containing the contents of other
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	Copy-assign another Result to this instance.	

⌋

8.4.4.2.1.6 Destructor

[SWS_CORE_00827] Definition of API function ara::core::Result< void, E >::~Result

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/result.h"
Scope:	`class ara::core::Result< void, E >`
Syntax:	`~Result () noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	Destructor. This destructor is trivial if std::is_trivially_destructible<E>::value is true.

⌋

8.4.4.2.2 Constructors

8.4.4.2.2.1 Result

[SWS_CORE_00824] Definition of API function ara::core::Result< void, E >::Result

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	
Syntax:	`explicit Result (E &&e) noexcept(std::is_nothrow_move_constructible< E >::value);`	
Parameters (in):	e	the error to put into the Result
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	Construct a new Result from the specified error (given as rvalue).	

⌋

8.4.4.2.2.2 Result

[SWS_CORE_00823] Definition of API function ara::core::Result< void, E >::Result

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	
Syntax:	`explicit Result (const E &e) noexcept(std::is_nothrow_copy_constructible< E >::value);`	
Parameters (in):	e	the error to put into the Result
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	Construct a new Result from the specified error (given as lvalue).	

⌋

8.4.4.2.3 Member Functions

8.4.4.2.3.1 Bind

[SWS_CORE_00870] Definition of API function ara::core::Result< void, E >::Bind

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	

Syntax:	`template <typename F>`
	`auto Bind (F &&f) const -> <see below>;`	

Template param:	F	the type of the Callable f
Parameters (in):	f	the Callable
Return value:	*<see below>*	a new Result instance of the possibly transformed type
Exception Safety:	not exception safe	
Thread Safety:	conditional,	depends on thread safety of f
Description:	Call the given Callable, and return a new Result with the result of the call. The Callable is expected to be compatible to one of these two interfaces: • `Result<XXX, E> f();` • `XXX f();` meaning that the Callable either returns a Result<XXX, E> or a XXX directly, where XXX can be any type that is suitable for use by class Result. The return type of this function is `decltype(f())` for a template argument F that returns a Result type, and it is `Result<decltype(f()), E>` for a template argument F that does not return a Result type. If this instance does not contain a value, a new Result<XXX, E> is still created and returned, with the original error contents of this instance being copied into the new instance.	

⌋

8.4.4.2.3.2 CheckError

[SWS_CORE_00865] Definition of API function ara::core::Result< void, E >::CheckError

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/result.h"
Scope:	`class ara::core::Result< void, E >`

▽

△

Syntax:	template <typename G> bool CheckError (G &&error) const noexcept;	
Template param:	G	the type of the error argument error
Parameters (in):	error	the error to check
Return value:	bool	true if *this contains an error that is equivalent to the given error, false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return whether this instance contains the given error. This call compares the argument error, static_cast'd to E, with the return value from Error().	

⌋

8.4.4.2.3.3 EmplaceError

[SWS_CORE_00844] Definition of API function ara::core::Result< void, E >::EmplaceError

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result< void, E >	
Syntax:	template <typename... Args> void EmplaceError (Args &&... args) noexcept(std::is_nothrow_constructible< E, Args... >::value);	
Template param:	Args...	the types of arguments given to this function
Parameters (in):	args	the arguments used for constructing the error
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	Put a new error into this instance, constructed in-place from the given arguments.	

⌋

8.4.4.2.3.4 EmplaceValue

[SWS_CORE_00843] Definition of API function ara::core::Result< void, E >::EmplaceValue

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	

Syntax:	`template <typename... Args>`
	`void EmplaceValue (Args &&... args) noexcept;`	

Template param:	Args...	the types of arguments given to this function
Parameters (in):	args	the arguments used for constructing the value
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Put a new value into this instance, constructed in-place from the given arguments.	

⌋

8.4.4.2.3.5 Err

[SWS_CORE_00868] Definition of API function ara::core::Result< void, E >::Err

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	
Syntax:	`Optional< E > Err () const &noexcept(std::is_nothrow_constructible< Optional< E >, const E &>::value);`	
Return value:	Optional< E >	an Optional with the error, if present
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return the contained error as an Optional.	

⌋

8.4.4.2.3.6 Err

[SWS_CORE_00869] Definition of API function ara::core::Result< void, E >::Err

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	
Syntax:	`Optional< E > Err () &&noexcept(std::is_nothrow_constructible< Optional< E >, E && >::value);`	
Return value:	Optional< E >	an Optional with the error, if present
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return the contained error as an Optional.	

⌋

8.4.4.2.3.7 Error

[SWS_CORE_00858] Definition of API function ara::core::Result< void, E >::Error

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	
Syntax:	`E && Error () &&noexcept;`	
Return value:	E &&	an rvalue reference to the contained error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`ResultWithNoErrorViolation`	If *this does not contain an error.
Description:	Access the contained error.	

⌋

8.4.4.2.3.8 Error

[SWS_CORE_00876] Definition of API function ara::core::Result< void, E >::Error

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	
Syntax:	`E & Error () & noexcept;`	
Return value:	E &	a reference to the contained error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`ResultWithNoErrorViolation`	If *this does not contain an error.
Description:	Access the contained error.	

⌋

8.4.4.2.3.9 Error

[SWS_CORE_00857] Definition of API function ara::core::Result< void, E >::Error

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	
Syntax:	`const E & Error () const & noexcept;`	
Return value:	const E &	a const reference to the contained error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`ResultWithNoErrorViolation`	If *this does not contain an error.
Description:	Access the contained error.	

⌋

8.4.4.2.3.10 ErrorOr

[SWS_CORE_00864] Definition of API function ara::core::Result< void, E >::ErrorOr

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	

Syntax:	`template <typename G>`
	`E ErrorOr (G &&defaultError) &&noexcept(std::is_nothrow_move_constructible< E >::value &&std::is_nothrow_constructible< E, G && >::value);`	

Template param:	G	the type of defaultError
Parameters (in):	defaultError	the error to use if *this does not contain an error
Return value:	E	the error
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return the contained error or the given default error. If *this contains an error, it is std::move'd into the return value. Otherwise, the specified default error is returned, static_cast'd to E.	

⌋

8.4.4.2.3.11 ErrorOr

[SWS_CORE_00863] Definition of API function ara::core::Result< void, E >::ErrorOr

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	

Syntax:	`template <typename G>`
	`E ErrorOr (G &&defaultError) const &noexcept(std::is_nothrow_copy_constructible< E >::value &&std::is_nothrow_constructible< E, G &&>::value);`	

Template param:	G	the type of defaultError
Parameters (in):	defaultError	the error to use if *this does not contain an error
Return value:	E	the error
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	

▽

△

Description:	Return the contained error or the given default error. If *this contains an error, it is returned. Otherwise, the specified default error is returned, static_cast'd to E.	

⌋

8.4.4.2.3.12 HasValue

[SWS_CORE_00851] Definition of API function ara::core::Result< void, E >::HasValue

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	
Syntax:	`bool HasValue () const noexcept;`	
Return value:	bool	true if *this contains a value, false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Check whether *this contains a value.	

⌋

8.4.4.2.3.13 Resolve

[SWS_CORE_00867] Definition of API function ara::core::Result< void, E >::Resolve

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	

Syntax:	`template <typename F>`
	`void Resolve (F &&f) const;`	

Template param:	F	the type of the Callable f
Parameters (in):	f	the Callable
Return value:	None	

▽

△

Exception Safety:	not exception safe	
Thread Safety:	conditional,	depends on thread safety of f
Description:	Do nothing or call a function. If *this contains a value, this function does nothing. Otherwise, the specified callable is invoked. The Callable is expected to be compatible to this interface: `void f(const E&);` This function only exists for helping with generic programming.	

⌋

8.4.4.2.3.14 Swap

[SWS_CORE_00845] Definition of API function ara::core::Result< void, E >::Swap

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	
Syntax:	`void Swap (Result &other) noexcept(std::is_nothrow_move_constructible< E >::value &&std::is_nothrow_move_assignable< E >::value);`	
Parameters (inout):	other	the other instance
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	Exchange the contents of this instance with those of other.	

⌋

8.4.4.2.3.15 Value

[SWS_CORE_00855] Definition of API function ara::core::Result< void, E >::Value

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/result.h"
Scope:	`class ara::core::Result< void, E >`
Syntax:	`void Value () const noexcept;`

▽

△

Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`ResultWithNoValueViolation`	If *this does not contain a value.
Description:	This function only exists for helping with generic programming.	

⌋

8.4.4.2.3.16 ValueOr

[SWS_CORE_00861] Definition of API function ara::core::Result< void, E >::ValueOr

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	

Syntax:	`template <typename U>`
	`void ValueOr (U &&defaultValue) const noexcept;`	

Template param:	U	the type of defaultValue
Parameters (in):	defaultValue	the value to use if *this does not contain a value
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Do nothing. This function only exists for helping with generic programming.	

⌋

8.4.4.2.3.17 ValueOrThrow

[SWS_CORE_00866] Definition of API function ara::core::Result< void, E >::ValueOrThrow

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	
Syntax:	`void ValueOrThrow () const noexcept(false);`	
Return value:	None	
Exceptions:	<TYPE>	the exception type associated with the contained error
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Return the contained value or throw an exception. This function shall not participate in overload resolution when C++ exceptions are disabled in the compiler toolchain.	

⌋

8.4.4.2.3.18 operator bool

[SWS_CORE_00852] Definition of API function ara::core::Result< void, E >::operator bool

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	
Syntax:	`explicit operator bool () const noexcept;`	
Return value:	bool	true if *this contains a value, false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Check whether *this contains a value.	

⌋

8.4.4.2.3.19 operator*

[SWS_CORE_00853] Definition of API function ara::core::Result< void, E >::operator*

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	
Syntax:	`void operator* () const noexcept;`	
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`ResultWithNoValueViolation`	If *this does not contain a value.
Description:	Access the contained value.	

⌋

8.4.4.2.4 Named Constructors

8.4.4.2.4.1 FromError

[SWS_CORE_00836] Definition of API function ara::core::Result< void, E >::FromError

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	

Syntax:	`template <typename... Args>`
	`static Result FromError (Args &&... args) noexcept(std::is_nothrow_constructible< E, Args... >::value);`	

Template param:	Args...	the types of arguments given to this function
Parameters (in):	args	the parameter pack used for constructing the error
Return value:	Result	a Result that contains an error
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	

▽

△

Description:	Build a new Result from an error that is constructed in-place from the given arguments. This function shall not participate in overload resolution unless: std::is_constructible<E, Args&&...>::value is true, and ● the first type of the expanded parameter pack is not E, and ● the first type of the expanded parameter pack is not a specialization of Result

⌋

8.4.4.2.4.2 FromError

[SWS_CORE_00834] Definition of API function ara::core::Result< void, E >::FromError

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	
Syntax:	`static Result FromError (const E &e) noexcept(std::is_nothrow_copy_constructible< E >::value);`	
Parameters (in):	e	the error to put into the Result
Return value:	Result	a Result that contains the error e
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	Build a new Result from the specified error (given as lvalue).	

⌋

8.4.4.2.4.3 FromError

[SWS_CORE_00835] Definition of API function ara::core::Result< void, E >::FromError

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/result.h"
Scope:	`class ara::core::Result< void, E >`

▽

△

Syntax:	`static Result FromError (E &&e) noexcept(std::is_nothrow_move_constructible< E >::value);`	
Parameters (in):	e	the error to put into the Result
Return value:	Result	a Result that contains the error e
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	Build a new Result from the specified error (given as rvalue).	

⌋

8.4.4.2.4.4 FromValue

[SWS_CORE_00831] Definition of API function ara::core::Result< void, E >::FromValue

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	
Syntax:	`static Result FromValue () noexcept;`	
Return value:	Result	a Result that contains a "void" value
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Build a new Result with "void" as value.	

⌋

8.5 Header: ara/core/core_error_domain.h

8.5.1 Non-Member Types

8.5.1.1 Enumeration: CoreErrc

[SWS_CORE_05200] Definition of API enum ara::core::CoreErrc

Upstream requirements: RS_AP_00130

⌈

Kind:	enumeration	
Header file:	#include "ara/core/core_error_domain.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace ara::core`	
Symbol:	CoreErrc	
Underlying type:	ErrorDomain::CodeType	
Syntax:	`enum class CoreErrc : ErrorDomain::CodeType {...};`	
Values:	kInvalidArgument= 22	An invalid argument was passed to a function
	kInvalidMetaModelShortname= 137	Given string is not a valid model element `shortName`
	kInvalidMetaModelPath= 138	Missing or invalid path to model element
	kAlreadyInitialized= 141	Initialization has already occurred and is complete
	kNotInitialized= 142	The subsystem is not initialized
Description:	Defines all errors of the `ara::core` Functional Cluster	

⌋

8.5.2 Non-Member Functions

8.5.2.1 Other

8.5.2.1.1 GetCoreErrorDomain

[SWS_CORE_05280] Definition of API function ara::core::GetCoreErrorDomain

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/core_error_domain.h"
Scope:	`namespace ara::core`
Syntax:	`constexpr const ErrorDomain & GetCoreErrorDomain () noexcept;`

▽

△

Return value:	const ErrorDomain &	the CoreErrorDomain
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a reference to the global CoreErrorDomain.	

⌋

8.5.2.1.2 MakeErrorCode

[SWS_CORE_05290] Definition of API function ara::core::MakeErrorCode

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/core_error_domain.h"	
Scope:	`namespace ara::core`	
Syntax:	`constexpr ErrorCode MakeErrorCode (CoreErrc code, ErrorDomain::SupportDataType data) noexcept;`	
Parameters (in):	code	the CoreErrorDomain-specific error code value
	data	optional vendor-specific error data
Return value:	ErrorCode	a new ErrorCode instance
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Create a new ErrorCode within CoreErrorDomain. This function is used internally by constructors of ErrorCode. It is usually not used directly by users.	

⌋

8.5.3 Class: CoreErrorDomain

[SWS_CORE_05221] Definition of API class ara::core::CoreErrorDomain

Upstream requirements: RS_AP_00130

⌈

Kind:	class
Header file:	#include "ara/core/core_error_domain.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	`namespace ara::core`
Symbol:	CoreErrorDomain

▽

△

Base class:	ErrorDomain
Syntax:	`class CoreErrorDomain final : public ErrorDomain {...};`
Unique ID:	As per `ara::core::CoreErrorDomain` in [SWS_CORE_90023]
Description:	An error domain for errors originating from the `ara::core` Functional Cluster.

⌋

8.5.3.1 Public Member Types

8.5.3.1.1 Type Alias: Errc

[SWS_CORE_05231] Definition of API type ara::core::CoreErrorDomain::Errc

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/core_error_domain.h"
Scope:	`class ara::core::CoreErrorDomain`
Symbol:	Errc
Syntax:	`using Errc = CoreErrc;`
Description:	Alias for the error code value enumeration

⌋

8.5.3.1.2 Type Alias: Exception

[SWS_CORE_05232] Definition of API type ara::core::CoreErrorDomain::Exception

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/core_error_domain.h"
Scope:	`class ara::core::CoreErrorDomain`
Symbol:	Exception
Syntax:	`using Exception = CoreException;`
Description:	Alias for the exception base class

⌋

8.5.3.2 Public Member Functions

8.5.3.2.1 Special Member Functions

8.5.3.2.1.1 Default Constructor

[SWS_CORE_05241] Definition of API function ara::core::CoreErrorDomain::CoreErrorDomain

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/core_error_domain.h"
Scope:	`class ara::core::CoreErrorDomain`
Syntax:	`constexpr CoreErrorDomain () noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	Default constructor

⌋

8.5.3.2.2 Member Functions

8.5.3.2.2.1 Message

[SWS_CORE_05243] Definition of API function ara::core::CoreErrorDomain::Message

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/core_error_domain.h"	
Scope:	`class ara::core::CoreErrorDomain`	
Syntax:	`const char * Message (ErrorDomain::CodeType errorCode) const noexcept override;`	
Parameters (in):	errorCode	the error code value
Return value:	const char *	the text message, never nullptr
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Translate an error code value into a text message	

⌋

8.5.3.2.2.2 Name

[SWS_CORE_05242] Definition of API function ara::core::CoreErrorDomain::Name

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/core_error_domain.h"	
Scope:	`class ara::core::CoreErrorDomain`	
Syntax:	`const char * Name () const noexcept override;`	
Return value:	const char *	"Core"
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return the `ApApplicationErrorDomain`. `shortName` of this error domain	

⌋

8.5.3.2.2.3 ThrowAsException

[SWS_CORE_05244] Definition of API function ara::core::CoreErrorDomain::ThrowAsException

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/core_error_domain.h"	
Scope:	`class ara::core::CoreErrorDomain`	
Syntax:	`void ThrowAsException (const ErrorCode &errorCode) const noexcept(false) override;`	
Parameters (in):	errorCode	the ErrorCode instance
Return value:	None	
Exceptions:	`CoreException`	an exception containing the given `ErrorCode`
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Throw the exception type corresponding to the given ErrorCode. As per [SWS_CORE_10304], this function does not participate in overload resolution when C++ exceptions are disabled in the compiler toolchain.	

⌋

8.5.4 Class: CoreException

[SWS_CORE_05211] Definition of API class ara::core::CoreException
Upstream requirements: RS_AP_00130

⌈

Kind:	class
Header file:	#include "ara/core/core_error_domain.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	`namespace ara::core`
Symbol:	CoreException
Base class:	Exception
Syntax:	`class CoreException :  public Exception {...};`
Description:	Exception type thrown for `ara::core` errors.

⌋

8.5.4.1 Public Member Functions

8.5.4.1.1 Constructors

8.5.4.1.1.1 CoreException

[SWS_CORE_05212] Definition of API function ara::core::CoreException::CoreException
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/core_error_domain.h"	
Scope:	`class ara::core::CoreException`	
Syntax:	`explicit CoreException (ErrorCode err) noexcept;`	
Parameters (in):	err	the ErrorCode
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Construct a new CoreException from an ErrorCode.	

⌋

8.6 Header: ara/core/future_error_domain.h

8.6.1 Non-Member Types

8.6.1.1 Enumeration: FutureErrc

[SWS_CORE_00400] Definition of API enum ara::core::FutureErrc

Upstream requirements: RS_AP_00130

⌈

Kind:	enumeration	
Header file:	#include "ara/core/future_error_domain.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace ara::core`	
Symbol:	FutureErrc	
Underlying type:	ara::core::ErrorDomain::CodeType	
Syntax:	`enum class FutureErrc : ara::core::ErrorDomain::CodeType {...};`	
Values:	kBrokenPromise= 101	The asynchronous task abandoned its `shared state`
	kNoState= 104	Attempt to access or without an associated `shared state`
Description:	Specifies the errors that can occur upon calling `Future::get` or `Future::GetResult`.	

⌋

8.6.2 Non-Member Functions

8.6.2.1 Other

8.6.2.1.1 GetFutureErrorDomain

[SWS_CORE_00480] Definition of API function ara::core::GetFutureErrorDomain

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future_error_domain.h"	
Scope:	`namespace ara::core`	
Syntax:	`constexpr const ErrorDomain & GetFutureErrorDomain () noexcept;`	
Return value:	const ErrorDomain &	reference to the FutureErrorDomain instance
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Obtain the reference to the single global FutureErrorDomain instance.	

⌋

8.6.2.1.2 MakeErrorCode

[SWS_CORE_00490] Definition of API function ara::core::MakeErrorCode

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future_error_domain.h"	
Scope:	`namespace ara::core`	
Syntax:	`constexpr ErrorCode MakeErrorCode (FutureErrc code, ErrorDomain::SupportDataType data) noexcept;`	
Parameters (in):	code	an enumeration value from `ara::core::FutureErrc`
	data	a vendor-defined supplementary value
Return value:	ErrorCode	the new ErrorCode instance
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Create a new ErrorCode for FutureErrorDomain with the given support data type.	

⌋

8.6.3 Class: FutureErrorDomain

[SWS_CORE_00421] Definition of API class ara::core::FutureErrorDomain

Upstream requirements: RS_AP_00130

⌈

Kind:	class
Header file:	#include "ara/core/future_error_domain.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	`namespace ara::core`
Symbol:	FutureErrorDomain
Base class:	ErrorDomain
Syntax:	`class FutureErrorDomain final : public ErrorDomain {...};`
Unique ID:	As per `ara::core::FutureErrorDomain` in [SWS_CORE_90023]
Description:	Error domain for errors originating from `ara::core::Future` and `ara::core::Promise`.

⌋

8.6.3.1 Public Member Types

8.6.3.1.1 Type Alias: Errc

[SWS_CORE_00431] Definition of API type ara::core::FutureErrorDomain::Errc

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/future_error_domain.h"
Scope:	`class ara::core::FutureErrorDomain`
Symbol:	Errc
Syntax:	`using Errc = FutureErrc;`
Description:	Alias for the error code value enumeration.

⌋

8.6.3.1.2 Type Alias: Exception

[SWS_CORE_00432] Definition of API type ara::core::FutureErrorDomain::Exception

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/future_error_domain.h"
Scope:	`class ara::core::FutureErrorDomain`
Symbol:	Exception
Syntax:	`using Exception = FutureException;`
Description:	Alias for the exception base class.

⌋

8.6.3.2 Public Member Functions

8.6.3.2.1 Special Member Functions

8.6.3.2.1.1 Default Constructor

[SWS_CORE_00441] Definition of API function ara::core::FutureErrorDomain::FutureErrorDomain

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/future_error_domain.h"
Scope:	`class ara::core::FutureErrorDomain`
Syntax:	`constexpr FutureErrorDomain () noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	Default constructor.

⌋

8.6.3.2.2 Member Functions

8.6.3.2.2.1 Message

[SWS_CORE_00443] Definition of API function ara::core::FutureErrorDomain::Message

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future_error_domain.h"	
Scope:	`class ara::core::FutureErrorDomain`	
Syntax:	`const char * Message (ErrorDomain::CodeType errorCode) const noexcept override;`	
Parameters (in):	errorCode	the error code value
Return value:	const char *	the text message, never nullptr
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Translate an error code value into a text message.	

⌋

8.6.3.2.2.2 Name

[SWS_CORE_00442] Definition of API function ara::core::FutureErrorDomain::Name

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future_error_domain.h"	
Scope:	`class ara::core::FutureErrorDomain`	
Syntax:	`const char * Name () const noexcept override;`	
Return value:	const char *	"Future"
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return the "shortname" ApApplicationErrorDomain.SN of this error domain.	

⌋

8.6.3.2.2.3 ThrowAsException

[SWS_CORE_00444] Definition of API function ara::core::FutureErrorDomain::ThrowAsException

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future_error_domain.h"	
Scope:	`class ara::core::FutureErrorDomain`	
Syntax:	`void ThrowAsException (const ErrorCode &errorCode) const noexcept(false) override;`	
Parameters (in):	errorCode	the ErrorCode instance
Return value:	None	
Exceptions:	`FutureException`	an exception containing the given `ErrorCode`
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Throw the exception type corresponding to the given ErrorCode. As per [SWS_CORE_10304], this function does not participate in overload resolution when C++ exceptions are disabled in the compiler toolchain.	

⌋

8.6.4 Class: FutureException

[SWS_CORE_00411] Definition of API class ara::core::FutureException

Upstream requirements: RS_AP_00130

⌈

Kind:	class
Header file:	#include "ara/core/future_error_domain.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	`namespace ara::core`
Symbol:	FutureException
Base class:	Exception
Syntax:	`class FutureException : public Exception {...};`
Description:	Exception type thrown by Future and Promise classes.

⌋

8.6.4.1 Public Member Functions

8.6.4.1.1 Constructors

8.6.4.1.1.1 FutureException

[SWS_CORE_00412] Definition of API function ara::core::FutureException::FutureException

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future_error_domain.h"	
Scope:	`class ara::core::FutureException`	
Syntax:	`explicit FutureException (ErrorCode err) noexcept;`	
Parameters (in):	err	the ErrorCode
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Construct a new FutureException from an ErrorCode.	

⌋

8.7 Header: ara/core/future.h

8.7.1 Non-Member Types

8.7.1.1 Enumeration: FutureStatus

[SWS_CORE_00361] Definition of API enum ara::core::FutureStatus

Upstream requirements: RS_AP_00130

⌈

Kind:	enumeration	
Header file:	#include "ara/core/future.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace ara::core`	
Symbol:	FutureStatus	
Underlying type:	std::uint8_t	
Syntax:	`enum class FutureStatus : std::uint8_t {...};`	
Values:	kReady	the `shared state` is ready
	kTimeout	the `shared state` did not become ready before the specified timeout has passed
Description:	Specifies the state of a Future as returned by wait_for() and wait_until(). These definitions are equivalent to the ones from `std::future_status`. However, no item equivalent to `std::future_status::deferred` is available here. The numerical values of the enum items are implementation-defined.	

⌋

8.7.2 Class: Future

[SWS_CORE_00321] Definition of API class ara::core::Future

Upstream requirements: RS_AP_00130

⌈

Kind:	class	
Header file:	#include "ara/core/future.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace ara::core`	
Symbol:	Future	

Syntax:	`template <typename T, typename E = ErrorCode>`
	`class Future final {...};`	

Template param:	typename T	the type of values
	typename E = ErrorCode	the type of errors

▽

△

Description:	Provides `ara::core` specific Future operations to collect the results of an asynchronous call. The implementation shall flag the condition `E != ara::core::ErrorCode` as a compile error.

⌋

8.7.2.1 Public Member Functions

8.7.2.1.1 Special Member Functions

8.7.2.1.1.1 Copy Constructor

[SWS_CORE_00334] Definition of API function ara::core::Future::Future

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	`class ara::core::Future`
Syntax:	`Future (const Future &)=delete;`
Description:	Copy constructor shall be disabled.

⌋

8.7.2.1.1.2 Default Constructor

[SWS_CORE_00322] Definition of API function ara::core::Future::Future

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	`class ara::core::Future`
Syntax:	`Future () noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	Default constructor. This function shall behave the same as the corresponding `std::future` function.

⌋

8.7.2.1.1.3 Move Constructor

[SWS_CORE_00323] Definition of API function ara::core::Future::Future

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future`	
Syntax:	`Future (Future &&other) noexcept;`	
Parameters (in):	other	the other instance
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Move constructor from another instance. This function shall behave the same as the corresponding `std::future` function.	

⌋

8.7.2.1.1.4 Copy Assignment Operator

[SWS_CORE_00335] Definition of API function ara::core::Future::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	`class ara::core::Future`
Syntax:	`Future & operator= (const Future &)=delete;`
Description:	Copy assignment operator shall be disabled.

⌋

8.7.2.1.1.5 Move Assignment Operator

[SWS_CORE_00325] Definition of API function ara::core::Future::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future`	
Syntax:	`Future & operator= (Future &&other) noexcept;`	
Parameters (in):	other	the other instance
Return value:	Future &	*this
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Move assign from another instance. This function shall behave the same as the corresponding `std::future` function.	

⌋

8.7.2.1.1.6 Destructor

[SWS_CORE_00333] Definition of API function ara::core::Future::~Future

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	`class ara::core::Future`
Syntax:	`~Future () noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	Destructor for Future objects. Abandons any `shared state`.

⌋

8.7.2.1.2 Member Functions

8.7.2.1.2.1 GetResult

[SWS_CORE_00336] Definition of API function ara::core::Future::GetResult

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future`	
Syntax:	`Result< T, E > GetResult () noexcept(std::is_nothrow_move_constructible< T >::value &&std::is_nothrow_move_constructible< E >::value);`	
Return value:	Result< T, E >	a Result as std::move(result from the `shared state`) with either a value or an error that has been put into the corresponding Promise. This can be because either: • explicit setting of the Error via Promise::SetError / Promise::SetResult or • the Promise was broken, meaning the `shared state` was abandoned by the corresponding Promise. Then the error is `kBrokenPromise` as defined in [SWS_CORE_00400].
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Errors:	FutureErrc::kBrokenPromise	rollback_semantics
		if the Promise was broken, meaning the `shared state` was abandoned by the corresponding Promise.
Violations:	`FutureInvalidViolation`	If the Future is invalid (`valid` returns false).
Description:	Get the result. Similar to get(), this call blocks until the value or an error is available. However, this call will never throw an exception.	

⌋

8.7.2.1.2.2 get

[SWS_CORE_00326] Definition of API function ara::core::Future::get

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	`class ara::core::Future`
Syntax:	`T get () noexcept(false);`

▽

△

Return value:	T	value of type T
Exceptions:	<TYPE>	an exception of the type associated with the error that has been put into the corresponding Promise. This can be because either: • explicit setting of the Error via Promise::SetError / Promise::SetResult or • the Promise was broken, meaning the shared state was abandoned by the corresponding Promise. Then the error is kBrokenPromise as defined in [SWS_CORE_00400].
	FutureException	in case the Future is invalid. The contained ErrorCode is kNoState
Exception Safety:	not exception safe	
Thread Safety:	not thread-safe	
Description:	Get the value. This function shall behave the same as the corresponding std::future function. This function shall not participate in overload resolution when C++ exceptions are disabled in the compiler toolchain.	

⌋

8.7.2.1.2.3 is_ready

[SWS_CORE_00332] Definition of API function ara::core::Future::is_ready

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Future	
Syntax:	bool is_ready () const noexcept;	
Return value:	bool	true if the Future contains a value or an error, false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	FutureInvalidViolation	If the Future is invalid (valid returns false).
Description:	Return whether the asynchronous operation has finished. If this function returns true, get(), GetResult() and the wait calls are guaranteed not to block.	

⌋

8.7.2.1.2.4 then

[SWS_CORE_00337] Definition of API function ara::core::Future::then

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Future	
Syntax:	template <typename F, typename ExecutorT> auto then (F &&func, ExecutorT &&executor) noexcept -> Future< *<see below>* >;	
Template param:	F	the type of the func argument
	ExecutorT	the type of the executor argument
Parameters (in):	func	a callable to register
	executor	the execution context in which to execute the Callable func
Return value:	Future< *<see below>* >	a new Future instance for the result of the continuation
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Violations:	FutureInvalidViolation	If the Future is invalid (valid returns false).
	FutureContinuationHasThrownViolation	If the continuation throws an exception.
Description:	Create a new Future with the content returned by func when passed the Result of this Future. func is called in the context of the provided execution context executor. valid() == false on the original future object immediately after it returns. The Callable input argument "func" takes a Result<T,E> object as parameter. This will be the Result obtained via GetResult from the Future instance itself, on which .then() is being called. The Result is passed to func as an rvalue expression. The return type of then depends on the return type of func (aka continuation). Let U be the return type of the continuation (i.e. a type equivalent to std::result_of_t<std::decay_t<F>(Result<T,E>)>). • If U is Future<T2,E2> for some types T2, E2, then the return type of then() is Future<T2,E2>. This is known as implicit Future unwrapping. • If U is Result<T2,E2> for some types T2, E2, then the return type of then() is Future<T2,E2>. This is known as implicit Result unwrapping. • Otherwise it is Future<U,E>. Continuations that are registered with Future::then() are registered in the Future F that Future::then() returns. No such continuation shall be executed after the destructor of the Future F returns. A continuation that is already being executed shall run to completion. Note: that means that the destructor of Future might block until the currently running continuation has been completed. The continuation shall not throw, except for the purpose of implementing a Violation. ▽	

▽

△

	△ Note: Exceptions can be used within the continuation, however if they do not realize a Violation, they must not escape the continuation. Note: Users who need to propagate information from closures' exceptions should translate them to an error and return an ara::core::Result or ara::core::Future from the continuation with the error stored in it.

⌋

8.7.2.1.2.5 then

[SWS_CORE_00331] Definition of API function ara::core::Future::then

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future`	

Syntax:	`template <typename F>`
	`auto then (F &&func) noexcept -> Future< <see below> >;`	

Parameters (in):	func	a callable to register
Return value:	Future< *<see below>* >	a new Future instance for the result of the continuation
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Violations:	`FutureInvalidViolation`	If the Future is invalid (`valid` returns false).
	`FutureContinuationHasThrownViolation`	If the continuation throws an exception.
Description:	Create a new Future with the content returned by func when passed the Result of this Future. func may be called in the context of this call or in the context of Promise::set_value() or Promise::SetError() or somewhere else. valid() == false on the original future object immediately after it returns. The Callable input argument "func" takes a Result<T,E> object as parameter. This will be the Result obtained via GetResult from the Future instance itself, on which .then() is being called. The Result is passed to func as an rvalue expression. The return type of then depends on the return type of func (aka continuation). Let U be the return type of the continuation (i.e. a type equivalent to std::result_of_t<std::decay_t<F>(Result<T,E>)>). • If U is Future<T2,E2> for some types T2, E2, then the return type of then() is Future<T2,E2>. This is known as implicit Future unwrapping. • If U is Result<T2,E2> for some types T2, E2, then the return type of then() is Future<T2,E2>. This is known as implicit Result unwrapping. • Otherwise it is Future<U,E>. Continuations that are registered with Future::then() are registered in the Future F that Future::then() returns. No such continuation shall be executed after the destructor of the Future F returns. A continuation that is already being executed shall run to completion. ▽	

▽

△

	△ Note: that means that the destructor of Future might block until the currently running continuation has been completed. The continuation shall not throw, except for the purpose of implementing a `Violation`. Note: Exceptions can be used within the continuation, however if they do not realize a `Violation`, they must not escape the continuation. Note: Users who need to propagate information from closures' exceptions should translate them to an error and return an `ara::core::Result` or `ara::core::Future` from the continuation with the error stored in it.

⌋

8.7.2.1.2.6 valid

[SWS_CORE_00327] Definition of API function ara::core::Future::valid

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future`	
Syntax:	`bool valid () const noexcept;`	
Return value:	bool	true if the Future is usable, false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Checks if the Future is valid, i.e. if it has a `shared state`. This function shall behave the same as the corresponding `std::future` function.	

⌋

8.7.2.1.2.7 wait

[SWS_CORE_00328] Definition of API function ara::core::Future::wait

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	`class ara::core::Future`
Syntax:	`void wait () const noexcept;`

▽

△

Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Violations:	`FutureInvalidViolation`	If the Future is invalid (`valid` returns false).
Description:	Wait for a value or an error to be available. This function shall behave the same as the corresponding `std::future` function.	

⌋

8.7.2.1.2.8 wait_for

[SWS_CORE_00329] Definition of API function ara::core::Future::wait_for

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future`	

Syntax:	`template <typename Rep, typename Period>`
	`FutureStatus wait_for (const std::chrono::duration< Rep, Period > &timeoutDuration) const noexcept;`	

Parameters (in):	timeoutDuration	maximal duration to wait for
Return value:	FutureStatus	status that indicates whether the timeout hit or if a value is available
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Violations:	`FutureInvalidViolation`	If the Future is invalid (`valid` returns false).
Description:	Wait for the given period, or until a value or an error is available. This function shall behave the same as the corresponding `std::future` function.	

⌋

8.7.2.1.2.9 wait_until

[SWS_CORE_00330] Definition of API function ara::core::Future::wait_until

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future`	

Syntax:	`template <typename Clock, typename Duration>`
	`FutureStatus wait_until (const std::chrono::time_point< Clock, Duration > &deadline) const noexcept;`	

Parameters (in):	deadline	latest point in time to wait
Return value:	FutureStatus	status that indicates whether the time was reached or if a value is available
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Violations:	`FutureInvalidViolation`	If the Future is invalid (`valid` returns false).
Description:	Wait until the given time, or until a value or an error is available. This function shall behave the same as the corresponding `std::future` function.	

⌋

8.7.3 Class: Future

[SWS_CORE_06221] Definition of API class ara::core::Future< void, E >

Upstream requirements: RS_AP_00130

⌈

Kind:	class	
Header file:	#include "ara/core/future.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace ara::core`	
Symbol:	Future< void, E >	

Syntax:	`template <typename E>`
	`class Future< void, E > final {...};`	

Template param:	typename E	the type of error
Description:	Specialization of class Future for "void" values. The implementation shall flag the condition `E != ara::core::ErrorCode` as a compile error.	

⌋

8.7.3.1 Public Member Functions

8.7.3.1.1 Special Member Functions

8.7.3.1.1.1 Copy Constructor

[SWS_CORE_06234] Definition of API function ara::core::Future< void, E >::Future

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	`class ara::core::Future< void, E >`
Syntax:	`Future (const Future &other)=delete;`
Description:	Copy constructor shall be disabled.

⌋

8.7.3.1.1.2 Default Constructor

[SWS_CORE_06222] Definition of API function ara::core::Future< void, E >::Future

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	`class ara::core::Future< void, E >`
Syntax:	`Future () noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	Default constructor. This function shall behave the same as the corresponding `std::future` function.

⌋

8.7.3.1.1.3 Move Constructor

[SWS_CORE_06223] Definition of API function ara::core::Future< void, E >::Future

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future< void, E >`	
Syntax:	`Future (Future &&other) noexcept;`	
Parameters (in):	other	the other instance
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Move constructor from another instance. This function shall behave the same as the corresponding `std::future` function.	

⌋

8.7.3.1.1.4 Copy Assignment Operator

[SWS_CORE_06235] Definition of API function ara::core::Future< void, E >::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	`class ara::core::Future< void, E >`
Syntax:	`Future & operator= (const Future &other)=delete;`
Description:	Copy assignment operator shall be disabled.

⌋

8.7.3.1.1.5 Move Assignment Operator

[SWS_CORE_06225] Definition of API function ara::core::Future< void, E >::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future< void, E >`	
Syntax:	`Future & operator= (Future &&other) noexcept;`	
Parameters (in):	other	the other instance
Return value:	Future &	*this
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Move assign from another instance. This function shall behave the same as the corresponding `std::future` function.	

⌋

8.7.3.1.1.6 Destructor

[SWS_CORE_06233] Definition of API function ara::core::Future< void, E >::~Future

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	`class ara::core::Future< void, E >`
Syntax:	`~Future () noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	Destructor for Future objects. Abandons any `shared state`.

⌋

8.7.3.1.2 Member Functions

8.7.3.1.2.1 GetResult

[SWS_CORE_06236] Definition of API function ara::core::Future< void, E >::GetResult

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future< void, E >`	
Syntax:	`Result< void, E > GetResult () noexcept(std::is_nothrow_move_constructible< E >::value);`	
Return value:	Result< void, E >	a Result as std::move(result from the `shared state`) with either a value or an error that has been put into the corresponding Promise. This can be because either: • explicit setting of the Error via Promise::SetError / Promise::SetResult or • the Promise was broken, meaning the `shared state` was abandoned by the corresponding Promise. Then the error is `kBrokenPromise` as defined in [SWS_CORE_00400]
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Errors:	FutureErrc::kBrokenPromise	rollback_semantics if the Promise was broken, meaning the `shared state` was abandoned by the corresponding Promise.
Violations:	`FutureInvalidViolation`	If the Future is invalid (`valid` returns false).
Description:	Get the result. Similar to get(), this call blocks until the value or an error is available. However, this call will never throw an exception.	

⌋

8.7.3.1.2.2 get

[SWS_CORE_06226] Definition of API function ara::core::Future< void, E >::get

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	`class ara::core::Future< void, E >`

▽

△

Syntax:	`void get () noexcept(false);`	
Return value:	None	
Exceptions:	<TYPE>	an exception of the type associated with the error that has been put into the corresponding Promise. This can be because either: ● explicit setting of the Error via Promise::SetError / Promise::SetResult or ● the Promise was broken, meaning the `shared state` was abandoned by the corresponding Promise. Then the error is `kBrokenPromise` as defined in [SWS_CORE_00400].
	FutureException	in case the Future is invalid. The contained ErrorCode is `kNoState`
Exception Safety:	not exception safe	
Thread Safety:	not thread-safe	
Description:	Get the value. This function shall behave the same as the corresponding `std::future` function. This function shall not participate in overload resolution when C++ exceptions are disabled in the compiler toolchain.	

⌋

8.7.3.1.2.3 is_ready

[SWS_CORE_06232] Definition of API function ara::core::Future< void, E >::is_ready

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future< void, E >`	
Syntax:	`bool is_ready () const noexcept;`	
Return value:	bool	true if the Future contains a value or an error, false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`FutureInvalidViolation`	If the Future is invalid (`valid` returns false).
Description:	Return whether the asynchronous operation has finished. If this function returns true, get(), GetResult() and the wait calls are guaranteed not to block.	

⌋

8.7.3.1.2.4 then

[SWS_CORE_06237] Definition of API function ara::core::Future< void, E >::then

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future< void, E >`	

Syntax:	`template <typename F, typename ExecutorT>`
	`auto then (F &&func, ExecutorT &&executor) noexcept -> Future<` *<see below>* `>;`	

Template param:	F	the type of the func argument
	ExecutorT	the type of the executor argument
Parameters (in):	func	a callable to register
	executor	the execution context in which to execute the Callable func
Return value:	Future< *<see below>* >	a new Future instance for the result of the continuation
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Violations:	`FutureInvalidViolation`	If the Future is invalid (`valid` returns false).
	`FutureContinuationHasThrownViolation`	If the continuation throws an exception.
Description:	Create a new Future with the content returned by func when passed the Result of this Future. func is called in the context of the provided execution context executor. valid() == false on the original future object immediately after it returns. The Callable input argument "func" takes a Result<void,E> object as parameter. This will be the Result obtained via GetResult from the Future instance itself, on which .then() is being called. The Result is passed to func as an rvalue expression. The return type of then depends on the return type of func (aka continuation). Let U be the return type of the continuation (i.e. a type equivalent to std::result_of_t<std::decay_t<F>(Result<void,E>)>). • If U is Future<T2,E2> for some types T2, E2, then the return type of then() is Future<T2,E2>. This is known as implicit Future unwrapping. • If U is Result<T2,E2> for some types T2, E2, then the return type of then() is Future<T2,E2>. This is known as implicit Result unwrapping. • Otherwise it is Future<U,E>. Continuations that are registered with Future::then() are registered in the Future F that Future::then() returns. No such continuation shall be executed after the destructor of the Future F returns. A continuation that is already being executed shall run to completion. Note: that means that the destructor of Future might block until the currently running continuation has been completed. The continuation shall not throw, except for the purpose of implementing a `Violation`. ▽	

▽

△

	△ Note: Exceptions can be used within the continuation, however if they do not realize a Violation, they must not escape the continuation. Note: Users who need to propagate information from closures' exceptions should translate them to an error and return an ara::core::Result or ara::core::Future from the continuation with the error stored in it.

⌋

8.7.3.1.2.5 then

[SWS_CORE_06231] Definition of API function ara::core::Future< void, E >::then

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future< void, E >`	

Syntax:	`template <typename F>`
	`auto then (F &&func) noexcept -> Future< <see below> >;`	

Parameters (in):	func	a callable to register
Return value:	Future< *<see below>* >	a new Future instance for the result of the continuation
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Violations:	FutureInvalidViolation	If the Future is invalid (valid returns false).
	FutureContinuationHasThrownViolation	If the continuation throws an exception.
Description:	Create a new Future with the content returned by func when passed the Result of this Future. func may be called in the context of this call or in the context of Promise::set_value() or Promise::SetError() or somewhere else. valid() == false on the original future object immediately after it returns. The Callable input argument "func" takes a Result<void,E> object as parameter. This will be the Result obtained via GetResult from the Future instance itself, on which .then() is being called. The Result is passed to func as an rvalue expression. The return type of then depends on the return type of func (aka continuation). Let U be the return type of the continuation (i.e. a type equivalent to std::result_of_t<std::decay_t<F>(Result<void,E>)>). • If U is Future<T2,E2> for some types T2, E2, then the return type of then() is Future<T2,E2>. This is known as implicit Future unwrapping. • If U is Result<T2,E2> for some types T2, E2, then the return type of then() is Future<T2,E2>. This is known as implicit Result unwrapping. • Otherwise it is Future<U,E>. Continuations that are registered with Future::then() are registered in the Future F that Future::then() returns. No such continuation shall be executed after the destructor of the Future F returns. A continuation that is already being executed shall run to completion. ▽	

▽

△

	△ Note: that means that the destructor of Future might block until the currently running continuation has been completed. The continuation shall not throw, except for the purpose of implementing a Violation. Note: Exceptions can be used within the continuation, however if they do not realize a Violation, they must not escape the continuation. Note: Users who need to propagate information from closures' exceptions should translate them to an error and return an ara::core::Result or ara::core::Future from the continuation with the error stored in it.

⌋

8.7.3.1.2.6 valid

[SWS_CORE_06227] Definition of API function ara::core::Future< void, E >::valid

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future< void, E >`	
Syntax:	`bool valid () const noexcept;`	
Return value:	bool	true if the Future is usable, false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Checks if the Future is valid, i.e. if it has a shared state. This function shall behave the same as the corresponding `std::future` function.	

⌋

8.7.3.1.2.7 wait

[SWS_CORE_06228] Definition of API function ara::core::Future< void, E >::wait

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	`class ara::core::Future< void, E >`
Syntax:	`void wait () const noexcept;`

▽

△

Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Violations:	FutureInvalidViolation	If the Future is invalid (valid returns false).
Description:	Wait for a value or an error to be available. This function shall behave the same as the corresponding std::future function.	

⌋

8.7.3.1.2.8 wait_for

[SWS_CORE_06229] Definition of API function ara::core::Future< void, E >::wait_for

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future< void, E >`	

Syntax:	`template <typename Rep, typename Period>`
	`FutureStatus wait_for (const std::chrono::duration< Rep, Period > &timeoutDuration) const noexcept;`	

Parameters (in):	timeoutDuration	maximal duration to wait for
Return value:	FutureStatus	status that indicates whether the timeout hit or if a value is available
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Violations:	FutureInvalidViolation	If the Future is invalid (valid returns false).
Description:	Wait for the given period, or until a value or an error is available. This function shall behave the same as the corresponding std::future function.	

⌋

8.7.3.1.2.9 wait_until

[SWS_CORE_06230] Definition of API function ara::core::Future< void, E >::wait_until

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future< void, E >`	

Syntax:	`template <typename Clock, typename Duration>`
	`FutureStatus wait_until (const std::chrono::time_point< Clock, Duration > &deadline) const noexcept;`	

Parameters (in):	deadline	latest point in time to wait
Return value:	FutureStatus	status that indicates whether the time was reached or if a value is available
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Violations:	`FutureInvalidViolation`	If the Future is invalid (`valid` returns false).
Description:	Wait until the given time, or until a value or an error is available. This function shall behave the same as the corresponding `std::future` function.	

⌋

8.7.4 Class: Promise

[SWS_CORE_00340] Definition of API class ara::core::Promise

Upstream requirements: RS_AP_00130

⌈

Kind:	class	
Header file:	#include "ara/core/future.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace ara::core`	
Symbol:	Promise	

Syntax:	`template <typename T, typename E = ErrorCode>`
	`class Promise final {...};`	

Template param:	typename T	the type of value
	typename E = ErrorCode	the type of error
Description:	ara::core specific variant of `std::promise` class The implementation shall flag the condition `E != ara::core::ErrorCode` as a compile error.	

⌋

8.7.4.1 Public Member Functions

8.7.4.1.1 Special Member Functions

8.7.4.1.1.1 Copy Constructor

[SWS_CORE_00350] Definition of API function ara::core::Promise::Promise

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	`class ara::core::Promise`
Syntax:	`Promise (const Promise &)=delete;`
Description:	Copy constructor shall be disabled.

⌋

8.7.4.1.1.2 Default Constructor

[SWS_CORE_00341] Definition of API function ara::core::Promise::Promise

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	`class ara::core::Promise`
Syntax:	`Promise () noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	Default constructor. This function shall behave the same as the corresponding `std::promise` function.

⌋

8.7.4.1.1.3 Move Constructor

[SWS_CORE_00342] Definition of API function ara::core::Promise::Promise

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise`	
Syntax:	`Promise (Promise &&other) noexcept;`	
Parameters (in):	other	the other instance
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Move constructor. This function shall behave the same as the corresponding `std::promise` function.	

⌋

8.7.4.1.1.4 Copy Assignment Operator

[SWS_CORE_00351] Definition of API function ara::core::Promise::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	`class ara::core::Promise`
Syntax:	`Promise & operator= (const Promise &)=delete;`
Description:	Copy assignment operator shall be disabled.

⌋

8.7.4.1.1.5 Move Assignment Operator

[SWS_CORE_00343] Definition of API function ara::core::Promise::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise`	
Syntax:	`Promise & operator= (Promise &&other) noexcept;`	
Parameters (in):	other	the other instance
Return value:	Promise &	*this
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Move assignment. Abandons any `shared state` and then as if Promise(std::move(other)).swap(*this).	

⌋

8.7.4.1.1.6 Destructor

[SWS_CORE_00349] Definition of API function ara::core::Promise::~Promise

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	`class ara::core::Promise`
Syntax:	`~Promise () noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	Destructor for Promise objects. Abandons any `shared state`.

⌋

8.7.4.1.2 Member Functions

8.7.4.1.2.1 SetError

[SWS_CORE_00353] Definition of API function ara::core::Promise::SetError

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise`	
Syntax:	`void SetError (E &&error) noexcept(std::is_nothrow_move_constructible< E >::value);`	
Parameters (in):	error	the error to store
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Violations:	`PromiseAlreadySatisfiedViolation`	If the `shared state` already has a stored value or error.
	`PromiseNoSharedStateToSetViolation`	If *this has no `shared state`.
Description:	Move an error into the `shared state` and make the `shared state` ready.	

⌋

8.7.4.1.2.2 SetError

[SWS_CORE_00354] Definition of API function ara::core::Promise::SetError

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise`	
Syntax:	`void SetError (const E &error) noexcept(std::is_nothrow_copy_constructible< E >::value);`	
Parameters (in):	error	the error to store
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Violations:	`PromiseAlreadySatisfiedViolation`	If the `shared state` already has a stored value or error.

▽

△

	PromiseNoSharedStateToSetViolation	If *this has no shared state.
Description:	Copy an error into the shared state and make the shared state ready.	

⌋

8.7.4.1.2.3 SetResult

[SWS_CORE_00356] Definition of API function ara::core::Promise::SetResult

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Promise	
Syntax:	void SetResult (Result< T, E > &&result) noexcept(std::is_nothrow_move_constructible< T >::value &&std::is_nothrow_move_constructible< E >::value);	
Parameters (in):	result	the result to store
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Violations:	PromiseAlreadySatisfiedViolation	If the shared state already has a stored value or error.
	PromiseNoSharedStateToSetViolation	If *this has no shared state.
Description:	Move a Result into the shared state and make the shared state ready.	

⌋

8.7.4.1.2.4 SetResult

[SWS_CORE_00355] Definition of API function ara::core::Promise::SetResult

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	class ara::core::Promise

▽

△

Syntax:	`void SetResult (const Result< T, E > &result) noexcept(std::is_nothrow_copy_constructible< T >::value &&std::is_nothrow_copy_constructible< E >::value);`	
Parameters (in):	result	the result to store
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Violations:	`PromiseAlreadySatisfiedViolation`	If the `shared state` already has a stored value or error.
	`PromiseNoSharedStateToSetViolation`	If *this has no `shared state`.
Description:	Copy a Result into the `shared state` and make the `shared state` ready.	

⌋

8.7.4.1.2.5 get_future

[SWS_CORE_00344] Definition of API function ara::core::Promise::get_future

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise`	
Syntax:	`Future< T, E > get_future () noexcept;`	
Return value:	Future< T, E >	a Future with the same `shared state` as *this
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Violations:	`FutureAlreadyRetrievedViolation`	If the function is called more than once on the same `shared state`.
	`PromiseWithNoSharedStateViolation`	If *this has no `shared state`.
Description:	Return an associated Future with the same `shared state` as *this. The returned Future is set as soon as this Promise receives the result, value, or an error. This fuction must only be called once as it is not allowed to have multiple Futures per Promise.	

⌋

8.7.4.1.2.6 set_value

[SWS_CORE_00345] Definition of API function ara::core::Promise::set_value

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise`	
Syntax:	`void set_value (const T &value) noexcept(std::is_nothrow_copy_constructible< T >::value);`	
Parameters (in):	value	the value to store
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Violations:	`PromiseAlreadySatisfiedViolation`	If the `shared state` already has a stored value or error.
	`PromiseNoSharedStateToSetViolation`	If *this has no `shared state`.
Description:	Copy a value into the `shared state` and make the `shared state` ready.	

⌋

8.7.4.1.2.7 set_value

[SWS_CORE_00346] Definition of API function ara::core::Promise::set_value

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise`	
Syntax:	`void set_value (T &&value) noexcept(std::is_nothrow_move_constructible< T >::value);`	
Parameters (in):	value	the value to store
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Violations:	`PromiseAlreadySatisfiedViolation`	If the `shared state` already has a stored value or error.
	`PromiseNoSharedStateToSetViolation`	If *this has no `shared state`.
Description:	Move a value into the `shared state` and make the `shared state` ready.	

⌋

8.7.4.1.2.8 swap

[SWS_CORE_00352] Definition of API function ara::core::Promise::swap

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise`	
Syntax:	`void swap (Promise &other) noexcept;`	
Parameters (inout):	other	the other instance
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Swap the contents of this instance with another one's. This function shall behave the same as the corresponding `std::promise` function.	

⌋

8.7.5 Class: Promise

[SWS_CORE_06340] Definition of API class ara::core::Promise< void, E >

Upstream requirements: RS_AP_00130

⌈

Kind:	class	
Header file:	#include "ara/core/future.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace ara::core`	
Symbol:	Promise< void, E >	

Syntax:	`template <typename E>`
	`class Promise< void, E > final {...};`	

Template param:	typename E	the type of error
Description:	Specialization of class Promise for "void" values. The implementation shall flag the condition `E != ara::core::ErrorCode` as a compile error.	

⌋

8.7.5.1 Public Member Functions

8.7.5.1.1 Special Member Functions

8.7.5.1.1.1 Move Constructor

[SWS_CORE_06342] Definition of API function ara::core::Promise< void, E >::Promise

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise< void, E >`	
Syntax:	`Promise (Promise &&other) noexcept;`	
Parameters (in):	other	the other instance
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Move constructor. This function shall behave the same as the corresponding `std::promise` function.	

⌋

8.7.5.1.1.2 Default Constructor

[SWS_CORE_06341] Definition of API function ara::core::Promise< void, E >::Promise

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	`class ara::core::Promise< void, E >`
Syntax:	`Promise () noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	Default constructor. This function shall behave the same as the corresponding `std::promise` function.

⌋

8.7.5.1.1.3 Copy Constructor

[SWS_CORE_06350] Definition of API function ara::core::Promise< void, E >::Promise

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	`class ara::core::Promise< void, E >`
Syntax:	`Promise (const Promise &)=delete;`
Description:	Copy constructor shall be disabled.

⌋

8.7.5.1.1.4 Copy Assignment Operator

[SWS_CORE_06351] Definition of API function ara::core::Promise< void, E >::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	`class ara::core::Promise< void, E >`
Syntax:	`Promise & operator= (const Promise &)=delete;`
Description:	Copy assignment operator shall be disabled.

⌋

8.7.5.1.1.5 Move Assignment Operator

[SWS_CORE_06343] Definition of API function ara::core::Promise< void, E >::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise< void, E >`	
Syntax:	`Promise & operator= (Promise &&other) noexcept;`	
Parameters (in):	other	the other instance
Return value:	Promise &	*this
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Move assignment. Abandons any `shared state` and then as if Promise(std::move(other)).swap(*this).	

⌋

8.7.5.1.1.6 Destructor

[SWS_CORE_06349] Definition of API function ara::core::Promise< void, E >::~Promise

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	`class ara::core::Promise< void, E >`
Syntax:	`~Promise () noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	Destructor for Promise objects. Abandons any `shared state`.

⌋

8.7.5.1.2 Member Functions

8.7.5.1.2.1 SetError

[SWS_CORE_06353] Definition of API function ara::core::Promise< void, E >::SetError

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise< void, E >`	
Syntax:	`void SetError (E &&error) noexcept(std::is_nothrow_move_constructible< E >::value);`	
Parameters (in):	error	the error to store
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Violations:	`PromiseAlreadySatisfiedViolation`	If the `shared state` already has a stored value or error.
	`PromiseNoSharedStateToSetViolation`	If *this has no `shared state`.
Description:	Move an error into the `shared state` and make the `shared state` ready.	

⌋

8.7.5.1.2.2 SetError

[SWS_CORE_06354] Definition of API function ara::core::Promise< void, E >::SetError

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise< void, E >`	
Syntax:	`void SetError (const E &error) noexcept(std::is_nothrow_copy_constructible< E >::value);`	
Parameters (in):	error	the error to store
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	

▽

△

Violations:	PromiseAlreadySatisfiedViolation	If the shared state already has a stored value or error.
	PromiseNoSharedStateToSetViolation	If *this has no shared state.
Description:	Copy an error into the shared state and make the shared state ready.	

⌋

8.7.5.1.2.3 SetResult

[SWS_CORE_06356] Definition of API function ara::core::Promise< void, E >::SetResult

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise< void, E >`	
Syntax:	`void SetResult (Result< void, E > &&result) noexcept(std::is_nothrow_move_constructible< E >::value);`	
Parameters (in):	result	the result to store
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Violations:	PromiseAlreadySatisfiedViolation	If the shared state already has a stored value or error.
	PromiseNoSharedStateToSetViolation	If *this has no shared state.
Description:	Move a Result into the shared state and make the shared state ready.	

⌋

8.7.5.1.2.4 SetResult

[SWS_CORE_06355] Definition of API function ara::core::Promise< void, E >::Set Result

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise< void, E >`	
Syntax:	`void SetResult (const Result< void, E > &result) noexcept(std::is_nothrow_copy_constructible< E >::value);`	
Parameters (in):	result	the result to store
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Violations:	`PromiseAlreadySatisfiedViolation`	If the `shared state` already has a stored value or error.
	`PromiseNoSharedStateToSetViolation`	If *this has no `shared state`.
Description:	Copy a Result into the `shared state` and make the `shared state` ready.	

⌋

8.7.5.1.2.5 get_future

[SWS_CORE_06344] Definition of API function ara::core::Promise< void, E >::get_future

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise< void, E >`	
Syntax:	`Future< void, E > get_future () noexcept;`	
Return value:	Future< void, E >	a Future with the same `shared state` as *this
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Violations:	`FutureAlreadyRetrievedViolation`	If the function is called more than once on the same `shared state`.
	`PromiseWithNoSharedStateViolation`	If *this has no `shared state`.

▽

△

Description:	Return an associated Future with the same shared state as *this. The returned Future is set as soon as this Promise receives the result, value, or an error. This fuction must only be called once as it is not allowed to have multiple Futures per Promise.

⌋

8.7.5.1.2.6 set_value

[SWS_CORE_06345] Definition of API function ara::core::Promise< void, E >::set_value

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise< void, E >`	
Syntax:	`void set_value () noexcept;`	
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`PromiseAlreadySatisfiedViolation`	If the shared state already has a stored value or error.
	`PromiseNoSharedStateToSetViolation`	If *this has no shared state.
Description:	Set the shared state value and make the shared state ready.	

⌋

8.7.5.1.2.7 swap

[SWS_CORE_06352] Definition of API function ara::core::Promise< void, E >::swap

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	`class ara::core::Promise< void, E >`
Syntax:	`void swap (Promise &other) noexcept;`

▽

△

Parameters (inout):	other	the other instance
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Swap the contents of this instance with another one's. This function shall behave the same as the corresponding `std::promise` function.	

⌋

8.8 Header: ara/core/array.h

8.8.1 Non-Member Functions

8.8.1.1 Other

8.8.1.1.1 get

[SWS_CORE_01282] Definition of API function ara::core::get

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <std::size_t I, typename T, std::size_t N>`
	`constexpr T & get (Array< T, N > &a) noexcept;`	

Template param:	I	the index into the Array whose element is desired
	T	the type of element in the Array
	N	the number of elements in the Array
Parameters (in):	a	the Array
Return value:	T &	a reference to the Ith element of the Array
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Overload of std::get for an lvalue mutable ara::core::Array. The implementation shall flag the condition I >= N as a compile error.	

⌋

8.8.1.1.2 get

[SWS_CORE_01283] Definition of API function ara::core::get

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <std::size_t I, typename T, std::size_t N>`
	`constexpr T && get (Array< T, N > &&a) noexcept;`	

Template param:	I	the index into the Array whose element is desired
	T	the type of element in the Array
	N	the number of elements in the Array
Parameters (in):	a	the Array
Return value:	T &&	an rvalue reference to the Ith element of the Array
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Overload of std::get for an rvalue ara::core::Array. The implementation shall flag the condition I >= N as a compile error.	

⌋

8.8.1.1.3 get

[SWS_CORE_01284] Definition of API function ara::core::get

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <std::size_t I, typename T, std::size_t N>`
	`constexpr T const & get (const Array< T, N > &a) noexcept;`	

Template param:	I	the index into the Array whose element is desired
	T	the type of element in the Array
	N	the number of elements in the Array
Parameters (in):	a	the Array
Return value:	T const &	a const_reference to the Ith element of the Array
Exception Safety:	exception safe	
Thread Safety:	thread-safe	

▽

△

Description:	Overload of std::get for an lvalue const ara::core::Array. The implementation shall flag the condition I >= N as a compile error.

⌋

8.8.1.1.4 operator!=

[SWS_CORE_01291] Definition of API function ara::core::operator!=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T, std::size_t N>`
	`bool operator!= (const Array< T, N > &lhs, const Array< T, N > &rhs) noexcept(noexcept(std::declval< T & >() !=std::declval< T & >()));`	

Template param:	T	the type of element in the Array
	N	the number of elements in the Array
Parameters (in):	lhs	the left-hand side of the comparison
	rhs	the right-hand side of the comparison
Return value:	bool	true if the Arrays are non-equal, false otherwise
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return true if the two Arrays have non-equal content.	

⌋

8.8.1.1.5 operator<

[SWS_CORE_01292] Definition of API function ara::core::operator<

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/array.h"
Scope:	`namespace ara::core`

▽

△

Syntax:	template <typename T, std::size_t N> bool operator< (const Array< T, N > &lhs, const Array< T, N > &rhs) noexcept(noexcept(std::declval< T & >()< std::declval< T & >()));	
Template param:	T	the type of element in the Array
	N	the number of elements in the Array
Parameters (in):	lhs	the left-hand side of the comparison
	rhs	the right-hand side of the comparison
Return value:	bool	true if lhs is less than rhs, false otherwise
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return true if the contents of lhs are lexicographically less than the contents of rhs.	

⌋

8.8.1.1.6 operator<=

[SWS_CORE_01294] Definition of API function ara::core::operator<=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	namespace ara::core	
Syntax:	template <typename T, std::size_t N> bool operator<= (const Array< T, N > &lhs, const Array< T, N > &rhs) noexcept(noexcept(std::declval< T & >()<=std::declval< T & >()));	
Template param:	T	the type of element in the Array
	N	the number of elements in the Array
Parameters (in):	lhs	the left-hand side of the comparison
	rhs	the right-hand side of the comparison
Return value:	bool	true if lhs is less than or equal to rhs, false otherwise
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return true if the contents of lhs are lexicographically less than or equal to the contents of rhs.	

⌋

8.8.1.1.7 operator==

[SWS_CORE_01290] Definition of API function ara::core::operator==

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T, std::size_t N>`
	`bool operator== (const Array< T, N > &lhs, const Array< T, N > &rhs) noexcept(noexcept(std::declval< T & >()==std::declval< T & >()));`	

Template param:	T	the type of element in the Array
	N	the number of elements in the Array
Parameters (in):	lhs	the left-hand side of the comparison
	rhs	the right-hand side of the comparison
Return value:	bool	true if the Arrays are equal, false otherwise
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return true if the two Arrays have equal content.	

⌋

8.8.1.1.8 operator>

[SWS_CORE_01293] Definition of API function ara::core::operator>

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T, std::size_t N>`
	`bool operator> (const Array< T, N > &lhs, const Array< T, N > &rhs) noexcept(noexcept(std::declval< T & >() > std::declval< T & >()));`	

Template param:	T	the type of elemenr in the Array
	N	the number of elements in the Array
Parameters (in):	lhs	the left-hand side of the comparison
	rhs	the right-hand side of the comparison
Return value:	bool	true if rhs is less than lhs, false otherwise
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	

▽

△

Description:	Return true if the contents of rhs are lexicographically less than the contents of lhs.

⌋

8.8.1.1.9 operator>=

[SWS_CORE_01295] Definition of API function ara::core::operator>=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T, std::size_t N>`
	`bool operator>= (const Array< T, N > &lhs, const Array< T, N > &rhs) noexcept(noexcept(std::declval< T & >() >=std::declval< T & >()));`	

Template param:	T	the type of element in the Array
	N	the number of elements in the Array
Parameters (in):	lhs	the left-hand side of the comparison
	rhs	the right-hand side of the comparison
Return value:	bool	true if rhs is less than or equal to lhs, false otherwise
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return true if the contents of rhs are lexicographically less than or equal to the contents of lhs.	

⌋

8.8.1.1.10 swap

[SWS_CORE_01296] Definition of API function ara::core::swap

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/array.h"
Scope:	`namespace ara::core`

Syntax:	`template <typename T, std::size_t N>`
	`void swap (Array< T, N > &lhs, Array< T, N > &rhs) noexcept(noexcept(lhs.swap(rhs)));`

▽

△

Template param:	T	the type of element in the Arrays
	N	the number of elements in the Arrays
Parameters (in):	lhs	the left-hand side of the call
	rhs	the right-hand side of the call
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Overload of std::swap for ara::core::Array.	

⌋

8.8.2 Class: Array

[SWS_CORE_01201] Definition of API class ara::core::Array

Upstream requirements: RS_AP_00130

⌈

Kind:	class	
Header file:	#include "ara/core/array.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace ara::core`	
Symbol:	Array	

Syntax:	`template <typename T, std::size_t N>`
	`class Array final {...};`	

Template param:	typename T	the type of element in the array
	std::size_t N	the number of elements in the array
Description:	Encapsulation of fixed size arrays.	

⌋

8.8.2.1 Public Member Types

8.8.2.1.1 Type Alias: const_iterator

[SWS_CORE_01213] Definition of API type ara::core::Array::const_iterator

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/array.h"
Scope:	`class ara::core::Array`
Symbol:	const_iterator
Syntax:	`using const_iterator = const T*;`
Description:	The type of a const_iterator to elements.

⌋

8.8.2.1.2 Type Alias: const_pointer

[SWS_CORE_01218] Definition of API type ara::core::Array::const_pointer

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/array.h"
Scope:	`class ara::core::Array`
Symbol:	const_pointer
Syntax:	`using const_pointer = const T*;`
Description:	Alias type for a pointer to a const element.

⌋

8.8.2.1.3 Type Alias: const_reference

[SWS_CORE_01211] Definition of API type ara::core::Array::const_reference

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/array.h"
Scope:	`class ara::core::Array`
Symbol:	const_reference
Syntax:	`using const_reference = const T&;`
Description:	Alias type for a const_reference to an element.

⌋

8.8.2.1.4 Type Alias: const_reverse_iterator

[SWS_CORE_01220] Definition of API type ara::core::Array::const_reverse_iterator

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/array.h"
Scope:	`class ara::core::Array`
Symbol:	const_reverse_iterator
Syntax:	`using const_reverse_iterator = std::reverse_iterator<const_iterator>;`
Description:	The type of a const_reverse_iterator to elements.

⌋

8.8.2.1.5 Type Alias: difference_type

[SWS_CORE_01215] Definition of API type ara::core::Array::difference_type

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/array.h"
Scope:	`class ara::core::Array`
Symbol:	difference_type
Syntax:	`using difference_type = std::ptrdiff_t;`
Description:	Alias for the type of parameters that indicate a difference of indexes into the Array.

⌋

8.8.2.1.6 Type Alias: iterator

[SWS_CORE_01212] Definition of API type ara::core::Array::iterator

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/array.h"
Scope:	`class ara::core::Array`
Symbol:	iterator
Syntax:	`using iterator = T*;`
Description:	The type of an iterator to elements.

⌋

8.8.2.1.7 Type Alias: pointer

[SWS_CORE_01217] Definition of API type ara::core::Array::pointer

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/array.h"
Scope:	`class ara::core::Array`
Symbol:	pointer
Syntax:	`using pointer = T*;`
Description:	Alias type for a pointer to an element.

⌋

8.8.2.1.8 Type Alias: reference

[SWS_CORE_01210] Definition of API type ara::core::Array::reference

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/array.h"
Scope:	`class ara::core::Array`
Symbol:	reference
Syntax:	`using reference = T&;`
Description:	Alias type for a reference to an element.

⌋

8.8.2.1.9 Type Alias: reverse_iterator

[SWS_CORE_01219] Definition of API type ara::core::Array::reverse_iterator

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/array.h"
Scope:	`class ara::core::Array`
Symbol:	reverse_iterator
Syntax:	`using reverse_iterator = std::reverse_iterator<iterator>;`
Description:	The type of a reverse_iterator to elements.

⌋

8.8.2.1.10 Type Alias: size_type

[SWS_CORE_01214] Definition of API type ara::core::Array::size_type

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/array.h"
Scope:	`class ara::core::Array`
Symbol:	size_type
Syntax:	`using size_type = std::size_t;`
Description:	Alias for the type of parameters that indicate an index into the Array.

⌋

8.8.2.1.11 Type Alias: value_type

[SWS_CORE_01216] Definition of API type ara::core::Array::value_type

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/array.h"
Scope:	`class ara::core::Array`
Symbol:	value_type
Syntax:	`using value_type = T;`
Description:	Alias for the type of elements in this Array.

⌋

8.8.2.2 Public Member Functions

8.8.2.2.1 Member Functions

8.8.2.2.1.1 at

[SWS_CORE_01274] Definition of API function ara::core::Array::at

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`class ara::core::Array`	
Syntax:	`constexpr const_reference at (size_type n) const noexcept;`	
Parameters (in):	n	the index into this Array
Return value:	const_reference	the const_reference
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`ArrayAccessOutOfRangeViolation`	If n is not within the range of the array.
Description:	Return a const_reference to the n-th element of this Array, with bound checking.	

⌋

8.8.2.2.1.2 at

[SWS_CORE_01273] Definition of API function ara::core::Array::at

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`class ara::core::Array`	
Syntax:	`reference at (size_type n) noexcept;`	
Parameters (in):	n	the index into this Array
Return value:	reference	the reference
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`ArrayAccessOutOfRangeViolation`	If n is not within the range of the array.
Description:	Return a reference to the n-th element of this Array, with bound checking.	

⌋

8.8.2.2.1.3 back

[SWS_CORE_01269] Definition of API function ara::core::Array::back

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`class ara::core::Array`	
Syntax:	`reference back () noexcept;`	
Return value:	reference	the reference
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a reference to the last element of this Array. Calling this function on an empty array shall be a compile-time error.	

⌋

8.8.2.2.1.4 back

[SWS_CORE_01270] Definition of API function ara::core::Array::back

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`class ara::core::Array`	
Syntax:	`constexpr const_reference back () const noexcept;`	
Return value:	const_reference	the reference
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a const_reference to the last element of this Array. Calling this function on an empty array shall be a compile-time error.	

⌋

8.8.2.2.1.5 begin

[SWS_CORE_01250] Definition of API function ara::core::Array::begin

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`class ara::core::Array`	
Syntax:	`iterator begin () noexcept;`	
Return value:	iterator	the iterator
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return an iterator pointing to the first element of this Array. De-referencing the returned iterator on an empty array is undefined behavior.	

⌋

8.8.2.2.1.6 begin

[SWS_CORE_01251] Definition of API function ara::core::Array::begin

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`class ara::core::Array`	
Syntax:	`const_iterator begin () const noexcept;`	
Return value:	const_iterator	the const_iterator
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a const_iterator pointing to the first element of this Array. De-referencing the returned iterator on an empty array is undefined behavior.	

⌋

8.8.2.2.1.7 cbegin

[SWS_CORE_01258] Definition of API function ara::core::Array::cbegin

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`class ara::core::Array`	
Syntax:	`const_iterator cbegin () const noexcept;`	
Return value:	const_iterator	the const_iterator
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a const_iterator pointing to the first element of this Array.	

⌋

8.8.2.2.1.8 cend

[SWS_CORE_01259] Definition of API function ara::core::Array::cend

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`class ara::core::Array`	
Syntax:	`const_iterator cend () const noexcept;`	
Return value:	const_iterator	the const_iterator
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a const_iterator pointing past the last element of this Array.	

⌋

8.8.2.2.1.9 crbegin

[SWS_CORE_01260] Definition of API function ara::core::Array::crbegin

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`class ara::core::Array`	
Syntax:	`const_reverse_iterator crbegin () const noexcept;`	
Return value:	const_reverse_iterator	the const_reverse_iterator
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a const_reverse_iterator pointing to the last element of this Array.	

⌋

8.8.2.2.1.10 crend

[SWS_CORE_01261] Definition of API function ara::core::Array::crend
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`class ara::core::Array`	
Syntax:	`const_reverse_iterator crend () const noexcept;`	
Return value:	const_reverse_iterator	the const_reverse_iterator
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a const_reverse_iterator pointing past the first element of this Array.	

⌋

8.8.2.2.1.11 data

[SWS_CORE_01272] Definition of API function ara::core::Array::data
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`class ara::core::Array`	
Syntax:	`const_pointer data () const noexcept;`	
Return value:	const_pointer	the const_pointer
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a const_pointer to the first element of this Array. This function shall return `nullptr` in case the size of the array is 0.	

⌋

8.8.2.2.1.12 data

[SWS_CORE_01271] Definition of API function ara::core::Array::data

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`class ara::core::Array`	
Syntax:	`pointer data () noexcept;`	
Return value:	pointer	the pointer
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a pointer to the first element of this Array. This function shall return `nullptr` in case the size of the array is 0.	

⌋

8.8.2.2.1.13 empty

[SWS_CORE_01264] Definition of API function ara::core::Array::empty

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`class ara::core::Array`	
Syntax:	`constexpr bool empty () const noexcept;`	
Return value:	bool	true if this Array contains 0 elements, false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return whether this Array is empty.	

⌋

8.8.2.2.1.14 end

[SWS_CORE_01253] Definition of API function ara::core::Array::end

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`class ara::core::Array`	
Syntax:	`const_iterator end () const noexcept;`	
Return value:	const_iterator	the const_iterator
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a const_iterator pointing past the last element of this Array.	

⌋

8.8.2.2.1.15 end

[SWS_CORE_01252] Definition of API function ara::core::Array::end

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`class ara::core::Array`	
Syntax:	`iterator end () noexcept;`	
Return value:	iterator	the iterator
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return an iterator pointing past the last element of this Array.	

⌋

8.8.2.2.1.16 fill

[SWS_CORE_01241] Definition of API function ara::core::Array::fill

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`class ara::core::Array`	
Syntax:	`void fill (const T &u) noexcept(std::is_nothrow_copy_assignable< T >::value);`	
Parameters (in):	u	the value
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	Assign the given value to all elements of this Array.	

⌋

8.8.2.2.1.17 front

[SWS_CORE_01268] Definition of API function ara::core::Array::front

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`class ara::core::Array`	
Syntax:	`constexpr const_reference front () const noexcept;`	
Return value:	const_reference	the reference
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a const_reference to the first element of this Array. Calling this function on an empty array shall be a compile-time error.	

⌋

8.8.2.2.1.18 front

[SWS_CORE_01267] Definition of API function ara::core::Array::front

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`class ara::core::Array`	
Syntax:	`reference front () noexcept;`	
Return value:	reference	the reference
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a reference to the first element of this Array. Calling this function on an empty array shall be a compile-time error.	

⌋

8.8.2.2.1.19 max_size

[SWS_CORE_01263] Definition of API function ara::core::Array::max_size

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`class ara::core::Array`	
Syntax:	`constexpr size_type max_size () const noexcept;`	
Return value:	size_type	N
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return the maximum number of elements supported by this Array.	

⌋

8.8.2.2.1.20 operator[]

[SWS_CORE_01266] Definition of API function ara::core::Array::operator[]

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`class ara::core::Array`	
Syntax:	`constexpr const_reference operator[] (size_type n) const noexcept;`	
Parameters (in):	n	the index into this Array
Return value:	const_reference	the const_reference
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a const_reference to the n-th element of this Array. Accessing a non-existing element through this operation is undefined behavior. Use the function `at` for checked access to the elements.	

⌋

8.8.2.2.1.21 operator[]

[SWS_CORE_01265] Definition of API function ara::core::Array::operator[]

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`class ara::core::Array`	
Syntax:	`reference operator[] (size_type n) noexcept;`	
Parameters (in):	n	the index into this Array
Return value:	reference	the reference
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a reference to the n-th element of this Array. Accessing a non-existing element through this operation is undefined behavior. Use the function `at` for checked access to the elements.	

⌋

8.8.2.2.1.22 rbegin

[SWS_CORE_01254] Definition of API function ara::core::Array::rbegin

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`class ara::core::Array`	
Syntax:	`reverse_iterator rbegin () noexcept;`	
Return value:	reverse_iterator	the reverse_iterator
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a reverse_iterator pointing to the last element of this Array.	

⌋

8.8.2.2.1.23 rbegin

[SWS_CORE_01255] Definition of API function ara::core::Array::rbegin

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`class ara::core::Array`	
Syntax:	`const_reverse_iterator rbegin () const noexcept;`	
Return value:	const_reverse_iterator	the const_reverse_iterator
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a const_reverse_iterator pointing to the last element of this Array.	

⌋

8.8.2.2.1.24 rend

[SWS_CORE_01257] Definition of API function ara::core::Array::rend

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`class ara::core::Array`	
Syntax:	`const_reverse_iterator rend () const noexcept;`	
Return value:	const_reverse_iterator	the const_reverse_iterator
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a const_reverse_iterator pointing past the first element of this Array.	

⌋

8.8.2.2.1.25 rend

[SWS_CORE_01256] Definition of API function ara::core::Array::rend

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`class ara::core::Array`	
Syntax:	`reverse_iterator rend () noexcept;`	
Return value:	reverse_iterator	the reverse_iterator
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a reverse_iterator pointing past the first element of this Array.	

⌋

8.8.2.2.1.26 size

[SWS_CORE_01262] Definition of API function ara::core::Array::size

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`class ara::core::Array`	
Syntax:	`constexpr size_type size () const noexcept;`	
Return value:	size_type	N
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return the number of elements in this Array.	

⌋

8.8.2.2.1.27 swap

[SWS_CORE_01242] Definition of API function ara::core::Array::swap

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`class ara::core::Array`	
Syntax:	`void swap (Array< T, N > &other) noexcept(noexcept(swap(std::declval< T & >(), std::declval< T & >())));`	
Parameters (inout):	other	the other Array
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	Exchange the contents of this Array with those of other. The noexcept specification shall make use of ADL for the swap() call.	

⌋

8.8.3 Struct: tuple_element

[SWS_CORE_01281] Definition of API class std::tuple_element< I, ara::core::Array< T, N > >

Upstream requirements: RS_AP_00130

⌈

Kind:	struct	
Header file:	#include "ara/core/array.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace std`	
Symbol:	tuple_element< I, ara::core::Array< T, N > >	

Syntax:	`template <size_t I, typename T, size_t N>`
	`struct tuple_element< I, ara::core::Array< T, N > > {...};`	

Template param:	size_t I	the index into the Array whose type is desired
	typename T	the type of element in the Array
	size_t N	the number of elements in the Array
Description:	Specialization of std::tuple_element for ara::core::Array. The implementation shall flag the condition I >= N as a compile error.	

⌋

8.8.3.1 Public Member Types

8.8.3.1.1 Type Alias: type

[SWS_CORE_01285] Definition of API type std::tuple_element< I, ara::core::Array< T, N > >::type

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/array.h"
Scope:	`struct std::tuple_element< I, ara::core::Array< T, N > >`
Symbol:	type
Syntax:	`using type = T;`
Description:	Alias for the type of the Array element with the given index.

⌋

8.8.4 Struct: tuple_size

[SWS_CORE_01280] Definition of API class std::tuple_size< ara::core::Array< T, N > >

Upstream requirements: RS_AP_00130

⌈

Kind:	struct	
Header file:	#include "ara/core/array.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace std`	
Symbol:	tuple_size< ara::core::Array< T, N > >	

Syntax:	`template <typename T, size_t N>`
	`struct tuple_size< ara::core::Array< T, N > > :  public integral_constant< size_t, N > {...};`	

Template param:	typename T	the type of element in the Array
	size_t N	the number of elements in the Array
Description:	Specialization of std::tuple_size for ara::core::Array. This specialization shall meet the C++14 UnaryTypeTrait requirements with a BaseCharacteristic of std::integral_constant<std::size_t, N>.	

⌋

8.9 Header: ara/core/vector.h

8.9.1 Non-Member Functions

8.9.1.1 Other

8.9.1.1.1 operator!=

[SWS_CORE_01391] Definition of API function ara::core::operator!=

Status: DRAFT

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/vector.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T, typename Allocator>`
	`bool operator!= (const Vector< T, Allocator > &lhs, const Vector< T, Allocator > &rhs);`	

Template param:	T	the type of element in the Vector

▽

△

	Allocator	the allocator to use for any memory allocations
Parameters (in):	lhs	the left-hand side of the comparison
	rhs	the right-hand side of the comparison
Return value:	bool	true if the Vectors are non-equal, false otherwise
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Return true if the two Vectors have non-equal content.	

⌋

8.9.1.1.2 operator<

[SWS_CORE_01392] Definition of API function ara::core::operator<

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/vector.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T, typename Allocator>`
	`bool operator< (const Vector< T, Allocator > &lhs, const Vector< T, Allocator > &rhs);`	

Template param:	T	the type of element in the Vector
	Allocator	the allocator to use for any memory allocations
Parameters (in):	lhs	the left-hand side of the comparison
	rhs	the right-hand side of the comparison
Return value:	bool	true if lhs is less than rhs, false otherwise
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Return true if the contents of lhs are lexicographically less than the contents of rhs.	

⌋

8.9.1.1.3 operator<=

[SWS_CORE_01393] Definition of API function ara::core::operator<=

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/vector.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T, typename Allocator>`
	`bool operator<= (const Vector< T, Allocator > &lhs, const Vector< T, Allocator > &rhs);`	

Template param:	T	the type of element in the Vector
	Allocator	the allocator to use for any memory allocations
Parameters (in):	lhs	the left-hand side of the comparison
	rhs	the right-hand side of the comparison
Return value:	bool	true if lhs is less than or equal to rhs, false otherwise
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Return true if the contents of lhs are lexicographically less than or equal to the contents of rhs.	

⌋

8.9.1.1.4 operator==

[SWS_CORE_01390] Definition of API function ara::core::operator==

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/vector.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T, typename Allocator>`
	`bool operator== (const Vector< T, Allocator > &lhs, const Vector< T, Allocator > &rhs);`	

Template param:	T	the type of element in the Vector
	Allocator	the allocator to use for any memory allocations
Parameters (in):	lhs	the left-hand side of the comparison
	rhs	the right-hand side of the comparison
Return value:	bool	true if the Vectors are equal, false otherwise
Exception Safety:	not exception safe	

▽

△

Thread Safety:	implementation defined
Description:	Return true if the two Vectors have equal content.

⌋

8.9.1.1.5 operator>

[SWS_CORE_01394] Definition of API function ara::core::operator>

Status: DRAFT

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/vector.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T, typename Allocator>`
	`bool operator> (const Vector< T, Allocator > &lhs, const Vector< T, Allocator > &rhs);`	

Template param:	T	the type of element in the Vector
	Allocator	the allocator to use for any memory allocations
Parameters (in):	lhs	the left-hand side of the comparison
	rhs	the right-hand side of the comparison
Return value:	bool	true if rhs is less than lhs, false otherwise
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Return true if the contents of rhs are lexicographically less than the contents of lhs.	

⌋

8.9.1.1.6 operator>=

[SWS_CORE_01395] Definition of API function ara::core::operator>=

Status: DRAFT

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`namespace ara::core`

▽

△

Syntax:	`template <typename T, typename Allocator>`
	`bool operator>= (const Vector< T, Allocator > &lhs, const Vector< T, Allocator > &rhs);`	

Template param:	T	the type of element in the Vector
	Allocator	the allocator to use for any memory allocations
Parameters (in):	lhs	the left-hand side of the comparison
	rhs	the right-hand side of the comparison
Return value:	bool	true if rhs is less than or equal to lhs, false otherwise
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Return true if the contents of rhs are lexicographically less than or equal to the contents of lhs.	

⌋

8.9.1.1.7 swap

[SWS_CORE_01396] Definition of API function ara::core::swap

Status: DRAFT

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/vector.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T, typename Allocator>`
	`void swap (Vector< T, Allocator > &lhs, Vector< T, Allocator > &rhs);`	

Template param:	T	the type of element in the Vector
	Allocator	the allocator to use for any memory allocations
Parameters (in):	lhs	the first Vector
	rhs	the second Vector
Return value:	None	
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Exchange the state of lhs with that of rhs.	

⌋

8.9.2 Class: Vector

[SWS_CORE_01301] Definition of API class ara::core::Vector

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	class	
Header file:	#include "ara/core/vector.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace ara::core`	
Symbol:	Vector	

Syntax:	`template <typename T, typename Allocator = <implementation-defined>>`
	`class Vector final {...};`	

Template param:	typename T	the type of element in the vector
	typename Allocator = *<implementation-defined>*	the allocator to use for any memory allocations
Description:	A growable container for contiguous elements.	

⌋

8.10 Header: ara/core/map.h

8.10.1 Non-Member Functions

8.10.1.1 Other

8.10.1.1.1 swap

[SWS_CORE_01496] Definition of API function ara::core::swap

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/map.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename K, typename V, typename C, typename Allocator>`
	`void swap (Map< K, V, C, Allocator > &lhs, Map< K, V, C, Allocator > &rhs);`	

Parameters (in):	lhs	the first Map
	rhs	the second Map

▽

△

Return value:	None
Exception Safety:	not exception safe
Thread Safety:	implementation defined
Description:	Exchange the state of lhs with that of rhs.

⌋

8.10.2 Class: Map

[SWS_CORE_01400] Definition of API class ara::core::Map

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	class	
Header file:	#include "ara/core/map.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace ara::core`	
Symbol:	Map	

Syntax:	`template <typename K, typename V, typename C = std::less<K>, typename Allocator = <implementation-defined>>`
	`class Map final {...};`	

Template param:	typename K	the type of keys in the map
	typename V	the type of values in the map
	typename C = std::less<K>	the comparator for key equality tests
	typename Allocator = *<implementation-defined>*	the allocator to use for any memory allocations
Description:	An ordered associative array.	

⌋

8.11 Header: ara/core/optional.h

8.11.1 Global Variables

8.11.1.1 nullopt

[SWS_CORE_01101] Definition of API variable ara::core::nullopt

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	variable
Header file:	#include "ara/core/optional.h"
Scope:	`namespace ara::core`
Symbol:	nullopt
Type:	`nullopt_t`
Syntax:	`constexpr nullopt_t nullopt {UNSPECIFIED};`
Description:	no-value state indicator, as per `std::nullopt` in [11]

⌋

8.11.2 Non-Member Functions

8.11.2.1 Other

8.11.2.1.1 make_optional

[SWS_CORE_01139] Definition of API function ara::core::make_optional

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T, typename... Args>`
	`constexpr Optional< T > make_optional (Args &&... args) noexcept(std::is_nothrow_constructible< T, Args... >::value);`	

DIRECTION NOT DEFINED	args	--
Return value:	Optional< T >	A new optional
Exception Safety:	conditionally exception safe	

▽

△

Thread Safety:	thread-safe
Description:	As per `std::make_optional(Args&&...)` in [11] except for the following deviations: 1. Function is conditionally noexcept

⌋

8.11.2.1.2 make_optional

[SWS_CORE_01140] Definition of API function ara::core::make_optional

Status: DRAFT

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`namespace ara::core`	
Syntax:	`template <typename T, typename U, typename... Args> constexpr Optional< T > make_optional (std::initializer_list< U > il, Args &&... args) noexcept(std::is_nothrow_constructible< T, std::initializer_list< U > &, Args &&... >::value);`	
DIRECTION NOT DEFINED	il	--
	args	--
Return value:	Optional< T >	A new optional
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	As per `std::make_optional(initializer_list<U>, Args&&...)` in [11] except for the following deviations: 1. Function is conditionally noexcept	

⌋

8.11.2.1.3 make_optional

[SWS_CORE_01138] Definition of API function ara::core::make_optional

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T>`
	`constexpr Optional< std::decay_t< T > > make_optional (T &&) noexcept(std::is_nothrow_move_constructible< T >::value);`	

DIRECTION NOT DEFINED	T &&	--
Return value:	Optional< std::decay_t< T > >	A new optional
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	As per `std::make_optional(T&&)` in [11] except for the following deviations: 1. Function is conditionally noexcept	

⌋

8.11.2.1.4 swap

[SWS_CORE_01096] Definition of API function ara::core::swap

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T>`
	`void swap (Optional< T > &lhs, Optional< T > &rhs) noexcept(std::is_nothrow_move_constructible< T >::value &&std::is_nothrow_move_assignable< T >::value);`	

Parameters (in):	lhs	the first Optional
	rhs	the second Optional
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	

▽

△

Description:	As per std::swap(Optional< T > &, Optional< T > &) in [11] except for the following deviations: 1. noexcept specifier conditions are modified

⌋

8.11.3 Class: Optional

[SWS_CORE_01033] Definition of API class ara::core::Optional

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	class
Header file:	#include "ara/core/optional.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	`namespace ara::core`
Symbol:	Optional

Syntax:	`template <typename T>`
	`class Optional final {...};`

Description:	Implements `std::optional` (see `[optional]` in [11]). Unless explicitly overriden in the member documentation, members always adhere in behavior to the ISO specification in [11].
See also:	[SWS_LBAP_00010], [SWS_LBAP_00011], [SWS_LBAP_00012]

⌋

8.11.3.1 Public Member Types

8.11.3.1.1 Type Alias: value_type

[SWS_CORE_01102] Definition of API type ara::core::Optional::value_type

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional`
Symbol:	value_type
Syntax:	`using value_type = T;`

▽

△

Description:	As per `std::optional::value_type` in [11]

⌋

8.11.3.2 Public Member Functions

8.11.3.2.1 Special Member Functions

8.11.3.2.1.1 Copy Constructor

[SWS_CORE_01105] Definition of API function ara::core::Optional::Optional

Status: DRAFT

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	
Syntax:	`constexpr Optional (const Optional &) noexcept(std::is_nothrow_copy_constructible< T >::value);`	
DIRECTION NOT DEFINED	const Optional &	--
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	As per `std::optional::optional(const optional& rhs)` in [11] except for the following deviations: 1. Function is conditionally noexcept	

⌋

8.11.3.2.1.2 Move Constructor

[SWS_CORE_01106] Definition of API function ara::core::Optional::Optional

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	
Syntax:	`constexpr Optional (Optional &&) noexcept(std::is_nothrow_move_constructible< T >::value);`	
DIRECTION NOT DEFINED	Optional &&	--
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	As per `std::optional::optional(optional&& rhs)` in [11] including noexcept conditions	

⌋

8.11.3.2.1.3 Copy Constructor

[SWS_CORE_01110] Definition of API function ara::core::Optional::Optional

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	

Syntax:	`template <typename U>`
	`EXPLICIT Optional (const Optional< U > &) noexcept(std::is_nothrow_constructible< T, const U & >::value);`	

DIRECTION NOT DEFINED	const Optional< U > &	--
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	As per `std::optional::optional(const optional<U>&)` in [11] except for the following deviations: 1. Function is conditionally noexcept This constructor is `explicit` if and only if `is_convertible<const U&, T>::value` is `false`	

⌋

8.11.3.2.1.4 Move Constructor

[SWS_CORE_01111] Definition of API function ara::core::Optional::Optional

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	

Syntax:	`template <typename U>`
	`EXPLICIT Optional (Optional< U > &&) noexcept(std::is_nothrow_constructible< T, U >::value);`	

DIRECTION NOT DEFINED	Optional< U > &&	--
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	As per `std::optional::optional(optional<U>&&)` in [11] except for the following deviations: 1. Function is conditionally noexcept This constructor is `explicit` if and only if `is_convertible<U&&, T>::value` is `false`	

⌋

8.11.3.2.1.5 Default Constructor

[SWS_CORE_01103] Definition of API function ara::core::Optional::Optional

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional`
Syntax:	`constexpr Optional () noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	Create an instance that does not contain a value, as per `std::optional::optional()` in [11]

⌋

8.11.3.2.1.6 Move Assignment Operator

[SWS_CORE_01118] Definition of API function ara::core::Optional::operator=

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	

Syntax:	`template <typename U>`
	`Optional & operator= (Optional< U > &&) &noexcept(std::is_nothrow_constructible< T, U >::value &&std::is_nothrow_assignable< T &, U >::value);`	

DIRECTION NOT DEFINED	Optional< U > &&	--
Return value:	Optional &	*this
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	As per `std::optional::operator=(optional<U>&&)` in [11] except for the following deviations: 1. Function is conditionally noexcept 2. Function is declared with the ref-qualifier &	

⌋

8.11.3.2.1.7 Copy Assignment Operator

[SWS_CORE_01114] Definition of API function ara::core::Optional::operator=

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	
Syntax:	`Optional & operator= (const Optional &other) &noexcept(std::is_nothrow_copy_constructible< T >::value &&std::is_nothrow_copy_assignable< T >::value);`	
Parameters (in):	other	the other instance
Return value:	Optional &	*this
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	

▽

△

Description:	As per `std::optional::operator=(const Optional&)` in [11] except for the following deviations: 1. Function is conditionally noexcept 2. Function is declared with the ref-qualifier &

⌋

8.11.3.2.1.8 Move Assignment Operator

[SWS_CORE_01115] Definition of API function ara::core::Optional::operator=

Status: DRAFT

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	
Syntax:	`Optional & operator= (Optional &&other) &noexcept(std::is_nothrow_move_assignable< T >::value &&std::is_nothrow_move_constructible< T >::value);`	
Parameters (in):	other	the other instance
Return value:	Optional &	*this
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	As per `std::optional::operator=(optional&&)` in [11] except for the following deviations: 1. Function is declared with the ref-qualifier &	

⌋

8.11.3.2.1.9 Copy Assignment Operator

[SWS_CORE_01117] Definition of API function ara::core::Optional::operator=

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	

Syntax:	`template <typename U>`
	`Optional & operator= (const Optional< U > &) &noexcept(std::is_nothrow_constructible< T, const U &>::value &&std::is_nothrow_assignable< T &, const U &>::value);`	

DIRECTION NOT DEFINED	const Optional< U > &	--
Return value:	Optional &	*this
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	As per `std::optional::operator=(const optional<U>&)` in [11] except for the following deviations: 1. Function is conditionally noexcept 2. Function is declared with the ref-qualifier &	

⌋

8.11.3.2.1.10 Destructor

[SWS_CORE_01112] Definition of API function ara::core::Optional::~Optional

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional`
Syntax:	`~Optional () noexcept(std::is_nothrow_destructible< T >::value);`
Exception Safety:	conditionally exception safe
Thread Safety:	not thread-safe
Description:	As per `std::optional::~optional()` in [11] except for the following deviations: 1. Function is conditionally noexcept

⌋

8.11.3.2.2 Constructors

8.11.3.2.2.1 Optional

[SWS_CORE_01107] Definition of API function ara::core::Optional::Optional

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	

Syntax:	`template <typename... Args>`
	`explicit constexpr Optional (in_place_t, Args &&...) noexcept(std::is_nothrow_constructible< T, Args... >::value);`	

DIRECTION NOT DEFINED	in_place_t	--
	...	--
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	As per `std::optional::optional(in_place_t, Args&&... )` in [11] except for the following deviations: 1. Function is conditionally noexcept	

⌋

8.11.3.2.2.2 Optional

[SWS_CORE_01108] Definition of API function ara::core::Optional::Optional

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	

Syntax:	`template <typename U, typename... Args>`
	`explicit constexpr Optional (in_place_t, std::initializer_list< U >, Args &&...) noexcept(std::is_nothrow_constructible< T, std::initializer_list< U > &, Args &&... >::value);`	

DIRECTION NOT DEFINED	in_place_t	--
	std::initializer_list< U >	--
	...	--
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	

▽

△

Description:	As per `std::optional::optional(in_place_t, initializer_list<U> il, Args&&... args)` in [11] except for the following deviations: 1. Function is conditionally noexcept

⌋

8.11.3.2.2.3 Optional

[SWS_CORE_01109] Definition of API function ara::core::Optional::Optional

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	

Syntax:	`template <typename U = T>`
	`EXPLICIT constexpr Optional (U &&) noexcept(std::is_nothrow_constructible< T, U >::value);`	

DIRECTION NOT DEFINED	U &&	--
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	As per `std::optional::optional(U&&)` in [11] except for the following deviations: 1. Function is conditionally noexcept This constructor is `explicit` if and only if `is_convertible<U&&, T>::value` is `false`	

⌋

8.11.3.2.2.4 Optional

[SWS_CORE_01104] Definition of API function ara::core::Optional::Optional

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional`

▽

△

Syntax:	`constexpr Optional (nullopt_t) noexcept;`	
DIRECTION NOT DEFINED	nullopt_t	--
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Create an instance that does not contain a value, as per `std::optional::optional(nullopt_t)` in [11]	

⌋

8.11.3.2.3 Member Functions

8.11.3.2.3.1 emplace

[SWS_CORE_01119] Definition of API function ara::core::Optional::emplace

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	

Syntax:	`template <typename... Args>`
	`T & emplace (Args &&...) noexcept(std::is_nothrow_constructible< T, Args &&... >::value);`	

DIRECTION NOT DEFINED	...	--
Return value:	T &	A reference to the new contained value
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	As per `std::optional::emplace(Args&&...)` in [11] except for the following deviations: 1. Function is conditionally noexcept	

⌋

8.11.3.2.3.2 emplace

[SWS_CORE_01120] Definition of API function ara::core::Optional::emplace

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	

Syntax:	`template <typename U, typename... Args>`
	`T & emplace (std::initializer_list< U >, Args &&...)`
	`noexcept(std::is_nothrow_constructible< T, std::initializer_list< U > &, Args &&... >::value);`	

DIRECTION NOT DEFINED	std::initializer_list< U >	--
	...	--
Return value:	T &	A reference to the new contained value
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	As per `std::optional::emplace(initializer_list<U>, Args&&...)` in [11] except for the following deviations: 1. Function is conditionally noexcept	

⌋

8.11.3.2.3.3 has_value

[SWS_CORE_01129] Definition of API function ara::core::Optional::has_value

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	
Syntax:	`constexpr bool has_value () const noexcept;`	
Return value:	bool	true if this instance contains a value
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	As per `std::optional::has_value()` in [11]	

⌋

8.11.3.2.3.4 operator bool

[SWS_CORE_01128] Definition of API function ara::core::Optional::operator bool

Status: DRAFT

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	
Syntax:	`explicit constexpr operator bool () const noexcept;`	
Return value:	bool	true if this instance contains a value
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	As per `std::optional::operator bool()` in [11]	

⌋

8.11.3.2.3.5 operator*

[SWS_CORE_01125] Definition of API function ara::core::Optional::operator*

Status: DRAFT

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	
Syntax:	`constexpr T & operator* () & noexcept;`	
Return value:	T &	A reference to the contained value
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`NoValueInOptionalViolation`	If *this does not contain a value.
Description:	As per `constexpr T& std::optional::operator*() &` in [11] except for the following deviations: 1. Function is noexcept 2. Function may result in a `Violation`	

⌋

8.11.3.2.3.6 operator*

[SWS_CORE_01124] Definition of API function ara::core::Optional::operator*

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	
Syntax:	`constexpr const T & operator* () const & noexcept;`	
Return value:	const T &	A const reference to the contained value
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`NoValueInOptionalViolation`	If *this does not contain a value.
Description:	As per `constexpr const T& std::optional::operator*() const&` in [11] except for the following deviations: 1. Function is noexcept 2. Function may result in a `Violation`	

⌋

8.11.3.2.3.7 operator*

[SWS_CORE_01126] Definition of API function ara::core::Optional::operator*

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	
Syntax:	`constexpr T && operator* () &&noexcept;`	
Return value:	T &&	An rvalue reference to the contained value
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`NoValueInOptionalViolation`	If *this does not contain a value.

▽

△

Description:	As per `constexpr T&& std::optional::operator*() &&` in [11] except for the following deviations: 1. Function is noexcept 2. Function may result in a `Violation`

⌋

8.11.3.2.3.8 operator*

[SWS_CORE_01127] Definition of API function ara::core::Optional::operator*

Status: DRAFT

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	
Syntax:	`constexpr const T && operator* () const &&noexcept;`	
Return value:	const T &&	An rvalue reference to the contained value
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`NoValueInOptionalViolation`	If *this does not contain a value.
Description:	As per `constexpr const T&& std::optional::operator*() const&&` in [11] except for the following deviations: 1. Function is noexcept 2. Function may result in a `Violation`	

⌋

8.11.3.2.3.9 operator->

[SWS_CORE_01122] Definition of API function ara::core::Optional::operator->

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	
Syntax:	`const T * operator-> () const noexcept;`	
Return value:	const T *	A const pointer to the contained value
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`NoValueInOptionalViolation`	If *this does not contain a value.
Description:	As per `constexpr const T* std::optional::operator->() const` in [11] except for the following deviations: 1. Function is noexcept 2. Function is not constexpr 3. Function may result in a `Violation`	

⌋

8.11.3.2.3.10 operator->

[SWS_CORE_01123] Definition of API function ara::core::Optional::operator->

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	
Syntax:	`T * operator-> () noexcept;`	
Return value:	T *	A pointer to the contained value
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`NoValueInOptionalViolation`	If *this does not contain a value.

▽

△

Description:	As per `constexpr T* std::optional::operator->()` in [11] except for the following deviations: 1. Function is noexcept 2. Function is not constexpr 3. Function may result in a `Violation`

⌋

8.11.3.2.3.11 operator=

[SWS_CORE_01116] Definition of API function ara::core::Optional::operator=

Status: DRAFT

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	

Syntax:	`template <typename U = T>`
	`Optional & operator= (U &&u) &noexcept(std::is_nothrow_constructible< T, U >::value &&std::is_nothrow_assignable< T &, U >::value);`	

Template param:	U	the type of u
Parameters (in):	u	the new value
Return value:	Optional &	*this
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	As per `std::optional::operator=(U&&)` in [11] except for the following deviations: 1. Function is conditionally noexcept 2. Function is declared with the ref-qualifier &	

⌋

8.11.3.2.3.12 operator=

[SWS_CORE_01113] Definition of API function ara::core::Optional::operator=

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	
Syntax:	`Optional & operator= (nullopt_t) & noexcept;`	
DIRECTION NOT DEFINED	nullopt_t	--
Return value:	Optional &	*this
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Clear this instance, as per `std::optional::operator=(nullopt_t)` in [11]	

⌋

8.11.3.2.3.13 reset

[SWS_CORE_01136] Definition of API function ara::core::Optional::reset

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional`
Syntax:	`void reset () noexcept;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per `std::optional::reset()` in [11]

⌋

8.11.3.2.3.14 swap

[SWS_CORE_01121] Definition of API function ara::core::Optional::swap

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional`
Syntax:	`void swap (Optional &) noexcept(std::is_nothrow_move_constructible< T >::value &&std::is_nothrow_move_assignable< T >::value);`
Return value:	None
Exception Safety:	conditionally exception safe
Thread Safety:	not thread-safe
Description:	As per `std::optional::swap(optional&)` in [11] except for the following deviations: 1. Conditions for the conditional exception safety are modified

⌋

8.11.3.2.3.15 value

[SWS_CORE_01132] Definition of API function ara::core::Optional::value

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	
Syntax:	`constexpr T && value () &&noexcept;`	
Return value:	T &&	An rvalue reference to the contained value
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`NoValueInOptionalViolation`	If *this does not contain a value.
Description:	As per `constexpr T&& std::optional::value() &&` in [11] except for the following deviations: 1. Function is noexcept 2. Function may result in a `Violation`	

⌋

8.11.3.2.3.16 value

[SWS_CORE_01133] Definition of API function ara::core::Optional::value

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	
Syntax:	`constexpr const T && value () const &&noexcept;`	
Return value:	const T &&	An rvalue reference to the contained value
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`NoValueInOptionalViolation`	If *this does not contain a value.
Description:	As per `constexpr const T&& std::optional::value() const&&` in [11] except for the following deviations: 1. Function is noexcept 2. Function may result in a `Violation`	

⌋

8.11.3.2.3.17 value

[SWS_CORE_01130] Definition of API function ara::core::Optional::value

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	
Syntax:	`constexpr const T & value () const & noexcept;`	
Return value:	const T &	A const reference to the contained value
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`NoValueInOptionalViolation`	If *this does not contain a value.

▽

△

Description:	As per `constexpr const T& std::optional::value() const&` in [11] except for the following deviations: 1. Function is noexcept 2. Function may result in a `Violation`

⌋

8.11.3.2.3.18 value

[SWS_CORE_01131] Definition of API function ara::core::Optional::value

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	
Syntax:	`constexpr T & value () & noexcept;`	
Return value:	T &	A reference to the contained value
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`NoValueInOptionalViolation`	If *this does not contain a value.
Description:	As per `constexpr T& std::optional::value() &` in [11] except for the following deviations: 1. Function is noexcept 2. Function may result in a `Violation`	

⌋

8.11.3.2.3.19 value_or

[SWS_CORE_01134] Definition of API function ara::core::Optional::value_or

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	

Syntax:	`template <typename U>`
	`constexpr T value_or (U &&) const &;`	

DIRECTION NOT DEFINED	U &&	--
Return value:	T	A copy of the contained value if there is one, or the given default.
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	As per `constexpr T std::optional::value_or(U&&) const&;` in [11]	

⌋

8.11.3.2.3.20 value_or

[SWS_CORE_01135] Definition of API function ara::core::Optional::value_or

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	

Syntax:	`template <typename U>`
	`constexpr T value_or (U &&) &&;`	

DIRECTION NOT DEFINED	U &&	--
Return value:	T	A copy of the contained value if there is one, or the given default.
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	As per `constexpr T std::optional::value_or(U&&) &&` in [11]	

⌋

8.11.4 Class: Optional

[SWS_CORE_01150] Definition of API class ara::core::Optional< T & >

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	class	
Header file:	#include "ara/core/optional.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace ara::core`	
Symbol:	Optional< T & >	

Syntax:	`template <typename T>`
	`class Optional< T & > final {...};`	

Template param:	typename T	the type of element in the container
Description:	Specialization of class Optional for lvalue references	

⌋

8.11.4.1 Public Member Types

8.11.4.1.1 Type Alias: value_type

[SWS_CORE_01151] Definition of API type ara::core::Optional< T & >::value_type

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional< T & >`
Symbol:	value_type
Syntax:	`using value_type = T&;`
Description:	The value type used by this specialization

⌋

8.11.4.2 Public Member Functions

8.11.4.2.1 Special Member Functions

8.11.4.2.1.1 Move Constructor

[SWS_CORE_01155] Definition of API function ara::core::Optional< T & >::Optional

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional< T & >`	
Syntax:	`constexpr Optional (Optional &&other) noexcept=default;`	
Parameters (in):	other	the other instance
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Move constructor The moved-from instance other will retain its state; if it contained a value before the move, its value is also retained.	

⌋

8.11.4.2.1.2 Copy Constructor

[SWS_CORE_01157] Definition of API function ara::core::Optional< T & >::Optional

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional< T & >`	

Syntax:	`template <typename U>`
	`explicit constexpr Optional (Optional< U > const &other) noexcept;`	

Template param:	U	the type of the other Optional's value
Parameters (in):	other	the other Optional to whose value to bind to (if there is one) This constructor is `explicit` if and only if `is_convertible<const U&, T>::value` is `false`
Exception Safety:	exception safe	

▽

△

Thread Safety:	implementation defined
Description:	Create an instance whose state is taken from another Optional

⌋

8.11.4.2.1.3 Default Constructor

[SWS_CORE_01152] Definition of API function ara::core::Optional< T & >::Optional

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional< T & >`
Syntax:	`constexpr Optional () noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	Create an instance that does not contain a value

⌋

8.11.4.2.1.4 Copy Constructor

[SWS_CORE_01154] Definition of API function ara::core::Optional< T & >::Optional

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional< T & >`	
Syntax:	`constexpr Optional (const Optional &other) noexcept=default;`	
Parameters (in):	other	the other instance
Exception Safety:	exception safe	
Thread Safety:	implementation defined	

▽

△

Description:	Copy constructor

⌋

8.11.4.2.1.5 Copy Assignment Operator

[SWS_CORE_01159] Definition of API function ara::core::Optional< T & >::operator=

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional< T & >`	
Syntax:	`constexpr Optional & operator= (const Optional &other) & noexcept=default;`	
Parameters (in):	other	the other instance
Return value:	Optional &	*this
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Copy assignment operator Rebinds this instance to the referee of other if there is one. Otherwise resets the stored value in *this.	

⌋

8.11.4.2.1.6 Copy Assignment Operator

[SWS_CORE_01163] Definition of API function ara::core::Optional< T & >::operator=

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional< T & >`

Syntax:	`template <typename U = T>`
	`Optional & operator= (const Optional< U > &other) & noexcept;`

▽

△

Template param:	U	the type of the other Optional’s value
Parameters (in):	other	the other Optional to whose value to bind to (if there is one)
Return value:	Optional &	*this
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Rebind this instance to the referee of another Optional	

⌋

8.11.4.2.1.7 Move Assignment Operator

[SWS_CORE_01160] Definition of API function ara::core::Optional< T & >::operator=

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional< T & >`	
Syntax:	`constexpr Optional & operator= (Optional &&other) & noexcept=default;`	
Parameters (in):	other	the other instance
Return value:	Optional &	*this
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Move assignment operator Rebinds this optional to the referee of other if there is one. Otherwise resets the stored value in *this. The moved-from instance other will retain its state; if it contained a value before the move, its value is also retained.	

⌋

8.11.4.2.1.8 Destructor

[SWS_CORE_01158] Definition of API function ara::core::Optional< T & >::~Optional

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional< T & >`
Syntax:	`~Optional () noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	Destructor The destructor is trivial.

⌋

8.11.4.2.2 Constructors

8.11.4.2.2.1 Optional

[SWS_CORE_01156] Definition of API function ara::core::Optional< T & >::Optional

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional< T & >`	

Syntax:	`template <typename U = T>`
	`explicit constexpr Optional (U &&u) noexcept;`	

Template param:	U	the type of u
Parameters (in):	u	the value to bind to
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Create an instance that contains a value If U is not an lvalue, the program is ill-formed.	

⌋

8.11.4.2.2.2 Optional

[SWS_CORE_01153] Definition of API function ara::core::Optional< T & >::Optional

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional< T & >`	
Syntax:	`constexpr Optional (nullopt_t) noexcept;`	
DIRECTION NOT DEFINED	nullopt_t	--
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Create an instance that does not contain a value	

⌋

8.11.4.2.3 Member Functions

8.11.4.2.3.1 emplace

[SWS_CORE_01164] Definition of API function ara::core::Optional< T & >::emplace

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional< T & >`	

Syntax:	`template <typename U = T>`
	`constexpr Optional & emplace (U &&u) noexcept;`	

Template param:	U	the type of u
Parameters (in):	u	the new value to bind to
Return value:	Optional &	*this
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Rebind this instance to u If U is not an lvalue, the program is ill-formed.	

⌋

8.11.4.2.3.2 has_value

[SWS_CORE_01172] Definition of API function ara::core::Optional< T & >::has_value

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional< T & >`	
Syntax:	`constexpr bool has_value () const noexcept;`	
Return value:	bool	true if this instance contains a value
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Returns whether this instance contains a value.	

⌋

8.11.4.2.3.3 operator bool

[SWS_CORE_01171] Definition of API function ara::core::Optional< T & >::operator bool

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional< T & >`	
Syntax:	`explicit constexpr operator bool () const noexcept;`	
Return value:	bool	true if this instance contains a value
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Returns whether this instance contains a value.	

⌋

8.11.4.2.3.4 operator*

[SWS_CORE_01169] Definition of API function ara::core::Optional< T & >::operator*

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional< T & >`	
Syntax:	`constexpr T const & operator* () const noexcept;`	
Return value:	T const &	the reference
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`NoValueInOptionalViolation`	If *this does not contain a value.
Description:	Return a const_reference to the referred-to object.	

⌋

8.11.4.2.3.5 operator*

[SWS_CORE_01170] Definition of API function ara::core::Optional< T & >::operator*

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional< T & >`	
Syntax:	`constexpr T & operator* () noexcept;`	
Return value:	T &	the reference
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`NoValueInOptionalViolation`	If *this does not contain a value.
Description:	Return a reference to the referred-to object.	

⌋

8.11.4.2.3.6 operator->

[SWS_CORE_01167] Definition of API function ara::core::Optional< T & >::operator->

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional< T & >`	
Syntax:	`T const * operator-> () const noexcept;`	
Return value:	T const *	the pointer
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`NoValueInOptionalViolation`	If *this does not contain a value.
Description:	Return a const_pointer to the referred-to object.	

⌋

8.11.4.2.3.7 operator->

[SWS_CORE_01168] Definition of API function ara::core::Optional< T & >::operator->

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional< T & >`	
Syntax:	`T * operator-> () noexcept;`	
Return value:	T *	the pointer
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`NoValueInOptionalViolation`	If *this does not contain a value.
Description:	Return a pointer to the referred-to object.	

⌋

8.11.4.2.3.8 operator=

[SWS_CORE_01161] Definition of API function ara::core::Optional< T & >::operator=

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional< T & >`	
Syntax:	`constexpr Optional & operator= (nullopt_t) & noexcept;`	
DIRECTION NOT DEFINED	nullopt_t	--
Return value:	Optional &	*this
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Clear this instance If this instance contains a reference value before this call, it is discarded.	

⌋

8.11.4.2.3.9 operator=

[SWS_CORE_01162] Definition of API function ara::core::Optional< T & >::operator=

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional< T & >`	

Syntax:	`template <typename U = T>`
	`constexpr Optional & operator= (U &&u) & noexcept;`	

Template param:	U	the type of u
Parameters (in):	u	the new value to bind to
Return value:	Optional &	*this
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Rebind this instance to u. If U is not an lvalue, the program is ill-formed.	

⌋

8.11.4.2.3.10 reset

[SWS_CORE_01165] Definition of API function ara::core::Optional< T & >::reset

Status: DRAFT

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional< T & >`
Syntax:	`constexpr void reset () noexcept;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	Clear this Optional If this instance contains a reference value before this call, it is discarded.

⌋

8.11.4.2.3.11 swap

[SWS_CORE_01166] Definition of API function ara::core::Optional< T & >::swap

Status: DRAFT

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional< T & >`	
Syntax:	`constexpr void swap (Optional &other) noexcept;`	
Parameters (inout):	other	the other instance
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Swap the contents of this instance with other	

⌋

8.11.4.2.3.12 value

[SWS_CORE_01174] Definition of API function ara::core::Optional< T & >::value

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional< T & >`	
Syntax:	`constexpr T const & value () const;`	
Return value:	T const &	the reference
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Violations:	`NoValueInOptionalViolation`	If *this does not contain a value.
Description:	Returns a const_reference to the referred-to object.	

⌋

8.11.4.2.3.13 value

[SWS_CORE_01173] Definition of API function ara::core::Optional< T & >::value

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional< T & >`	
Syntax:	`constexpr T & value ();`	
Return value:	T &	the reference
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Violations:	`NoValueInOptionalViolation`	If *this does not contain a value.
Description:	Returns a reference to the referred-to object.	

⌋

8.11.4.2.3.14 value_or

[SWS_CORE_01175] Definition of API function ara::core::Optional< T & >::value_or

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional< T & >`	

Syntax:	`template <typename U>`
	`constexpr T value_or (U &&u) const noexcept;`	

DIRECTION NOT DEFINED	u	--
Return value:	T	A copy of the referred-to object if there is one, or the given default
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a copy of the referred-to object if there is one, or the given default. The program is ill-formed if T is not copy-constructible, or not convertible from U&&.	

⌋

8.11.5 Struct: nullopt_t

[SWS_CORE_01100] Definition of API class ara::core::nullopt_t

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	struct
Header file:	#include "ara/core/optional.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	`namespace ara::core`
Symbol:	nullopt_t
Syntax:	`struct nullopt_t {...};`
Description:	The struct nullopt_t is an empty structure type used as a unique type to indicate the state of not containing a value for Optional objects, as per `std::nullopt_t` in [11].

⌋

8.12 Header: ara/core/variant.h

8.12.1 Non-Member Types

8.12.1.1 Type Alias: variant_alternative_t

[SWS_CORE_01612] Definition of API type ara::core::variant_alternative_t

Status: DRAFT

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/variant.h"
Scope:	`namespace ara::core`
Symbol:	variant_alternative_t
Syntax:	`using variant_alternative_t = typename variant_alternative<I, T>::type;`
Description:	Variant helper typename, as per `std::variant_alternative<I, variant<Types...>>` in [11].

⌋

8.12.2 Global Variables

8.12.2.1 variant_size_v

[SWS_CORE_01606] Definition of API variable ara::core::variant_size_v

Status: DRAFT

Upstream requirements: RS_AP_00130

⌈

Kind:	variable
Header file:	#include "ara/core/variant.h"
Scope:	`namespace ara::core`
Symbol:	variant_size_v
Type:	`std::size_t`

Syntax:	`template <typename T>`
	`constexpr std::size_t variant_size_v = variant_size<T>::value;`

Description:	Variant helper variable, as per `std::variant_size_v` in [11].

⌋

8.12.3 Non-Member Functions

8.12.3.1 Other

8.12.3.1.1 get

[SWS_CORE_01613] Definition of API function ara::core::get

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`namespace ara::core`	
Syntax:	`template <std::size_t I, typename... Types> constexpr variant_alternative_t< I, Variant< Types... > > & get ( Variant< Types... > &variant) noexcept;`	
DIRECTION NOT DEFINED	variant	--
Return value:	variant_alternative_t< I, Variant< Types... > > &	the reference
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`BadVariantAccessViolation`	If variant.index() is not equal to I.
Description:	Return a reference, as per `template <size_t I, class... Types> constexpr variant_alternative_t<I, variant<Types...>>& get(variant<Types...>& v)` in [11] except for the following deviations: 1. Function is noexcept 2. Function may result in a `Violation`	

⌋

8.12.3.1.2 get

[SWS_CORE_01614] Definition of API function ara::core::get

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/variant.h"
Scope:	`namespace ara::core`

▽

△

Syntax:	template <std::size_t I, typename... Types> constexpr variant_alternative_t< I, Variant< Types... > > && get (Variant< Types... > &&variant) noexcept;	
DIRECTION NOT DEFINED	variant	--
Return value:	variant_alternative_t< I, Variant< Types... > > &&	the reference
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	BadVariantAccessViolation	If variant.index() is not equal to I.
Description:	Return a reference, as per template <size_t I, class... Types> constexpr variant_alternative_t<I, variant<Types...>>&& get(variant<Types...>&& v) in [11] except for the following deviations: 1. Function is noexcept 2. Function may result in a Violation	

⌋

8.12.3.1.3 get

[SWS_CORE_01615] Definition of API function ara::core::get

Status: DRAFT

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	namespace ara::core	
Syntax:	template <std::size_t I, typename... Types> constexpr const variant_alternative_t< I, Variant< Types... > > & get (const Variant< Types... > &variant) noexcept;	
DIRECTION NOT DEFINED	variant	--
Return value:	const variant_alternative_t< I, Variant< Types... > > &	the reference
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	BadVariantAccessViolation	If variant.index() is not equal to I.
Description:	Return a reference, as per template <size_t I, class... Types> constexpr const variant_alternative_t<I, variant<Types... >>& get(const variant<Types... >& v) in [11] except for the following deviations: 1. Function is noexcept 2. Function may result in a Violation	

⌋

8.12.3.1.4 get

[SWS_CORE_01616] Definition of API function ara::core::get

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <std::size_t I, typename... Types>`
	`constexpr const variant_alternative_t< I, Variant< Types... > > &&`
	`get (const Variant< Types... > &&variant) noexcept;`	

DIRECTION NOT DEFINED	variant	--
Return value:	const variant_alternative_t< I, Variant< Types... > > &&	the reference
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`BadVariantAccessViolation`	If variant.index() is not equal to I.
Description:	Return a reference, as per `template <size_t I, class... Types> constexpr const variant_alternative_t<I, variant<Types...>>&& get(const variant <Types...>&& v)` in [11] except for the following deviations: 1. Function is noexcept 2. Function may result in a `Violation`	

⌋

8.12.3.1.5 get

[SWS_CORE_01617] Definition of API function ara::core::get

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T, typename... Types>`
	`constexpr T & get (Variant< Types... > &variant) noexcept;`	

DIRECTION NOT DEFINED	variant	--
Return value:	T &	the reference

▽

△

Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	BadVariantAccessViolation	If variant does not hold a value of type T.
Description:	Return a reference, as per `template <class T, class...Types> constexpr T& get(variant<Types...>& v)` in [11] except for the following deviations: 1. Function is noexcept 2. Function may result in a `Violation`	

⌋

8.12.3.1.6 get

[SWS_CORE_01618] Definition of API function ara::core::get

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T, typename... Types>`
	`constexpr T && get (Variant< Types... > &&variant) noexcept;`	

DIRECTION NOT DEFINED	variant	--
Return value:	T &&	the reference
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	BadVariantAccessViolation	If variant does not hold a value of type T.
Description:	Return a reference, as per `template <class T, class...Types> constexpr T&& get(variant<Types...>&& v)` in [11] except for the following deviations: 1. Function is noexcept 2. Function may result in a `Violation`	

⌋

8.12.3.1.7 get

[SWS_CORE_01619] Definition of API function ara::core::get

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T, typename... Types>`
	`constexpr const T & get (const Variant< Types... > &variant) noexcept;`	

DIRECTION NOT DEFINED	variant	--
Return value:	const T &	the reference
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`BadVariantAccessViolation`	If variant does not hold a value of type T.
Description:	Return a reference, as per `template <class T, class...Types> constexpr const T& get(const variant<Types...>& v)` in [11] except for the following deviations: 1. Function is noexcept 2. Function may result in a `Violation`	

⌋

8.12.3.1.8 get

[SWS_CORE_01620] Definition of API function ara::core::get

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T, typename... Types>`
	`constexpr const T && get (const Variant< Types... > &&variant) noexcept;`	

DIRECTION NOT DEFINED	variant	--
Return value:	const T &&	the reference
Exception Safety:	exception safe	
Thread Safety:	thread-safe	

▽

△

Violations:	`BadVariantAccessViolation`	If variant does not hold a value of type T.
Description:	Return a reference, as per `template <class T, class...Types> constexpr const T&& get(const variant<Types...>&& v)` in [11] except for the following deviations: 1. Function is noexcept 2. Function may result in a `Violation`	

⌋

8.12.3.1.9 get_if

[SWS_CORE_01623] Definition of API function ara::core::get_if

Status: DRAFT

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T, typename... Types>`
	`std::add_pointer_t< T > get_if (Variant< Types... > *variant)`
	`noexcept;`	

DIRECTION NOT DEFINED	variant	--
Return value:	std::add_pointer_t< T >	A pointer to the value stored or nullptr
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a pointer, as per `template <class T, class... Types> constexpr add_pointer_t<T> get_if(variant<Types...>* v) noexcept` in [11]	

⌋

8.12.3.1.10 get_if

[SWS_CORE_01624] Definition of API function ara::core::get_if

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`namespace ara::core`	
Syntax:	`template <typename T, typename... Types> std::add_pointer_t< const T > get_if (const Variant< Types... > *variant) noexcept;`	
DIRECTION NOT DEFINED	variant	--
Return value:	std::add_pointer_t< const T >	A pointer to the value stored or nullptr
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a pointer, as per `template <class T, class... Types> constexpr add_pointer_t<const T> get_if(const variant<Types...>* v) noexcept` in [11]	

⌋

8.12.3.1.11 get_if

[SWS_CORE_01621] Definition of API function ara::core::get_if

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`namespace ara::core`	
Syntax:	`template <std::size_t I, typename... Types> std::add_pointer_t< variant_alternative_t< I, Variant< Types... > > > get_if (Variant< Types... > *variant) noexcept;`	
DIRECTION NOT DEFINED	variant	--
Return value:	std::add_pointer_t< variant_alternative_t< I, Variant< Types... > > >	A pointer to the value stored or nullptr
Exception Safety:	exception safe	
Thread Safety:	thread-safe	

▽

△

Description:	Return a pointer, as per `template <size_t I, class... Types> constexpr add_pointer_t<variant_alternative_t<I, variant<Types...>>> get_if(variant<Types...>* v) noexcept` in [11]

⌋

8.12.3.1.12 get_if

[SWS_CORE_01622] Definition of API function ara::core::get_if

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`namespace ara::core`	
Syntax:	`template <std::size_t I, typename... Types> std::add_pointer_t< const variant_alternative_t< I, Variant< Types... > > > get_if (const Variant< Types... > *variant) noexcept;`	
DIRECTION NOT DEFINED	variant	--
Return value:	std::add_pointer_t< const variant_alternative_t< I, Variant< Types... > > >	A pointer to the value stored or nullptr
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a pointer, as per `template <size_t I, class... Types> constexpr add_pointer_t<variant_alternative_t<I, variant<Types...>>> get_if(variant<Types...>* v noexcept)` in [11]	

⌋

8.12.3.1.13 holds_alternative

[SWS_CORE_01626] Definition of API function ara::core::holds_alternative

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T, typename... Types>`
	`constexpr bool holds_alternative (const Variant< Types... > &variant) noexcept;`	

DIRECTION NOT DEFINED	variant	--
Return value:	bool	true if index() is equal to the zero-based index of T in Types...
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Check for contained value type, as per `holds_alternative(const variant<Types...>& v)` in [11]	

⌋

8.12.3.1.14 operator!=

[SWS_CORE_01642] Definition of API function ara::core::operator!=

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`namespace ara::core`	
Syntax:	`constexpr bool operator!= (Monostate lhs, Monostate rhs) noexcept;`	
DIRECTION NOT DEFINED	lhs	--
	rhs	--
Return value:	bool	false
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Comparison operator, as per `operator!=(monostate, monostate)` in [11]	

⌋

8.12.3.1.15 operator!=

[SWS_CORE_01628] Definition of API function ara::core::operator!=

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename... Types>`
	`constexpr bool operator!= (const Variant< Types... > &lhs, const Variant< Types... > &rhs);`	

DIRECTION NOT DEFINED	lhs	--
	rhs	--
Return value:	bool	boolean result of the comparison as defined in [11]
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Comparison operator, as per `operator!=(const variant<Types...>& v, const variant<Types...>& w)` in [11]	

⌋

8.12.3.1.16 operator<

[SWS_CORE_01629] Definition of API function ara::core::operator<

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename... Types>`
	`constexpr bool operator< (const Variant< Types... > &lhs, const Variant< Types... > &rhs);`	

DIRECTION NOT DEFINED	lhs	--
	rhs	--
Return value:	bool	boolean result of the comparison as defined in [11]
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Comparison operator, as per `operator<(const variant<Types...>& v, const variant<Types...>& w)` in [11]	

⌋

8.12.3.1.17 operator<

[SWS_CORE_01643] Definition of API function ara::core::operator<

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`namespace ara::core`	
Syntax:	`constexpr bool operator< (Monostate lhs, Monostate rhs) noexcept;`	
DIRECTION NOT DEFINED	lhs	--
	rhs	--
Return value:	bool	false
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Comparison operator, as per `operator<(monostate, monostate)` in [11]	

⌋

8.12.3.1.18 operator<=

[SWS_CORE_01645] Definition of API function ara::core::operator<=

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`namespace ara::core`	
Syntax:	`constexpr bool operator<= (Monostate lhs, Monostate rhs) noexcept;`	
DIRECTION NOT DEFINED	lhs	--
	rhs	--
Return value:	bool	true
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Comparison operator, as per `operator<=(monostate, monostate)` in [11]	

⌋

8.12.3.1.19 operator<=

[SWS_CORE_01631] Definition of API function ara::core::operator<=

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename... Types>`
	`constexpr bool operator<= (const Variant< Types... > &lhs, const Variant< Types... > &rhs);`	

DIRECTION NOT DEFINED	lhs	--
	rhs	--
Return value:	bool	boolean result of the comparison as defined in [11]
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Comparison operator, as per `operator<=(const variant<Types...>& v, const variant<Types...>& w)` in [11]	

⌋

8.12.3.1.20 operator==

[SWS_CORE_01641] Definition of API function ara::core::operator==

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`namespace ara::core`	
Syntax:	`constexpr bool operator== (Monostate lhs, Monostate rhs) noexcept;`	
DIRECTION NOT DEFINED	lhs	--
	rhs	--
Return value:	bool	true
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Comparison operator, as per `operator==(monostate, monostate)` in [11]	

⌋

8.12.3.1.21 operator==

[SWS_CORE_01627] Definition of API function ara::core::operator==

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename... Types>`
	`constexpr bool operator== (const Variant< Types... > &lhs, const Variant< Types... > &rhs);`	

DIRECTION NOT DEFINED	lhs	--
	rhs	--
Return value:	bool	boolean result of the comparison as defined in [11]
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Comparison operator, as per `operator==(const variant<Types...>& v, const variant<Types...>& w)` in [11]	

⌋

8.12.3.1.22 operator>

[SWS_CORE_01644] Definition of API function ara::core::operator>

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`namespace ara::core`	
Syntax:	`constexpr bool operator> (Monostate lhs, Monostate rhs) noexcept;`	
DIRECTION NOT DEFINED	lhs	--
	rhs	--
Return value:	bool	false
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Comparison operator, as per `operator>(monostate, monostate)` in [11]	

⌋

8.12.3.1.23 operator>

[SWS_CORE_01630] Definition of API function ara::core::operator>

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename... Types>`
	`constexpr bool operator> (const Variant< Types... > &lhs, const Variant< Types... > &rhs);`	

DIRECTION NOT DEFINED	lhs	--
	rhs	--
Return value:	bool	boolean result of the comparison as defined in [11]
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Comparison operator, as per `operator>(const variant<Types...>& v, const variant<Types...>& w)` in [11]	

⌋

8.12.3.1.24 operator>=

[SWS_CORE_01646] Definition of API function ara::core::operator>=

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`namespace ara::core`	
Syntax:	`constexpr bool operator>= (Monostate lhs, Monostate rhs) noexcept;`	
DIRECTION NOT DEFINED	lhs	--
	rhs	--
Return value:	bool	true
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Comparison operator, as per `operator>=(monostate, monostate)` in [11]	

⌋

8.12.3.1.25 operator>=

[SWS_CORE_01632] Definition of API function ara::core::operator>=

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename... Types>`
	`constexpr bool operator>= (const Variant< Types... > &lhs, const Variant< Types... > &rhs);`	

DIRECTION NOT DEFINED	lhs	--
	rhs	--
Return value:	bool	boolean result of the comparison as defined in [11]
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Comparison operator, as per `operator>=(const variant<Types...>& v, const variant<Types...>& w)` in [11]	

⌋

8.12.3.1.26 swap

[SWS_CORE_01696] Definition of API function ara::core::swap

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename... Types>`
	`void swap (Variant< Types... > &lhs, Variant< Types... > &rhs) noexcept(noexcept(lhs.swap(rhs)));`	

DIRECTION NOT DEFINED	lhs	--
	rhs	--
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	Swap the contained value, as per `swap(variant<Types...>& v, variant<Types...>& w)` in [11]	

⌋

8.12.3.1.27 visit

[SWS_CORE_01634] Definition of API function ara::core::visit
Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename Visitor, typename... Variants>`
	`constexpr see_below_ visit (Visitor &&visitor, Variants &&... variants);`	

DIRECTION NOT DEFINED	visitor	--
	variants	--
Return value:	see_below_	As specified in [16]
Exception Safety:	not exception safe	
Thread Safety:	not thread-safe	
Description:	As per `template<class R, class Visitor, class... Variants> constexpr R visit(Visitor&& vis, Variants&&... vars)` in [16]	

⌋

8.12.3.1.28 visit

[SWS_CORE_01633] Definition of API function ara::core::visit
Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename ReturnType, typename Visitor, typename... Variants>`
	`constexpr see_below_ visit (Visitor &&visitor, Variants &&... variants);`	

DIRECTION NOT DEFINED	visitor	--
	variants	--
Return value:	see_below_	As specified in [16]
Exception Safety:	not exception safe	
Thread Safety:	not thread-safe	
Description:	As per `template<class Visitor, class... Variants> constexpr see below visit(Visitor&& vis, Variants&&... vars)` in [16]	

⌋

8.12.4 Struct: Monostate

[SWS_CORE_01640] Definition of API class ara::core::Monostate

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	struct
Header file:	#include "ara/core/variant.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	`namespace ara::core`
Symbol:	Monostate
Syntax:	`struct Monostate final {...};`
Description:	A candidate for an empty, default-constructible first alternative type for a Variant, as per `std::monostate` in [11]

⌋

8.12.5 Class: Variant

[SWS_CORE_01601] Definition of API class ara::core::Variant

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	class	
Header file:	#include "ara/core/variant.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace ara::core`	
Symbol:	Variant	

Syntax:	`template <typename... Types>`
	`class Variant {...};`	

Template param:	typename... Types	the types contained in the variant
Description:	A type-safe union. Implements `std::variant` (see `[variant.variant]` in [11]) Unless explicitly overriden in the member documentation, members always adhere in behavior to the ISO specification in [11].	

⌋

8.12.5.1 Public Member Functions

8.12.5.1.1 Special Member Functions

8.12.5.1.1.1 Move Constructor

[SWS_CORE_01651] Definition of API function ara::core::Variant::Variant

Status: DRAFT

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`class ara::core::Variant`	
Syntax:	`Variant (Variant &&other) noexcept(see_below);`	
DIRECTION NOT DEFINED	other	--
Exceptions:	<TYPE>	any exception thrown by the move-initialization of any Ti
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	Move constructor, as per in [11] including noexcept conditions	

⌋

8.12.5.1.1.2 Default Constructor

[SWS_CORE_01649] Definition of API function ara::core::Variant::Variant

Status: DRAFT

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`class ara::core::Variant`	
Syntax:	`constexpr Variant () noexcept(see_below);`	
Exceptions:	<TYPE>	any exception thrown by the value-initialization of T0
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Default constructor, as per `variant()` in [11]	

⌋

8.12.5.1.1.3 Copy Constructor

[SWS_CORE_01650] Definition of API function ara::core::Variant::Variant

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`class ara::core::Variant`	
Syntax:	`Variant (const Variant &other) noexcept(see_below);`	
DIRECTION NOT DEFINED	other	--
Exceptions:	<TYPE>	any exception thrown by the direct-initialization of any Ti
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	Copy constructor, as per `variant(const variant& w)` in [11] except for the following deviations: 1. Function is conditionally noexcept The expression inside `noexcept` is equivalent to the logical AND of is_nothrow_copy_constructible<Ti>::value for all i.	

⌋

8.12.5.1.1.4 Move Assignment Operator

[SWS_CORE_01659] Definition of API function ara::core::Variant::operator=

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`class ara::core::Variant`	
Syntax:	`constexpr Variant & operator= (Variant &&other) noexcept(see_below);`	
DIRECTION NOT DEFINED	other	--
Return value:	Variant &	*this
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	Move-assignment operator, as per `operator=(variant&& rhs)` in [11] including noexcept conditions	

⌋

8.12.5.1.1.5 Copy Assignment Operator

[SWS_CORE_01658] Definition of API function ara::core::Variant::operator=

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`class ara::core::Variant`	
Syntax:	`constexpr Variant & operator= (const Variant &other) noexcept(see_below);`	
DIRECTION NOT DEFINED	other	--
Return value:	Variant &	*this
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	Copy-assignment operator, as per `operator=(const variant& rhs)` in [11] except for the following deviations: 1. Function is conditionally noexcept The expression inside `noexcept` is equivalent to the logical AND of is_nothrow_copy_constructible<Ti>::value && is_nothrow_copy_assignable<Ti>::value for all i.	

⌋

8.12.5.1.1.6 Destructor

[SWS_CORE_01657] Definition of API function ara::core::Variant::~Variant

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/variant.h"
Scope:	`class ara::core::Variant`
Syntax:	`~Variant () noexcept(see_below);`
Exception Safety:	conditionally exception safe
Thread Safety:	not thread-safe
Description:	Destructor, as per `~variant()` in [11] except for the following deviations: 1. Function is conditionally noexcept The expression inside `noexcept` is equivalent to the logical AND of is_nothrow_destructible<Ti>::value for all i.

⌋

8.12.5.1.2 Constructors

8.12.5.1.2.1 Variant

[SWS_CORE_01653] Definition of API function ara::core::Variant::Variant

Status: DRAFT

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`class ara::core::Variant`	

Syntax:	`template <typename T, typename... Args>`
	`explicit constexpr Variant (in_place_type_t< T > tag, Args &&... args) noexcept(std::is_nothrow_constructible< T, Args... >::value);`	

DIRECTION NOT DEFINED	tag	--
	args	--
Exceptions:	Any	exception thrown by calling the selected constructor of T
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	As per `variant(in_place_type_t<T>, Args&&... args)` in [11] except for the following deviations: 1. Function is conditionally noexcept	

⌋

8.12.5.1.2.2 Variant

[SWS_CORE_01654] Definition of API function ara::core::Variant::Variant

Status: DRAFT

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`class ara::core::Variant`	

Syntax:	`template <typename T, typename U, typename... Args>`
	`explicit constexpr Variant (in_place_type_t< T > tag, std::initializer_list< U > il, Args &&... args) noexcept(std::is_nothrow_constructible< T, std::initializer_list< U > &, Args &&... >::value);`	

DIRECTION NOT DEFINED	tag	--
	il	--
	args	--

▽

△

Exceptions:	Any	exception thrown by calling the selected constructor of T.
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	As per `variant(in_place_type_t<T>, initializer_list<U> il, Args&&... args)` in [11] except for the following deviations: 1. Function is conditionally noexcept	

⌋

8.12.5.1.2.3 Variant

[SWS_CORE_01655] Definition of API function ara::core::Variant::Variant

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`class ara::core::Variant`	

Syntax:	`template <std::size_t I, typename... Args>`
	`explicit constexpr Variant (in_place_index_t< I > tag, Args &&... args) noexcept(see_below);`	

DIRECTION NOT DEFINED	tag	--
	args	--
Exceptions:	Any	exception thrown by calling the selected constructor of T_I
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	As per `variant(in_place_index_t<I>, Args&&... args)` in [11] except for the following deviations: 1. Function is conditionally noexcept The expression inside `noexcept` is equivalent to `std::is_nothrow_constructible<T_I, Args...>::value`.	

⌋

8.12.5.1.2.4 Variant

[SWS_CORE_01656] Definition of API function ara::core::Variant::Variant

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`class ara::core::Variant`	

Syntax:	`template <std::size_t I, typename U, typename... Args>`
	`explicit constexpr Variant (in_place_index_t< I > tag,`
	`std::initializer_list< U > il, Args &&... args) noexcept(see_below);`	

DIRECTION NOT DEFINED	tag	--
	il	--
	args	--
Exceptions:	Any	exception thrown by calling the selected constructor of T_I.
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	
Description:	As per `variant(in_place_index_t<I>, initializer_list<U> il, Args&&... args)` in [11] except for the following deviations: 1. Function is conditionally noexcept The expression inside `noexcept` is equivalent to `std::is_nothrow_constructible<T_I, std::initializer_list<U>&, Args&&...>::value`.	

⌋

8.12.5.1.2.5 Variant

[SWS_CORE_01652] Definition of API function ara::core::Variant::Variant

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`class ara::core::Variant`	

Syntax:	`template <typename T>`
	`constexpr Variant (T &&value) noexcept(see_below);`	

DIRECTION NOT DEFINED	value	--
Exceptions:	Any	exception thrown by the initialization of the selected alternative Tj
Exception Safety:	conditionally exception safe	
Thread Safety:	implementation defined	

▽

△

Description:	As per `constexpr variant(T&& t)` in [11] including noexcept conditions

⌋

8.12.5.1.3 Member Functions

8.12.5.1.3.1 emplace

[SWS_CORE_01663] Definition of API function ara::core::Variant::emplace

Status: DRAFT

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`class ara::core::Variant`	

Syntax:	`template <std::size_t I, typename... Args>`
	`variant_alternative_t< I, Variant< Types... > > & emplace (Args &&... args) noexcept(see_below);`	

DIRECTION NOT DEFINED	args	--
Return value:	variant_alternative_t< I, Variant< Types... > > &	*this
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	As per `template <size_t I, class... Args> variant_alternative_t<I, variant<Types...>>& emplace(Args&&... args)` in [11] except for the following deviations: 1. Function is conditionally noexcept The expression inside `noexcept` is equivalent to `std::is_nothrow_constructible<T_I, Args...>::value`.	

⌋

8.12.5.1.3.2 emplace

[SWS_CORE_01662] Definition of API function ara::core::Variant::emplace

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`class ara::core::Variant`	
Syntax:	`template <typename T, typename U, typename... Args> T & emplace (std::initializer_list< U > il, Args &&... args) noexcept(std::is_nothrow_constructible< T, std::initializer_list< U > &, Args &&... >::value);`	
DIRECTION NOT DEFINED	il	--
	args	--
Return value:	T &	*this
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	As per `template <class T, class U, class... Args> T& emplace(initializer_list<U> il, Args&&... args)` in [11] except for the following deviations: 1. Function is conditionally noexcept	

⌋

8.12.5.1.3.3 emplace

[SWS_CORE_01661] Definition of API function ara::core::Variant::emplace

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`class ara::core::Variant`	
Syntax:	`template <typename T, typename... Args> T & emplace (Args &&... args) noexcept(std::is_nothrow_constructible< T, Args... >::value);`	
DIRECTION NOT DEFINED	args	--
Return value:	T &	*this
Exceptions:	Any	exception thrown by calling the selected constructor of T
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	

▽

△

Description:	As per template <class T, class... Args> T& emplace(Args&&... args) in [11] except for the following deviations: 1. Function is conditionally noexcept

⌋

8.12.5.1.3.4 emplace

[SWS_CORE_01664] Definition of API function ara::core::Variant::emplace

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	class ara::core::Variant	
Syntax:	template <std::size_t I, typename U, typename... Args> variant_alternative_t< I, Variant< Types... > > & emplace (std::initializer_list< U > il, Args &&... args) noexcept(see_below);	
DIRECTION NOT DEFINED	il	--
	args	--
Return value:	variant_alternative_t< I, Variant< Types... > > &	*this
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	As per template <size_t I, class U, class... Args> variant_alternative_t<I, variant<Types...>>& emplace(initializer_list<U> il, Args&&... args) in [11] except for the following deviations: 1. Function is conditionally noexcept The expression inside noexcept is equivalent to std::is_nothrow_constructible<T_I, std::initializer_list<U>&, Args&&...>::value.	

⌋

8.12.5.1.3.5 index

[SWS_CORE_01665] Definition of API function ara::core::Variant::index

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`class ara::core::Variant`	
Syntax:	`constexpr std::size_t index () const noexcept;`	
Return value:	std::size_t	the index as in [11]
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Get the index of the currently contained value, as per `index()` in [11]	

⌋

8.12.5.1.3.6 operator=

[SWS_CORE_01660] Definition of API function ara::core::Variant::operator=

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`class ara::core::Variant`	

Syntax:	`template <typename T>`
	`Variant & operator= (T &&value) noexcept(see_below);`	

DIRECTION NOT DEFINED	value	--
Return value:	Variant &	*this
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	As per `operator=(T&& t)` in [11] including noexcept conditions	

⌋

8.12.5.1.3.7 swap

[SWS_CORE_01667] Definition of API function ara::core::Variant::swap

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`class ara::core::Variant`	
Syntax:	`void swap (Variant &other) noexcept(see_below);`	
DIRECTION NOT DEFINED	other	--
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	As per `swap(variant& rhs)` in [11] except for the following deviations: 1. Conditions for the conditional exception safety are modified The expression inside `noexcept` is equivalent to the logical AND of is_nothrow_move_constructible<Ti>::value && is_nothrow_move_assignable<Ti>::value for all i.	

⌋

8.12.5.1.3.8 valueless_by_exception

[SWS_CORE_01666] Definition of API function ara::core::Variant::valueless_by_exception

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`class ara::core::Variant`	
Syntax:	`constexpr bool valueless_by_exception () const noexcept;`	
Return value:	bool	<ARTechTerm{false}>} if *this holds a value
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Check if *this holds a value, as per `valueless_by_exception()` in [11]	

⌋

8.12.6 Struct: variant_alternative

[SWS_CORE_01607] Definition of API class ara::core::variant_alternative

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	struct	
Header file:	#include "ara/core/variant.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace ara::core`	
Symbol:	variant_alternative	

Syntax:	`template <std::size_t I, typename T>`
	`struct variant_alternative;`	

Template param:	std::size_t I	--
	typename T	--
Description:	Variant helper class, as per `std::variant_alternative` in [11].	

⌋

8.12.7 Struct: variant_alternative

[SWS_CORE_01611] Definition of API class ara::core::variant_alternative< I, Variant< Types... > >

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	struct	
Header file:	#include "ara/core/variant.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace ara::core`	
Symbol:	variant_alternative< I, Variant< Types... > >	

Syntax:	`template <std::size_t I, typename...  Types>`
	`struct variant_alternative< I, Variant< Types...  > > {...};`	

Template param:	std::size_t I	--
	typename... Types	--
Description:	Variant helper class, as per `std::variant_alternative<I, variant<Types...>>` in [11].	

⌋

8.12.8 Struct: variant_alternative

[SWS_CORE_01608] Definition of API class ara::core::variant_alternative< I, const T >

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	struct	
Header file:	#include "ara/core/variant.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace ara::core`	
Symbol:	variant_alternative< I, const T >	

Syntax:	`template <std::size_t I, typename T>`
	`struct variant_alternative< I, const T > {...};`	

Template param:	std::size_t I	--
	typename T	--
Description:	Variant helper class, as per `std::variant_alternative<I, const T>` in [11].	

⌋

8.12.9 Struct: variant_alternative

[SWS_CORE_01610] Definition of API class ara::core::variant_alternative< I, const volatile T >

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	struct	
Header file:	#include "ara/core/variant.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace ara::core`	
Symbol:	variant_alternative< I, const volatile T >	

Syntax:	`template <std::size_t I, typename T>`
	`struct variant_alternative< I, const volatile T > {...};`	

Template param:	std::size_t I	--
	typename T	--
Description:	Variant helper class, as per `std::variant_alternative<I, const volatile T>` in [11].	

⌋

8.12.10 Struct: variant_alternative

[SWS_CORE_01609] Definition of API class ara::core::variant_alternative< I, volatile T >

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	struct	
Header file:	#include "ara/core/variant.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace ara::core`	
Symbol:	variant_alternative< I, volatile T >	

Syntax:	`template <std::size_t I, typename T>`
	`struct variant_alternative< I, volatile T > {...};`	

Template param:	std::size_t I	--
	typename T	--
Description:	Variant helper class, as per `std::variant_alternative<I, volatile T>` in [11].	

⌋

8.12.11 Struct: variant_size

[SWS_CORE_01600] Definition of API class ara::core::variant_size

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	struct	
Header file:	#include "ara/core/variant.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace ara::core`	
Symbol:	variant_size	

Syntax:	`template <typename T>`
	`struct variant_size;`	

Template param:	typename T	--
Description:	Variant helper class, as per `std::variant_size` in [11].	

⌋

8.12.12 Struct: variant_size

[SWS_CORE_01605] Definition of API class ara::core::variant_size< Variant< Types... > >

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	struct	
Header file:	#include "ara/core/variant.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace ara::core`	
Symbol:	variant_size< Variant< Types... > >	

Syntax:	`template <typename... Types>`
	`struct variant_size< Variant< Types... > > {...};`	

Template param:	typename... Types	--
Description:	Variant helper class, as per `std::variant_size<variant<Types...>>` in [11].	

⌋

8.12.13 Struct: variant_size

[SWS_CORE_01602] Definition of API class ara::core::variant_size< const T >

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	struct	
Header file:	#include "ara/core/variant.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace ara::core`	
Symbol:	variant_size< const T >	

Syntax:	`template <typename T>`
	`struct variant_size< const T > {...};`	

Template param:	typename T	--
Description:	Variant helper class, as per `std::variant_size<const T>` in [11].	

⌋

8.12.14 Struct: variant_size

[SWS_CORE_01604] Definition of API class ara::core::variant_size< const volatile T >

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	struct	
Header file:	#include "ara/core/variant.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace ara::core`	
Symbol:	variant_size< const volatile T >	

Syntax:	`template <typename T>`
	`struct variant_size< const volatile T > {...};`	

Template param:	typename T	--
Description:	Variant helper class, as per `std::variant_size<const volatile T>` in [11].	

⌋

8.12.15 Struct: variant_size

[SWS_CORE_01603] Definition of API class ara::core::variant_size< volatile T >

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	struct	
Header file:	#include "ara/core/variant.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace ara::core`	
Symbol:	variant_size< volatile T >	

Syntax:	`template <typename T>`
	`struct variant_size< volatile T > {...};`	

Template param:	typename T	--
Description:	Variant helper class, as per `std::variant_size<volatile T>` in [11].	

⌋

8.12.16 Struct: hash

[SWS_CORE_01670] Definition of API class std::hash<::ara::core::Monostate >

Upstream requirements: RS_AP_00130

⌈

Kind:	struct
Header file:	#include "ara/core/variant.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	`namespace std`
Symbol:	hash<::ara::core::Monostate >

Syntax:	`template <>`
	`struct hash<::ara::core::Monostate > {...};`

Description:	As per corresponding declaration in `[variant.variant]` in [11]

⌋

8.12.16.1 Public Member Functions

8.12.16.1.1 Member Functions

8.12.16.1.1.1 operator()

[SWS_CORE_01671] Definition of API function std::hash<::ara::core::Monostate >::operator()

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/variant.h"
Scope:	`struct std::hash<::ara::core::Monostate >`
Syntax:	`size_t operator() (const ::ara::core::Monostate &monostate) const;`
Exception Safety:	not exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[variant.variant]` in [11]

⌋

8.12.17 Struct: hash

[SWS_CORE_01668] Definition of API class std::hash<::ara::core::Variant< Types... > >

Upstream requirements: RS_AP_00130

⌈

Kind:	struct	
Header file:	#include "ara/core/variant.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace std`	
Symbol:	hash<::ara::core::Variant< Types... > >	

Syntax:	`template <typename... Types>`
	`struct hash<::ara::core::Variant< Types... > > {...};`	

Template param:	typename... Types	--
Description:	As per corresponding declaration in `[variant.variant]` in [11]	

⌋

8.12.17.1 Public Member Functions

8.12.17.1.1 Member Functions

8.12.17.1.1.1 operator()

[SWS_CORE_01669] Definition of API function std::hash<::ara::core::Variant< Types... > >::operator()

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/variant.h"
Scope:	`struct std::hash<::ara::core::Variant< Types... > >`
Syntax:	`size_t operator() (const ::ara::core::Variant< Types... > &variant) const;`
Exception Safety:	not exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[variant.variant]` in [11]

⌋

8.13 Header: ara/core/string_view.h

8.13.1 Non-Member Functions

8.13.1.1 Other

8.13.1.1.1 operator!=

[SWS_CORE_02182] Definition of API function ara::core::operator!=

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`namespace ara::core`
Syntax:	`constexpr bool operator!= (StringView lhs, StringView rhs) noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.1.1.2 operator""_SV

[SWS_CORE_02188] Definition of API function ara::core::literals::operator""_SV

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`namespace ara::core::literals`
Syntax:	`constexpr StringView operator""_SV (const char *str, std::size_t len) noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per `std::operator""sv` in [11]

⌋

8.13.1.1.3 operator<

[SWS_CORE_02183] Definition of API function ara::core::operator<

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`namespace ara::core`
Syntax:	`constexpr bool operator< (StringView lhs, StringView rhs) noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.1.1.4 operator

[SWS_CORE_02187] Definition of API function ara::core::operator<<

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`namespace ara::core`
Syntax:	`std::ostream & operator<< (std::ostream &os, StringView sv);`
Exception Safety:	not exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.1.1.5 operator<=

[SWS_CORE_02185] Definition of API function ara::core::operator<=

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`namespace ara::core`
Syntax:	`constexpr bool operator<= (StringView lhs, StringView rhs) noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.1.1.6 operator==

[SWS_CORE_02181] Definition of API function ara::core::operator==

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`namespace ara::core`
Syntax:	`constexpr bool operator== (StringView lhs, StringView rhs) noexcept;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.1.1.7 operator>

[SWS_CORE_02184] Definition of API function ara::core::operator>

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`namespace ara::core`
Syntax:	`constexpr bool operator> (StringView lhs, StringView rhs) noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.1.1.8 operator>=

[SWS_CORE_02186] Definition of API function ara::core::operator>=

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`namespace ara::core`
Syntax:	`constexpr bool operator>= (StringView lhs, StringView rhs) noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.2 Class: StringView

[SWS_CORE_02001] Definition of API class ara::core::StringView

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	class
Header file:	#include "ara/core/string_view.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	`namespace ara::core`
Symbol:	StringView
Syntax:	`class StringView final {...};`
Description:	Implements `std::string_view` in [11] Unless explicitly overriden in the member documentation, members always adhere in behavior to the ISO specification in [11].

⌋

8.13.2.1 Public Member Types

8.13.2.1.1 Type Alias: const_iterator

[SWS_CORE_02105] Definition of API type ara::core::StringView::const_iterator

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Symbol:	const_iterator
Syntax:	`using const_iterator = implementation_defined;`
Description:	As per corresponding declaration in `[string.view]` in [11] except for the following deviations: 1. The value_type of the iterator is const char

⌋

8.13.2.1.2 Type Alias: const_pointer

[SWS_CORE_02102] Definition of API type ara::core::StringView::const_pointer

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Symbol:	const_pointer
Syntax:	`using const_pointer = const value_type*;`
Description:	As per corresponding declaration in `[string.view]` in [11]

⌋

8.13.2.1.3 Type Alias: const_reference

[SWS_CORE_02104] Definition of API type ara::core::StringView::const_reference

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Symbol:	const_reference
Syntax:	`using const_reference = const value_type&;`
Description:	As per corresponding declaration in `[string.view]` in [11]

⌋

8.13.2.1.4 Type Alias: const_reverse_iterator

[SWS_CORE_02107] Definition of API type ara::core::StringView::const_reverse_iterator

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Symbol:	const_reverse_iterator
Syntax:	`using const_reverse_iterator = std::reverse_iterator<const_iterator>;`
Description:	As per corresponding declaration in `[string.view]` in [11]

⌋

8.13.2.1.5 Type Alias: difference_type

[SWS_CORE_02110] Definition of API type ara::core::StringView::difference_type

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Symbol:	difference_type
Syntax:	`using difference_type = std::ptrdiff_t;`
Description:	As per corresponding declaration in `[string.view]` in [11]

⌋

8.13.2.1.6 Type Alias: iterator

[SWS_CORE_02106] Definition of API type ara::core::StringView::iterator

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Symbol:	iterator
Syntax:	`using iterator = const_iterator;`
Description:	As per corresponding declaration in `[string.view]` in [11]

⌋

8.13.2.1.7 Type Alias: pointer

[SWS_CORE_02101] Definition of API type ara::core::StringView::pointer

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Symbol:	pointer
Syntax:	`using pointer = value_type*;`
Description:	As per corresponding declaration in `[string.view]` in [11]

⌋

8.13.2.1.8 Type Alias: reference

[SWS_CORE_02103] Definition of API type ara::core::StringView::reference

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Symbol:	reference
Syntax:	`using reference = value_type&;`
Description:	As per corresponding declaration in `[string.view]` in [11]

⌋

8.13.2.1.9 Type Alias: reverse_iterator

[SWS_CORE_02108] Definition of API type ara::core::StringView::reverse_iterator

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Symbol:	reverse_iterator
Syntax:	`using reverse_iterator = const_reverse_iterator;`
Description:	As per corresponding declaration in `[string.view]` in [11]

⌋

8.13.2.1.10 Type Alias: size_type

[SWS_CORE_02109] Definition of API type ara::core::StringView::size_type

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Symbol:	size_type
Syntax:	`using size_type = std::size_t;`
Description:	As per corresponding declaration in `[string.view]` in [11]

⌋

8.13.2.1.11 Type Alias: value_type

[SWS_CORE_02100] Definition of API type ara::core::StringView::value_type

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Symbol:	value_type
Syntax:	`using value_type = char;`
Description:	The type of characters within this StringView

⌋

8.13.2.2 Public Member Variables

8.13.2.2.1 npos

[SWS_CORE_02111] Definition of API variable ara::core::StringView::npos

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	variable
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Symbol:	npos
Type:	`size_type`
Syntax:	`static constexpr size_type npos = size_type(-1);`
Description:	As per corresponding declaration in `[string.view]` in [11]

⌋

8.13.2.3 Public Member Functions

8.13.2.3.1 Special Member Functions

8.13.2.3.1.1 Move Constructor

[SWS_CORE_02117] Definition of API function ara::core::StringView::StringView

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr StringView (StringView &&other) noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	(AUTOSAR defined) move constructor - not present in [11]

⌋

8.13.2.3.1.2 Default Constructor

[SWS_CORE_02112] Definition of API function ara::core::StringView::StringView

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr StringView () noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.2.3.1.3 Copy Constructor

[SWS_CORE_02115] Definition of API function ara::core::StringView::StringView

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr StringView (const StringView &other) noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.2.3.1.4 Move Assignment Operator

[SWS_CORE_02118] Definition of API function ara::core::StringView::operator=

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr StringView & operator= (StringView &&other) noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	(AUTOSAR defined) move assignment operator - not present in [11]

⌋

8.13.2.3.1.5 Copy Assignment Operator

[SWS_CORE_02116] Definition of API function ara::core::StringView::operator=

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr StringView & operator= (const StringView &other) noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.2.3.1.6 Destructor

[SWS_CORE_02119] Definition of API function ara::core::StringView::~String View
Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`~StringView () noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	(AUTOSAR defined) destructor - not present in [11]

⌋

8.13.2.3.2 Constructors

8.13.2.3.2.1 StringView

[SWS_CORE_02114] Definition of API function ara::core::StringView::StringView
Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr StringView (const char *str, size_type len) noexcept;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[string.view]` in [11] except for the following deviations: 1. Function is noexcept

⌋

8.13.2.3.2.2 StringView

[SWS_CORE_02113] Definition of API function ara::core::StringView::StringView

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr StringView (const char *str) noexcept;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[string.view]` in [11] except for the following deviations: 1. Function is noexcept

⌋

8.13.2.3.3 Member Functions

8.13.2.3.3.1 at

[SWS_CORE_02133] Definition of API function ara::core::StringView::at

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/string_view.h"	
Scope:	`class ara::core::StringView`	
Syntax:	`constexpr const_reference at (size_type pos) const noexcept;`	
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`StringViewOutOfRangeViolation`	In case of an out-of-bounds access
Description:	As per corresponding function in `[string.view]` in [11] except for the following deviations: 1. Function is noexcept 2. Function uses Violations instead of exceptions as the error handling mechanism	

⌋

8.13.2.3.3.2 back

[SWS_CORE_02135] Definition of API function ara::core::StringView::back

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr const_reference back () const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11] except for the following deviations: 1. Function is noexcept The behavior of this function is undefined if empty() == true.

⌋

8.13.2.3.3.3 begin

[SWS_CORE_02120] Definition of API function ara::core::StringView::begin

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr const_iterator begin () const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.2.3.3.4 cbegin

[SWS_CORE_02122] Definition of API function ara::core::StringView::cbegin

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr const_iterator cbegin () const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.2.3.3.5 cend

[SWS_CORE_02123] Definition of API function ara::core::StringView::cend

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr const_iterator cend () const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.2.3.3.6 compare

[SWS_CORE_02147] Definition of API function ara::core::StringView::compare

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/string_view.h"	
Scope:	`class ara::core::StringView`	
Syntax:	`constexpr int compare (size_type pos1, size_type n1, const char *s, size_type n2) const noexcept;`	
Parameters (in):	pos1	position of the first character to consider within this StringView
	n1	number of characters to consider within this StringView
	s	the other StringView
	n2	number of characters to consider within the other character sequence
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`StringViewOutOfRangeViolation`	If pos1 value exceeds the size of this StringView or pos2 value exceeds the size of a StringView created from s
Description:	As per corresponding function in `[string.view]` in [11] except for the following deviations: 1. Function is noexcept	

⌋

8.13.2.3.3.7 compare

[SWS_CORE_02143] Definition of API function ara::core::StringView::compare

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/string_view.h"	
Scope:	`class ara::core::StringView`	
Syntax:	`constexpr int compare (size_type pos1, size_type n1, StringView s) const noexcept;`	
Parameters (in):	pos1	position of the first character to consider within this StringView
	n1	number of characters to consider within this StringView
	s	the other StringView
Exception Safety:	exception safe	
Thread Safety:	thread-safe	

▽

△

Violations:	`StringViewOutOfRangeViolation`	If pos1 value exceeds the size of this StringView
Description:	As per corresponding function in `[string.view]` in [11] except for the following deviations: 1. Function is noexcept	

⌋

8.13.2.3.3.8 compare

[SWS_CORE_02142] Definition of API function ara::core::StringView::compare

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr int compare (StringView s) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.2.3.3.9 compare

[SWS_CORE_02145] Definition of API function ara::core::StringView::compare

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr int compare (const char *s) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe

▽

△

Description:	As per corresponding function in [string.view] in [11] except for the following deviations: 1. Function is noexcept

⌋

8.13.2.3.3.10 compare

[SWS_CORE_02146] Definition of API function ara::core::StringView::compare

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/string_view.h"	
Scope:	class ara::core::StringView	
Syntax:	constexpr int compare (size_type pos1, size_type n1, const char *s) const noexcept;	
Parameters (in):	pos1	position of the first character to consider within this StringView
	n1	number of characters to consider within this StringView
	s	the null-terminated other character sequence
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	StringViewOutOfRangeViolation	If pos1 value exceeds the size of this StringView
Description:	As per corresponding function in [string.view] in [11] except for the following deviations: 1. Function is noexcept	

⌋

8.13.2.3.3.11 compare

[SWS_CORE_02144] Definition of API function ara::core::StringView::compare

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	class ara::core::StringView

▽

△

Syntax:	`constexpr int compare (size_type pos1, size_type n1, StringView s, size_type pos2, size_type n2) const noexcept;`	
Parameters (in):	pos1	position of the first character to consider within this StringView
	n1	number of characters to consider within this StringView
	s	the other StringView
	pos2	position of the first character to consider within the other StringView
	n2	number of characters to consider within the other StringView
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`StringViewOutOfRangeViolation`	If pos1 value exceeds the size of this StringView or pos2 value exceeds the size of s
Description:	As per corresponding function in `[string.view]` in [11] except for the following deviations: 1. Function is noexcept	

⌋

8.13.2.3.3.12 contains

[SWS_CORE_02154] Definition of API function ara::core::StringView::contains

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr bool contains (StringView sv) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [17]

⌋

8.13.2.3.3.13 contains

[SWS_CORE_02156] Definition of API function ara::core::StringView::contains

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr bool contains (const char *str) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [17] except for the following deviations: 1. Function is noexcept

⌋

8.13.2.3.3.14 contains

[SWS_CORE_02155] Definition of API function ara::core::StringView::contains

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr bool contains (char c) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [17]

⌋

8.13.2.3.3.15 copy

[SWS_CORE_02140] Definition of API function ara::core::StringView::copy

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/string_view.h"	
Scope:	`class ara::core::StringView`	
Syntax:	`size_type copy (char *s, size_type n, size_type pos=0) const noexcept;`	
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`StringViewOutOfRangeViolation`	If pos value exceeds the size of this StringView
Description:	As per corresponding function in `[string.view]` in [11] except for the following deviations: 1. Function is noexcept 2. Function uses Violations instead of exceptions as the error handling mechanism	

⌋

8.13.2.3.3.16 crbegin

[SWS_CORE_02126] Definition of API function ara::core::StringView::crbegin

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr const_reverse_iterator crbegin () const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.2.3.3.17 crend

[SWS_CORE_02127] Definition of API function ara::core::StringView::crend

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr const_reverse_iterator crend () const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.2.3.3.18 data

[SWS_CORE_02136] Definition of API function ara::core::StringView::data

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr const_pointer data () const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.2.3.3.19 empty

[SWS_CORE_02131] Definition of API function ara::core::StringView::empty

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr bool empty () const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.2.3.3.20 end

[SWS_CORE_02121] Definition of API function ara::core::StringView::end

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr const_iterator end () const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.2.3.3.21 ends_with

[SWS_CORE_02151] Definition of API function ara::core::StringView::ends_with

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr bool ends_with (StringView sv) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [16]

⌋

8.13.2.3.3.22 ends_with

[SWS_CORE_02152] Definition of API function ara::core::StringView::ends_with

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr bool ends_with (char c) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [16]

⌋

8.13.2.3.3.23 ends_with

[SWS_CORE_02153] Definition of API function ara::core::StringView::ends_with

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr bool ends_with (const char *str) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [16] except for the following deviations: 1. Function is noexcept

⌋

8.13.2.3.3.24 find

[SWS_CORE_02160] Definition of API function ara::core::StringView::find

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find (const char *s, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11] except for the following deviations: 1. Function is noexcept

⌋

8.13.2.3.3.25 find

[SWS_CORE_02158] Definition of API function ara::core::StringView::find

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find (char c, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.2.3.3.26 find

[SWS_CORE_02157] Definition of API function ara::core::StringView::find

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find (StringView s, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.2.3.3.27 find

[SWS_CORE_02159] Definition of API function ara::core::StringView::find

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find (const char *s, size_type pos, size_type n) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11] except for the following deviations: 1. Function is noexcept

⌋

8.13.2.3.3.28 find_first_not_of

[SWS_CORE_02173] Definition of API function ara::core::StringView::find_first_not_of

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find_first_not_of (StringView s, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.2.3.3.29 find_first_not_of

[SWS_CORE_02176] Definition of API function ara::core::StringView::find_first_not_of

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find_first_not_of (const char *s, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11] except for the following deviations: 1. Function is noexcept

⌋

8.13.2.3.3.30 find_first_not_of

[SWS_CORE_02175] Definition of API function ara::core::StringView::find_first_not_of

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find_first_not_of (const char *s, size_type pos, size_type n) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11] except for the following deviations: 1. Function is noexcept

⌋

8.13.2.3.3.31 find_first_not_of

[SWS_CORE_02174] Definition of API function ara::core::StringView::find_first_not_of

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find_first_not_of (char c, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.2.3.3.32 find_first_of

[SWS_CORE_02166] Definition of API function ara::core::StringView::find_first_of

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find_first_of (char c, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.2.3.3.33 find_first_of

[SWS_CORE_02168] Definition of API function ara::core::StringView::find_first_of

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find_first_of (const char *s, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11] except for the following deviations: 1. Function is noexcept

⌋

8.13.2.3.3.34 find_first_of

[SWS_CORE_02167] Definition of API function ara::core::StringView::find_first_of

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find_first_of (const char *s, size_type pos, size_type n) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11] except for the following deviations: 1. Function is noexcept

⌋

8.13.2.3.3.35 find_first_of

[SWS_CORE_02165] Definition of API function ara::core::StringView::find_first_of

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find_first_of (StringView s, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.2.3.3.36 find_last_not_of

[SWS_CORE_02177] Definition of API function ara::core::StringView::find_last_not_of

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find_last_not_of (StringView s, size_type pos=npos) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.2.3.3.37 find_last_not_of

[SWS_CORE_02180] Definition of API function ara::core::StringView::find_last_not_of

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find_last_not_of (const char *s, size_type pos=npos) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11] except for the following deviations: 1. Function is noexcept

⌋

8.13.2.3.3.38 find_last_not_of

[SWS_CORE_02178] Definition of API function ara::core::StringView::find_last_not_of

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find_last_not_of (char c, size_type pos=npos) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.2.3.3.39 find_last_not_of

[SWS_CORE_02179] Definition of API function ara::core::StringView::find_last_not_of

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find_last_not_of (const char *s, size_type pos, size_type n) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11] except for the following deviations: 1. Function is noexcept

⌋

8.13.2.3.3.40 find_last_of

[SWS_CORE_02170] Definition of API function ara::core::StringView::find_last_of

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find_last_of (char c, size_type pos=npos) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.2.3.3.41 find_last_of

[SWS_CORE_02171] Definition of API function ara::core::StringView::find_last_of

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find_last_of (const char *s, size_type pos, size_type n) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11] except for the following deviations: 1. Function is noexcept

⌋

8.13.2.3.3.42 find_last_of

[SWS_CORE_02169] Definition of API function ara::core::StringView::find_last_of

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find_last_of (StringView s, size_type pos=npos) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.2.3.3.43 find_last_of

[SWS_CORE_02172] Definition of API function ara::core::StringView::find_last_of

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find_last_of (const char *s, size_type pos=npos) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11] except for the following deviations: 1. Function is noexcept

⌋

8.13.2.3.3.44 front

[SWS_CORE_02134] Definition of API function ara::core::StringView::front

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr const_reference front () const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11] except for the following deviations: 1. Function is noexcept The behavior of this function is undefined if empty() == true.

⌋

8.13.2.3.3.45 length

[SWS_CORE_02129] Definition of API function ara::core::StringView::length

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type length () const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.2.3.3.46 max_size

[SWS_CORE_02130] Definition of API function ara::core::StringView::max_size

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type max_size () const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.2.3.3.47 operator[]

[SWS_CORE_02132] Definition of API function ara::core::StringView::operator[]

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr const_reference operator[] (size_type pos) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11] except for the following deviations: 1. Function is noexcept The behavior of this function is undefined if pos >= size().

⌋

8.13.2.3.3.48 rbegin

[SWS_CORE_02124] Definition of API function ara::core::StringView::rbegin

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr const_reverse_iterator rbegin () const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.2.3.3.49 remove_prefix

[SWS_CORE_02137] Definition of API function ara::core::StringView::remove_prefix

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr void remove_prefix (size_type n) noexcept;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding function in `[string.view]` in [11] except for the following deviations: 1. Function is noexcept The behavior of this function is undefined if n > size().

⌋

8.13.2.3.3.50 remove_suffix

[SWS_CORE_02138] Definition of API function ara::core::StringView::remove_suffix

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr void remove_suffix (size_type n) noexcept;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding function in `[string.view]` in [11] except for the following deviations: 1. Function is noexcept The behavior of this function is undefined if n > size().

⌋

8.13.2.3.3.51 rend

[SWS_CORE_02125] Definition of API function ara::core::StringView::rend

Status: DRAFT

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr const_reverse_iterator rend () const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.2.3.3.52 rfind

[SWS_CORE_02164] Definition of API function ara::core::StringView::rfind

Status: DRAFT

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type rfind (const char *s, size_type pos=npos) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11] except for the following deviations: 1. Function is noexcept

⌋

8.13.2.3.3.53 rfind

[SWS_CORE_02163] Definition of API function ara::core::StringView::rfind

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type rfind (const char *s, size_type pos, size_type n) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11] except for the following deviations: 1. Function is noexcept

⌋

8.13.2.3.3.54 rfind

[SWS_CORE_02162] Definition of API function ara::core::StringView::rfind

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type rfind (char c, size_type pos=npos) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.2.3.3.55 rfind

[SWS_CORE_02161] Definition of API function ara::core::StringView::rfind

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type rfind (StringView s, size_type pos=npos) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.2.3.3.56 size

[SWS_CORE_02128] Definition of API function ara::core::StringView::size

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type size () const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.2.3.3.57 starts_with

[SWS_CORE_02148] Definition of API function ara::core::StringView::starts_with

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr bool starts_with (StringView sv) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [16]

⌋

8.13.2.3.3.58 starts_with

[SWS_CORE_02150] Definition of API function ara::core::StringView::starts_with

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr bool starts_with (const char *str) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [16] except for the following deviations: 1. Function is noexcept

⌋

8.13.2.3.3.59 starts_with

[SWS_CORE_02149] Definition of API function ara::core::StringView::starts_with

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr bool starts_with (char c) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [16]

⌋

8.13.2.3.3.60 substr

[SWS_CORE_02141] Definition of API function ara::core::StringView::substr

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/string_view.h"	
Scope:	`class ara::core::StringView`	
Syntax:	`constexpr StringView substr (size_type pos=0, size_type n=npos) const noexcept;`	
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`StringViewOutOfRangeViolation`	If pos value exceeds the size of this StringView
Description:	As per corresponding function in `[string.view]` in [11] except for the following deviations: 1. Function is noexcept	

⌋

8.13.2.3.3.61 swap

[SWS_CORE_02139] Definition of API function ara::core::StringView::swap

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr void swap (StringView &other) noexcept;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.13.3 Struct: hash

[SWS_CORE_02189] Definition of API class std::hash< ara::core::StringView >

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	struct
Header file:	#include "ara/core/string_view.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	`namespace std`
Symbol:	hash< ara::core::StringView >

Syntax:	`template <>`
	`struct hash< ara::core::StringView > final {...};`

Description:	As per corresponding declaration in `[string.view]` in [11]

⌋

8.13.3.1 Public Member Functions

8.13.3.1.1 Member Functions

8.13.3.1.1.1 operator()

[SWS_CORE_02190] Definition of API function std::hash< ara::core::StringView >::operator()

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`struct std::hash< ara::core::StringView >`
Syntax:	`size_t operator() (ara::core::StringView const &v) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[string.view]` in [11]

⌋

8.14 Header: ara/core/string.h

8.14.1 Non-Member Types

8.14.1.1 Type Alias: String

[SWS_CORE_03001] Definition of API type ara::core::String

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/string.h"
Scope:	`namespace ara::core`
Symbol:	String
Syntax:	`using String = BasicString<>;`
Description:	String type.

⌋

8.14.2 Non-Member Functions

8.14.2.1 Other

8.14.2.1.1 swap

[SWS_CORE_03296] Definition of API function ara::core::swap

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename Allocator>`
	`void swap (BasicString< Allocator > &lhs, BasicString< Allocator > &rhs);`	

Template param:	Allocator	the allocator to use for any memory allocations
Parameters (in):	lhs	the first BasicString
	rhs	the second BasicString
Return value:	None	
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Exchange the state of lhs with that of rhs.	

⌋

8.14.3 Class: BasicString

[SWS_CORE_03000] Definition of API class ara::core::BasicString

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	class
Header file:	#include "ara/core/string.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	`namespace ara::core`
Symbol:	BasicString

Syntax:	`template <typename Allocator = <implementation-defined>>`
	`class BasicString final {...};`

▽

△

Template param:	typename Allocator = *<implementation-defined>*	the allocator type to use for any memory allocations
Description:	BasicString type	

⌋

8.14.3.1 Public Member Types

8.14.3.1.1 Type Alias: const_iterator

[SWS_CORE_00072] Definition of API type ara::core::BasicString::const_iterator

Status: DRAFT

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Symbol:	const_iterator
Syntax:	`using const_iterator = <implementation-defined>_ITERATOR;`
Description:	As per corresponding member in `[basic.string]` in [11]

⌋

8.14.3.1.2 Type Alias: iterator

[SWS_CORE_00071] Definition of API type ara::core::BasicString::iterator

Status: DRAFT

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Symbol:	iterator
Syntax:	`using iterator = <implementation-defined>;`
Description:	As per corresponding member in `[basic.string]` in [11]

⌋

8.14.3.1.3 Type Alias: size_type

[SWS_CORE_03012] Definition of API type ara::core::BasicString::size_type

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Symbol:	size_type
Syntax:	`using size_type = std::size_t;`
Description:	As per corresponding member in `[basic.string]` in [11] except for the following deviations: 1. Type shall be fixed to `std::size_t`

⌋

8.14.3.2 Public Member Variables

8.14.3.2.1 npos

[SWS_CORE_00073] Definition of API variable ara::core::BasicString::npos

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	variable
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Symbol:	npos
Type:	`size_type const`
Syntax:	`static constexpr size_type const npos = size_type(-1);`
Description:	As per corresponding member in `[basic.string]` in [11]

⌋

8.14.3.3 Public Member Functions

8.14.3.3.1 Constructors

8.14.3.3.1.1 BasicString

[SWS_CORE_03302] Definition of API function ara::core::BasicString::BasicString

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`explicit BasicString (StringView sv);`	
Parameters (in):	sv	a StringView
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Constructor from StringView.	

⌋

8.14.3.3.1.2 BasicString

[SWS_CORE_03303] Definition of API function ara::core::BasicString::BasicString

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	

Syntax:	`template <typename T>`
	`BasicString (const T &t, size_type pos, size_type n, const Allocator &alloc=Allocator());`	

Template param:	T	a type that is implicitly convertible to StringView
Parameters (in):	t	an instance of T
	pos	offset into t from where to start reading
	n	number of chars to read from t + pos
	alloc	the allocator instance to use

▽

△

Exception Safety:	not exception safe
Thread Safety:	implementation defined
Description:	Constructor from implicit StringView.

⌋

8.14.3.3.2 Member Functions

8.14.3.3.2.1 append

[SWS_CORE_03309] Definition of API function ara::core::BasicString::append

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	

Syntax:	`template <typename T>`
	`BasicString & append (const T &t, size_type pos, size_type n=npos);`	

Template param:	T	a type that is implicitly convertible to StringView
Parameters (in):	t	an instance of T
	pos	offset into t from where to start reading
	n	number of chars to read from t + pos
Return value:	BasicString &	*this
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Concatenation from implicit StringView.	

⌋

8.14.3.3.2.2 append

[SWS_CORE_03308] Definition of API function ara::core::BasicString::append

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`BasicString & append (StringView sv);`	
Parameters (in):	sv	the StringView
Return value:	BasicString &	*this
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Concatenation from StringView.	

⌋

8.14.3.3.2.3 assign

[SWS_CORE_03305] Definition of API function ara::core::BasicString::assign

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`BasicString & assign (StringView sv);`	
Parameters (in):	sv	the StringView
Return value:	BasicString &	*this
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Assignment from StringView.	

⌋

8.14.3.3.2.4 assign

[SWS_CORE_03306] Definition of API function ara::core::BasicString::assign

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	

Syntax:	`template <typename T>`
	`BasicString & assign (const T &t, size_type pos, size_type n=npos);`	

Template param:	T	a type that is implicitly convertible to StringView
Parameters (in):	t	an instance of T
	pos	offset into t from where to start reading
	n	number of chars to read from t + pos
Return value:	BasicString &	*this
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Assignment from implicit StringView.	

⌋

8.14.3.3.2.5 compare

[SWS_CORE_03321] Definition of API function ara::core::BasicString::compare

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`int compare (StringView sv) const noexcept;`	
Parameters (in):	sv	the StringView
Return value:	int	as per description of std::string::compare
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Compare with a StringView.	

⌋

8.14.3.3.2.6 compare

[SWS_CORE_03322] Definition of API function ara::core::BasicString::compare

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`int compare (size_type pos1, size_type n1, StringView sv) const;`	
Parameters (in):	pos1	index into *this from where to start comparing
	n1	number of chars at *this + pos1 to compare
	sv	the StringView
Return value:	int	as per description of std::string::compare
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Compare with a StringView.	

⌋

8.14.3.3.2.7 compare

[SWS_CORE_03323] Definition of API function ara::core::BasicString::compare

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	

Syntax:	`template <typename T>`
	`int compare (size_type pos1, size_type n1, const T &t, size_type pos2, size_type n2=npos) const;`	

Parameters (in):	pos1	index into *this from where to start comparing
	n1	number of chars at *this + pos1 to compare
	t	an instance of T
	pos2	index into t from where to start reading
	n2	number of chars to read from t + pos2
Return value:	int	as per description of std::string::compare
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	

▽

△

Description:	Compare with an implicit StringView.

⌋

8.14.3.3.2.8 find

[SWS_CORE_03315] Definition of API function ara::core::BasicString::find

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`size_type find (StringView sv, size_type pos=0) const noexcept;`	
Parameters (in):	sv	the StringView
	pos	index into *this from where to start searching
Return value:	size_type	index of the first character of the found substring, or npos if no such substring is found
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Find the first substring equal to the given StringView.	

⌋

8.14.3.3.2.9 find_first_not_of

[SWS_CORE_03319] Definition of API function ara::core::BasicString::find_first_not_of

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`size_type find_first_not_of (StringView sv, size_type pos=0) const noexcept;`	
Parameters (in):	sv	the StringView

▽

△

	pos	index into *this from where to start searching
Return value:	size_type	index of the found character, or npos if no such character is found
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Find the first character that is not one of the characters in the given StringView.	

⌋

8.14.3.3.2.10 find_first_of

[SWS_CORE_03317] Definition of API function ara::core::BasicString::find_first_of

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`size_type find_first_of (StringView sv, size_type pos=0) const noexcept;`	
Parameters (in):	sv	the StringView
	pos	index into *this from where to start searching
Return value:	size_type	index of the found character, or npos if no such character is found
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Find the first character equal to one of the characters in the given StringView.	

⌋

8.14.3.3.2.11 find_last_not_of

[SWS_CORE_03320] Definition of API function ara::core::BasicString::find_last_not_of

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`size_type find_last_not_of (StringView sv, size_type pos=npos) const noexcept;`	
Parameters (in):	sv	the StringView
	pos	index into *this from where to start searching
Return value:	size_type	index of the found character, or npos if no such character is found
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Find the last character that is not one of the characters in the given StringView.	

⌋

8.14.3.3.2.12 find_last_of

[SWS_CORE_03318] Definition of API function ara::core::BasicString::find_last_of

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`size_type find_last_of (StringView sv, size_type pos=npos) const noexcept;`	
Parameters (in):	sv	the StringView
	pos	index into *this from where to start searching
Return value:	size_type	index of the found character, or npos if no such character is found
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Find the last character equal to one of the characters in the given StringView.	

⌋

8.14.3.3.2.13 insert

[SWS_CORE_03311] Definition of API function ara::core::BasicString::insert

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	

Syntax:	`template <typename T>`
	`BasicString & insert (size_type pos1, const T &t, size_type pos2, size_type n=npos);`	

Template param:	T	a type that is implicitly convertible to StringView
Parameters (in):	pos1	index into *this before which to insert
	t	an instance of T
	pos2	index into t from where to start reading
	n	number of chars to read from t + pos
Return value:	BasicString &	*this
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Insertion of implicit StringView.	

⌋

8.14.3.3.2.14 insert

[SWS_CORE_03310] Definition of API function ara::core::BasicString::insert

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`BasicString & insert (size_type pos, StringView sv);`	
Parameters (in):	pos	position in *this before which to insert
	sv	the StringView
Return value:	BasicString &	*this
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Insertion of StringView.	

⌋

8.14.3.3.2.15 operator StringView

[SWS_CORE_03301] Definition of API function ara::core::BasicString::operator StringView

Status: DRAFT

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`operator StringView () const noexcept;`	
Return value:	StringView	a StringView
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Implicit conversion to StringView.	

⌋

8.14.3.3.2.16 operator+=

[SWS_CORE_03307] Definition of API function ara::core::BasicString::operator+=

Status: DRAFT

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`BasicString & operator+= (StringView sv);`	
Parameters (in):	sv	the StringView
Return value:	BasicString &	*this
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Concatenation operator from StringView.	

⌋

8.14.3.3.2.17 operator=

[SWS_CORE_03304] Definition of API function ara::core::BasicString::operator=

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`BasicString & operator= (StringView sv);`	
Parameters (in):	sv	the StringView
Return value:	BasicString &	*this
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Assignment operator from StringView.	

⌋

8.14.3.3.2.18 replace

[SWS_CORE_03314] Definition of API function ara::core::BasicString::replace

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`BasicString & replace (const_iterator i1, const_iterator i2, StringView sv);`	
Parameters (in):	i1	iterator pointing into *this to where replacement will start
	i2	iterator pointing into *this to where replacement will end
	sv	the StringView
Return value:	BasicString &	*this
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Replacement of iterator range with StringView.	

⌋

8.14.3.3.2.19 replace

[SWS_CORE_03313] Definition of API function ara::core::BasicString::replace

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	

Syntax:	`template <typename T>`
	`BasicString & replace (size_type pos1, size_type n1, const T &t, size_type pos2, size_type n2=npos);`	

Template param:	T	a type that is implicitly convertible to StringView
Parameters (in):	pos1	index into *this before where replacement will start
	n1	number of chars to replace from *this + pos1
	t	an instance of T
	pos2	index into t from where to start reading
	n2	number of chars to read from t + pos2
Return value:	BasicString &	*this
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Replacement with implicit StringView.	

⌋

8.14.3.3.2.20 replace

[SWS_CORE_03312] Definition of API function ara::core::BasicString::replace

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`BasicString & replace (size_type pos1, size_type n1, StringView sv);`	
Parameters (in):	pos1	index into *this where replacement will start
	n1	index into sv from where to start reading
	sv	the StringView
Return value:	BasicString &	*this
Exception Safety:	not exception safe	

▽

△

Thread Safety:	implementation defined
Description:	Replacement with StringView.

⌋

8.14.3.3.2.21 rfind

[SWS_CORE_03316] Definition of API function ara::core::BasicString::rfind

Status: DRAFT

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`size_type rfind (StringView sv, size_type pos=npos) const noexcept;`	
Parameters (in):	sv	the StringView
	pos	index into *this from where to start searching
Return value:	size_type	index of the first character of the found substring, or npos if no such substring is found
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Find the last substring equal to the given StringView.	

⌋

8.15 Header: ara/core/span.h

8.15.1 Global Variables

8.15.1.1 dynamic_extent

[SWS_CORE_01901] Definition of API variable ara::core::dynamic_extent

Upstream requirements: RS_AP_00130

⌈

Kind:	variable
Header file:	#include "ara/core/span.h"
Scope:	`namespace ara::core`
Symbol:	dynamic_extent
Type:	`std::size_t`
Syntax:	`constexpr std::size_t dynamic_extent = std::numeric_limits<std::size_t>::max();`
Description:	A constant for creating Spans with dynamic sizes. The constant is always set to std::numeric_limits<std::size_t>::max().

⌋

8.15.2 Non-Member Functions

8.15.2.1 Other

8.15.2.1.1 MakeSpan

[SWS_CORE_01990] Definition of API function ara::core::MakeSpan

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T>`
	`constexpr Span< T > MakeSpan (T *ptr, typename Span< T >::size_type count) noexcept;`	

Template param:	T	the type of elements
Parameters (in):	ptr	the pointer
	count	the number of elements to take from ptr
Return value:	Span< T >	the new Span

▽

△

Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	Create a new Span from the given pointer and size.

⌋

8.15.2.1.2 MakeSpan

[SWS_CORE_01991] Definition of API function ara::core::MakeSpan

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T>`
	`constexpr Span< T > MakeSpan (T *firstElem, T *lastElem) noexcept;`	

Template param:	T	the type of elements
Parameters (in):	firstElem	pointer to the first element
	lastElem	pointer to past the last element
Return value:	Span< T >	the new Span
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Create a new Span from the open range between [firstElem, lastElem).	

⌋

8.15.2.1.3 MakeSpan

[SWS_CORE_01992] Definition of API function ara::core::MakeSpan

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`namespace ara::core`

Syntax:	`template <typename T, std::size_t N>`
	`constexpr Span< T, N > MakeSpan (T(&arr)[N]) noexcept;`

▽

△

Template param:	T	the type of elements
	N	the size of the raw array
Parameters (in):	arr	the raw array
Return value:	Span< T, N >	the new Span
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Create a new Span from the given raw array.	

⌋

8.15.2.1.4 MakeSpan

[SWS_CORE_01993] Definition of API function ara::core::MakeSpan

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename Container>`
	`constexpr Span< typename Container::value_type > MakeSpan (Container &cont) noexcept;`	

Template param:	Container	the type of container
Parameters (in):	cont	the container
Return value:	Span< typename Container::value_type >	the new Span
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Create a new Span from the given container.	

⌋

8.15.2.1.5 MakeSpan

[SWS_CORE_01994] Definition of API function ara::core::MakeSpan

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	namespace ara::core	
Syntax:	template <typename Container> constexpr Span< typename Container::value_type const > MakeSpan (const Container &cont) noexcept;	
Template param:	Container	the type of container
Parameters (in):	cont	the container
Return value:	Span< typename Container::value_type const >	the new Span
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Create a new Span from the given const container.	

⌋

8.15.2.1.6 as_bytes

[SWS_CORE_01980] Definition of API function ara::core::as_bytes

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	namespace ara::core	
Syntax:	template <typename ElementType, std::size_t Extent> Span< const Byte, Extent==dynamic_extent ? dynamic_extent :sizeof(ElementType) *Extent > as_bytes (Span< ElementType, Extent > s) noexcept;	
Parameters (in):	s	the input Span<T>
Return value:	Span< const Byte, Extent==dynamic_extent ? dynamic_extent :sizeof(ElementType) *Extent >	a Span<const Byte>
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a read-only Span<Byte> over the object representation of the input Span<T>	

⌋

8.15.2.1.7 as_writable_bytes

[SWS_CORE_01981] Definition of API function ara::core::as_writable_bytes

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename ElementType, std::size_t Extent>`
	`Span< Byte, Extent==dynamic_extent ? dynamic_extent :sizeof(ElementType) *Extent > as_writable_bytes (Span< ElementType, Extent > s) noexcept;`	

Parameters (in):	s	the input Span<T>
Return value:	Span< Byte, Extent==dynamic_extent ? dynamic_extent :sizeof(ElementType) *Extent >	a Span<Byte>
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a writable Span<Byte> over the object representation of the input Span<T>	

⌋

8.15.3 Class: Span

[SWS_CORE_01900] Definition of API class ara::core::Span

Upstream requirements: RS_AP_00130

⌈

Kind:	class	
Header file:	#include "ara/core/span.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace ara::core`	
Symbol:	Span	

Syntax:	`template <typename T, std::size_t Extent = dynamic_extent>`
	`class Span {...};`	

Template param:	typename T	the type of elements in the Span
	std::size_t Extent = dynamic_extent	the extent to use for this Span
Description:	A view over a contiguous sequence of objects. The type T is required to be a complete object type that is not an abstract class type.	

⌋

8.15.3.1 Public Member Types

8.15.3.1.1 Type Alias: const_iterator

[SWS_CORE_01918] Definition of API type ara::core::Span::const_iterator

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Symbol:	const_iterator
Syntax:	`using const_iterator = <implementation-defined>;`
Description:	The type of a const_iterator to elements. This iterator shall implement the concepts RandomAccessIterator, ContiguousIterator, and ConstexprIterator.

⌋

8.15.3.1.2 Type Alias: const_pointer

[SWS_CORE_01922] Definition of API type ara::core::Span::const_pointer

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Symbol:	const_pointer
Syntax:	`using const_pointer = const element_type*;`
Description:	Alias type for a pointer to a constant element.

⌋

8.15.3.1.3 Type Alias: const_reference

[SWS_CORE_01923] Definition of API type ara::core::Span::const_reference

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Symbol:	const_reference
Syntax:	`using const_reference = const element_type&;`
Description:	Alias type for a reference to a constant element.

⌋

8.15.3.1.4 Type Alias: const_reverse_iterator

[SWS_CORE_01920] Definition of API type ara::core::Span::const_reverse_iterator

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Symbol:	const_reverse_iterator
Syntax:	`using const_reverse_iterator = std::reverse_iterator<const_iterator>;`
Description:	The type of a const_reverse_iterator to elements.

⌋

8.15.3.1.5 Type Alias: difference_type

[SWS_CORE_01914] Definition of API type ara::core::Span::difference_type

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Symbol:	difference_type
Syntax:	`using difference_type = std::ptrdiff_t;`
Description:	Alias for the type of parameters that indicate a difference of indexes into the Span.

⌋

8.15.3.1.6 Type Alias: element_type

[SWS_CORE_01911] Definition of API type ara::core::Span::element_type

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Symbol:	element_type
Syntax:	`using element_type = T;`
Description:	Alias for the type of elements in this Span.

⌋

8.15.3.1.7 Type Alias: iterator

[SWS_CORE_01917] Definition of API type ara::core::Span::iterator

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Symbol:	iterator
Syntax:	`using iterator = <implementation-defined>;`
Description:	The type of an iterator to elements. This iterator shall implement the concepts RandomAccessIterator, ContiguousIterator, and ConstexprIterator.

⌋

8.15.3.1.8 Type Alias: pointer

[SWS_CORE_01915] Definition of API type ara::core::Span::pointer

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Symbol:	pointer
Syntax:	`using pointer = element_type*;`
Description:	Alias type for a pointer to an element.

⌋

8.15.3.1.9 Type Alias: reference

[SWS_CORE_01916] Definition of API type ara::core::Span::reference

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Symbol:	reference
Syntax:	`using reference = element_type&;`
Description:	Alias type for a reference to an element.

⌋

8.15.3.1.10 Type Alias: reverse_iterator

[SWS_CORE_01919] Definition of API type ara::core::Span::reverse_iterator

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Symbol:	reverse_iterator
Syntax:	`using reverse_iterator = std::reverse_iterator<iterator>;`
Description:	The type of a reverse_iterator to elements.

⌋

8.15.3.1.11 Type Alias: size_type

[SWS_CORE_01921] Definition of API type ara::core::Span::size_type

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Symbol:	size_type
Syntax:	`using size_type = std::size_t;`
Description:	Alias for the type of parameters that indicate a size or a number of values.

⌋

8.15.3.1.12 Type Alias: value_type

[SWS_CORE_01912] Definition of API type ara::core::Span::value_type

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Symbol:	value_type
Syntax:	`using value_type = typename std::remove_cv<element_type>::type;`
Description:	Alias for the type of values in this Span.

⌋

8.15.3.2 Public Member Variables

8.15.3.2.1 extent

[SWS_CORE_01931] Definition of API variable ara::core::Span::extent

Upstream requirements: RS_AP_00130

⌈

Kind:	variable
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Symbol:	extent
Type:	`size_type`
Syntax:	`static constexpr size_type extent = Extent;`
Description:	A constant reflecting the configured Extent of this Span.

⌋

8.15.3.3 Public Member Functions

8.15.3.3.1 Special Member Functions

8.15.3.3.1.1 Copy Constructor

[SWS_CORE_01949] Definition of API function ara::core::Span::Span

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	
Syntax:	`constexpr Span (const Span &other) noexcept=default;`	
Parameters (in):	other	the other instance
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Copy construct a new Span from another instance.	

⌋

8.15.3.3.1.2 Copy Constructor

[SWS_CORE_01950] Definition of API function ara::core::Span::Span

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	

Syntax:	`template <typename U, std::size_t N>`
	`constexpr Span (const Span< U, N > &s) noexcept;`	

Template param:	U	the type of elements within the other Span
	N	the Extent of the other Span
Parameters (in):	s	the other Span instance
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Converting constructor. This ctor allows construction of a cv-qualified Span from a normal Span, and also of a dynamic_extent-Span<> from a static extent-one. This constructor shall not participate in overload resolution unless: • Extent == dynamic_extent || Extent == N is true, • U(*)[] is convertible to T(*)[]	

⌋

8.15.3.3.1.3 Default Constructor

[SWS_CORE_01941] Definition of API function ara::core::Span::Span

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`constexpr Span () noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	Default constructor. This constructor shall not participate in overload resolution unless (Extent == dynamic_extent || Extent == 0) is true.

⌋

8.15.3.3.1.4 Copy Assignment Operator

[SWS_CORE_01952] Definition of API function ara::core::Span::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	
Syntax:	`constexpr Span & operator= (const Span &other) noexcept=default;`	
Parameters (in):	other	the other instance
Return value:	Span &	*this
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Copy assignment operator.	

⌋

8.15.3.3.1.5 Destructor

[SWS_CORE_01951] Definition of API function ara::core::Span::~Span

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`~Span () noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	Destructor.

⌋

8.15.3.3.2 Constructors

8.15.3.3.2.1 Span

[SWS_CORE_01942] Definition of API function ara::core::Span::Span

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	
Syntax:	`constexpr Span (pointer ptr, size_type count) noexcept;`	
Parameters (in):	ptr	the pointer
	count	the number of elements to take from ptr
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Construct a new Span from the given pointer and size. [ptr, ptr + count) shall be a valid range. If extent is not equal to dynamic_extent, then count shall be equal to Extent.	

⌋

8.15.3.3.2.2 Span

[SWS_CORE_01943] Definition of API function ara::core::Span::Span

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	
Syntax:	`constexpr Span (pointer firstElem, pointer lastElem) noexcept;`	
Parameters (in):	firstElem	pointer to the first element
	lastElem	pointer to past the last element
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Construct a new Span from the open range between [firstElem, lastElem). [firstElem, lastElem) shall be a valid range. If extent is not equal to dynamic_extent, then (lastElem - firstElem) shall be equal to extent.	

⌋

8.15.3.3.2.3 Span

[SWS_CORE_01944] Definition of API function ara::core::Span::Span

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	

Syntax:	`template <std::size_t N>`
	`constexpr Span (element_type(&arr)[N]) noexcept;`	

Template param:	N	the size of the raw array
Parameters (in):	arr	the raw array
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Construct a new Span from the given raw array. This constructor shall not participate in overload resolution unless: • extent == dynamic_extent || N == extent is true, and • std::remove_pointer_t<decltype(ara::core::data(arr))>(*)[] is convertible to T(*)[].	

⌋

8.15.3.3.2.4 Span

[SWS_CORE_01953] Definition of API function ara::core::Span::Span

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	

Syntax:	`template <typename U, std::size_t N>`
	`constexpr Span (std::array< U, N > &arr) noexcept;`	

Template param:	U	the type of elements within the std::array
	N	the size of the std::array
Parameters (in):	arr	the std::array
Exception Safety:	exception safe	
Thread Safety:	implementation defined	

▽

△

Description:	Construct a new Span from the given std::array. This constructor shall not participate in overload resolution unless: • extent == dynamic_extent || N == extent is true, and • std::remove_pointer_t<decltype(std::data(arr))>(*)[] is convertible to T(*)[].

⌋

8.15.3.3.2.5 Span

[SWS_CORE_01954] Definition of API function ara::core::Span::Span

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	

Syntax:	`template <typename U, std::size_t N>`
	`constexpr Span (const std::array< U, N > &arr) noexcept;`	

Template param:	U	the type of elements within the std::array
	N	the size of the std::array
Parameters (in):	arr	the std::array
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Construct a new Span from the given const std::array. This constructor shall not participate in overload resolution unless: • extent == dynamic_extent || N == extent is true, and • std::remove_pointer_t<decltype(std::data(arr))>(*)[] is convertible to T(*)[].	

⌋

8.15.3.3.2.6 Span

[SWS_CORE_01945] Definition of API function ara::core::Span::Span

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	

Syntax:	`template <typename U, std::size_t N>`
	`constexpr Span (Array< U, N > &arr) noexcept;`	

Template param:	U	the type of elements within the Array
	N	the size of the Array
Parameters (in):	arr	the array
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Construct a new Span from the given Array. This constructor shall not participate in overload resolution unless: • extent == dynamic_extent || N == extent is true, and • std::remove_pointer_t<decltype(ara::core::data(arr))>(*)[] is convertible to T(*)[].	

⌋

8.15.3.3.2.7 Span

[SWS_CORE_01946] Definition of API function ara::core::Span::Span

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	

Syntax:	`template <typename U, std::size_t N>`
	`constexpr Span (const Array< U, N > &arr) noexcept;`	

Template param:	U	the type of elements within the Array
	N	the size of the Array
Parameters (in):	arr	the array
Exception Safety:	exception safe	
Thread Safety:	implementation defined	

▽

△

Description:	Construct a new Span from the given const Array. This constructor shall not participate in overload resolution unless: • extent == dynamic_extent || N == extent is true, and • std::remove_pointer_t<decltype(ara::core::data(arr))>(*)[] is convertible to T(*)[].

⌋

8.15.3.3.2.8 Span

[SWS_CORE_01947] Definition of API function ara::core::Span::Span

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	

Syntax:	`template <typename Container>`
	`constexpr Span (Container &cont) noexcept;`	

Template param:	Container	the type of container
Parameters (in):	cont	the container
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Construct a new Span from the given container. [ara::core::data(cont), ara::core::data(cont) + ara::core::size(cont)) shall be a valid range. This constructor shall not participate in overload resolution unless: • extent == dynamic_extent is true, • Container is not a specialization of Span, • Container is not a specialization of Array, • Container is not a specialization of std::array, • std::is_array<Container>::value is false, • ara::core::data(cont) and ara::core::size(cont) are both well-formed, and • std::remove_pointer_t<decltype(ara::core::data(cont))>(*)[] is convertible to T(*)[].	

⌋

8.15.3.3.2.9 Span

[SWS_CORE_01948] Definition of API function ara::core::Span::Span

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	

Syntax:	`template <typename Container>`
	`constexpr Span (const Container &cont) noexcept;`	

Template param:	Container	the type of container
Parameters (in):	cont	the container
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Construct a new Span from the given const container. [ara::core::data(cont), ara::core::data(cont) + ara::core::size(cont)) shall be a valid range. This constructor shall not participate in overload resolution unless: • extent == dynamic_extent is true, • Container is not a specialization of Span, • Container is not a specialization of Array, • Container is not a specialization of std::array, • std::is_array<Container>::value is false, • ara::core::data(cont) and ara::core::size(cont) are both well-formed, and • std::remove_pointer<decltype(ara::core::data(cont))>::type(*)[] is convertible to T(*)[].	

⌋

8.15.3.3.3 Member Functions

8.15.3.3.3.1 back

[SWS_CORE_01960] Definition of API function ara::core::Span::back

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	
Syntax:	`constexpr reference back () const noexcept;`	
Return value:	reference	the reference

▽

△

Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	Return a reference to the last element of this Span. The behavior of this function is undefined if empty() is true.

⌋

8.15.3.3.3.2 begin

[SWS_CORE_01972] Definition of API function ara::core::Span::begin

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	
Syntax:	`constexpr iterator begin () const noexcept;`	
Return value:	iterator	the iterator
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return an iterator pointing to the first element of this Span.	

⌋

8.15.3.3.3.3 cbegin

[SWS_CORE_01974] Definition of API function ara::core::Span::cbegin

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	
Syntax:	`constexpr const_iterator cbegin () const noexcept;`	
Return value:	const_iterator	the const_iterator
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a const_iterator pointing to the first element of this Span.	

⌋

8.15.3.3.3.4 cend

[SWS_CORE_01975] Definition of API function ara::core::Span::cend

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	
Syntax:	`constexpr const_iterator cend () const noexcept;`	
Return value:	const_iterator	the const_iterator
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a const_iterator pointing past the last element of this Span.	

⌋

8.15.3.3.3.5 crbegin

[SWS_CORE_01978] Definition of API function ara::core::Span::crbegin

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	
Syntax:	`constexpr const_reverse_iterator crbegin () const noexcept;`	
Return value:	const_reverse_iterator	the const_reverse_iterator
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a const_reverse_iterator pointing to the last element of this Span.	

⌋

8.15.3.3.3.6 crend

[SWS_CORE_01979] Definition of API function ara::core::Span::crend

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	
Syntax:	`constexpr const_reverse_iterator crend () const noexcept;`	
Return value:	const_reverse_iterator	the reverse_iterator
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a const_reverse_iterator pointing past the first element of this Span.	

⌋

8.15.3.3.3.7 data

[SWS_CORE_01971] Definition of API function ara::core::Span::data

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	
Syntax:	`constexpr pointer data () const noexcept;`	
Return value:	pointer	the pointer
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a pointer to the start of the memory block covered by this Span.	

⌋

8.15.3.3.3.8 empty

[SWS_CORE_01969] Definition of API function ara::core::Span::empty

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	
Syntax:	`constexpr bool empty () const noexcept;`	
Return value:	bool	true if this Span contains 0 elements, false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return whether this Span is empty.	

⌋

8.15.3.3.3.9 end

[SWS_CORE_01973] Definition of API function ara::core::Span::end

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	
Syntax:	`constexpr iterator end () const noexcept;`	
Return value:	iterator	the iterator
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return an iterator pointing past the last element of this Span.	

⌋

8.15.3.3.3.10 first

[SWS_CORE_01962] Definition of API function ara::core::Span::first

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	
Syntax:	`constexpr Span< element_type, dynamic_extent > first (size_type count) const noexcept;`	
Parameters (in):	count	the number of elements to take over
Return value:	Span< element_type, dynamic_extent >	the subspan
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a subspan containing only the first elements of this Span. The behavior of this function is undefined if (count > size()).	

⌋

8.15.3.3.3.11 first

[SWS_CORE_01961] Definition of API function ara::core::Span::first

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	

Syntax:	`template <std::size_t Count>`
	`constexpr Span< element_type, Count > first () const noexcept;`	

Template param:	Count	the number of elements to take over
Return value:	Span< element_type, Count >	the subspan
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a subspan containing only the first elements of this Span. The implementation shall ensure that (Count <= Extent) is true. The behavior of this function is undefined if (Count > size()).	

⌋

8.15.3.3.3.12 front

[SWS_CORE_01959] Definition of API function ara::core::Span::front

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	
Syntax:	`constexpr reference front () const noexcept;`	
Return value:	reference	the reference
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a reference to the first element of this Span. The behavior of this function is undefined if empty() is true.	

⌋

8.15.3.3.3.13 last

[SWS_CORE_01963] Definition of API function ara::core::Span::last

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	

Syntax:	`template <std::size_t Count>`
	`constexpr Span< element_type, Count > last () const noexcept;`	

Template param:	Count	the number of elements to take over
Return value:	Span< element_type, Count >	the subspan
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a subspan containing only the last elements of this Span. The implementation shall ensure that (Count <= Extent) is true. The behavior of this function is undefined if (Count > size()).	

⌋

8.15.3.3.3.14 last

[SWS_CORE_01964] Definition of API function ara::core::Span::last

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	
Syntax:	`constexpr Span< element_type, dynamic_extent > last (size_type count) const noexcept;`	
Parameters (in):	count	the number of elements to take over
Return value:	Span< element_type, dynamic_extent >	the subspan
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a subspan containing only the last elements of this Span. The behavior of this function is undefined if (count > size()).	

⌋

8.15.3.3.3.15 operator[]

[SWS_CORE_01970] Definition of API function ara::core::Span::operator[]

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	
Syntax:	`constexpr reference operator[] (size_type idx) const noexcept;`	
Parameters (in):	idx	the index into this Span
Return value:	reference	the reference
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a reference to the n-th element of this Span.	

⌋

8.15.3.3.3.16 rbegin

[SWS_CORE_01976] Definition of API function ara::core::Span::rbegin

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	
Syntax:	`constexpr reverse_iterator rbegin () const noexcept;`	
Return value:	reverse_iterator	the reverse_iterator
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a reverse_iterator pointing to the last element of this Span.	

⌋

8.15.3.3.3.17 rend

[SWS_CORE_01977] Definition of API function ara::core::Span::rend

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	
Syntax:	`constexpr reverse_iterator rend () const noexcept;`	
Return value:	reverse_iterator	the reverse_iterator
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a reverse_iterator pointing past the first element of this Span.	

⌋

8.15.3.3.3.18 size

[SWS_CORE_01967] Definition of API function ara::core::Span::size

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	
Syntax:	`constexpr size_type size () const noexcept;`	
Return value:	size_type	the number of elements contained in this Span
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return the size of this Span.	

⌋

8.15.3.3.3.19 size_bytes

[SWS_CORE_01968] Definition of API function ara::core::Span::size_bytes

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	
Syntax:	`constexpr size_type size_bytes () const noexcept;`	
Return value:	size_type	the number of bytes covered by this Span
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return the size of this Span in bytes.	

⌋

8.15.3.3.3.20 subspan

[SWS_CORE_01966] Definition of API function ara::core::Span::subspan

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	
Syntax:	`constexpr Span< element_type, dynamic_extent > subspan (size_type offset, size_type count=dynamic_extent) const noexcept;`	
Parameters (in):	offset	offset into this Span from which to start
	count	the number of elements to take over
Return value:	Span< element_type, dynamic_extent >	the subspan
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a subspan of this Span. The behavior of this function is undefined unless (offset <= size() && (count == dynamic_extent || count <= size() - offset)) is true.	

⌋

8.15.3.3.3.21 subspan

[SWS_CORE_01965] Definition of API function ara::core::Span::subspan

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	
Syntax:	`template <std::size_t Offset, std::size_t Count = dynamic_extent> constexpr auto subspan () const noexcept -> Span< element_type, <see below> >;`	
Template param:	Offset	offset into this Span from which to start
	Count	the number of elements to take over
Return value:	Span< element_type, *<see below>* >	the subspan
Exception Safety:	exception safe	
Thread Safety:	thread-safe	

▽

△

Description:	Return a subspan of this Span. The second template argument of the returned Span type is: Count != dynamic_extent ? Count : (Extent != dynamic_extent ? Extent - Offset : dynamic_extent) The implementation shall ensure that (Offset <= Extent && (Count == dynamic_extent || Count <= Extent - Offset)) is true. The behavior of this function is undefined unless (Offset <= size() && (Count == dynamic_extent || Count <= size() - Offset)) is true.

⌋

8.16 Header: ara/core/steady_clock.h

8.16.1 Class: SteadyClock

[SWS_CORE_06401] Definition of API class ara::core::SteadyClock

Upstream requirements: RS_AP_00130

⌈

Kind:	class
Header file:	#include "ara/core/steady_clock.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	`namespace ara::core`
Symbol:	SteadyClock
Syntax:	`class SteadyClock final {...};`
Description:	This clock represents a monotonic clock. The time points of this clock cannot decrease as physical time moves forward and the time between ticks of this clock is constant.

⌋

8.16.1.1 Public Member Types

8.16.1.1.1 Type Alias: duration

[SWS_CORE_06411] Definition of API type ara::core::SteadyClock::duration

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/steady_clock.h"
Scope:	`class ara::core::SteadyClock`
Symbol:	duration
Syntax:	`using duration = std::chrono::duration<rep, period>;`
Description:	std::chrono::duration<rep, period>

⌋

8.16.1.1.2 Type Alias: period

[SWS_CORE_06413] Definition of API type ara::core::SteadyClock::period

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/steady_clock.h"
Scope:	`class ara::core::SteadyClock`
Symbol:	period
Syntax:	`using period = std::nano;`
Description:	A std::ratio type representing the tick period of the clock, in seconds .

⌋

8.16.1.1.3 Type Alias: rep

[SWS_CORE_06412] Definition of API type ara::core::SteadyClock::rep

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/steady_clock.h"
Scope:	`class ara::core::SteadyClock`
Symbol:	rep
Syntax:	`using rep = std::int64_t;`
Description:	An arithmetic type representing the number of ticks in the clock's duration .

⌋

8.16.1.1.4 Type Alias: time_point

[SWS_CORE_06414] Definition of API type ara::core::SteadyClock::time_point

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/steady_clock.h"
Scope:	`class ara::core::SteadyClock`
Symbol:	time_point
Syntax:	`using time_point = std::chrono::time_point<SteadyClock, duration>;`
Description:	std::chrono::time_point<ara::core::SteadyClock>

⌋

8.16.1.2 Public Member Variables

8.16.1.2.1 is_steady

[SWS_CORE_06431] Definition of API variable ara::core::SteadyClock::is_steady

Upstream requirements: RS_AP_00130

⌈

Kind:	variable
Header file:	#include "ara/core/steady_clock.h"
Scope:	`class ara::core::SteadyClock`
Symbol:	is_steady
Type:	`bool`
Syntax:	`static constexpr bool is_steady = true;`
Description:	steady clock flag, always true

⌋

8.16.1.3 Public Member Functions

8.16.1.3.1 Member Functions

8.16.1.3.1.1 now

[SWS_CORE_06432] Definition of API function ara::core::SteadyClock::now

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/steady_clock.h"	
Scope:	`class ara::core::SteadyClock`	
Syntax:	`static time_point now () noexcept;`	
Return value:	time_point	a time_point
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a time_point representing the current value of the clock.	

⌋

8.17 Header: ara/core/instance_specifier.h

8.17.1 Non-Member Functions

8.17.1.1 Other

8.17.1.1.1 operator!=

[SWS_CORE_08082] Definition of API function ara::core::operator!=

Upstream requirements: RS_Main_00320

⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	`namespace ara::core`	
Syntax:	`bool operator!= (ara::core::StringView lhs, const InstanceSpecifier &rhs) noexcept;`	
Parameters (in):	lhs	stringified form of a `ara::core::InstanceSpecifier`
	rhs	an `ara::core::InstanceSpecifier`
Return value:	bool	true in case rhs string representation not equals lhs
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Negative equality comparison of two `ara::core::InstanceSpecifier`s	

⌋

8.17.1.1.2 operator==

[SWS_CORE_08081] Definition of API function ara::core::operator==

Upstream requirements: RS_Main_00320

⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	`namespace ara::core`	
Syntax:	`bool operator== (ara::core::StringView lhs, const InstanceSpecifier &rhs) noexcept;`	
Parameters (in):	lhs	stringified form of a `ara::core::InstanceSpecifier`
	rhs	an `ara::core::InstanceSpecifier`
Return value:	bool	true in case rhs string representation equals lhs
Exception Safety:	exception safe	
Thread Safety:	thread-safe	

▽

△

Description:	Equality comparison of two ara::core::InstanceSpecifiers

⌋

8.17.2 Class: InstanceSpecifier

[SWS_CORE_08001] Definition of API class ara::core::InstanceSpecifier

Upstream requirements: RS_AP_00140, RS_Main_00320

⌈

Kind:	class
Header file:	#include "ara/core/instance_specifier.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	`namespace ara::core`
Symbol:	InstanceSpecifier
Syntax:	`class InstanceSpecifier final {...};`
Description:	Representation of an AUTOSAR Instance Specifier, which is basically an AUTOSAR shortName path wrapper.

⌋

8.17.2.1 Public Member Functions

8.17.2.1.1 Special Member Functions

8.17.2.1.1.1 Move Constructor

[SWS_CORE_08023] Definition of API function ara::core::InstanceSpecifier::InstanceSpecifier

Upstream requirements: RS_Main_00320

⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	`class ara::core::InstanceSpecifier`	
Syntax:	`InstanceSpecifier (InstanceSpecifier &&other) noexcept;`	
Parameters (in):	other	the other instance
Exception Safety:	exception safe	

▽

△

Thread Safety:	implementation defined
Description:	Move constructor

⌋

8.17.2.1.1.2 Copy Constructor

[SWS_CORE_08022] Definition of API function ara::core::InstanceSpecifier::InstanceSpecifier

Upstream requirements: RS_Main_00320

⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	`class ara::core::InstanceSpecifier`	
Syntax:	`InstanceSpecifier (const InstanceSpecifier &other) noexcept;`	
Parameters (in):	other	the other instance
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Copy constructor	

⌋

8.17.2.1.1.3 Copy Assignment Operator

[SWS_CORE_08024] Definition of API function ara::core::InstanceSpecifier::operator=

Upstream requirements: RS_Main_00320

⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	`class ara::core::InstanceSpecifier`	
Syntax:	`InstanceSpecifier & operator= (const InstanceSpecifier &other) noexcept;`	
Parameters (in):	other	the other instance
Return value:	InstanceSpecifier &	*this
Exception Safety:	exception safe	

▽

△

Thread Safety:	not thread-safe	
Description:	Copy assignment operator	

⌋

8.17.2.1.1.4 Move Assignment Operator

[SWS_CORE_08025] Definition of API function ara::core::InstanceSpecifier::operator=

Upstream requirements: RS_Main_00320

⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	`class ara::core::InstanceSpecifier`	
Syntax:	`InstanceSpecifier & operator= (InstanceSpecifier &&other) noexcept;`	
Parameters (in):	other	the other instance
Return value:	InstanceSpecifier &	*this
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Move assignment operator	

⌋

8.17.2.1.1.5 Destructor

[SWS_CORE_08029] Definition of API function ara::core::InstanceSpecifier::~InstanceSpecifier

Upstream requirements: RS_AP_00134, RS_Main_00320

⌈

Kind:	function
Header file:	#include "ara/core/instance_specifier.h"
Scope:	`class ara::core::InstanceSpecifier`
Syntax:	`~InstanceSpecifier () noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	Destructor

⌋

8.17.2.1.2 Constructors

8.17.2.1.2.1 InstanceSpecifier

[SWS_CORE_08021] Definition of API function ara::core::InstanceSpecifier::InstanceSpecifier

Upstream requirements: RS_Main_00320

⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	`class ara::core::InstanceSpecifier`	
Syntax:	`explicit InstanceSpecifier (ara::core::StringView qualifiedShortName) noexcept(false);`	
Parameters (in):	qualifiedShortName	String representation of a `ara::core::InstanceSpecifier`, according to the syntax rules in [SWS_CORE_10200] and [SWS_CORE_10203].
Exceptions:	`ara::core::CoreException`	As per `ara::core::CoreErrc`::`kInvalidMetaModelPath` :- the `qualifiedShortName` is not a valid `shortName` path to a model element.
	`ara::core::CoreException`	As per `ara::core::CoreErrc`::`kInvalidMetaModelShortname` :- the `qualifiedShortName` contains invalid characters or missing `shortName` path elements.
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Construct an `ara::core::InstanceSpecifier` from a `ara::core::StringView`	

⌋

8.17.2.1.3 Member Functions

8.17.2.1.3.1 ToString

[SWS_CORE_08041] Definition of API function ara::core::InstanceSpecifier::ToString

Upstream requirements: RS_Main_00320

⌈

Kind:	function
Header file:	#include "ara/core/instance_specifier.h"
Scope:	`class ara::core::InstanceSpecifier`
Syntax:	`ara::core::StringView ToString () const noexcept;`

▽

△

Return value:	ara::core::StringView	Stringified form of `ara::core::InstanceSpecifier`. Lifetime of the underlying string is only guaranteed for the lifetime of the underlying string of the `ara::core::StringView` passed to the constructor.
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	method to return the stringified form of `ara::core::InstanceSpecifier`	

⌋

8.17.2.1.3.2 operator!=

[SWS_CORE_08045] Definition of API function ara::core::InstanceSpecifier::operator!=

Upstream requirements: RS_Main_00320

⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	`class ara::core::InstanceSpecifier`	
Syntax:	`bool operator!= (ara::core::StringView other) const noexcept;`	
Parameters (in):	other	string representation to compare this one with
Return value:	bool	false in case this `ara::core::InstanceSpecifier` is denoting exactly the same model element as other, true otherwise.
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	uneq operator to compare with other `ara::core::InstanceSpecifier` string representation.	

⌋

8.17.2.1.3.3 operator!=

[SWS_CORE_08044] Definition of API function ara::core::InstanceSpecifier::operator!=

Upstream requirements: RS_Main_00320

⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	`class ara::core::InstanceSpecifier`	
Syntax:	`bool operator!= (const InstanceSpecifier &other) const noexcept;`	
Parameters (in):	other	`ara::core::InstanceSpecifier` instance to compare this one with.
Return value:	bool	false in case both `ara::core::InstanceSpecifier`s are denoting exactly the same model element, true otherwise.
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	uneq operator to compare with other `ara::core::InstanceSpecifier` instance.	

⌋

8.17.2.1.3.4 operator<

[SWS_CORE_08046] Definition of API function ara::core::InstanceSpecifier::operator<

Upstream requirements: RS_Main_00320

⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	`class ara::core::InstanceSpecifier`	
Syntax:	`bool operator< (const InstanceSpecifier &other) const noexcept;`	
Parameters (in):	other	`ara::core::InstanceSpecifier` instance to compare this one with.
Return value:	bool	true in case this `ara::core::InstanceSpecifier` is lexically lower than other, false otherwise.
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	lower than operator to compare with other `ara::core::InstanceSpecifier` for ordering purposes, for instance, when collecting identifiers in maps).	

⌋

8.17.2.1.3.5 operator==

[SWS_CORE_08042] Definition of API function ara::core::InstanceSpecifier::operator==

Upstream requirements: RS_Main_00320

⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	`class ara::core::InstanceSpecifier`	
Syntax:	`bool operator== (const InstanceSpecifier &other) const noexcept;`	
Parameters (in):	other	`ara::core::InstanceSpecifier` instance to compare this one with.
Return value:	bool	true in case both `ara::core::InstanceSpecifier`s are denoting exactly the same model element, false otherwise.
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	eq operator to compare with other `ara::core::InstanceSpecifier` instance.	

⌋

8.17.2.1.3.6 operator==

[SWS_CORE_08043] Definition of API function ara::core::InstanceSpecifier::operator==

Upstream requirements: RS_Main_00320

⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	`class ara::core::InstanceSpecifier`	
Syntax:	`bool operator== (ara::core::StringView other) const noexcept;`	
Parameters (in):	other	string representation to compare this one with.
Return value:	bool	true in case this `ara::core::InstanceSpecifier` is denoting exactly the same model element as other, false otherwise.
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	eq operator to compare with other `ara::core::InstanceSpecifier` instance.	

⌋

8.17.2.1.4 Named Constructors

8.17.2.1.4.1 Create

[SWS_CORE_08032] Definition of API function ara::core::InstanceSpecifier::Create

Upstream requirements: RS_Main_00150, RS_AP_00137, RS_AP_00136

⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	`class ara::core::InstanceSpecifier`	
Syntax:	`static ara::core::Result< InstanceSpecifier > Create (ara::core::StringView qualifiedShortName) noexcept;`	
Parameters (in):	qualifiedShortName	string representation of a valid
Return value:	ara::core::Result< InstanceSpecifier >	a Result, containing either a syntactically valid `ara::core::InstanceSpecifier`, or an ErrorCode
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	CoreErrc::kInvalidMetaModelShortname	rollback_semantics
		if any of the path elements of `qualifiedShortName` is missing or contains invalid characters
	CoreErrc::kInvalidMetaModelPath	rollback_semantics
		if the `qualifiedShortName` is not a valid path to a model element
Description:	Create a new instance of this class `ara::core::InstanceSpecifier`, according to the syntax rules given by [SWS_CORE_10200] and [SWS_CORE_10203].	

⌋

8.18 Header: ara/core/memory_resource.h

8.18.1 Non-Member Functions

8.18.1.1 Other

8.18.1.1.1 GetDefaultResource

[SWS_CORE_06565] Definition of API function ara::core::GetDefaultResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`namespace ara::core`
Syntax:	`MemoryResource * GetDefaultResource () noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[mem.res]` in [16]

⌋

8.18.1.1.2 NewDeleteResource

[SWS_CORE_06562] Definition of API function ara::core::NewDeleteResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`namespace ara::core`
Syntax:	`MemoryResource * NewDeleteResource () noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[mem.res]` in [16]

⌋

8.18.1.1.3 NullMemoryResource

[SWS_CORE_06563] Definition of API function ara::core::NullMemoryResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`namespace ara::core`
Syntax:	`MemoryResource * NullMemoryResource () noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[mem.res]` in [16]

⌋

8.18.1.1.4 SetDefaultResource

[SWS_CORE_06564] Definition of API function ara::core::SetDefaultResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`namespace ara::core`
Syntax:	`MemoryResource * SetDefaultResource (MemoryResource *r) noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[mem.res]` in [16]

⌋

8.18.1.1.5 operator==

[SWS_CORE_06561] Definition of API function ara::core::operator==

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`namespace ara::core`
Syntax:	`bool operator== (const MemoryResource &a, const MemoryResource &b) noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[mem.res]` in [16]

⌋

8.18.1.1.6 operator==

[SWS_CORE_06560] Definition of API function ara::core::operator==

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`namespace ara::core`

Syntax:	`template <class T1, class T2>`
	`bool operator== (const PolymorphicAllocator< T1 > &a, const PolymorphicAllocator< T2 > &b) noexcept;`

Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[mem.res]` in [16]

⌋

8.18.2 Class: MemoryResource

[SWS_CORE_06500] Definition of API class ara::core::MemoryResource

Upstream requirements: RS_AP_00130

⌈

Kind:	class
Header file:	#include "ara/core/memory_resource.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	`namespace ara::core`
Symbol:	MemoryResource
Syntax:	`class MemoryResource {...};`
Description:	Implements `std::pmr::memory_resource` (see `[mem.res.class]` in [16]). Unless explicitly overriden in the member documentation, members always adhere in behavior to the ISO specification in [16].

⌋

8.18.2.1 Public Member Functions

8.18.2.1.1 Special Member Functions

8.18.2.1.1.1 Move Constructor

[SWS_CORE_06574] Definition of API function ara::core::MemoryResource::MemoryResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MemoryResource`
Syntax:	`MemoryResource (MemoryResource &&other) noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	(AUTOSAR defined) move constructor - not present in [16]

⌋

8.18.2.1.1.2 Default Constructor

[SWS_CORE_06501] Definition of API function ara::core::MemoryResource::MemoryResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MemoryResource`
Syntax:	`MemoryResource () noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[mem.res.class]` in [16] except for the following deviations: 1. Function is noexcept

⌋

8.18.2.1.1.3 Copy Constructor

[SWS_CORE_06502] Definition of API function ara::core::MemoryResource::MemoryResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MemoryResource`
Syntax:	`MemoryResource (const MemoryResource &other) noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[mem.res.class]` in [16] except for the following deviations: 1. Function is noexcept

⌋

8.18.2.1.1.4 Move Assignment Operator

[SWS_CORE_06575] Definition of API function ara::core::MemoryResource::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MemoryResource`
Syntax:	`MemoryResource & operator= (MemoryResource &&other) noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	(AUTOSAR defined) move assignment operator - not present in [16]

⌋

8.18.2.1.1.5 Copy Assignment Operator

[SWS_CORE_06507] Definition of API function ara::core::MemoryResource::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MemoryResource`
Syntax:	`MemoryResource & operator= (const MemoryResource &other) noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding function in `[mem.res.class]` in [16] except for the following deviations: 1. Function is noexcept

⌋

8.18.2.1.1.6 Destructor

[SWS_CORE_06506] Definition of API function ara::core::MemoryResource::~MemoryResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MemoryResource`
Syntax:	`virtual ~MemoryResource () noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding function in `[mem.res.class]` in [16] except for the following deviations: 1. Function is noexcept

⌋

8.18.2.1.2 Member Functions

8.18.2.1.2.1 allocate

[SWS_CORE_06503] Definition of API function ara::core::MemoryResource::allocate

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MemoryResource`
Syntax:	`ARA_COMPILER_DEFINED_NODISCARD void * allocate (std::size_t bytes, std::size_t alignment=alignof(std::max_align_t)) noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding function in `[mem.res.class]` in [16] except for the following deviations: 1. Function is `noexcept` 2. If unsuccessful: `nullptr` is returned, instead of raising any `Exception` 3. `ARA_COMPILER_DEFINED_NODISCARD` - see [SWS_CORE_11952]

⌋

8.18.2.1.2.2 deallocate

[SWS_CORE_06504] Definition of API function ara::core::MemoryResource::deallocate

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MemoryResource`
Syntax:	`void deallocate (void *p, std::size_t bytes, std::size_t alignment=max_align) noexcept;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding function in `[mem.res.class]` in [16] except for the following deviations: 1. Function is `noexcept` 2. If unsuccessful: errors will be silently ignored

⌋

8.18.2.1.2.3 is_equal

[SWS_CORE_06505] Definition of API function ara::core::MemoryResource::is_equal

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MemoryResource`
Syntax:	`bool is_equal (const MemoryResource &other) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[mem.res.class]` in [16]

⌋

8.18.2.2 Private Member Functions

8.18.2.2.1 Member Functions

8.18.2.2.1.1 do_allocate

[SWS_CORE_06566] Definition of API function ara::core::MemoryResource::do_allocate

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MemoryResource`
Syntax:	`virtual void * do_allocate (std::size_t bytes, std::size_t alignment) noexcept=0;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding function in `[mem.res.class]` in [16] except for the following deviations: 1. Function is `noexcept` 2. If unsuccessful: `nullptr` is returned, instead of raising any `Exception`
Visibility:	private

⌋

8.18.2.2.1.2 do_deallocate

[SWS_CORE_06567] Definition of API function ara::core::MemoryResource::do_deallocate

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MemoryResource`
Syntax:	`virtual void do_deallocate (void *p, std::size_t bytes, std::size_t alignment) noexcept=0;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	not thread-safe

▽

△

Description:	As per corresponding function in `[mem.res.class]` in [16] except for the following deviations: 1. Function is `noexcept` 2. If unsuccessful: errors will be silently ignored
Visibility:	private

⌋

8.18.2.2.1.3 do_is_equal

[SWS_CORE_06568] Definition of API function ara::core::MemoryResource::do_is_equal

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MemoryResource`
Syntax:	`virtual bool do_is_equal (const MemoryResource &other) const noexcept=0;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[mem.res.class]` in [16]
Visibility:	private

⌋

8.18.3 Class: MonotonicBufferResource

[SWS_CORE_06520] Definition of API class ara::core::MonotonicBufferResource

Upstream requirements: RS_AP_00130

⌈

Kind:	class
Header file:	#include "ara/core/memory_resource.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	`namespace ara::core`
Symbol:	MonotonicBufferResource
Base class:	MemoryResource

▽

△

Syntax:	`class MonotonicBufferResource : public MemoryResource {...};`
Description:	Implements `std::pmr::monotonic_buffer_resource` (see `[mem.res.monotonic.buffer]` in [16]). Unless explicitly overriden in the member documentation, members always adhere in behavior to the ISO specification in [16].

⌋

8.18.3.1 Public Member Functions

8.18.3.1.1 Special Member Functions

8.18.3.1.1.1 Copy Constructor

[SWS_CORE_06527] Definition of API function ara::core::MonotonicBufferResource::MonotonicBufferResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MonotonicBufferResource`
Syntax:	`MonotonicBufferResource (const MonotonicBufferResource &)=delete;`
Description:	As per corresponding function in `[mem.res.monotonic.buffer]` in [16]

⌋

8.18.3.1.1.2 Move Constructor

[SWS_CORE_06576] Definition of API function ara::core::MonotonicBufferResource::MonotonicBufferResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MonotonicBufferResource`
Syntax:	`MonotonicBufferResource (MonotonicBufferResource &&other) noexcept=default;`
Exception Safety:	exception safe

▽

△

Thread Safety:	implementation defined
Description:	(AUTOSAR defined) move constructor - not present in [16]

⌋

8.18.3.1.1.3 Default Constructor

[SWS_CORE_06524] Definition of API function ara::core::MonotonicBufferResource::MonotonicBufferResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MonotonicBufferResource`
Syntax:	`MonotonicBufferResource () noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[mem.res.monotonic.buffer]` in [16] except for the following deviations: 1. Function is noexcept

⌋

8.18.3.1.1.4 Copy Assignment Operator

[SWS_CORE_06529] Definition of API function ara::core::MonotonicBufferResource::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MonotonicBufferResource`
Syntax:	`MonotonicBufferResource & operator= (const MonotonicBufferResource &)=delete;`
Description:	As per corresponding function in `[mem.res.monotonic.buffer]` in [16]

⌋

8.18.3.1.1.5 Move Assignment Operator

[SWS_CORE_06577] Definition of API function ara::core::MonotonicBufferResource::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MonotonicBufferResource`
Syntax:	`MonotonicBufferResource & operator= (MonotonicBufferResource &&other) noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	(AUTOSAR defined) move assignment operator - not present in [16]

⌋

8.18.3.1.1.6 Destructor

[SWS_CORE_06528] Definition of API function ara::core::MonotonicBufferResource::~MonotonicBufferResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MonotonicBufferResource`
Syntax:	`virtual ~MonotonicBufferResource () noexcept override;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding function in `[mem.res.monotonic.buffer]` in [16] except for the following deviations: 1. Function is noexcept

⌋

8.18.3.1.2 Constructors

8.18.3.1.2.1 MonotonicBufferResource

[SWS_CORE_06525] Definition of API function ara::core::MonotonicBufferResource::MonotonicBufferResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MonotonicBufferResource`
Syntax:	`explicit MonotonicBufferResource (std::size_t initial_size) noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[mem.res.monotonic.buffer]` in [16] except for the following deviations: 1. Function is noexcept

⌋

8.18.3.1.2.2 MonotonicBufferResource

[SWS_CORE_06523] Definition of API function ara::core::MonotonicBufferResource::MonotonicBufferResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MonotonicBufferResource`
Syntax:	`MonotonicBufferResource (void *buffer, std::size_t buffer_size, MemoryResource *upstream) noexcept;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[mem.res.monotonic.buffer]` in [16]

⌋

8.18.3.1.2.3 MonotonicBufferResource

[SWS_CORE_06521] Definition of API function ara::core::MonotonicBufferResource::MonotonicBufferResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MonotonicBufferResource`
Syntax:	`explicit MonotonicBufferResource (MemoryResource *upstream) noexcept;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[mem.res.monotonic.buffer]` in [16]

⌋

8.18.3.1.2.4 MonotonicBufferResource

[SWS_CORE_06522] Definition of API function ara::core::MonotonicBufferResource::MonotonicBufferResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MonotonicBufferResource`
Syntax:	`MonotonicBufferResource (std::size_t initial_size, MemoryResource *upstream) noexcept;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[mem.res.monotonic.buffer]` in [16]

⌋

8.18.3.1.2.5 MonotonicBufferResource

[SWS_CORE_06526] Definition of API function ara::core::MonotonicBufferResource::MonotonicBufferResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MonotonicBufferResource`
Syntax:	`MonotonicBufferResource (void *buffer, std::size_t buffer_size) noexcept;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[mem.res.monotonic.buffer]` in [16] except for the following deviations: 1. Function is noexcept

⌋

8.18.3.1.3 Member Functions

8.18.3.1.3.1 release

[SWS_CORE_06530] Definition of API function ara::core::MonotonicBufferResource::release

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MonotonicBufferResource`
Syntax:	`void release () noexcept;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding function in `[mem.res.monotonic.buffer]` in [16]

⌋

8.18.3.1.3.2 upstream_resource

[SWS_CORE_06531] Definition of API function ara::core::MonotonicBufferResource::upstream_resource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MonotonicBufferResource`
Syntax:	`MemoryResource * upstream_resource () const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[mem.res.monotonic.buffer]` in [16]

⌋

8.18.3.2 Protected Member Functions

8.18.3.2.1 Member Functions

8.18.3.2.1.1 do_allocate

[SWS_CORE_06569] Definition of API function ara::core::MonotonicBufferResource::do_allocate

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MonotonicBufferResource`
Syntax:	`void * do_allocate (std::size_t bytes, std::size_t alignment) noexcept override;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding function in `[mem.res.monotonic.buffer]` in [16] except for the following deviations: 1. Function is `noexcept` 2. If unsuccessful: `nullptr` is returned, instead of raising any `Exception`
Visibility:	protected

⌋

8.18.3.2.1.2 do_deallocate

[SWS_CORE_06570] Definition of API function ara::core::MonotonicBufferResource::do_deallocate

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MonotonicBufferResource`
Syntax:	`void do_deallocate (void *p, std::size_t bytes, std::size_t alignment) noexcept override;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding function in `[mem.res.monotonic.buffer]` in [16] except for the following deviations: 1. Function is `noexcept` 2. If unsuccessful: errors will be silently ignored
Visibility:	protected

⌋

8.18.3.2.1.3 do_is_equal

[SWS_CORE_06571] Definition of API function ara::core::MonotonicBufferResource::do_is_equal

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MonotonicBufferResource`
Syntax:	`bool do_is_equal (const MemoryResource &other) const noexcept override;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[mem.res.monotonic.buffer]` in [16]
Visibility:	protected

⌋

8.18.4 Class: PolymorphicAllocator

[SWS_CORE_06540] Definition of API class ara::core::PolymorphicAllocator

Upstream requirements: RS_AP_00130

⌈

Kind:	class	
Header file:	#include "ara/core/memory_resource.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace ara::core`	
Symbol:	PolymorphicAllocator	

Syntax:	`template <class Tp = ara::core::Byte>`
	`class PolymorphicAllocator {...};`	

Template param:	Tp = ara::core::Byte	--
Description:	Implements `std::pmr::polymorphic_allocator` (see `[mem.poly.allocator.class]` in [16]). Unless explicitly overriden in the member documentation, members always adhere in behavior to the ISO specification in [16].	

⌋

8.18.4.1 Public Member Types

8.18.4.1.1 Type Alias: value_type

[SWS_CORE_06572] Definition of API type ara::core::PolymorphicAllocator::value_type

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`
Symbol:	value_type
Syntax:	`using value_type = Tp;`
Description:	As per corresponding function in `[mem.poly.allocator.class]` in [16]

⌋

8.18.4.2 Public Member Functions

8.18.4.2.1 Special Member Functions

8.18.4.2.1.1 Move Constructor

[SWS_CORE_06546] Definition of API function ara::core::PolymorphicAllocator::PolymorphicAllocator

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`
Syntax:	`PolymorphicAllocator (PolymorphicAllocator &&other) noexcept;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	(AUTOSAR defined) move constructor - not present in [16]
See also:	[RS_AP_00145]

⌋

8.18.4.2.1.2 Copy Constructor

[SWS_CORE_06544] Definition of API function ara::core::PolymorphicAllocator::PolymorphicAllocator

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`

Syntax:	`template <class U>`
	`PolymorphicAllocator (const PolymorphicAllocator< U > &other) noexcept;`

Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[mem.poly.allocator.class]` in [16]

⌋

8.18.4.2.1.3 Default Constructor

[SWS_CORE_06541] Definition of API function ara::core::PolymorphicAllocator::PolymorphicAllocator

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`
Syntax:	`PolymorphicAllocator () noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[mem.poly.allocator.class]` in [16]

⌋

8.18.4.2.1.4 Copy Constructor

[SWS_CORE_06543] Definition of API function ara::core::PolymorphicAllocator::PolymorphicAllocator

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`
Syntax:	`PolymorphicAllocator (const PolymorphicAllocator &other) noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[mem.poly.allocator.class]` in [16] except for the following deviations: 1. Function is noexcept

⌋

8.18.4.2.1.5 Move Assignment Operator

[SWS_CORE_00066] Definition of API function ara::core::PolymorphicAllocator::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`
Syntax:	`PolymorphicAllocator & operator= (PolymorphicAllocator &&other) noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	(AUTOSAR defined) move assignment operator - not present in [16]
See also:	[RS_AP_00145]

⌋

8.18.4.2.1.6 Copy Assignment Operator

[SWS_CORE_06545] Definition of API function ara::core::PolymorphicAllocator::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`
Syntax:	`PolymorphicAllocator & operator= (const PolymorphicAllocator &)=delete;`
Description:	As per corresponding function in `[mem.poly.allocator.class]` in [16]

⌋

8.18.4.2.2 Constructors

8.18.4.2.2.1 PolymorphicAllocator

[SWS_CORE_06542] Definition of API function ara::core::PolymorphicAllocator::PolymorphicAllocator

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`
Syntax:	`PolymorphicAllocator (MemoryResource *r) noexcept;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[mem.poly.allocator.class]` in [16] except for the following deviations: 1. Function is noexcept

⌋

8.18.4.2.3 Member Functions

8.18.4.2.3.1 allocate

[SWS_CORE_06547] Definition of API function ara::core::PolymorphicAllocator::allocate

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	`class ara::core::PolymorphicAllocator`	
Syntax:	`ARA_COMPILER_DEFINED_NODISCARD Tp * allocate (std::size_t n) noexcept;`	
Return value:	ARA_COMPILER_DEFINED_NODISCARD Tp *	• If successful: As per corresponding function in `[mem.poly.allocator.class]` in [16] • If unsuccessful: the returned pointer shall be `nullptr`
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	

▽

△

Description:	As per corresponding function in `[mem.poly.allocator.class]` in [16] except for the following deviations: 1. Function is `noexcept` 2. If unsuccessful: `nullptr` is returned, instead of raising any `Exception` 3. `ARA_COMPILER_DEFINED_NODISCARD` - see [SWS_CORE_11952]

⌋

8.18.4.2.3.2 allocate_bytes

[SWS_CORE_06549] Definition of API function ara::core::PolymorphicAllocator::allocate_bytes

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`
Syntax:	`ARA_COMPILER_DEFINED_NODISCARD void * allocate_bytes (std::size_t nbytes, std::size_t alignment=alignof(std::max_align_t)) noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding function in `[mem.poly.allocator.class]` in [16] except for the following deviations: 1. Function is `noexcept` 2. If unsuccessful: the returned pointer shall be `nullptr` 3. `ARA_COMPILER_DEFINED_NODISCARD` - see [SWS_CORE_11952]

⌋

8.18.4.2.3.3 allocate_object

[SWS_CORE_06551] Definition of API function ara::core::PolymorphicAllocator::allocate_object

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`

Syntax:	`template <class T>`
	`ARA_COMPILER_DEFINED_NODISCARD T * allocate_object (std::size_t n=1) noexcept;`

Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding function in `[mem.poly.allocator.class]` in [16] except for the following deviations: 1. Function is `noexcept` 2. If unsuccessful: the returned pointer shall be `nullptr` 3. `ARA_COMPILER_DEFINED_NODISCARD` - see [SWS_CORE_11952]

⌋

8.18.4.2.3.4 construct

[SWS_CORE_06555] Definition of API function ara::core::PolymorphicAllocator::construct

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`

Syntax:	`template <class T, class... Args>`
	`void construct (T *p, Args &&... args) noexcept(std::is_nothrow_constructible< T, Args... >::value);`

Return value:	None
Exception Safety:	conditionally exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding function in `[mem.poly.allocator.class]` in [16] except for the following deviations: 1. Function is conditionally `noexcept`

⌋

8.18.4.2.3.5 deallocate

[SWS_CORE_06548] Definition of API function ara::core::PolymorphicAllocator::deallocate

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`
Syntax:	`void deallocate (Tp *p, std::size_t n) noexcept;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding function in `[mem.poly.allocator.class]` in [16] except for the following deviations: 1. Function is `noexcept` 2. If unsuccessful: errors will be silently ignored

⌋

8.18.4.2.3.6 deallocate_bytes

[SWS_CORE_06550] Definition of API function ara::core::PolymorphicAllocator::deallocate_bytes

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`
Syntax:	`void deallocate_bytes (void *p, std::size_t nbytes, std::size_t alignment=alignof(std::max_align_t)) noexcept;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding function in `[mem.poly.allocator.class]` in [16] except for the following deviations: 1. Function is `noexcept` 2. If unsuccessful: errors will be silently ignored

⌋

8.18.4.2.3.7 deallocate_object

[SWS_CORE_06552] Definition of API function ara::core::PolymorphicAllocator::deallocate_object

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`

Syntax:	`template <class T>`
	`void deallocate_object (T *p, std::size_t n=1) noexcept;`

Return value:	None
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding function in `[mem.poly.allocator.class]` in [16] except for the following deviations: 1. Function is `nullptr` 2. If unsuccessful: errors will be silently ignored

⌋

8.18.4.2.3.8 delete_object

[SWS_CORE_06554] Definition of API function ara::core::PolymorphicAllocator::delete_object

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`

Syntax:	`template <class T>`
	`void delete_object (T *p) noexcept(std::is_nothrow_destructible< T >::value);`

Return value:	None
Exception Safety:	conditionally exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding function in `[mem.poly.allocator.class]` in [16] except for the following deviations: 1. Function is conditionally `noexcept` 2. If unsuccessful: errors will be silently ignored

⌋

8.18.4.2.3.9 destroy

[SWS_CORE_06556] Definition of API function ara::core::PolymorphicAllocator::destroy

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`

Syntax:	`template <class T>`
	`void destroy (T *p) noexcept(std::is_nothrow_destructible< T >::value);`

Return value:	None
Exception Safety:	conditionally exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding function in `[mem.poly.allocator.class]` in [16] except for the following deviations: 1. Function is conditionally `noexcept`

⌋

8.18.4.2.3.10 new_object

[SWS_CORE_06553] Definition of API function ara::core::PolymorphicAllocator::new_object

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`

Syntax:	`template <class T, class... CtorArgs>`
	`ARA_COMPILER_DEFINED_NODISCARD T * new_object (CtorArgs &&... ctor_args) noexcept(std::is_nothrow_constructible< T, CtorArgs... >::value);`

Exception Safety:	conditionally exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding function in `[mem.poly.allocator.class]` in [16] except for the following deviations: 1. Function is conditionally `noexcept` 2. If unsuccessful: `nullptr` is returned, instead of raising any `Exception` 3. `ARA_COMPILER_DEFINED_NODISCARD` - see [SWS_CORE_11952]

⌋

8.18.4.2.3.11 resource

[SWS_CORE_06557] Definition of API function ara::core::PolymorphicAllocator::resource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`
Syntax:	`MemoryResource * resource () const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding function in `[mem.poly.allocator.class]` in [16]

⌋

8.18.4.2.3.12 select_on_container_copy_construction

[SWS_CORE_06573] Definition of API function ara::core::PolymorphicAllocator::select_on_container_copy_construction

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`
Syntax:	`PolymorphicAllocator select_on_container_copy_construction () const noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding function in `[mem.poly.allocator.class]` in [16] except for the following deviations: 1. Function is noexcept

⌋

8.18.5 Struct: PoolOptions

[SWS_CORE_00026] Definition of API class ara::core::PoolOptions

Upstream requirements: RS_AP_00130

⌈

Kind:	struct
Header file:	#include "ara/core/memory_resource.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	`namespace ara::core`
Symbol:	PoolOptions
Syntax:	`struct PoolOptions {...};`
Description:	Implements `std::pmr::pool_options` (see `[mem.poly.pool_options.class]` in [16]). Unless explicitly overriden in the member documentation, members always adhere in behavior to the ISO specification in [16].

⌋

8.18.5.1 Public Member Variables

8.18.5.1.1 largest_required_pool_block

[SWS_CORE_00028] Definition of API variable ara::core::PoolOptions::largest_required_pool_block

Upstream requirements: RS_AP_00130

⌈

Kind:	variable
Header file:	#include "ara/core/memory_resource.h"
Scope:	`struct ara::core::PoolOptions`
Symbol:	largest_required_pool_block
Type:	`std::size_t`
Syntax:	`std::size_t largest_required_pool_block = 0;`
Description:	As per corresponding member in `[mem.poly.pool_options.class]` in [16]

⌋

8.18.5.1.2 max_blocks_per_chunk

[SWS_CORE_00027] Definition of API variable ara::core::PoolOptions::max_blocks_per_chunk

Upstream requirements: RS_AP_00130

⌈

Kind:	variable
Header file:	#include "ara/core/memory_resource.h"
Scope:	`struct ara::core::PoolOptions`
Symbol:	max_blocks_per_chunk
Type:	`std::size_t`
Syntax:	`std::size_t max_blocks_per_chunk = 0;`
Description:	As per corresponding member in `[mem.poly.pool_options.class]` in [16]

⌋

8.18.6 Class: SynchronizedPoolResource

[SWS_CORE_00029] Definition of API class ara::core::SynchronizedPoolResource

Upstream requirements: RS_AP_00130

⌈

Kind:	class
Header file:	#include "ara/core/memory_resource.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	`namespace ara::core`
Symbol:	SynchronizedPoolResource
Base class:	MemoryResource
Syntax:	`class SynchronizedPoolResource : public MemoryResource {...};`
Description:	Implements `std::pmr::synchronized_pool_resource` (see `[mem.res.pool]` in [16]). Unless explicitly overriden in the member documentation, members always adhere in behavior to the ISO specification in [16].

⌋

8.18.6.1 Public Member Functions

8.18.6.1.1 Special Member Functions

8.18.6.1.1.1 Move Constructor

[SWS_CORE_00043] Definition of API function ara::core::SynchronizedPoolResource::SynchronizedPoolResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::SynchronizedPoolResource`
Syntax:	`SynchronizedPoolResource (SynchronizedPoolResource &&other) noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	(AUTOSAR defined) move constructor - not present in [16]

⌋

8.18.6.1.1.2 Default Constructor

[SWS_CORE_00031] Definition of API function ara::core::SynchronizedPoolResource::SynchronizedPoolResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::SynchronizedPoolResource`
Syntax:	`SynchronizedPoolResource ();`
Exception Safety:	not exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[mem.res.pool]` in [16]

⌋

8.18.6.1.1.3 Copy Constructor

[SWS_CORE_00034] Definition of API function ara::core::SynchronizedPoolResource::SynchronizedPoolResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::SynchronizedPoolResource`
Syntax:	`SynchronizedPoolResource (const SynchronizedPoolResource &)=delete;`
Description:	As per corresponding function in `[mem.res.pool]` in [16]

⌋

8.18.6.1.1.4 Copy Assignment Operator

[SWS_CORE_00036] Definition of API function ara::core::SynchronizedPoolResource::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::SynchronizedPoolResource`
Syntax:	`SynchronizedPoolResource & operator= (const SynchronizedPoolResource &)=delete;`
Description:	As per corresponding function in `[mem.res.pool]` in [16]

⌋

8.18.6.1.1.5 Move Assignment Operator

[SWS_CORE_00044] Definition of API function ara::core::SynchronizedPoolResource::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::SynchronizedPoolResource`
Syntax:	`SynchronizedPoolResource & operator= (SynchronizedPoolResource &&other) noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	(AUTOSAR defined) move assignment operator - not present in [16]

⌋

8.18.6.1.1.6 Destructor

[SWS_CORE_00035] Definition of API function ara::core::SynchronizedPoolResource::~SynchronizedPoolResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::SynchronizedPoolResource`
Syntax:	`virtual ~SynchronizedPoolResource () noexcept override;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[mem.res.pool]` in [16] except for the following deviations: 1. Function is noexcept

⌋

8.18.6.1.2 Constructors

8.18.6.1.2.1 SynchronizedPoolResource

[SWS_CORE_00032] Definition of API function ara::core::SynchronizedPoolResource::SynchronizedPoolResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::SynchronizedPoolResource`
Syntax:	`explicit SynchronizedPoolResource (MemoryResource *upstream);`
Exception Safety:	not exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[mem.res.pool]` in [16]

⌋

8.18.6.1.2.2 SynchronizedPoolResource

[SWS_CORE_00030] Definition of API function ara::core::SynchronizedPoolResource::SynchronizedPoolResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::SynchronizedPoolResource`
Syntax:	`SynchronizedPoolResource (const PoolOptions &opts, MemoryResource *upstream) noexcept;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[mem.res.pool]` in [16] except for the following deviations: 1. Function is noexcept

⌋

8.18.6.1.2.3 SynchronizedPoolResource

[SWS_CORE_00033] Definition of API function ara::core::SynchronizedPoolResource::SynchronizedPoolResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::SynchronizedPoolResource`
Syntax:	`explicit SynchronizedPoolResource (const PoolOptions &opts);`
Exception Safety:	not exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[mem.res.pool]` in [16]

⌋

8.18.6.1.3 Member Functions

8.18.6.1.3.1 options

[SWS_CORE_00039] Definition of API function ara::core::SynchronizedPoolResource::options

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::SynchronizedPoolResource`
Syntax:	`PoolOptions options () const noexcept;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[mem.res.pool]` in [16] except for the following deviations: 1. Function is noexcept

⌋

8.18.6.1.3.2 release

[SWS_CORE_00037] Definition of API function ara::core::SynchronizedPoolResource::release

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::SynchronizedPoolResource`
Syntax:	`void release () noexcept;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[mem.res.pool]` in [16] except for the following deviations: 1. Function is noexcept

⌋

8.18.6.1.3.3 upstream_resource

[SWS_CORE_00038] Definition of API function ara::core::SynchronizedPoolResource::upstream_resource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::SynchronizedPoolResource`
Syntax:	`MemoryResource * upstream_resource () const noexcept;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[mem.res.pool]` in [16] except for the following deviations: 1. Function is noexcept

⌋

8.18.6.2 Protected Member Functions

8.18.6.2.1 Member Functions

8.18.6.2.1.1 do_allocate

[SWS_CORE_00061] Definition of API function ara::core::SynchronizedPoolResource::do_allocate

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::SynchronizedPoolResource`
Syntax:	`void * do_allocate (std::size_t bytes, std::size_t alignment) noexcept override;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[mem.res.pool]` in [16] except for the following deviations: 1. Function is noexcept 2. If unsuccessful: the returned pointer shall be `nullptr`
Visibility:	protected

⌋

8.18.6.2.1.2 do_deallocate

[SWS_CORE_00041] Definition of API function ara::core::SynchronizedPoolResource::do_deallocate

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::SynchronizedPoolResource`
Syntax:	`void do_deallocate (void *p, std::size_t bytes, std::size_t alignment) noexcept override;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	implementation defined

▽

△

Description:	As per corresponding function in `[mem.res.pool]` in [16] except for the following deviations: 1. Function is noexcept 2. If unsuccessful: errors will be silently ignored
Visibility:	protected

⌋

8.18.6.2.1.3 do_is_equal

[SWS_CORE_00042] Definition of API function ara::core::SynchronizedPoolResource::do_is_equal

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::SynchronizedPoolResource`
Syntax:	`bool do_is_equal (const MemoryResource &other) const noexcept override;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[mem.res.pool]` in [16]
Visibility:	protected

⌋

8.18.7 Class: UnsynchronizedPoolResource

[SWS_CORE_00045] Definition of API class ara::core::UnsynchronizedPoolResource

Upstream requirements: RS_AP_00130

⌈

Kind:	class
Header file:	#include "ara/core/memory_resource.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	`namespace ara::core`
Symbol:	UnsynchronizedPoolResource

▽

△

Base class:	MemoryResource
Syntax:	`class UnsynchronizedPoolResource : public MemoryResource {...};`
Description:	Implements `std::pmr::unsynchronized_pool_resource` (see `[mem.res.pool]` in [16]). Unless explicitly overriden in the member documentation, members always adhere in behavior to the ISO specification in [16].

⌋

8.18.7.1 Public Member Functions

8.18.7.1.1 Special Member Functions

8.18.7.1.1.1 Move Constructor

[SWS_CORE_00048] Definition of API function ara::core::UnsynchronizedPoolResource::UnsynchronizedPoolResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::UnsynchronizedPoolResource`
Syntax:	`UnsynchronizedPoolResource (UnsynchronizedPoolResource &&other) noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	(AUTOSAR defined) move constructor - not present in [16]

⌋

8.18.7.1.1.2 Default Constructor

[SWS_CORE_00063] Definition of API function ara::core::UnsynchronizedPoolResource::UnsynchronizedPoolResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::UnsynchronizedPoolResource`
Syntax:	`UnsynchronizedPoolResource ();`
Exception Safety:	not exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[mem.res.pool]` in [16]

⌋

8.18.7.1.1.3 Copy Constructor

[SWS_CORE_00046] Definition of API function ara::core::UnsynchronizedPoolResource::UnsynchronizedPoolResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::UnsynchronizedPoolResource`
Syntax:	`UnsynchronizedPoolResource (const UnsynchronizedPoolResource &)=delete;`
Description:	As per corresponding function in `[mem.res.pool]` in [16]

⌋

8.18.7.1.1.4 Copy Assignment Operator

[SWS_CORE_00047] Definition of API function ara::core::UnsynchronizedPoolResource::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::UnsynchronizedPoolResource`
Syntax:	`UnsynchronizedPoolResource & operator= (const UnsynchronizedPoolResource &)=delete;`
Description:	As per corresponding function in `[mem.res.pool]` in [16]

⌋

8.18.7.1.1.5 Move Assignment Operator

[SWS_CORE_00049] Definition of API function ara::core::UnsynchronizedPoolResource::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::UnsynchronizedPoolResource`
Syntax:	`UnsynchronizedPoolResource & operator= (UnsynchronizedPoolResource &&other) noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	(AUTOSAR defined) move assignment operator - not present in [16]

⌋

8.18.7.1.1.6 Destructor

[SWS_CORE_00062] Definition of API function ara::core::UnsynchronizedPoolResource::~UnsynchronizedPoolResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::UnsynchronizedPoolResource`
Syntax:	`virtual ~UnsynchronizedPoolResource () noexcept override;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[mem.res.pool]` in [16] except for the following deviations: 1. Function is noexcept

⌋

8.18.7.1.2 Constructors

8.18.7.1.2.1 UnsynchronizedPoolResource

[SWS_CORE_00064] Definition of API function ara::core::UnsynchronizedPoolResource::UnsynchronizedPoolResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::UnsynchronizedPoolResource`
Syntax:	`explicit UnsynchronizedPoolResource (MemoryResource *upstream);`
Exception Safety:	not exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[mem.res.pool]` in [16]

⌋

8.18.7.1.2.2 UnsynchronizedPoolResource

[SWS_CORE_00067] Definition of API function ara::core::UnsynchronizedPoolResource::UnsynchronizedPoolResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::UnsynchronizedPoolResource`
Syntax:	`UnsynchronizedPoolResource (const PoolOptions &opts, MemoryResource *upstream) noexcept;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[mem.res.pool]` in [16] except for the following deviations: 1. Function is noexcept

⌋

8.18.7.1.2.3 UnsynchronizedPoolResource

[SWS_CORE_00068] Definition of API function ara::core::UnsynchronizedPoolResource::UnsynchronizedPoolResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::UnsynchronizedPoolResource`
Syntax:	`explicit UnsynchronizedPoolResource (const PoolOptions &opts);`
Exception Safety:	not exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[mem.res.pool]` in [16]

⌋

8.18.7.1.3 Member Functions

8.18.7.1.3.1 options

[SWS_CORE_00057] Definition of API function ara::core::UnsynchronizedPoolResource::options

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::UnsynchronizedPoolResource`
Syntax:	`PoolOptions options () const noexcept;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[mem.res.pool]` in [16] except for the following deviations: 1. Function is noexcept

⌋

8.18.7.1.3.2 release

[SWS_CORE_00065] Definition of API function ara::core::UnsynchronizedPoolResource::release

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::UnsynchronizedPoolResource`
Syntax:	`void release () noexcept;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[mem.res.pool]` in [16] except for the following deviations: 1. Function is noexcept

⌋

8.18.7.1.3.3 upstream_resource

[SWS_CORE_00056] Definition of API function ara::core::UnsynchronizedPoolResource::upstream_resource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::UnsynchronizedPoolResource`
Syntax:	`MemoryResource * upstream_resource () const noexcept;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[mem.res.pool]` in [16] except for the following deviations: 1. Function is noexcept

⌋

8.18.7.2 Protected Member Functions

8.18.7.2.1 Member Functions

8.18.7.2.1.1 do_allocate

[SWS_CORE_00058] Definition of API function ara::core::UnsynchronizedPoolResource::do_allocate

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::UnsynchronizedPoolResource`
Syntax:	`void * do_allocate (size_t bytes, size_t alignment) noexcept override;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[mem.res.pool]` in [16] except for the following deviations: 1. Function is noexcept 2. If unsuccessful: the returned pointer shall be `nullptr`
Visibility:	protected

⌋

8.18.7.2.1.2 do_deallocate

[SWS_CORE_00059] Definition of API function ara::core::UnsynchronizedPoolResource::do_deallocate

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::UnsynchronizedPoolResource`
Syntax:	`void do_deallocate (void *p, size_t bytes, size_t alignment) noexcept override;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[mem.res.pool]` in [16] except for the following deviations: 1. Function is noexcept 2. If unsuccessful: errors will be silently ignored
Visibility:	protected

⌋

8.18.7.2.1.3 do_is_equal

[SWS_CORE_00060] Definition of API function ara::core::UnsynchronizedPoolResource::do_is_equal

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::UnsynchronizedPoolResource`
Syntax:	`bool do_is_equal (const MemoryResource &other) const noexcept override;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	As per corresponding function in `[mem.res.pool]` in [16]
Visibility:	protected

⌋

8.19 Header: ara/core/executor.h

8.19.1 Class: Executor

[SWS_CORE_00008] Definition of API class ara::core::Executor

Upstream requirements: RS_AP_00130

⌈

Kind:	class
Header file:	#include "ara/core/executor.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	`namespace ara::core`
Symbol:	Executor
Syntax:	`class Executor {...};`
Description:	Provides an interface for a executing context, which asynchronously invokes `Callable` ([5] `[func.wrap.func]`) objects in a guaranteed thread-safe context. The `execute(...)` method in [SWS_CORE_00024] returns a `ara::core::Future` object to the caller for further result/error extraction or management of the asynchronous-context; the `execute_oneway(...)` method in [SWS_CORE_00025] imply fire-and-forget semantics and thus no asynchronous-context (`ara::core::Future`) is returned.

⌋

8.19.1.1 Public Member Functions

8.19.1.1.1 Special Member Functions

8.19.1.1.1.1 Copy Constructor

[SWS_CORE_00009] Definition of API function ara::core::Executor::Executor

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/executor.h"	
Scope:	`class ara::core::Executor`	
Syntax:	`Executor (const Executor &other);`	
Parameters (in):	other	Other `ara::core::Executor` to copy
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Copy constructor	

⌋

8.19.1.1.1.2 Move Constructor

[SWS_CORE_00015] Definition of API function ara::core::Executor::Executor

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/executor.h"	
Scope:	`class ara::core::Executor`	
Syntax:	`Executor (Executor &&other) noexcept;`	
Parameters (in):	other	Other `ara::core::Executor` to move
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Move constructor	

⌋

8.19.1.1.1.3 Copy Assignment Operator

[SWS_CORE_00017] Definition of API function ara::core::Executor::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/executor.h"	
Scope:	`class ara::core::Executor`	
Syntax:	`Executor & operator= (const Executor &other);`	
Parameters (in):	other	Other `ara::core::Executor` to copy
Return value:	Executor &	The copied object
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Copy assignment operator	

⌋

8.19.1.1.1.4 Move Assignment Operator

[SWS_CORE_00018] Definition of API function ara::core::Executor::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/executor.h"	
Scope:	`class ara::core::Executor`	
Syntax:	`Executor & operator= (Executor &&other) noexcept;`	
Parameters (in):	other	Other `ara::core::Executor` to move
Return value:	Executor &	The moved object
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Move assignment operator	

⌋

8.19.1.1.1.5 Destructor

[SWS_CORE_00019] Definition of API function ara::core::Executor::~Executor

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/executor.h"
Scope:	`class ara::core::Executor`
Syntax:	`~Executor () noexcept;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Destructor

⌋

8.19.1.1.2 Member Functions

8.19.1.1.2.1 execute

[SWS_CORE_00024] Definition of API function ara::core::Executor::execute

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/executor.h"	
Scope:	`class ara::core::Executor`	

Syntax:	`template <class F, class... P>`
	`ara::core::Future< std::result_of_t< std::decay_t< F >(std::decay_t< P >...)> > execute (F &&f, P &&... p) noexcept;`	

Template param:	F	a `Callable` type
	P	argument list to the `Callable`
Parameters (in):	f	a `Callable` to be executed
	p	parameters to be supplied with the `Callable`
Return value:	ara::core::Future< std::result_of_t< std::decay_t< F >(std::decay_t< P >...)> >	• If successful: an `ara::core::Future` containing a `ara::core::Result::value_type` containing the result of the callable `F` • If unsuccessful: an `ara::core::Future` containing an `ErrorCode`
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Asynchronously invoke `F` with the provided arguments `P...`.	

⌋

8.19.1.1.2.2 oneway_execute

[SWS_CORE_00025] Definition of API function ara::core::Executor::oneway_execute

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/executor.h"	
Scope:	`class ara::core::Executor`	

Syntax:	`template <class F, class... P>`
	`void oneway_execute (F &&f, P &&... p) noexcept;`	

Template param:	F	a `Callable` type
	P	argument list to the `Callable`
Parameters (in):	f	a `Callable` to be executed

▽

△

	p	parameters to be supplied with the Callable
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Asynchronously invoke `F` with the provided arguments `P...`.	

⌋

8.20 Header: ara/core/utility.h

8.20.1 Non-Member Types

8.20.1.1 Type Alias: Byte

[SWS_CORE_04200] Definition of API type ara::core::Byte

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/utility.h"
Scope:	`namespace ara::core`
Symbol:	Byte
Syntax:	`using Byte = <implementation-defined>;`
Description:	A non-integral binary type. The exact setup of this type is implementation-defined; the specifications in Chapter 7 define the expected behavior.

⌋

8.20.2 Global Variables

8.20.2.1 in_place

[SWS_CORE_04013] Definition of API variable ara::core::in_place

Upstream requirements: RS_AP_00130

⌈

Kind:	variable
Header file:	#include "ara/core/utility.h"
Scope:	`namespace ara::core`
Symbol:	in_place
Type:	`in_place_t`
Syntax:	`constexpr in_place_t in_place;`
Description:	The singleton instance of in_place_t.

⌋

8.20.2.2 in_place_index

[SWS_CORE_04033] Definition of API variable ara::core::in_place_index

Upstream requirements: RS_AP_00130

⌈

Kind:	variable	
Header file:	#include "ara/core/utility.h"	
Scope:	`namespace ara::core`	
Symbol:	in_place_index	
Type:	`in_place_index_t< I >`	

Syntax:	`template <std::size_t I>`
	`constexpr in_place_index_t<I> in_place_index {};`	

Template param:	std::size_t I	the index to address
Description:	The singleton instances (one for each I) of in_place_index_t.	

⌋

8.20.2.3 in_place_type

[SWS_CORE_04023] Definition of API variable ara::core::in_place_type

Upstream requirements: RS_AP_00130

⌈

Kind:	variable	
Header file:	#include "ara/core/utility.h"	
Scope:	`namespace ara::core`	
Symbol:	in_place_type	
Type:	`in_place_type_t< T >`	

Syntax:	`template <typename T>`
	`constexpr in_place_type_t<T> in_place_type;`	

Template param:	typename T	the type to address
Description:	The singleton instances (one for each T) of in_place_type_t.	

⌋

8.20.3 Non-Member Functions

8.20.3.1 Other

8.20.3.1.1 data

[SWS_CORE_04112] Definition of API function ara::core::data

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/utility.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T, std::size_t N>`
	`constexpr T * data (T(&array)[N]) noexcept;`	

Template param:	T	the type of array elements
	N	the number of elements in the array
Parameters (in):	array	reference to a raw array
Return value:	T *	a pointer to the first element of the array
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a pointer to the block of memory that contains the elements of a raw array.	

⌋

8.20.3.1.2 data

[SWS_CORE_04113] Definition of API function ara::core::data

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/utility.h"	
Scope:	namespace ara::core	
Syntax:	template <typename E> constexpr const E * data (std::initializer_list< E > il) noexcept;	
Template param:	E	the type of elements in the std::initializer_list
Parameters (in):	il	the std::initializer_list
Return value:	const E *	a pointer to the first element of the std::initializer_list
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a pointer to the block of memory that contains the elements of a std::initializer_list.	

⌋

8.20.3.1.3 data

[SWS_CORE_04110] Definition of API function ara::core::data

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/utility.h"	
Scope:	namespace ara::core	
Syntax:	template <typename Container> constexpr auto data (Container &c) noexcept(noexcept(c.data())) -> decltype(c.data());	
Template param:	Container	a type with a data() method
Parameters (in):	c	an instance of Container
Return value:	decltype(c.data())	a pointer to the first element of the container
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return a pointer to the block of memory that contains the elements of a container.	

⌋

8.20.3.1.4 data

[SWS_CORE_04111] Definition of API function ara::core::data

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/utility.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename Container>`
	`constexpr auto data (const Container &c) noexcept(noexcept(c.data()))`
	`-> decltype(c.data());`	

Template param:	Container	a type with a data() method
Parameters (in):	c	an instance of Container
Return value:	decltype(c.data())	a pointer to the first element of the container
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return a const_pointer to the block of memory that contains the elements of a container.	

⌋

8.20.3.1.5 empty

[SWS_CORE_04132] Definition of API function ara::core::empty

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/utility.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename E>`
	`constexpr bool empty (std::initializer_list< E > il) noexcept;`	

Template param:	E	the type of elements in the std::initializer_list
Parameters (in):	il	the std::initializer_list
Return value:	bool	true if the std::initializer_list is empty, false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return whether the given std::initializer_list is empty.	

⌋

8.20.3.1.6 empty

[SWS_CORE_04131] Definition of API function ara::core::empty
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/utility.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T, std::size_t N>`
	`constexpr bool empty (const T(&array)[N]) noexcept;`	

Template param:	T	the type of array elements
	N	the number of elements in the array
Parameters (in):	array	the raw array
Return value:	bool	false
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return whether the given raw array is empty. As raw arrays cannot have zero elements in C++, this function always returns false.	

⌋

8.20.3.1.7 empty

[SWS_CORE_04130] Definition of API function ara::core::empty
Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/utility.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename Container>`
	`constexpr auto empty (const Container &c)`
	`noexcept(noexcept(c.empty())) -> decltype(c.empty());`	

Template param:	Container	a type with a empty() method
Parameters (in):	c	an instance of Container
Return value:	decltype(c.empty())	true if the container is empty, false otherwise
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return whether the given container is empty.	

⌋

8.20.3.1.8 size

[SWS_CORE_04121] Definition of API function ara::core::size

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/utility.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename T, std::size_t N>`
	`constexpr std::size_t size (const T(&array)[N]) noexcept;`	

Template param:	T	the type of array elements
	N	the number of elements in the array
Parameters (in):	array	reference to a raw array
Return value:	std::size_t	the size of the array, i.e. N
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return the size of a raw array.	

⌋

8.20.3.1.9 size

[SWS_CORE_04120] Definition of API function ara::core::size

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/utility.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename Container>`
	`constexpr auto size (const Container &c) noexcept(noexcept(c.size()))`
	`-> decltype(c.size());`	

Template param:	Container	a type with a size() method
Parameters (in):	c	an instance of Container
Return value:	decltype(c.size())	the size of the container
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return the size of a container.	

⌋

8.20.4 Struct: in_place_index_t

[SWS_CORE_04031] Definition of API class ara::core::in_place_index_t

Upstream requirements: RS_AP_00130

⌈

Kind:	struct	
Header file:	#include "ara/core/utility.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace ara::core`	
Symbol:	in_place_index_t	

Syntax:	`template <std::size_t I>`
	`struct in_place_index_t {...};`	

Template param:	std::size_t I	--
Description:	Denote an index-distinguishing operation to be performed in-place. An instance of this type can be passed to certain constructors of ara::core::Variant to denote the intention that construction of the contained type shall be done in-place, i.e. without any copying taking place.	

⌋

8.20.4.1 Public Member Functions

8.20.4.1.1 Special Member Functions

8.20.4.1.1.1 Default Constructor

[SWS_CORE_04032] Definition of API function ara::core::in_place_index_t::in_place_index_t

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/utility.h"
Scope:	`struct ara::core::in_place_index_t`
Syntax:	`explicit in_place_index_t () noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	Default constructor.

⌋

8.20.5 Struct: in_place_t

[SWS_CORE_04011] Definition of API class ara::core::in_place_t

Upstream requirements: RS_AP_00130

⌈

Kind:	struct
Header file:	#include "ara/core/utility.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	`namespace ara::core`
Symbol:	in_place_t
Syntax:	`struct in_place_t {...};`
Description:	Denote an operation to be performed in-place. An instance of this type can be passed to certain constructors of ara::core::Optional to denote the intention that construction of the contained type shall be done in-place, i.e. without any copying taking place.

⌋

8.20.5.1 Public Member Functions

8.20.5.1.1 Special Member Functions

8.20.5.1.1.1 Default Constructor

[SWS_CORE_04012] Definition of API function ara::core::in_place_t::in_place_t

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/utility.h"
Scope:	`struct ara::core::in_place_t`
Syntax:	`explicit in_place_t () noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	Default constructor.

⌋

8.20.6 Struct: in_place_type_t

[SWS_CORE_04021] Definition of API class ara::core::in_place_type_t
Upstream requirements: RS_AP_00130

⌈

Kind:	struct	
Header file:	#include "ara/core/utility.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	`namespace ara::core`	
Symbol:	in_place_type_t	

Syntax:	`template <typename T>`
	`struct in_place_type_t {...};`	

Template param:	typename T	--
Description:	Denote a type-distinguishing operation to be performed in-place. An instance of this type can be passed to certain constructors of ara::core::Variant to denote the intention that construction of the contained type shall be done in-place, i.e. without any copying taking place.	

⌋

8.20.6.1 Public Member Functions

8.20.6.1.1 Special Member Functions

8.20.6.1.1.1 Default Constructor

[SWS_CORE_04022] Definition of API function ara::core::in_place_type_t::in_place_type_t
Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/utility.h"
Scope:	`struct ara::core::in_place_type_t`
Syntax:	`explicit in_place_type_t () noexcept=default;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	Default constructor.

⌋

8.21 Header: ara/core/initialization.h

8.21.1 Non-Member Functions

8.21.1.1 Other

8.21.1.1.1 Deinitialize

[SWS_CORE_10002] Definition of API function ara::core::Deinitialize

Upstream requirements: RS_Main_00011

⌈

Kind:	function	
Header file:	#include "ara/core/initialization.h"	
Scope:	`namespace ara::core`	
Syntax:	`Result< void > Deinitialize () noexcept;`	
Return value:	Result< void >	a Result with an error code, in case an error occurred
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Errors:	CoreErrc::kNotInitialized	rollback_semantics
		if the function is called for a framework instance that has already been successfully deinitialized, or has never been initialized
Description:	Shutdown of the ARA Framework. After this call, no interaction with the ARA is allowed with the exception of types intended to be used independently of initialization as defined in [SWS_CORE_15002]. As a prerequisite to calling this API it is expected that the use of ARA interfaces is completed (with the given exceptions). It is strongly recommended to make this call in a place where it is guaranteed that the static initialization has completed and destruction of statically initialized data has not yet started.	

⌋

8.21.1.1.2 Initialize

[SWS_CORE_10001] Definition of API function ara::core::Initialize

Status: OBSOLETE

Upstream requirements: RS_Main_00011

⌈

Kind:	function
Header file:	#include "ara/core/initialization.h"
Scope:	`namespace ara::core`
Syntax:	`Result< void > Initialize () noexcept;`

▽

△

Return value:	Result< void >	a Result with an error code, in case an error occurred
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Errors:	CoreErrc::kAlready Initialized	rollback_semantics
		if the function is called for a framework instance that has already been successfully initialized
Description:	(Pre-)Initialization of the ARA Framework. Prior to this call, interaction with the ARA is not allowed with the exception of types intended to be used independently of initialization as defined in [SWS_CORE_15002]. It is strongly recommended to make this call in a place where it is guaranteed that static initialization has completed.	

⌋

8.21.1.1.3 Initialize

[SWS_CORE_10003] Definition of API function ara::core::Initialize

Upstream requirements: RS_Main_00011

⌈

Kind:	function	
Header file:	#include "ara/core/initialization.h"	
Scope:	`namespace ara::core`	
Syntax:	`Result< void > Initialize (int &argc, char **argv) noexcept;`	
Parameters (inout):	argc	number of elements in argv
	argv	the process's command-line arguments, as received in main()
Return value:	Result< void >	a Result with an error code, in case an error occurred
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Errors:	CoreErrc::kAlready Initialized	rollback_semantics
		if the function is called for a framework instance that has already been successfully initialized
Description:	(Pre-)Initialization of the ARA Framework. Prior to this call, interaction with the ARA is not allowed with the exception of types intended to be used independently of initialization as defined in [SWS_CORE_15002]. It is strongly recommended to make this call in a place where it is guaranteed that static initialization has completed. Please note that argc is passed by mutable reference, so both argc and argv may be modified within the function. This is done to allow injection and interpretation of vendor specific ARA configuration information. The injected arguments are removed in accordance with [SWS_CORE_15006].	

⌋

8.22 Header: ara/core/abort.h

8.22.1 Non-Member Types

8.22.1.1 Type Alias: AbortHandler

[SWS_CORE_00050] Definition of API type ara::core::AbortHandler

Upstream requirements: RS_AP_00159

⌈

Kind:	type alias
Header file:	#include "ara/core/abort.h"
Scope:	`namespace ara::core`
Symbol:	AbortHandler
Syntax:	`using AbortHandler = decltype(&AbortHandlerPrototype);`
Description:	The type of a handler for `ara::core::SetAbortHandler`

⌋

8.22.2 Non-Member Functions

8.22.2.1 Other

8.22.2.1.1 Abort

[SWS_CORE_00052] Definition of API function ara::core::Abort

Upstream requirements: RS_AP_00127, RS_AP_00159, RS_AP_00136

⌈

Kind:	function	
Header file:	#include "ara/core/abort.h"	
Scope:	`namespace ara::core`	

Syntax:	`template <typename...  Args>`
	`void Abort (const Args &...  args) noexcept;`	

Template param:	Args...	the types of arguments given to this function
Parameters (in):	args	custom texts to be added in the log message being output
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	thread-safe	

▽

△

Description:	Abort the current operation This function will never return to its caller. The stack is not unwound: destructors of variables with automatic storage duration are not called. Calling this function is ill-formed if any of the arguments is not convertible to `ara::core::StringView`.

⌋

8.22.2.1.2 AbortHandlerPrototype

[SWS_CORE_00053] Definition of API function ara::core::AbortHandlerPrototype

Upstream requirements: RS_AP_00159

⌈

Kind:	function
Header file:	#include "ara/core/abort.h"
Scope:	`namespace ara::core`
Syntax:	`void AbortHandlerPrototype () noexcept;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	A function declaration with the correct prototype for `ara::core::SetAbortHandler` This declaration exists only for providing a function type that includes `noexcept` and that acts as base type for a type alias, which is defined in [SWS_CORE_00050]. This compensates for the fact that the C++ standard (up to and including C++14) prohibits that `noexcept` appears in an alias-declaration. There is no implementation of this function.

⌋

8.22.2.1.3 AddAbortHandler

[SWS_CORE_00054] Definition of API function ara::core::AddAbortHandler

Upstream requirements: RS_AP_00159

⌈

Kind:	function	
Header file:	#include "ara/core/abort.h"	
Scope:	`namespace ara::core`	
Syntax:	`bool AddAbortHandler (AbortHandler handler) noexcept;`	
Parameters (in):	handler	a custom Abort handler
Return value:	bool	true if the given handler was successfully installed; false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Add a custom Abort handler function `false` is returned when either the implementation-defined limit for number of abort handlers would be exceeded, or if `nullptr` is passed to this function Implementations support at least 8 AbortHandlers.	

⌋

8.22.2.1.4 SetAbortHandler

[SWS_CORE_00051] Definition of API function ara::core::SetAbortHandler

Upstream requirements: RS_AP_00159

⌈

Kind:	function	
Header file:	#include "ara/core/abort.h"	
Scope:	`namespace ara::core`	
Syntax:	`AbortHandler SetAbortHandler (AbortHandler handler) noexcept;`	
Parameters (in):	handler	a custom Abort handler (or `nullptr`)
Return value:	AbortHandler	the most recently installed Abort handler (or `nullptr` if none was installed)
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Add a custom Abort handler function and return the most recently added one. By setting `nullptr`, the implementation may restore the default handler instead; this will remove all previously installed handlers. This function can be called from multiple threads simultaneously; these calls are performed in an implementation-defined sequence.	

⌋

9 Service Interfaces

This functional cluster does not define any provided or required service interfaces.

10 Configuration

The configuration model of this functional cluster is defined in [18]. This chapter defines the default values for attributes and semantic constraints for elements specified in [18] that are part of the configuration model of this functional cluster.

10.1 Default Values

This functional cluster does not define any default values for attributes specified in [18].

10.2 Semantic Constraints

This functional cluster does not define any semantic constraints for elements specified in [18].

A Mentioned Manifest Elements

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

Chapter is generated.

Class	**ApApplicationErrorDomain**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This meta-class represents the ability to define a global error domain for an ApApplicationError. **Tags:** atp.recommendedPackage=ApplicationErrorDomains			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
namespace (ordered)	SymbolProps	*	aggr	This aggregation defines the namespace of the ApApplicationErrorDomain
value	PositiveUnlimitedInteger	0..1	attr	This attribute identifies the error category.

Table A.1: ApApplicationErrorDomain

Class	**CompositionSwComponentType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition			
Note	A CompositionSwComponentType aggregates SwComponentPrototypes (that in turn are typed by SwComponentTypes) as well as SwConnectors for primarily connecting SwComponentPrototypes among each others and towards the surface of the CompositionSwComponentType. By this means, a hierarchical structures of software-components can be created. **Tags:** atp.recommendedPackage=SwComponentTypes			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
component	SwComponentPrototype	*	aggr	The instantiated components that are part of this composition. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=component.shortName, component.variationPoint.shortLabel vh.latestBindingTime=postBuild

▽

△

Class	**CompositionSwComponentType**			
connector	SwConnector	*	aggr	SwConnectors have the principal ability to establish a connection among PortPrototypes. They can have many roles in the context of a CompositionSwComponentType. Details are refined by subclasses. The aggregation of SwConnectors is subject to variability with the purpose to support variant data flow. The aggregation is marked as atpSplitable in order to allow the extension of the ECU extract with AssemblySwConnectors between ApplicationSwComponentTypes and ServiceSwComponentTypes during the ECU integration. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=connector.shortName, connector.variationPoint.shortLabel vh.latestBindingTime=postBuild
constantValueMapping	ConstantSpecificationMappingSet	*	ref	Reference to the ConstantSpecificationMapping to be applied for initValues of PPortComSpecs and RPortComSpec. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=constantValueMapping
dataTypeMapping	DataTypeMappingSet	*	ref	Reference to the DataTypeMapping to be applied for the used ApplicationDataTypes in ServiceInterfaces. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=dataTypeMapping
physicalDimensionMapping	PhysicalDimensionMappingSet	0..1	ref	This reference identifies the PhysicalDimensionMappingSet that is applicable in the context of the enclosing CompositionSwComponentType. The PhysicalDimensionMappings contained in the PhysicalDimensionMappingSet shall be taken into account for the assessment of the compatibility of PhysicalDimensions in the context of creation of a PortInterfaceMapping in the scope of the CompositionSwComponentType.

Table A.2: CompositionSwComponentType

Class	***CppImplementationDataType*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::CppImplementationDataType			
Note	This meta-class represents the way to specify a reusable data type definition taken as a the basis for a C++ language binding			
Base	*ARElement*, *ARObject*, *AbstractImplementationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *CppImplementationDataTypeContextTarget*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	CustomCppImplementationDataType, StdCppImplementationDataType			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	***CppImplementationDataType*** (abstract)			
arraySize	PositiveInteger	0..1	attr	This attribute can be used to specify the array size if the enclosing CppImplementationDataType has array semantics. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
headerFile	String	0..1	attr	Configuration of the Header File with the custom class declaration.
namespace (ordered)	SymbolProps	*	aggr	This aggregation allows for the definition an own namespace for the enclosing CppImplementationData Type.
subElement (ordered)	CppImplementation DataTypeElement	*	aggr	This represents the collection of sub-elements of the enclosing CppImplementationDataType
template Argument (ordered)	CppTemplateArgument	*	aggr	This aggregation allows for the specification of properties of template arguments
typeEmitter	NameToken	0..1	attr	This attribute can be taken to control how the respective CppImplementationDataType is contributed to the language binding.
typeReference	CppImplementation DataType	0..1	ref	This reference shall be defined to define a type reference (a.k.a. typedef).

Table A.3: CppImplementationDataType

Class	**DltMessage**			
Package	M2::AUTOSARTemplates::LogAndTraceExtract			
Note	This element defines a DltMessage.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	LogAndTraceMessageCollectionSet.dltMessage			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
dltArgument (ordered)	DltArgument	*	aggr	Ordered collection of DltArguments in the DltMessage.
messageId	PositiveInteger	0..1	attr	This attribute defines the unique Id for the DltMessage.
messageLine Number	PositiveInteger	0..1	attr	This attribute describes the position in the source file in which this log message was called.
messageSource File	String	0..1	attr	This attribute describes the source file in which this log message was called.
messageType Info	String	0..1	attr	This attribute describes the message Type
privacyLevel	PrivacyLevel	0..1	aggr	The Privacy Level helps to identify the Log and Trace content towards the degree of privacy to it.

Table A.4: DltMessage

Class	**Executable**
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::ApplicationStructure
Note	This meta-class represents an executable program. **Tags:** atp.recommendedPackage=Executables
Base	*ARElement*, *ARObject*, *AtpClassifier*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDesignElement*, *UploadablePackageElement*
Aggregated by	ARPackage.element

▽

△

Class	**Executable**			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
buildType	BuildTypeEnum	0..1	attr	This attribute describes the buildType of a module and/or platform implementation.
implementationProps	ExecutableImplementationProps	*	aggr	This aggregation contains the collection of implementation-specific properties necessary to properly build the enclosing Executable.
minimumTimerGranularity	TimeValue	0..1	attr	This attribute describes the minimum timer resolution (TimeValue of one tick) that is required by the Executable.
reportingBehavior	ExecutionStateReportingBehaviorEnum	0..1	attr	this attribute controls the execution state reporting behavior of the enclosing Executable.
rootSwComponentPrototype	RootSwComponentPrototype	0..1	aggr	This represents the root SwCompositionPrototype of the Executable. This aggregation is required (in contrast to a direct reference of a SwComponentType) in order to support the definition of instanceRefs in Executable context.
traceSwitchConfiguration	TraceSwitchConfiguration	*	aggr	Configuration of the MsgId based trace switch **Tags:** atp.Status=draft
version	StrongRevisionLabelString	0..1	attr	Version of the executable.

Table A.5: Executable

Class	***FunctionalClusterInteractsWithFunctionalClusterMapping*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::FunctionalClusterInteractsWithFunctionalClusterMapping			
Note	This meta-class identifies a relation between functional clusters on the adaptive platform such one functional cluster can call APIs of the other functional cluster.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDeploymentElement*, *UploadablePackageElement*			
Subclasses	ArtifactChecksumToCryptoProviderMapping, ComCertificateToCryptoCertificateMapping, ComKeyToCryptoKeySlotMapping, ComSecOcToCryptoKeySlotMapping, FunctionalClusterInteractsWithDiagnosticEventMapping, FunctionalClusterInteractsWithPersistencyDeploymentMapping, FunctionalClusterToSecurityEventDefinitionMapping, NmInteractsWithSmMapping, PersistencyDeploymentElementToCryptoKeySlotMapping, PersistencyDeploymentToCryptoKeySlotMapping, PersistencyDeploymentToDltLogSinkMapping, SmInteractsWithNmMapping, TimeBaseProviderToPersistencyMapping, UcmToTimeBaseResourceMapping			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.6: FunctionalClusterInteractsWithFunctionalClusterMapping

Class	***Identifiable*** (abstract)
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable
Note	Instances of this class can be referred to by their identifier (within the namespace borders). In addition to this, Identifiables are objects which contribute significantly to the overall structure of an AUTOSAR description. In particular, Identifiables might contain Identifiables.
Base	*ARObject*, *MultilanguageReferrable*, *Referrable*

▽

△

Class	***Identifiable*** (abstract)			
Subclasses	ARPackage, *AbstractDoIpLogicAddressProps*, *AbstractEvent*, *AbstractFunctionalClusterDesign*, *AbstractImplementationDataTypeElement*, *AbstractSecurityEventFilter*, *AbstractSecurityIdsmInstanceFilter*, *AbstractServiceInstance*, *AbstractSignalBasedToISignalTriggeringMapping*, AdaptiveSwcInternalBehavior, ApApplicationEndpoint, *ApmcAbstractDefinition*, *ApmcConfigurationElementDef*, *ApmcContainerElementValue*, ApmcContainerValue, ApmcEnumerationLiteralDef, ApplicationEndpoint, ApplicationError, AppliedStandard, ArtifactChecksum, ArtifactLocator, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpFeature*, AutosarOperationArgumentInstance, AutosarVariableInstance, *BuildActionEntity*, BuildActionEnvironment, Chapter, CheckpointTransition, ClassContentConditional, ClientIdDefinition, ClientServerOperation, Code, *CollectableElement*, ComManagementMapping, *CommConnectorPort*, *CommunicationConnector*, *CommunicationController*, Compiler, ConsistencyNeeds, ConsumedEventGroup, CouplingPort, *CouplingPortAbstractShaper*, *CouplingPortStructuralElement*, CryptoCertificate, CryptoKeySlot, CryptoKeySlotDesign, CryptoKeySlotUsageDesign, CryptoProvider, *CryptoServiceMapping*, DataPrototypeGroup, DataPrototypeTransformationPropsIdent, DataTransformation, DdsCpDomain, DdsCpPartition, DdsCpQosProfile, DdsCpTopic, DdsDomainRange, DependencyOnArtifact, *DiagEventDebounceAlgorithm*, DiagnosticAuthTransmitCertificateEvaluation, DiagnosticConnectedIndicator, DiagnosticDataElement, DiagnosticDebounceAlgorithmProps, DiagnosticFunctionInhibitSource, DiagnosticParameterElement, *DiagnosticRoutineSubfunction*, DiagnosticSovdMethodPrimitive, DltApplication, DltArgument, DltMessage, DoIpInterface, DoIpLogicAddress, DoIpLogicalAddress, DoIpNetworkConfigurationDesign, DoIpRoutingActivation, E2EProfileConfiguration, End2EndEventProtectionProps, End2EndMethodProtectionProps, EndToEndProtection, EthernetWakeupSleepOnDatalineConfig, EventHandler, EventMapping, ExclusiveArea, *ExecutableEntity*, *ExecutionTime*, FMAttributeDef, FMFeatureMapAssertion, FMFeatureMapCondition, FMFeatureMapElement, FMFeatureRelation, FMFeatureRestriction, FMFeatureSelection, FieldMapping, FireAndForgetMethodMapping, FlexrayArTpNode, FlexrayTpPduPool, *FrameTriggering*, GeneralParameter, GlobalSupervision, GlobalTimeGateway, *GlobalTimeMaster*, *GlobalTimeSlave*, *HealthChannel*, *HeapUsage*, HwAttributeDef, HwAttributeLiteralDef, HwPin, HwPinGroup, *IEEE1722TpAcfBus*, *IEEE1722TpAcfBusPart*, IPSecRule, IPv6ExtHeaderFilterList, ISignalToIPduMapping, ISignalTriggering, *IdentCaption*, ImpositionTime, InternalTriggeringPoint, Keyword, LifeCycleState, Linker, MacAddressVlanMembership, MacMulticastGroup, MacSecKayParticipant, McDataInstance, MemorySection, MemoryUsage, MethodMapping, ModeDeclaration, ModeDeclarationMapping, ModeSwitchPoint, NetworkEndpoint, *NmCluster*, *NmNode*, *PackageableElement*, ParameterAccess, PduActivationRoutingGroup, PduToFrameMapping, PduTriggering, PerInstanceMemory, *PersistencyDeploymentElement*, *PersistencyInterfaceElement*, *PhmSupervision*, *PhysicalChannel*, PortGroup, *PortInterfaceMapping*, ProcessToMachineMapping, Processor, ProcessorCore, PskIdentityToKeySlotMapping, ResourceConsumption, ResourceGroup, RootSwClusterDesignComponentPrototype, RootSwComponentPrototype, RootSwCompositionPrototype, RptComponent, RptContainer, RptExecutableEntity, RptExecutableEntityEvent, RptExecutionContext, RptProfile, RptServicePoint, RunnableEntityGroup, *SdgAttribute*, SdgClass, SecOcJobMapping, SecOcJobRequirement, SecureCommunicationAuthenticationProps, *SecureCommunicationDeployment*, SecureCommunicationFreshnessProps, SecurityEventContextDataElement, SecurityEventContextProps, *ServiceEventDeployment*, *ServiceFieldDeployment*, ServiceInterfaceElementSecureComConfig, *ServiceMethodDeployment*, *ServiceNeeds*, SignalServiceTranslationEventProps, SignalServiceTranslationProps, SocketAddress, SoftwarePackageStep, SomeipEventGroup, SomeipProvidedEventGroup, SomeipTpChannel, *SpecElementReference*, *StackUsage*, *StateManagementActionItem*, StateManagementActionList, StateManagementStateNotification, *StateManagementStateRequest*, StaticSocketConnection, StructuredReq, SupervisionCheckpoint, SupervisionMode, SupervisionModeCondition, SwGenericAxisParamType, SwServiceArg, SwcServiceDependency, SwitchAsynchronousTrafficShaperGroupEntry, SystemMapping, *TimeBaseResource*, *TimingClock*, TimingClockSyncAccuracy, TimingCondition, *TimingConstraint*, *TimingDescription*, TimingExtensionResource, TimingModeInstance, TlsCryptoCipherSuite, TlsCryptoCipherSuiteProps, TlsJobMapping, Topic1, TpAddress, TraceableTable, TraceableText, *TracedFailure*, TransformationISignalPropsIdent, *TransformationProps*, TransformationTechnology, Trigger, UcmDescription, UcmRetryStrategy, UcmStep, VariableAccess, VariationPointProxy, VehicleRolloutStep, ViewMap, VlanConfig, WaitPoint			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
adminData	AdminData	0..1	aggr	This represents the administrative data for the identifiable object. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=adminData xml.sequenceOffset=-40

▽

△

Class	***Identifiable*** (abstract)			
annotation	Annotation	*	aggr	Possibility to provide additional notes while defining a model element (e.g. the ECU Configuration Parameter Values). These are not intended as documentation but are mere design notes. **Tags:** xml.sequenceOffset=-25
category	CategoryString	0..1	attr	The category is a keyword that specializes the semantics of the Identifiable. It affects the expected existence of attributes and the applicability of constraints. **Tags:** xml.sequenceOffset=-50
desc	MultiLanguageOverviewParagraph	0..1	aggr	This represents a general but brief (one paragraph) description what the object in question is about. It is only one paragraph! Desc is intended to be collected into overview tables. This property helps a human reader to identify the object in question. More elaborate documentation, (in particular how the object is built or used) should go to "introduction". **Tags:** xml.sequenceOffset=-60
introduction	DocumentationBlock	0..1	aggr	This represents more information about how the object in question is built or is used. Therefore it is a DocumentationBlock. **Tags:** xml.sequenceOffset=-30
uuid	String	0..1	attr	The purpose of this attribute is to provide a globally unique identifier for an instance of a meta-class. The values of this attribute should be globally unique strings prefixed by the type of identifier. For example, to include a DCE UUID as defined by The Open Group, the UUID would be preceded by "DCE:". The values of this attribute may be used to support merging of different AUTOSAR models. The form of the UUID (Universally Unique Identifier) is taken from a standard defined by the Open Group (was Open Software Foundation). This standard is widely used, including by Microsoft for COM (GUIDs) and by many companies for DCE, which is based on CORBA. The method for generating these 128-bit IDs is published in the standard and the effectiveness and uniqueness of the IDs is not in practice disputed. If the id namespace is omitted, DCE is assumed. An example is "DCE:2fac1234-31f8-11b4-a222-08002b34c003". The uuid attribute has no semantic meaning for an AUTOSAR model and there is no requirement for AUTOSAR tools to manage the timestamp. **Tags:** xml.attribute=true

Table A.7: Identifiable

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

▽

△

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

Table A.8: PortPrototype

Class	***Referrable*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (while adhering to namespace borders).			
Base	*ARObject*			
Subclasses	*AtpDefinition*, BswDistinguishedPartition, *BswModuleCallPoint*, BswModuleClientServerEntry, BswVariableAccess, CouplingPortTrafficClassAssignment, *CppImplementationDataTypeContextTarget*, *DiagnosticEnvModeElement*, EthernetPriorityRegeneration, ExclusiveAreaNestingOrder, *HwDescriptionEntity*, *ImplementationProps*, ModeTransition, *MultilanguageReferrable*, NmNetworkHandle, PncMappingIdent, *SingleLanguageReferrable*, SoConIPduIdentifier, SocketConnectionBundle, SomeipRequiredEventGroup, TimeSyncServerConfiguration, TpConnectionIdent			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
shortName	Identifier	1	attr	This specifies an identifying shortName for the object. It needs to be unique within its context and is intended for humans but even more for technical reference. **Stereotypes:** atpIdentityContributor **Tags:** xml.enforceMinMultiplicity=true xml.sequenceOffset=-100
shortNameFragment	ShortNameFragment	*	aggr	This specifies how the Referrable.shortName is composed of several shortNameFragments. **Tags:** xml.sequenceOffset=-90

Table A.9: Referrable

Class	**RootSwComponentPrototype**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::ApplicationStructure			
Note	The RootSwCompositionPrototype represents the top-level-composition of software components within an Executable. The contained SwComponentPrototypes are fully specified by their SwComponentTypes (including Port Prototypes, PortInterfaces, VariableDataPrototypes, etc.).			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, Executable.rootSwComponentPrototype			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
applicationType	SwComponentType	0..1	tref	This SwComponentType acts as the Type of the RootSw ComponentPrototype. **Stereotypes:** isOfType

Table A.10: RootSwComponentPrototype

Class	**StdCppImplementationDataType**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::CppImplementationDataType			
Note	This meta-class represents the way to specify a data type definition that is taken as the basis for a C++ language binding to a C++ Standard Library feature. **Tags:** atp.recommendedPackage=CppImplementationDataTypes			
Base	*ARElement*, *ARObject*, *AbstractImplementationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *CppImplementationDataType*, *CppImplementationData TypeContextTarget*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.11: StdCppImplementationDataType

Class	**SwComponentPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition			
Note	Role of a software component within a composition.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, CompositionSwComponentType.component			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
type	SwComponentType	0..1	tref	Type of the instance. **Stereotypes:** isOfType

Table A.12: SwComponentPrototype

B Demands and constraints on Base Software (normative)

This functional cluster defines no demands or constraints for the Base Software on which the AUTOSAR Adaptive Platform is running on (usually a POSIX-compatible operating system).

C Platform Extension Interfaces (normative)

This functional cluster does not specify any Platform Extension Interfaces.

D Not implemented requirements

This functional cluster implements all functional requirements specified in the corresponding requirement specifications.

E Change History

Please note that the lists in this chapter also include specification items that have been removed from the specification in a later version. These specification items do not appear as hyperlinks in the document.

E.1 Change History of this document according to AUTOSAR Release R19-11

E.1.1 Added Specification Items in R19-11

Number	Heading
[SWS_CORE_00003]	Handling of Violations
[SWS_CORE_00004]	Handling of Corruptions
[SWS_CORE_00005]	Handling of failed default allocations
[SWS_CORE_00014]	The Core error domain
[SWS_CORE_00050]	
[SWS_CORE_00051]	
[SWS_CORE_00052]	
[SWS_CORE_00131]	
[SWS_CORE_00132]	
[SWS_CORE_00133]	
[SWS_CORE_00134]	
[SWS_CORE_00135]	
[SWS_CORE_00136]	
[SWS_CORE_00137]	
[SWS_CORE_00138]	
[SWS_CORE_00151]	
[SWS_CORE_00152]	
[SWS_CORE_00153]	
[SWS_CORE_00154]	
[SWS_CORE_00322]	
[SWS_CORE_00323]	
[SWS_CORE_00325]	
[SWS_CORE_00326]	
[SWS_CORE_00327]	
[SWS_CORE_00328]	
[SWS_CORE_00329]	
[SWS_CORE_00330]	

▽

△

Number	Heading
[SWS_CORE_00331]	
[SWS_CORE_00332]	
[SWS_CORE_00333]	
[SWS_CORE_00334]	
[SWS_CORE_00335]	
[SWS_CORE_00336]	
[SWS_CORE_00341]	
[SWS_CORE_00342]	
[SWS_CORE_00343]	
[SWS_CORE_00344]	
[SWS_CORE_00345]	
[SWS_CORE_00346]	
[SWS_CORE_00349]	
[SWS_CORE_00350]	
[SWS_CORE_00351]	
[SWS_CORE_00352]	
[SWS_CORE_00353]	
[SWS_CORE_00354]	
[SWS_CORE_00412]	
[SWS_CORE_00441]	
[SWS_CORE_00442]	
[SWS_CORE_00443]	
[SWS_CORE_00444]	
[SWS_CORE_00480]	
[SWS_CORE_00490]	
[SWS_CORE_00512]	
[SWS_CORE_00513]	
[SWS_CORE_00514]	
[SWS_CORE_00515]	
[SWS_CORE_00516]	
[SWS_CORE_00518]	
[SWS_CORE_00519]	
[SWS_CORE_00571]	
[SWS_CORE_00572]	
[SWS_CORE_00611]	
[SWS_CORE_00612]	
[SWS_CORE_00613]	
[SWS_CORE_00721]	
[SWS_CORE_00722]	

▽

△

Number	Heading
[SWS_CORE_00723]	
[SWS_CORE_00724]	
[SWS_CORE_00725]	
[SWS_CORE_00726]	
[SWS_CORE_00727]	
[SWS_CORE_00731]	
[SWS_CORE_00732]	
[SWS_CORE_00733]	
[SWS_CORE_00734]	
[SWS_CORE_00735]	
[SWS_CORE_00736]	
[SWS_CORE_00741]	
[SWS_CORE_00742]	
[SWS_CORE_00743]	
[SWS_CORE_00744]	
[SWS_CORE_00745]	
[SWS_CORE_00751]	
[SWS_CORE_00752]	
[SWS_CORE_00753]	
[SWS_CORE_00754]	
[SWS_CORE_00755]	
[SWS_CORE_00756]	
[SWS_CORE_00757]	
[SWS_CORE_00758]	
[SWS_CORE_00759]	
[SWS_CORE_00761]	
[SWS_CORE_00762]	
[SWS_CORE_00763]	
[SWS_CORE_00765]	
[SWS_CORE_00766]	
[SWS_CORE_00767]	
[SWS_CORE_00768]	
[SWS_CORE_00769]	
[SWS_CORE_00780]	
[SWS_CORE_00781]	
[SWS_CORE_00782]	
[SWS_CORE_00783]	
[SWS_CORE_00784]	
[SWS_CORE_00785]	

▽

△

Number	Heading
[SWS_CORE_00786]	
[SWS_CORE_00787]	
[SWS_CORE_00788]	
[SWS_CORE_00789]	
[SWS_CORE_00796]	
[SWS_CORE_00821]	
[SWS_CORE_00823]	
[SWS_CORE_00824]	
[SWS_CORE_00825]	
[SWS_CORE_00826]	
[SWS_CORE_00827]	
[SWS_CORE_00831]	
[SWS_CORE_00834]	
[SWS_CORE_00835]	
[SWS_CORE_00836]	
[SWS_CORE_00841]	
[SWS_CORE_00842]	
[SWS_CORE_00843]	
[SWS_CORE_00844]	
[SWS_CORE_00845]	
[SWS_CORE_00851]	
[SWS_CORE_00852]	
[SWS_CORE_00853]	
[SWS_CORE_00855]	
[SWS_CORE_00857]	
[SWS_CORE_00858]	
[SWS_CORE_00861]	
[SWS_CORE_00863]	
[SWS_CORE_00865]	
[SWS_CORE_00866]	
[SWS_CORE_00867]	
[SWS_CORE_01941]	
[SWS_CORE_01942]	
[SWS_CORE_01943]	
[SWS_CORE_01944]	
[SWS_CORE_01945]	
[SWS_CORE_01946]	
[SWS_CORE_01947]	
[SWS_CORE_01948]	

▽

△

Number	Heading
[SWS_CORE_01949]	
[SWS_CORE_01950]	
[SWS_CORE_01951]	
[SWS_CORE_01952]	
[SWS_CORE_01961]	
[SWS_CORE_01962]	
[SWS_CORE_01963]	
[SWS_CORE_01964]	
[SWS_CORE_01965]	
[SWS_CORE_01966]	
[SWS_CORE_01967]	
[SWS_CORE_01968]	
[SWS_CORE_01969]	
[SWS_CORE_01970]	
[SWS_CORE_01971]	
[SWS_CORE_01972]	
[SWS_CORE_01973]	
[SWS_CORE_01974]	
[SWS_CORE_01975]	
[SWS_CORE_01976]	
[SWS_CORE_01977]	
[SWS_CORE_01978]	
[SWS_CORE_01979]	
[SWS_CORE_01990]	
[SWS_CORE_01991]	
[SWS_CORE_01992]	
[SWS_CORE_01993]	
[SWS_CORE_01994]	
[SWS_CORE_03000]	BasicString type
[SWS_CORE_04012]	
[SWS_CORE_04022]	
[SWS_CORE_04032]	
[SWS_CORE_04110]	
[SWS_CORE_04111]	
[SWS_CORE_04112]	
[SWS_CORE_04113]	
[SWS_CORE_04120]	
[SWS_CORE_04121]	
[SWS_CORE_04130]	

▽

△

Number	Heading
[SWS_CORE_04131]	
[SWS_CORE_04132]	
[SWS_CORE_04200]	
[SWS_CORE_05200]	
[SWS_CORE_05211]	
[SWS_CORE_05212]	
[SWS_CORE_05221]	
[SWS_CORE_05231]	
[SWS_CORE_05232]	
[SWS_CORE_05241]	
[SWS_CORE_05242]	
[SWS_CORE_05243]	
[SWS_CORE_05244]	
[SWS_CORE_05280]	
[SWS_CORE_05290]	
[SWS_CORE_06221]	
[SWS_CORE_06222]	
[SWS_CORE_06223]	
[SWS_CORE_06225]	
[SWS_CORE_06226]	
[SWS_CORE_06227]	
[SWS_CORE_06228]	
[SWS_CORE_06229]	
[SWS_CORE_06230]	
[SWS_CORE_06231]	
[SWS_CORE_06232]	
[SWS_CORE_06233]	
[SWS_CORE_06234]	
[SWS_CORE_06235]	
[SWS_CORE_06236]	
[SWS_CORE_06340]	
[SWS_CORE_06341]	
[SWS_CORE_06342]	
[SWS_CORE_06343]	
[SWS_CORE_06344]	
[SWS_CORE_06345]	
[SWS_CORE_06349]	
[SWS_CORE_06350]	
[SWS_CORE_06351]	

▽

△

Number	Heading
[SWS_CORE_06352]	
[SWS_CORE_06353]	
[SWS_CORE_06354]	
[SWS_CORE_08021]	
[SWS_CORE_08029]	
[SWS_CORE_08032]	
[SWS_CORE_08041]	
[SWS_CORE_08042]	
[SWS_CORE_08043]	
[SWS_CORE_08044]	
[SWS_CORE_08045]	
[SWS_CORE_08046]	
[SWS_CORE_10001]	
[SWS_CORE_10002]	
[SWS_CORE_10100]	Type property of ara::core::Byte
[SWS_CORE_10101]	Size of type ara::core::Byte
[SWS_CORE_10102]	Value range of type ara::core::Byte
[SWS_CORE_10103]	Creation of ara::core::Byte instances
[SWS_CORE_10104]	Default-constructed ara::core::Byte instances
[SWS_CORE_10105]	Destructor of type ara::core::Byte
[SWS_CORE_10106]	Implicit conversion from other types
[SWS_CORE_10107]	Implicit conversion to other types
[SWS_CORE_10108]	Conversion to unsigned char
[SWS_CORE_10109]	Equality comparison for byte ara::core::Byte
[SWS_CORE_10110]	Non-equality comparison for byte ara::core::Byte
[SWS_CORE_10200]	Valid InstanceSpecifier representations
[SWS_CORE_10201]	Validation of meta-model paths
[SWS_CORE_10202]	Construction of InstanceSpecifier objects

Table E.1: Added Specification Items in R19-11

E.1.2 Changed Specification Items in R19-11

Number	Heading
[SWS_CORE_00002]	Handling of Errors
[SWS_CORE_00040]	Errors originating from C++ standard classes
[SWS_CORE_03001]	`String` type

▽

△

Number	Heading
[SWS_CORE_03296]	swap overload for BasicString
[SWS_CORE_03301]	Implicit conversion to StringView
[SWS_CORE_03302]	Constructor from StringView
[SWS_CORE_03303]	Constructor from implicit StringView
[SWS_CORE_03304]	operator= from StringView
[SWS_CORE_03305]	Assignment from StringView
[SWS_CORE_03306]	Assignment from implicit StringView
[SWS_CORE_03307]	operator+ from StringView
[SWS_CORE_03308]	Concatenation of StringView
[SWS_CORE_03309]	Concatenation of implicit StringView
[SWS_CORE_03310]	Insertion of StringView
[SWS_CORE_03311]	Insertion of implicit StringView
[SWS_CORE_03312]	Replacement with StringView
[SWS_CORE_03313]	Replacement with implicit StringView
[SWS_CORE_03314]	Replacement of iterator range with StringView
[SWS_CORE_03315]	Forward-find a StringView
[SWS_CORE_03316]	Reverse-find a StringView
[SWS_CORE_03317]	Forward-find of character set within a StringView
[SWS_CORE_03318]	Reverse-find of character set within a StringView
[SWS_CORE_03319]	Forward-find of character set not within a StringView
[SWS_CORE_03320]	Reverse-find of character set not within a StringView
[SWS_CORE_03321]	Comparison with a StringView
[SWS_CORE_03322]	Comparison of subsequence with a StringView
[SWS_CORE_03323]	Comparison of subsequence with a subsequence of a StringView

Table E.2: Changed Specification Items in R19-11

E.1.3 Deleted Specification Items in R19-11

Number	Heading
[SWS_CORE_00001]	Handling of Fatal Errors
[SWS_CORE_00012]	The POSIX error domain

Table E.3: Deleted Specification Items in R19-11

E.2 Change History of this document according to AUTOSAR Release R20-11

E.2.1 Added Specification Items in R20-11

Number	Heading
[SWS_CORE_00011]	AUTOSAR error domain range
[SWS_CORE_00016]	Vendor-defined error domain range
[SWS_CORE_00053]	
[SWS_CORE_00337]	
[SWS_CORE_00355]	
[SWS_CORE_00356]	
[SWS_CORE_00614]	
[SWS_CORE_00764]	
[SWS_CORE_00770]	
[SWS_CORE_00771]	
[SWS_CORE_00772]	
[SWS_CORE_00773]	
[SWS_CORE_00864]	
[SWS_CORE_00868]	
[SWS_CORE_00869]	
[SWS_CORE_00870]	
[SWS_CORE_01210]	
[SWS_CORE_01211]	
[SWS_CORE_01212]	
[SWS_CORE_01213]	
[SWS_CORE_01214]	
[SWS_CORE_01215]	
[SWS_CORE_01216]	
[SWS_CORE_01217]	
[SWS_CORE_01218]	
[SWS_CORE_01219]	
[SWS_CORE_01220]	
[SWS_CORE_01241]	
[SWS_CORE_01242]	
[SWS_CORE_01250]	
[SWS_CORE_01251]	
[SWS_CORE_01252]	
[SWS_CORE_01253]	

▽

△

Number	Heading
[SWS_CORE_01254]	
[SWS_CORE_01255]	
[SWS_CORE_01256]	
[SWS_CORE_01257]	
[SWS_CORE_01258]	
[SWS_CORE_01259]	
[SWS_CORE_01260]	
[SWS_CORE_01261]	
[SWS_CORE_01262]	
[SWS_CORE_01263]	
[SWS_CORE_01264]	
[SWS_CORE_01265]	
[SWS_CORE_01266]	
[SWS_CORE_01267]	
[SWS_CORE_01268]	
[SWS_CORE_01269]	
[SWS_CORE_01270]	
[SWS_CORE_01271]	
[SWS_CORE_01272]	
[SWS_CORE_01280]	
[SWS_CORE_01281]	
[SWS_CORE_01282]	
[SWS_CORE_01283]	
[SWS_CORE_01284]	
[SWS_CORE_01285]	
[SWS_CORE_01290]	
[SWS_CORE_01291]	
[SWS_CORE_01292]	
[SWS_CORE_01293]	
[SWS_CORE_01294]	
[SWS_CORE_01295]	
[SWS_CORE_01980]	
[SWS_CORE_01981]	
[SWS_CORE_04023]	
[SWS_CORE_04033]	
[SWS_CORE_06237]	
[SWS_CORE_06355]	
[SWS_CORE_06356]	
[SWS_CORE_06401]	

▽

△

Number	Heading
[SWS_CORE_06411]	
[SWS_CORE_06412]	
[SWS_CORE_06413]	
[SWS_CORE_06414]	
[SWS_CORE_06431]	
[SWS_CORE_06432]	
[SWS_CORE_08022]	
[SWS_CORE_08023]	
[SWS_CORE_08024]	
[SWS_CORE_08025]	
[SWS_CORE_08081]	
[SWS_CORE_08082]	
[SWS_CORE_10300]	ErrorCode type properties
[SWS_CORE_10400]	ErrorDomain type properties
[SWS_CORE_10900]	Error condition enumeration type
[SWS_CORE_10901]	Error condition enumeration naming
[SWS_CORE_10902]	Error condition enumeration contents
[SWS_CORE_10903]	Error condition enumeration numbers
[SWS_CORE_10910]	ErrorDomain exception base type
[SWS_CORE_10911]	ErrorDomain exception base type naming
[SWS_CORE_10912]	ErrorDomain exception type hierarchy
[SWS_CORE_10930]	ErrorDomain subclass type
[SWS_CORE_10931]	ErrorDomain subclass naming
[SWS_CORE_10932]	ErrorDomain subclass non-extensibility
[SWS_CORE_10933]	ErrorDomain subclass Errc symbol
[SWS_CORE_10934]	ErrorDomain subclass Exception symbol
[SWS_CORE_10950]	ErrorDomain subclass member function property
[SWS_CORE_10951]	ErrorDomain subclass shortname retrieval
[SWS_CORE_10952]	ErrorDomain subclass unique identifier retrieval
[SWS_CORE_10953]	Throwing ErrorCodes as exceptions
[SWS_CORE_10980]	ErrorDomain subclass accessor function
[SWS_CORE_10981]	ErrorDomain subclass accessor function naming
[SWS_CORE_10982]	ErrorDomain subclass accessor function
[SWS_CORE_10990]	MakeErrorCode overload for new error domains
[SWS_CORE_10991]	MakeErrorCode overload signature
[SWS_CORE_10999]	Custom error domain scope
[SWS_CORE_11200]	Array base behavior
[SWS_CORE_11800]	SteadyClock type requirements
[SWS_CORE_11801]	Epoch of SteadyClock

▽

△

Number	Heading
[SWS_CORE_12402]	“Noreturn” property for Abort
[SWS_CORE_12403]	Logging of Explicit Operation Abortion
[SWS_CORE_12404]	AbortHandler invocation
[SWS_CORE_12405]	Final action without AbortHandler
[SWS_CORE_12406]	Final action with a returning AbortHandler
[SWS_CORE_12407]	Thread-safety of Explicit Operation Abortion
[SWS_CORE_90001]	Include folder structure
[SWS_CORE_90002]	Prevent multiple inclusion of header file
[SWS_CORE_90003]	

Table E.4: Added Specification Items in R20-11

E.2.2 Changed Specification Items in R20-11

Number	Heading
[SWS_CORE_00010]	Error domain identifier
[SWS_CORE_00050]	
[SWS_CORE_00051]	
[SWS_CORE_00052]	
[SWS_CORE_00110]	
[SWS_CORE_00121]	
[SWS_CORE_00122]	
[SWS_CORE_00123]	
[SWS_CORE_00131]	
[SWS_CORE_00132]	
[SWS_CORE_00133]	
[SWS_CORE_00134]	
[SWS_CORE_00135]	
[SWS_CORE_00136]	
[SWS_CORE_00137]	
[SWS_CORE_00138]	
[SWS_CORE_00151]	
[SWS_CORE_00152]	
[SWS_CORE_00153]	
[SWS_CORE_00154]	
[SWS_CORE_00321]	
[SWS_CORE_00322]	

▽

△

Number	Heading
[SWS_CORE_00323]	
[SWS_CORE_00325]	
[SWS_CORE_00326]	
[SWS_CORE_00327]	
[SWS_CORE_00328]	
[SWS_CORE_00329]	
[SWS_CORE_00330]	
[SWS_CORE_00331]	
[SWS_CORE_00332]	
[SWS_CORE_00333]	
[SWS_CORE_00334]	
[SWS_CORE_00335]	
[SWS_CORE_00336]	
[SWS_CORE_00340]	
[SWS_CORE_00341]	
[SWS_CORE_00342]	
[SWS_CORE_00343]	
[SWS_CORE_00344]	
[SWS_CORE_00345]	
[SWS_CORE_00346]	
[SWS_CORE_00349]	
[SWS_CORE_00350]	
[SWS_CORE_00351]	
[SWS_CORE_00352]	
[SWS_CORE_00353]	
[SWS_CORE_00354]	
[SWS_CORE_00361]	
[SWS_CORE_00400]	
[SWS_CORE_00411]	
[SWS_CORE_00412]	
[SWS_CORE_00421]	
[SWS_CORE_00431]	
[SWS_CORE_00432]	
[SWS_CORE_00441]	
[SWS_CORE_00442]	
[SWS_CORE_00443]	
[SWS_CORE_00444]	
[SWS_CORE_00480]	
[SWS_CORE_00490]	

▽

△

Number	Heading
[SWS_CORE_00501]	
[SWS_CORE_00512]	
[SWS_CORE_00513]	
[SWS_CORE_00514]	
[SWS_CORE_00515]	
[SWS_CORE_00516]	
[SWS_CORE_00518]	
[SWS_CORE_00519]	
[SWS_CORE_00571]	
[SWS_CORE_00572]	
[SWS_CORE_00601]	
[SWS_CORE_00611]	
[SWS_CORE_00612]	
[SWS_CORE_00613]	
[SWS_CORE_00701]	
[SWS_CORE_00711]	
[SWS_CORE_00712]	
[SWS_CORE_00721]	
[SWS_CORE_00722]	
[SWS_CORE_00723]	
[SWS_CORE_00724]	
[SWS_CORE_00725]	
[SWS_CORE_00726]	
[SWS_CORE_00727]	
[SWS_CORE_00731]	
[SWS_CORE_00732]	
[SWS_CORE_00733]	
[SWS_CORE_00734]	
[SWS_CORE_00735]	
[SWS_CORE_00736]	
[SWS_CORE_00741]	
[SWS_CORE_00742]	
[SWS_CORE_00743]	
[SWS_CORE_00744]	
[SWS_CORE_00745]	
[SWS_CORE_00751]	
[SWS_CORE_00752]	
[SWS_CORE_00753]	
[SWS_CORE_00754]	

▽

△

Number	Heading
[SWS_CORE_00755]	
[SWS_CORE_00756]	
[SWS_CORE_00757]	
[SWS_CORE_00758]	
[SWS_CORE_00759]	
[SWS_CORE_00761]	
[SWS_CORE_00762]	
[SWS_CORE_00763]	
[SWS_CORE_00765]	
[SWS_CORE_00766]	
[SWS_CORE_00767]	
[SWS_CORE_00768]	
[SWS_CORE_00769]	
[SWS_CORE_00780]	
[SWS_CORE_00781]	
[SWS_CORE_00782]	
[SWS_CORE_00783]	
[SWS_CORE_00784]	
[SWS_CORE_00785]	
[SWS_CORE_00786]	
[SWS_CORE_00787]	
[SWS_CORE_00788]	
[SWS_CORE_00789]	
[SWS_CORE_00796]	
[SWS_CORE_00801]	
[SWS_CORE_00811]	
[SWS_CORE_00812]	
[SWS_CORE_00821]	
[SWS_CORE_00823]	
[SWS_CORE_00824]	
[SWS_CORE_00825]	
[SWS_CORE_00826]	
[SWS_CORE_00827]	
[SWS_CORE_00831]	
[SWS_CORE_00834]	
[SWS_CORE_00835]	
[SWS_CORE_00836]	
[SWS_CORE_00841]	
[SWS_CORE_00842]	

▽

△

Number	Heading
[SWS_CORE_00843]	
[SWS_CORE_00844]	
[SWS_CORE_00845]	
[SWS_CORE_00851]	
[SWS_CORE_00852]	
[SWS_CORE_00853]	
[SWS_CORE_00855]	
[SWS_CORE_00857]	
[SWS_CORE_00858]	
[SWS_CORE_00861]	
[SWS_CORE_00863]	
[SWS_CORE_00865]	
[SWS_CORE_00866]	
[SWS_CORE_00867]	
[SWS_CORE_01201]	
[SWS_CORE_01296]	
[SWS_CORE_01390]	Global `operator==` for `Vector`
[SWS_CORE_01391]	Global `operator!=` for `Vector`
[SWS_CORE_01392]	Global `operator<` for `Vector`
[SWS_CORE_01393]	Global `operator<=` for `Vector`
[SWS_CORE_01394]	Global `operator>` for `Vector`
[SWS_CORE_01395]	Global `operator>=` for `Vector`
[SWS_CORE_01900]	
[SWS_CORE_01901]	
[SWS_CORE_01911]	
[SWS_CORE_01912]	
[SWS_CORE_01913]	
[SWS_CORE_01914]	
[SWS_CORE_01915]	
[SWS_CORE_01916]	
[SWS_CORE_01917]	
[SWS_CORE_01918]	
[SWS_CORE_01919]	
[SWS_CORE_01920]	
[SWS_CORE_01921]	
[SWS_CORE_01931]	
[SWS_CORE_01941]	
[SWS_CORE_01942]	
[SWS_CORE_01943]	

▽

△

Number	Heading
[SWS_CORE_01944]	
[SWS_CORE_01945]	
[SWS_CORE_01946]	
[SWS_CORE_01947]	
[SWS_CORE_01948]	
[SWS_CORE_01949]	
[SWS_CORE_01950]	
[SWS_CORE_01951]	
[SWS_CORE_01952]	
[SWS_CORE_01961]	
[SWS_CORE_01962]	
[SWS_CORE_01963]	
[SWS_CORE_01964]	
[SWS_CORE_01965]	
[SWS_CORE_01966]	
[SWS_CORE_01967]	
[SWS_CORE_01968]	
[SWS_CORE_01969]	
[SWS_CORE_01970]	
[SWS_CORE_01971]	
[SWS_CORE_01972]	
[SWS_CORE_01973]	
[SWS_CORE_01974]	
[SWS_CORE_01975]	
[SWS_CORE_01976]	
[SWS_CORE_01977]	
[SWS_CORE_01978]	
[SWS_CORE_01979]	
[SWS_CORE_01990]	
[SWS_CORE_01991]	
[SWS_CORE_01992]	
[SWS_CORE_01993]	
[SWS_CORE_01994]	
[SWS_CORE_03303]	Constructor from implicit `StringView`
[SWS_CORE_03306]	Assignment from implicit `StringView`
[SWS_CORE_03309]	Concatenation of implicit `StringView`
[SWS_CORE_03311]	Insertion of implicit `StringView`
[SWS_CORE_03313]	Replacement with implicit `StringView`
[SWS_CORE_03323]	Comparison of subsequence with a subsequence of a `StringView`

▽

△

Number	Heading
[SWS_CORE_04011]	
[SWS_CORE_04012]	
[SWS_CORE_04013]	
[SWS_CORE_04021]	
[SWS_CORE_04022]	
[SWS_CORE_04031]	
[SWS_CORE_04032]	
[SWS_CORE_04110]	
[SWS_CORE_04111]	
[SWS_CORE_04112]	
[SWS_CORE_04113]	
[SWS_CORE_04120]	
[SWS_CORE_04121]	
[SWS_CORE_04130]	
[SWS_CORE_04131]	
[SWS_CORE_04132]	
[SWS_CORE_04200]	
[SWS_CORE_05200]	
[SWS_CORE_05211]	
[SWS_CORE_05212]	
[SWS_CORE_05221]	
[SWS_CORE_05231]	
[SWS_CORE_05232]	
[SWS_CORE_05241]	
[SWS_CORE_05242]	
[SWS_CORE_05243]	
[SWS_CORE_05244]	
[SWS_CORE_05280]	
[SWS_CORE_05290]	
[SWS_CORE_06221]	
[SWS_CORE_06222]	
[SWS_CORE_06223]	
[SWS_CORE_06225]	
[SWS_CORE_06226]	
[SWS_CORE_06227]	
[SWS_CORE_06228]	
[SWS_CORE_06229]	
[SWS_CORE_06230]	
[SWS_CORE_06231]	

▽

△

Number	Heading
[SWS_CORE_06232]	
[SWS_CORE_06233]	
[SWS_CORE_06234]	
[SWS_CORE_06235]	
[SWS_CORE_06236]	
[SWS_CORE_06340]	
[SWS_CORE_06341]	
[SWS_CORE_06342]	
[SWS_CORE_06343]	
[SWS_CORE_06344]	
[SWS_CORE_06345]	
[SWS_CORE_06349]	
[SWS_CORE_06350]	
[SWS_CORE_06351]	
[SWS_CORE_06352]	
[SWS_CORE_06353]	
[SWS_CORE_06354]	
[SWS_CORE_08001]	
[SWS_CORE_08021]	
[SWS_CORE_08029]	
[SWS_CORE_08032]	
[SWS_CORE_08041]	
[SWS_CORE_08042]	
[SWS_CORE_08043]	
[SWS_CORE_08044]	
[SWS_CORE_08045]	
[SWS_CORE_08046]	
[SWS_CORE_10001]	
[SWS_CORE_10002]	
[SWS_CORE_10109]	Equality comparison for ara::core::Byte
[SWS_CORE_10110]	Non-equality comparison for ara::core::Byte

Table E.5: Changed Specification Items in R20-11

E.2.3 Deleted Specification Items in R20-11

none

E.3 Change History of this document according to AUTOSAR Release R21-11

E.3.1 Added Specification Items in R21-11

Number	Heading
[SWS_CORE_00020]	Semantics of an Error
[SWS_CORE_00021]	Semantics of a Violation
[SWS_CORE_00022]	Semantics of a Corruption
[SWS_CORE_00023]	Semantics of a Failed Default Allocation
[SWS_CORE_01922]	
[SWS_CORE_01923]	
[SWS_CORE_01953]	
[SWS_CORE_01954]	
[SWS_CORE_01959]	
[SWS_CORE_01960]	
[SWS_CORE_08101]	
[SWS_CORE_08111]	
[SWS_CORE_08121]	
[SWS_CORE_08122]	
[SWS_CORE_08123]	
[SWS_CORE_08124]	
[SWS_CORE_08125]	
[SWS_CORE_08126]	
[SWS_CORE_08127]	
[SWS_CORE_08128]	
[SWS_CORE_08129]	
[SWS_CORE_08141]	
[SWS_CORE_08180]	
[SWS_CORE_08181]	
[SWS_CORE_08182]	
[SWS_CORE_08183]	
[SWS_CORE_08184]	
[SWS_CORE_08185]	
[SWS_CORE_08186]	
[SWS_CORE_08187]	
[SWS_CORE_08188]	
[SWS_CORE_08189]	
[SWS_CORE_08190]	

▽

△

Number	Heading
[SWS_CORE_08191]	
[SWS_CORE_08192]	
[SWS_CORE_08193]	
[SWS_CORE_08194]	
[SWS_CORE_08195]	
[SWS_CORE_08196]	
[SWS_CORE_08197]	
[SWS_CORE_08198]	
[SWS_CORE_08199]	
[SWS_CORE_10301]	Comparison of ara::core::ErrorCode instances
[SWS_CORE_10302]	Semantics of ErrorCode
[SWS_CORE_10303]	Semantics of ErrorDomain
[SWS_CORE_10401]	Identity of ErrorDomains
[SWS_CORE_10600]	Semantics of ara::core::Result
[SWS_CORE_10800]	Semantics of ara::core::Future and ara::core::Promise
[SWS_CORE_15001]	Handling of interaction with the ARA of an un-/deinitialized runtime
[SWS_CORE_15002]	Special ara::core types to be used without initialization
[SWS_CORE_15003]	Startup and initialization of ARA
[SWS_CORE_15004]	Shutdown and de-initialization of ARA
[SWS_CORE_90004]	Implementation-defined declaration classifiers
[SWS_CORE_90020]	

Table E.6: Added Specification Items in R21-11

E.3.2 Changed Specification Items in R21-11

Number	Heading
[SWS_CORE_00002]	Handling of Errors
[SWS_CORE_00003]	Handling of Violations
[SWS_CORE_00013]	The Future error domain
[SWS_CORE_00014]	The Core error domain
[SWS_CORE_00040]	Errors originating from C++ standard classes
[SWS_CORE_00050]	
[SWS_CORE_00051]	
[SWS_CORE_00052]	
[SWS_CORE_00053]	
[SWS_CORE_00110]	

▽

△

Number	Heading
[SWS_CORE_00121]	
[SWS_CORE_00122]	
[SWS_CORE_00123]	
[SWS_CORE_00131]	
[SWS_CORE_00132]	
[SWS_CORE_00133]	
[SWS_CORE_00134]	
[SWS_CORE_00135]	
[SWS_CORE_00136]	
[SWS_CORE_00137]	
[SWS_CORE_00138]	
[SWS_CORE_00151]	
[SWS_CORE_00152]	
[SWS_CORE_00153]	
[SWS_CORE_00154]	
[SWS_CORE_00321]	
[SWS_CORE_00322]	
[SWS_CORE_00323]	
[SWS_CORE_00325]	
[SWS_CORE_00326]	
[SWS_CORE_00327]	
[SWS_CORE_00328]	
[SWS_CORE_00329]	
[SWS_CORE_00330]	
[SWS_CORE_00331]	
[SWS_CORE_00332]	
[SWS_CORE_00333]	
[SWS_CORE_00334]	
[SWS_CORE_00335]	
[SWS_CORE_00336]	
[SWS_CORE_00337]	
[SWS_CORE_00340]	
[SWS_CORE_00341]	
[SWS_CORE_00342]	
[SWS_CORE_00343]	
[SWS_CORE_00344]	
[SWS_CORE_00345]	
[SWS_CORE_00346]	
[SWS_CORE_00349]	

▽

△

Number	Heading
[SWS_CORE_00350]	
[SWS_CORE_00351]	
[SWS_CORE_00352]	
[SWS_CORE_00353]	
[SWS_CORE_00354]	
[SWS_CORE_00355]	
[SWS_CORE_00356]	
[SWS_CORE_00361]	
[SWS_CORE_00400]	
[SWS_CORE_00411]	
[SWS_CORE_00412]	
[SWS_CORE_00421]	
[SWS_CORE_00431]	
[SWS_CORE_00432]	
[SWS_CORE_00441]	
[SWS_CORE_00442]	
[SWS_CORE_00443]	
[SWS_CORE_00444]	
[SWS_CORE_00480]	
[SWS_CORE_00490]	
[SWS_CORE_00501]	
[SWS_CORE_00512]	
[SWS_CORE_00513]	
[SWS_CORE_00514]	
[SWS_CORE_00515]	
[SWS_CORE_00516]	
[SWS_CORE_00518]	
[SWS_CORE_00519]	
[SWS_CORE_00571]	
[SWS_CORE_00572]	
[SWS_CORE_00601]	
[SWS_CORE_00611]	
[SWS_CORE_00612]	
[SWS_CORE_00613]	
[SWS_CORE_00614]	
[SWS_CORE_00701]	
[SWS_CORE_00711]	
[SWS_CORE_00712]	
[SWS_CORE_00721]	

▽

△

Number	Heading
[SWS_CORE_00722]	
[SWS_CORE_00723]	
[SWS_CORE_00724]	
[SWS_CORE_00725]	
[SWS_CORE_00726]	
[SWS_CORE_00727]	
[SWS_CORE_00731]	
[SWS_CORE_00732]	
[SWS_CORE_00733]	
[SWS_CORE_00734]	
[SWS_CORE_00735]	
[SWS_CORE_00736]	
[SWS_CORE_00741]	
[SWS_CORE_00742]	
[SWS_CORE_00743]	
[SWS_CORE_00744]	
[SWS_CORE_00745]	
[SWS_CORE_00751]	
[SWS_CORE_00752]	
[SWS_CORE_00753]	
[SWS_CORE_00754]	
[SWS_CORE_00755]	
[SWS_CORE_00756]	
[SWS_CORE_00757]	
[SWS_CORE_00758]	
[SWS_CORE_00759]	
[SWS_CORE_00761]	
[SWS_CORE_00762]	
[SWS_CORE_00763]	
[SWS_CORE_00764]	
[SWS_CORE_00765]	
[SWS_CORE_00766]	
[SWS_CORE_00767]	
[SWS_CORE_00768]	
[SWS_CORE_00769]	
[SWS_CORE_00770]	
[SWS_CORE_00771]	
[SWS_CORE_00772]	
[SWS_CORE_00773]	

▽

△

Number	Heading
[SWS_CORE_00780]	
[SWS_CORE_00781]	
[SWS_CORE_00782]	
[SWS_CORE_00783]	
[SWS_CORE_00784]	
[SWS_CORE_00785]	
[SWS_CORE_00786]	
[SWS_CORE_00787]	
[SWS_CORE_00788]	
[SWS_CORE_00789]	
[SWS_CORE_00796]	
[SWS_CORE_00801]	
[SWS_CORE_00811]	
[SWS_CORE_00812]	
[SWS_CORE_00821]	
[SWS_CORE_00823]	
[SWS_CORE_00824]	
[SWS_CORE_00825]	
[SWS_CORE_00826]	
[SWS_CORE_00827]	
[SWS_CORE_00831]	
[SWS_CORE_00834]	
[SWS_CORE_00835]	
[SWS_CORE_00836]	
[SWS_CORE_00841]	
[SWS_CORE_00842]	
[SWS_CORE_00843]	
[SWS_CORE_00844]	
[SWS_CORE_00845]	
[SWS_CORE_00851]	
[SWS_CORE_00852]	
[SWS_CORE_00853]	
[SWS_CORE_00855]	
[SWS_CORE_00857]	
[SWS_CORE_00858]	
[SWS_CORE_00861]	
[SWS_CORE_00863]	
[SWS_CORE_00864]	
[SWS_CORE_00865]	

▽

△

Number	Heading
[SWS_CORE_00866]	
[SWS_CORE_00867]	
[SWS_CORE_00868]	
[SWS_CORE_00869]	
[SWS_CORE_00870]	
[SWS_CORE_01201]	
[SWS_CORE_01210]	
[SWS_CORE_01211]	
[SWS_CORE_01212]	
[SWS_CORE_01213]	
[SWS_CORE_01214]	
[SWS_CORE_01215]	
[SWS_CORE_01216]	
[SWS_CORE_01217]	
[SWS_CORE_01218]	
[SWS_CORE_01219]	
[SWS_CORE_01220]	
[SWS_CORE_01241]	
[SWS_CORE_01242]	
[SWS_CORE_01250]	
[SWS_CORE_01251]	
[SWS_CORE_01252]	
[SWS_CORE_01253]	
[SWS_CORE_01254]	
[SWS_CORE_01255]	
[SWS_CORE_01256]	
[SWS_CORE_01257]	
[SWS_CORE_01258]	
[SWS_CORE_01259]	
[SWS_CORE_01260]	
[SWS_CORE_01261]	
[SWS_CORE_01262]	
[SWS_CORE_01263]	
[SWS_CORE_01264]	
[SWS_CORE_01265]	
[SWS_CORE_01266]	
[SWS_CORE_01267]	
[SWS_CORE_01268]	
[SWS_CORE_01269]	

▽

△

Number	Heading
[SWS_CORE_01270]	
[SWS_CORE_01271]	
[SWS_CORE_01272]	
[SWS_CORE_01280]	
[SWS_CORE_01281]	
[SWS_CORE_01282]	
[SWS_CORE_01283]	
[SWS_CORE_01284]	
[SWS_CORE_01285]	
[SWS_CORE_01290]	
[SWS_CORE_01291]	
[SWS_CORE_01292]	
[SWS_CORE_01293]	
[SWS_CORE_01294]	
[SWS_CORE_01295]	
[SWS_CORE_01296]	
[SWS_CORE_01900]	
[SWS_CORE_01901]	
[SWS_CORE_01911]	
[SWS_CORE_01912]	
[SWS_CORE_01914]	
[SWS_CORE_01915]	
[SWS_CORE_01916]	
[SWS_CORE_01917]	
[SWS_CORE_01918]	
[SWS_CORE_01919]	
[SWS_CORE_01920]	
[SWS_CORE_01921]	
[SWS_CORE_01931]	
[SWS_CORE_01941]	
[SWS_CORE_01942]	
[SWS_CORE_01943]	
[SWS_CORE_01944]	
[SWS_CORE_01945]	
[SWS_CORE_01946]	
[SWS_CORE_01947]	
[SWS_CORE_01948]	
[SWS_CORE_01949]	
[SWS_CORE_01950]	

▽

△

Number	Heading
[SWS_CORE_01951]	
[SWS_CORE_01952]	
[SWS_CORE_01961]	
[SWS_CORE_01962]	
[SWS_CORE_01963]	
[SWS_CORE_01964]	
[SWS_CORE_01965]	
[SWS_CORE_01966]	
[SWS_CORE_01967]	
[SWS_CORE_01968]	
[SWS_CORE_01969]	
[SWS_CORE_01970]	
[SWS_CORE_01971]	
[SWS_CORE_01972]	
[SWS_CORE_01973]	
[SWS_CORE_01974]	
[SWS_CORE_01975]	
[SWS_CORE_01976]	
[SWS_CORE_01977]	
[SWS_CORE_01978]	
[SWS_CORE_01979]	
[SWS_CORE_01980]	
[SWS_CORE_01981]	
[SWS_CORE_01990]	
[SWS_CORE_01991]	
[SWS_CORE_01992]	
[SWS_CORE_01993]	
[SWS_CORE_01994]	
[SWS_CORE_03000]	`BasicString` type
[SWS_CORE_04011]	
[SWS_CORE_04012]	
[SWS_CORE_04013]	
[SWS_CORE_04021]	
[SWS_CORE_04022]	
[SWS_CORE_04023]	
[SWS_CORE_04031]	
[SWS_CORE_04032]	
[SWS_CORE_04033]	
[SWS_CORE_04110]	

▽

△

Number	Heading
[SWS_CORE_04111]	
[SWS_CORE_04112]	
[SWS_CORE_04113]	
[SWS_CORE_04120]	
[SWS_CORE_04121]	
[SWS_CORE_04130]	
[SWS_CORE_04131]	
[SWS_CORE_04132]	
[SWS_CORE_04200]	
[SWS_CORE_05200]	
[SWS_CORE_05211]	
[SWS_CORE_05212]	
[SWS_CORE_05221]	
[SWS_CORE_05231]	
[SWS_CORE_05232]	
[SWS_CORE_05241]	
[SWS_CORE_05242]	
[SWS_CORE_05243]	
[SWS_CORE_05244]	
[SWS_CORE_05280]	
[SWS_CORE_05290]	
[SWS_CORE_06221]	
[SWS_CORE_06222]	
[SWS_CORE_06223]	
[SWS_CORE_06225]	
[SWS_CORE_06226]	
[SWS_CORE_06227]	
[SWS_CORE_06228]	
[SWS_CORE_06229]	
[SWS_CORE_06230]	
[SWS_CORE_06231]	
[SWS_CORE_06232]	
[SWS_CORE_06233]	
[SWS_CORE_06234]	
[SWS_CORE_06235]	
[SWS_CORE_06236]	
[SWS_CORE_06237]	
[SWS_CORE_06340]	
[SWS_CORE_06341]	

▽

△

Number	Heading
[SWS_CORE_06342]	
[SWS_CORE_06343]	
[SWS_CORE_06344]	
[SWS_CORE_06345]	
[SWS_CORE_06349]	
[SWS_CORE_06350]	
[SWS_CORE_06351]	
[SWS_CORE_06352]	
[SWS_CORE_06353]	
[SWS_CORE_06354]	
[SWS_CORE_06355]	
[SWS_CORE_06356]	
[SWS_CORE_06401]	
[SWS_CORE_06411]	
[SWS_CORE_06412]	
[SWS_CORE_06413]	
[SWS_CORE_06414]	
[SWS_CORE_06431]	
[SWS_CORE_06432]	
[SWS_CORE_08001]	
[SWS_CORE_08021]	
[SWS_CORE_08022]	
[SWS_CORE_08023]	
[SWS_CORE_08024]	
[SWS_CORE_08025]	
[SWS_CORE_08029]	
[SWS_CORE_08032]	
[SWS_CORE_08041]	
[SWS_CORE_08042]	
[SWS_CORE_08043]	
[SWS_CORE_08044]	
[SWS_CORE_08045]	
[SWS_CORE_08046]	
[SWS_CORE_08081]	
[SWS_CORE_08082]	
[SWS_CORE_10001]	
[SWS_CORE_10002]	
[SWS_CORE_10100]	Type property of ara::core::Byte
[SWS_CORE_10101]	Size of type ara::core::Byte

▽

△

Number	Heading
[SWS_CORE_10102]	Value range of type ara::core::Byte
[SWS_CORE_10103]	Creation of ara::core::Byte instances
[SWS_CORE_10104]	Default-constructed ara::core::Byte instances
[SWS_CORE_10105]	Destructor of type ara::core::Byte
[SWS_CORE_10106]	Implicit conversion from other types
[SWS_CORE_10107]	Implicit conversion to other types
[SWS_CORE_10108]	Conversion to unsigned char
[SWS_CORE_10109]	Equality comparison for ara::core::Byte
[SWS_CORE_10110]	Non-equality comparison for ara::core::Byte
[SWS_CORE_10200]	Valid InstanceSpecifier representations
[SWS_CORE_10201]	Validation of meta-model paths
[SWS_CORE_10202]	Construction of InstanceSpecifier objects
[SWS_CORE_10300]	ErrorCode type properties
[SWS_CORE_10400]	ErrorDomain type properties
[SWS_CORE_10900]	Error condition enumeration type
[SWS_CORE_10901]	Error condition enumeration naming
[SWS_CORE_10910]	ErrorDomain exception base type
[SWS_CORE_10911]	ErrorDomain exception base type naming
[SWS_CORE_10930]	ErrorDomain subclass type
[SWS_CORE_10931]	ErrorDomain subclass naming
[SWS_CORE_10932]	ErrorDomain subclass non-extensibility
[SWS_CORE_10933]	ErrorDomain subclass Errc symbol
[SWS_CORE_10934]	ErrorDomain subclass Exception symbol
[SWS_CORE_10950]	ErrorDomain subclass member function property
[SWS_CORE_10951]	ErrorDomain subclass shortname retrieval
[SWS_CORE_10952]	ErrorDomain subclass unique identifier retrieval
[SWS_CORE_10953]	Throwing ErrorCodes as exceptions
[SWS_CORE_10980]	ErrorDomain subclass accessor function
[SWS_CORE_10981]	ErrorDomain subclass accessor function naming
[SWS_CORE_10982]	ErrorDomain subclass accessor function
[SWS_CORE_10990]	MakeErrorCode overload for new error domains
[SWS_CORE_10991]	MakeErrorCode overload signature
[SWS_CORE_10999]	Custom error domain scope
[SWS_CORE_11800]	SteadyClock type requirements
[SWS_CORE_12403]	Logging of Explicit Operation Abortion

Table E.7: Changed Specification Items in R21-11

E.3.3 Deleted Specification Items in R21-11

Number	Heading
[SWS_CORE_01913]	

Table E.8: Deleted Specification Items in R21-11

E.4 Change History of this document according to AUTOSAR Release R22-11

E.4.1 Added Specification Items in R22-11

Number	Heading
[SWS_CORE_00054]	
[SWS_CORE_00615]	
[SWS_CORE_00616]	
[SWS_CORE_00617]	
[SWS_CORE_00618]	
[SWS_CORE_03012]	
[SWS_CORE_10203]	Valid InstanceSpecifier representations - functional cluster interaction
[SWS_CORE_11000]	Optional base behavior
[SWS_CORE_11300]	Vector base behavior
[SWS_CORE_11400]	Map base behavior
[SWS_CORE_11600]	Variant base behavior
[SWS_CORE_11900]	Span base behavior
[SWS_CORE_12000]	String base behavior
[SWS_CORE_12200]	StringView base behavior
[SWS_CORE_90005]	Custom declarations and definitions
[SWS_CORE_90006]	

Table E.9: Added Specification Items in R22-11

E.4.2 Changed Specification Items in R22-11

Number	Heading
[SWS_CORE_00051]	
[SWS_CORE_00052]	
[SWS_CORE_00340]	
[SWS_CORE_00341]	
[SWS_CORE_00342]	
[SWS_CORE_00343]	
[SWS_CORE_00344]	
[SWS_CORE_00345]	
[SWS_CORE_00346]	
[SWS_CORE_00349]	
[SWS_CORE_00350]	
[SWS_CORE_00351]	
[SWS_CORE_00352]	
[SWS_CORE_00353]	
[SWS_CORE_00354]	
[SWS_CORE_00355]	
[SWS_CORE_00356]	
[SWS_CORE_00571]	
[SWS_CORE_00572]	
[SWS_CORE_00614]	
[SWS_CORE_01033]	
[SWS_CORE_01096]	
[SWS_CORE_01301]	
[SWS_CORE_01390]	
[SWS_CORE_01391]	
[SWS_CORE_01392]	
[SWS_CORE_01393]	
[SWS_CORE_01394]	
[SWS_CORE_01395]	
[SWS_CORE_01396]	
[SWS_CORE_01400]	
[SWS_CORE_01496]	
[SWS_CORE_01601]	
[SWS_CORE_01696]	
[SWS_CORE_02001]	
[SWS_CORE_03000]	

▽

△

Number	Heading
[SWS_CORE_03001]	
[SWS_CORE_03296]	
[SWS_CORE_03301]	
[SWS_CORE_03302]	
[SWS_CORE_03303]	
[SWS_CORE_03304]	
[SWS_CORE_03305]	
[SWS_CORE_03306]	
[SWS_CORE_03307]	
[SWS_CORE_03308]	
[SWS_CORE_03309]	
[SWS_CORE_03310]	
[SWS_CORE_03311]	
[SWS_CORE_03312]	
[SWS_CORE_03313]	
[SWS_CORE_03314]	
[SWS_CORE_03315]	
[SWS_CORE_03316]	
[SWS_CORE_03317]	
[SWS_CORE_03318]	
[SWS_CORE_03319]	
[SWS_CORE_03320]	
[SWS_CORE_03321]	
[SWS_CORE_03322]	
[SWS_CORE_03323]	
[SWS_CORE_05244]	
[SWS_CORE_06340]	
[SWS_CORE_06341]	
[SWS_CORE_06342]	
[SWS_CORE_06343]	
[SWS_CORE_06344]	
[SWS_CORE_06345]	
[SWS_CORE_06349]	
[SWS_CORE_06350]	
[SWS_CORE_06351]	
[SWS_CORE_06352]	
[SWS_CORE_06353]	
[SWS_CORE_06354]	
[SWS_CORE_06355]	

▽

△

Number	Heading
[SWS_CORE_06356]	
[SWS_CORE_08021]	
[SWS_CORE_08032]	
[SWS_CORE_10200]	Valid InstanceSpecifier representations - application interaction
[SWS_CORE_10980]	ErrorDomain subclass accessor function
[SWS_CORE_10990]	MakeErrorCode overload for new error domains
[SWS_CORE_10991]	MakeErrorCode overload signature
[SWS_CORE_10999]	Custom error domain scope
[SWS_CORE_11200]	Array base behavior
[SWS_CORE_12404]	AbortHandler invocation
[SWS_CORE_12405]	Final action without AbortHandler
[SWS_CORE_12406]	Final action with returning AbortHandlers
[SWS_CORE_15002]	Special ara::core types to be used without initialization
[SWS_CORE_90003]	

Table E.10: Changed Specification Items in R22-11

E.4.3 Deleted Specification Items in R22-11

none

E.5 Change History of this document according to AUTOSAR Release R23-11

E.5.1 Added Specification Items in R23-11

Number	Heading
[SWS_CORE_00774]	Definition of API function ara::core::Result::operator*
[SWS_CORE_00775]	Definition of API function ara::core::Result::Value
[SWS_CORE_00776]	Definition of API function ara::core::Result::Error
[SWS_CORE_00876]	Definition of API function ara::core::Result< void, E >::Error
[SWS_CORE_01273]	Definition of API function ara::core::Array::at
[SWS_CORE_01274]	Definition of API function ara::core::Array::at
[SWS_CORE_06500]	Definition of API class ara::core::MemoryResource
[SWS_CORE_06501]	Definition of API function ara::core::MemoryResource::MemoryResource
[SWS_CORE_06502]	Definition of API function ara::core::MemoryResource::MemoryResource

▽

△

Number	Heading
[SWS_CORE_06503]	Definition of API function ara::core::MemoryResource::allocate
[SWS_CORE_06504]	Definition of API function ara::core::MemoryResource::deallocate
[SWS_CORE_06505]	Definition of API function ara::core::MemoryResource::is_equal
[SWS_CORE_06506]	Definition of API function ara::core::MemoryResource::~MemoryResource
[SWS_CORE_06507]	Definition of API function ara::core::MemoryResource::operator=
[SWS_CORE_06520]	Definition of API class ara::core::MonotonicBufferResource
[SWS_CORE_06521]	Definition of API function ara::core::MonotonicBufferResource::MonotonicBufferResource
[SWS_CORE_06522]	Definition of API function ara::core::MonotonicBufferResource::MonotonicBufferResource
[SWS_CORE_06523]	Definition of API function ara::core::MonotonicBufferResource::MonotonicBufferResource
[SWS_CORE_06524]	Definition of API function ara::core::MonotonicBufferResource::MonotonicBufferResource
[SWS_CORE_06525]	Definition of API function ara::core::MonotonicBufferResource::MonotonicBufferResource
[SWS_CORE_06526]	Definition of API function ara::core::MonotonicBufferResource::MonotonicBufferResource
[SWS_CORE_06527]	Definition of API function ara::core::MonotonicBufferResource::MonotonicBufferResource
[SWS_CORE_06528]	Definition of API function ara::core::MonotonicBufferResource::~MonotonicBufferResource
[SWS_CORE_06529]	Definition of API function ara::core::MonotonicBufferResource::operator=
[SWS_CORE_06530]	Definition of API function ara::core::MonotonicBufferResource::release
[SWS_CORE_06531]	Definition of API function ara::core::MonotonicBufferResource::upstream_resource
[SWS_CORE_06540]	Definition of API class ara::core::PolymorphicAllocator
[SWS_CORE_06541]	Definition of API function ara::core::PolymorphicAllocator::PolymorphicAllocator
[SWS_CORE_06542]	Definition of API function ara::core::PolymorphicAllocator::PolymorphicAllocator
[SWS_CORE_06543]	Definition of API function ara::core::PolymorphicAllocator::PolymorphicAllocator
[SWS_CORE_06544]	Definition of API function ara::core::PolymorphicAllocator::PolymorphicAllocator
[SWS_CORE_06545]	Definition of API function ara::core::PolymorphicAllocator::operator=
[SWS_CORE_06546]	Definition of API function ara::core::PolymorphicAllocator::PolymorphicAllocator
[SWS_CORE_06547]	Definition of API function ara::core::PolymorphicAllocator::allocate
[SWS_CORE_06548]	Definition of API function ara::core::PolymorphicAllocator::deallocate
[SWS_CORE_06549]	Definition of API function ara::core::PolymorphicAllocator::allocate_bytes
[SWS_CORE_06550]	Definition of API function ara::core::PolymorphicAllocator::deallocate_bytes
[SWS_CORE_06551]	Definition of API function ara::core::PolymorphicAllocator::allocate_object
[SWS_CORE_06552]	Definition of API function ara::core::PolymorphicAllocator::deallocate_object

▽

△

Number	Heading
[SWS_CORE_06553]	Definition of API function ara::core::PolymorphicAllocator::new_object
[SWS_CORE_06554]	Definition of API function ara::core::PolymorphicAllocator::delete_object
[SWS_CORE_06555]	Definition of API function ara::core::PolymorphicAllocator::construct
[SWS_CORE_06556]	Definition of API function ara::core::PolymorphicAllocator::destroy
[SWS_CORE_06557]	Definition of API function ara::core::PolymorphicAllocator::resource
[SWS_CORE_06560]	Definition of API function ara::core::operator==
[SWS_CORE_06561]	Definition of API function ara::core::operator==
[SWS_CORE_06562]	Definition of API function ara::core::NewDeleteResource
[SWS_CORE_06563]	Definition of API function ara::core::NullMemoryResource
[SWS_CORE_06564]	Definition of API function ara::core::SetDefaultResource
[SWS_CORE_06565]	Definition of API function ara::core::GetDefaultResource
[SWS_CORE_11950]	MemoryResource base behavior
[SWS_CORE_11951]	MemoryResource error behavior
[SWS_CORE_11952]	Resolution of macro ARA_COMPILER_DEFINED_NODISCARD
[SWS_CORE_15005]	
[SWS_CORE_90021]	
[SWS_CORE_90022]	

Table E.11: Added Specification Items in R23-11

E.5.2 Changed Specification Items in R23-11

Number	Heading
[SWS_CORE_00122]	Definition of API type ara::core::ErrorDomain::CodeType
[SWS_CORE_00123]	Definition of API type ara::core::ErrorDomain::SupportDataType
[SWS_CORE_00151]	Definition of API function ara::core::ErrorDomain::Id
[SWS_CORE_00152]	Definition of API function ara::core::ErrorDomain::Name
[SWS_CORE_00154]	Definition of API function ara::core::ErrorDomain::ThrowAsException
[SWS_CORE_00326]	Definition of API function ara::core::Future::get
[SWS_CORE_00328]	Definition of API function ara::core::Future::wait
[SWS_CORE_00329]	Definition of API function ara::core::Future::wait_for
[SWS_CORE_00330]	Definition of API function ara::core::Future::wait_until
[SWS_CORE_00331]	Definition of API function ara::core::Future::then
[SWS_CORE_00332]	Definition of API function ara::core::Future::is_ready
[SWS_CORE_00333]	Definition of API function ara::core::Future::~Future
[SWS_CORE_00336]	Definition of API function ara::core::Future::GetResult
[SWS_CORE_00337]	Definition of API function ara::core::Future::then

▽

△

Number	Heading
[SWS_CORE_00343]	Definition of API function ara::core::Promise::operator=
[SWS_CORE_00344]	Definition of API function ara::core::Promise::get_future
[SWS_CORE_00345]	Definition of API function ara::core::Promise::set_value
[SWS_CORE_00346]	Definition of API function ara::core::Promise::set_value
[SWS_CORE_00349]	Definition of API function ara::core::Promise::~Promise
[SWS_CORE_00353]	Definition of API function ara::core::Promise::SetError
[SWS_CORE_00354]	Definition of API function ara::core::Promise::SetError
[SWS_CORE_00355]	Definition of API function ara::core::Promise::SetResult
[SWS_CORE_00356]	Definition of API function ara::core::Promise::SetResult
[SWS_CORE_00400]	Definition of API enum ara::core::future_errc
[SWS_CORE_00444]	Definition of API function ara::core::FutureErrorDomain::ThrowAsException
[SWS_CORE_00519]	Definition of API function ara::core::ErrorCode::ThrowAsException
[SWS_CORE_00571]	Definition of API function ara::core::operator==
[SWS_CORE_00572]	Definition of API function ara::core::operator!=
[SWS_CORE_00711]	Definition of API type ara::core::Result::value_type
[SWS_CORE_00712]	Definition of API type ara::core::Result::error_type
[SWS_CORE_00721]	Definition of API function ara::core::Result::Result
[SWS_CORE_00722]	Definition of API function ara::core::Result::Result
[SWS_CORE_00723]	Definition of API function ara::core::Result::Result
[SWS_CORE_00724]	Definition of API function ara::core::Result::Result
[SWS_CORE_00725]	Definition of API function ara::core::Result::Result
[SWS_CORE_00726]	Definition of API function ara::core::Result::Result
[SWS_CORE_00731]	Definition of API function ara::core::Result::FromValue
[SWS_CORE_00732]	Definition of API function ara::core::Result::FromValue
[SWS_CORE_00733]	Definition of API function ara::core::Result::FromValue
[SWS_CORE_00734]	Definition of API function ara::core::Result::FromError
[SWS_CORE_00735]	Definition of API function ara::core::Result::FromError
[SWS_CORE_00736]	Definition of API function ara::core::Result::FromError
[SWS_CORE_00741]	Definition of API function ara::core::Result::operator=
[SWS_CORE_00742]	Definition of API function ara::core::Result::operator=
[SWS_CORE_00743]	Definition of API function ara::core::Result::EmplaceValue
[SWS_CORE_00744]	Definition of API function ara::core::Result::EmplaceError
[SWS_CORE_00745]	Definition of API function ara::core::Result::Swap
[SWS_CORE_00751]	Definition of API function ara::core::Result::HasValue
[SWS_CORE_00752]	Definition of API function ara::core::Result::operator bool
[SWS_CORE_00753]	Definition of API function ara::core::Result::operator*
[SWS_CORE_00754]	Definition of API function ara::core::Result::operator->
[SWS_CORE_00755]	Definition of API function ara::core::Result::Value
[SWS_CORE_00756]	Definition of API function ara::core::Result::Value

▽

△

Number	Heading
[SWS_CORE_00757]	Definition of API function ara::core::Result::Error
[SWS_CORE_00758]	Definition of API function ara::core::Result::Error
[SWS_CORE_00759]	Definition of API function ara::core::Result::operator*
[SWS_CORE_00761]	Definition of API function ara::core::Result::ValueOr
[SWS_CORE_00762]	Definition of API function ara::core::Result::ValueOr
[SWS_CORE_00763]	Definition of API function ara::core::Result::ErrorOr
[SWS_CORE_00764]	Definition of API function ara::core::Result::ErrorOr
[SWS_CORE_00765]	Definition of API function ara::core::Result::CheckError
[SWS_CORE_00766]	Definition of API function ara::core::Result::ValueOrThrow
[SWS_CORE_00767]	Definition of API function ara::core::Result::Resolve
[SWS_CORE_00768]	Definition of API function ara::core::Result::Bind
[SWS_CORE_00769]	Definition of API function ara::core::Result::ValueOrThrow
[SWS_CORE_00770]	Definition of API function ara::core::Result::Ok
[SWS_CORE_00771]	Definition of API function ara::core::Result::Ok
[SWS_CORE_00772]	Definition of API function ara::core::Result::Err
[SWS_CORE_00773]	Definition of API function ara::core::Result::Err
[SWS_CORE_00853]	Definition of API function ara::core::Result< void, E >::operator*
[SWS_CORE_00855]	Definition of API function ara::core::Result< void, E >::Value
[SWS_CORE_00857]	Definition of API function ara::core::Result< void, E >::Error
[SWS_CORE_00858]	Definition of API function ara::core::Result< void, E >::Error
[SWS_CORE_01265]	Definition of API function ara::core::Array::operator[]
[SWS_CORE_01266]	Definition of API function ara::core::Array::operator[]
[SWS_CORE_05244]	Definition of API function ara::core::CoreErrorDomain::ThrowAsException
[SWS_CORE_06226]	Definition of API function ara::core::Future< void, E >::get
[SWS_CORE_06228]	Definition of API function ara::core::Future< void, E >::wait
[SWS_CORE_06229]	Definition of API function ara::core::Future< void, E >::wait_for
[SWS_CORE_06230]	Definition of API function ara::core::Future< void, E >::wait_until
[SWS_CORE_06231]	Definition of API function ara::core::Future< void, E >::then
[SWS_CORE_06232]	Definition of API function ara::core::Future< void, E >::is_ready
[SWS_CORE_06233]	Definition of API function ara::core::Future< void, E >::~Future
[SWS_CORE_06236]	Definition of API function ara::core::Future< void, E >::GetResult
[SWS_CORE_06237]	Definition of API function ara::core::Future< void, E >::then
[SWS_CORE_06343]	Definition of API function ara::core::Promise< void, E >::operator=
[SWS_CORE_06344]	Definition of API function ara::core::Promise< void, E >::get_future
[SWS_CORE_06345]	Definition of API function ara::core::Promise< void, E >::set_value
[SWS_CORE_06349]	Definition of API function ara::core::Promise< void, E >::~Promise
[SWS_CORE_06353]	Definition of API function ara::core::Promise< void, E >::SetError
[SWS_CORE_06354]	Definition of API function ara::core::Promise< void, E >::SetError
[SWS_CORE_06355]	Definition of API function ara::core::Promise< void, E >::SetResult

▽

△

Number	Heading
[SWS_CORE_06356]	Definition of API function ara::core::Promise< void, E >::SetResult
[SWS_CORE_10001]	Definition of API function ara::core::Initialize
[SWS_CORE_10002]	Definition of API function ara::core::Deinitialize
[SWS_CORE_15002]	Special ara::core types to be used independently of initialization

Table E.12: Changed Specification Items in R23-11

E.5.3 Deleted Specification Items in R23-11

Number	Heading
[SWS_CORE_08101]	
[SWS_CORE_08111]	
[SWS_CORE_08121]	
[SWS_CORE_08122]	
[SWS_CORE_08123]	
[SWS_CORE_08124]	
[SWS_CORE_08125]	
[SWS_CORE_08126]	
[SWS_CORE_08127]	
[SWS_CORE_08128]	
[SWS_CORE_08129]	
[SWS_CORE_08141]	
[SWS_CORE_08180]	
[SWS_CORE_08181]	
[SWS_CORE_08182]	
[SWS_CORE_08183]	
[SWS_CORE_08184]	
[SWS_CORE_08185]	
[SWS_CORE_08186]	
[SWS_CORE_08187]	
[SWS_CORE_08188]	
[SWS_CORE_08189]	
[SWS_CORE_08190]	
[SWS_CORE_08191]	
[SWS_CORE_08192]	
[SWS_CORE_08193]	
[SWS_CORE_08194]	

▽

△

Number	Heading
[SWS_CORE_08195]	
[SWS_CORE_08196]	
[SWS_CORE_08197]	
[SWS_CORE_08198]	
[SWS_CORE_08199]	
[SWS_CORE_15001]	Handling of interaction with the ARA of an un-/deinitialized runtime
[SWS_CORE_90020]	

Table E.13: Deleted Specification Items in R23-11

E.6 Change History of this document according to AUTOSAR Release R24-11

E.6.1 Added Specification Items in R24-11

Number	Heading
[SWS_CORE_00006]	Handling of exception-based Violations
[SWS_CORE_00007]	Handling of exception-based Failed Default Allocations
[SWS_CORE_00008]	Definition of API class ara::core::Executor
[SWS_CORE_00009]	Definition of API function ara::core::Executor::Executor
[SWS_CORE_00015]	Definition of API function ara::core::Executor::Executor
[SWS_CORE_00017]	Definition of API function ara::core::Executor::operator=
[SWS_CORE_00018]	Definition of API function ara::core::Executor::operator=
[SWS_CORE_00019]	Definition of API function ara::core::Executor::~Executor
[SWS_CORE_00024]	Definition of API function ara::core::Executor::execute
[SWS_CORE_00025]	Definition of API function ara::core::Executor::oneway_execute
[SWS_CORE_00026]	Definition of API class ara::core::PoolOptions
[SWS_CORE_00027]	Definition of API variable ara::core::PoolOptions::max_blocks_per_chunk
[SWS_CORE_00028]	Definition of API variable ara::core::PoolOptions::largest_required_pool_block
[SWS_CORE_00029]	Definition of API class ara::core::SynchronizedPoolResource
[SWS_CORE_00030]	Definition of API function ara::core::SynchronizedPoolResource::SynchronizedPoolResource
[SWS_CORE_00031]	Definition of API function ara::core::SynchronizedPoolResource::SynchronizedPoolResource
[SWS_CORE_00032]	Definition of API function ara::core::SynchronizedPoolResource::SynchronizedPoolResource

▽

△

Number	Heading
[SWS_CORE_00033]	Definition of API function ara::core::SynchronizedPoolResource::SynchronizedPoolResource
[SWS_CORE_00034]	Definition of API function ara::core::SynchronizedPoolResource::SynchronizedPoolResource
[SWS_CORE_00035]	Definition of API function ara::core::SynchronizedPoolResource::~SynchronizedPoolResource
[SWS_CORE_00036]	Definition of API function ara::core::SynchronizedPoolResource::operator=
[SWS_CORE_00037]	Definition of API function ara::core::SynchronizedPoolResource::release
[SWS_CORE_00038]	Definition of API function ara::core::SynchronizedPoolResource::upstream_resource
[SWS_CORE_00039]	Definition of API function ara::core::SynchronizedPoolResource::options
[SWS_CORE_00041]	Definition of API function ara::core::SynchronizedPoolResource::do_deallocate
[SWS_CORE_00042]	Definition of API function ara::core::SynchronizedPoolResource::do_is_equal
[SWS_CORE_00043]	Definition of API function ara::core::SynchronizedPoolResource::SynchronizedPoolResource
[SWS_CORE_00044]	Definition of API function ara::core::SynchronizedPoolResource::operator=
[SWS_CORE_00045]	Definition of API class ara::core::UnsynchronizedPoolResource
[SWS_CORE_00046]	Definition of API function ara::core::UnsynchronizedPoolResource::UnsynchronizedPoolResource
[SWS_CORE_00047]	Definition of API function ara::core::UnsynchronizedPoolResource::operator=
[SWS_CORE_00048]	Definition of API function ara::core::UnsynchronizedPoolResource::UnsynchronizedPoolResource
[SWS_CORE_00049]	Definition of API function ara::core::UnsynchronizedPoolResource::operator=
[SWS_CORE_00055]	Functional Cluster Log and Trace messages
[SWS_CORE_00056]	Definition of API function ara::core::UnsynchronizedPoolResource::upstream_resource
[SWS_CORE_00057]	Definition of API function ara::core::UnsynchronizedPoolResource::options
[SWS_CORE_00058]	Definition of API function ara::core::UnsynchronizedPoolResource::do_allocate
[SWS_CORE_00059]	Definition of API function ara::core::UnsynchronizedPoolResource::do_deallocate
[SWS_CORE_00060]	Definition of API function ara::core::UnsynchronizedPoolResource::do_is_equal
[SWS_CORE_00061]	Definition of API function ara::core::SynchronizedPoolResource::do_allocate
[SWS_CORE_00062]	Definition of API function ara::core::UnsynchronizedPoolResource::~UnsynchronizedPoolResource
[SWS_CORE_00063]	Definition of API function ara::core::UnsynchronizedPoolResource::UnsynchronizedPoolResource

▽

△

Number	Heading
[SWS_CORE_00064]	Definition of API function ara::core::UnsynchronizedPoolResource::UnsynchronizedPoolResource
[SWS_CORE_00065]	Definition of API function ara::core::UnsynchronizedPoolResource::release
[SWS_CORE_00066]	Definition of API function ara::core::PolymorphicAllocator::operator=
[SWS_CORE_00067]	Definition of API function ara::core::UnsynchronizedPoolResource::UnsynchronizedPoolResource
[SWS_CORE_00068]	Definition of API function ara::core::UnsynchronizedPoolResource::UnsynchronizedPoolResource
[SWS_CORE_00069]	ara::core::Result<T&> may hold lvalue reference types
[SWS_CORE_00070]	Assigning to ara::core::Result<T&>
[SWS_CORE_00071]	Definition of API type ara::core::BasicString::iterator
[SWS_CORE_00072]	Definition of API type ara::core::BasicString::const_iterator
[SWS_CORE_00073]	Definition of API variable ara::core::BasicString::npos
[SWS_CORE_00090]	Handling of Standardized Violations
[SWS_CORE_00091]	Messages for Violations
[SWS_CORE_00092]	Vendor-specific errors
[SWS_CORE_00093]	Vendor header file
[SWS_CORE_00777]	Definition of API function ara::core::Result::operator->
[SWS_CORE_00901]	Definition of API class ara::core::Result< T &, E >
[SWS_CORE_00902]	Definition of API type ara::core::Result< T &, E >::value_type
[SWS_CORE_00903]	Definition of API type ara::core::Result< T &, E >::error_type
[SWS_CORE_00904]	Definition of API function ara::core::Result< T &, E >::FromValue
[SWS_CORE_00905]	Definition of API function ara::core::Result< T &, E >::FromError
[SWS_CORE_00906]	Definition of API function ara::core::Result< T &, E >::FromError
[SWS_CORE_00907]	Definition of API function ara::core::Result< T &, E >::FromError
[SWS_CORE_00908]	Definition of API function ara::core::Result< T &, E >::Result
[SWS_CORE_00909]	Definition of API function ara::core::Result< T &, E >::Result
[SWS_CORE_00910]	Definition of API function ara::core::Result< T &, E >::Result
[SWS_CORE_00911]	Definition of API function ara::core::Result< T &, E >::Result
[SWS_CORE_00912]	Definition of API function ara::core::Result< T &, E >::Result
[SWS_CORE_00913]	Definition of API function ara::core::Result< T &, E >::~Result
[SWS_CORE_00914]	Definition of API function ara::core::Result< T &, E >::operator=
[SWS_CORE_00915]	Definition of API function ara::core::Result< T &, E >::operator=
[SWS_CORE_00916]	Definition of API function ara::core::Result< T &, E >::EmplaceError
[SWS_CORE_00917]	Definition of API function ara::core::Result< T &, E >::Swap
[SWS_CORE_00918]	Definition of API function ara::core::Result< T &, E >::HasValue
[SWS_CORE_00919]	Definition of API function ara::core::Result< T &, E >::operator bool
[SWS_CORE_00920]	Definition of API function ara::core::Result< T &, E >::operator*
[SWS_CORE_00921]	Definition of API function ara::core::Result< T &, E >::operator*

▽

△

Number	Heading
[SWS_CORE_00922]	Definition of API function ara::core::Result< T &, E >::operator->
[SWS_CORE_00923]	Definition of API function ara::core::Result< T &, E >::operator->
[SWS_CORE_00924]	Definition of API function ara::core::Result< T &, E >::Value
[SWS_CORE_00925]	Definition of API function ara::core::Result< T &, E >::Value
[SWS_CORE_00926]	Definition of API function ara::core::Result< T &, E >::Ok
[SWS_CORE_00927]	Definition of API function ara::core::Result< T &, E >::Ok
[SWS_CORE_00928]	Definition of API function ara::core::Result< T &, E >::Error
[SWS_CORE_00929]	Definition of API function ara::core::Result< T &, E >::Error
[SWS_CORE_00930]	Definition of API function ara::core::Result< T &, E >::Error
[SWS_CORE_00931]	Definition of API function ara::core::Result< T &, E >::Err
[SWS_CORE_00932]	Definition of API function ara::core::Result< T &, E >::Err
[SWS_CORE_00933]	Definition of API function ara::core::Result< T &, E >::ValueOr
[SWS_CORE_00934]	Definition of API function ara::core::Result< T &, E >::ValueOr
[SWS_CORE_00935]	Definition of API function ara::core::Result< T &, E >::ErrorOr
[SWS_CORE_00936]	Definition of API function ara::core::Result< T &, E >::ErrorOr
[SWS_CORE_00937]	Definition of API function ara::core::Result< T &, E >::CheckError
[SWS_CORE_00938]	Definition of API function ara::core::Result< T &, E >::ValueOrThrow
[SWS_CORE_00939]	Definition of API function ara::core::Result< T &, E >::Resolve
[SWS_CORE_00940]	Definition of API function ara::core::Result< T &, E >::Resolve
[SWS_CORE_00941]	Definition of API function ara::core::Result< T &, E >::Bind
[SWS_CORE_01032]	Optional references
[SWS_CORE_01034]	Assignment behavior of Optional references
[SWS_CORE_01100]	Definition of API class ara::core::nullopt_t
[SWS_CORE_01101]	Definition of API variable ara::core::nullopt
[SWS_CORE_01102]	Definition of API type ara::core::Optional::value_type
[SWS_CORE_01103]	Definition of API function ara::core::Optional::Optional
[SWS_CORE_01104]	Definition of API function ara::core::Optional::Optional
[SWS_CORE_01105]	Definition of API function ara::core::Optional::Optional
[SWS_CORE_01106]	Definition of API function ara::core::Optional::Optional
[SWS_CORE_01107]	Definition of API function ara::core::Optional::Optional
[SWS_CORE_01108]	Definition of API function ara::core::Optional::Optional
[SWS_CORE_01109]	Definition of API function ara::core::Optional::Optional
[SWS_CORE_01110]	Definition of API function ara::core::Optional::Optional
[SWS_CORE_01111]	Definition of API function ara::core::Optional::Optional
[SWS_CORE_01112]	Definition of API function ara::core::Optional::~Optional
[SWS_CORE_01113]	Definition of API function ara::core::Optional::operator=
[SWS_CORE_01114]	Definition of API function ara::core::Optional::operator=
[SWS_CORE_01115]	Definition of API function ara::core::Optional::operator=
[SWS_CORE_01116]	Definition of API function ara::core::Optional::operator=

▽

△

Number	Heading
[SWS_CORE_01117]	Definition of API function ara::core::Optional::operator=
[SWS_CORE_01118]	Definition of API function ara::core::Optional::operator=
[SWS_CORE_01119]	Definition of API function ara::core::Optional::emplace
[SWS_CORE_01120]	Definition of API function ara::core::Optional::emplace
[SWS_CORE_01121]	Definition of API function ara::core::Optional::swap
[SWS_CORE_01122]	Definition of API function ara::core::Optional::operator->
[SWS_CORE_01123]	Definition of API function ara::core::Optional::operator->
[SWS_CORE_01124]	Definition of API function ara::core::Optional::operator*
[SWS_CORE_01125]	Definition of API function ara::core::Optional::operator*
[SWS_CORE_01126]	Definition of API function ara::core::Optional::operator*
[SWS_CORE_01127]	Definition of API function ara::core::Optional::operator*
[SWS_CORE_01128]	Definition of API function ara::core::Optional::operator bool
[SWS_CORE_01129]	Definition of API function ara::core::Optional::has_value
[SWS_CORE_01130]	Definition of API function ara::core::Optional::value
[SWS_CORE_01131]	Definition of API function ara::core::Optional::value
[SWS_CORE_01132]	Definition of API function ara::core::Optional::value
[SWS_CORE_01133]	Definition of API function ara::core::Optional::value
[SWS_CORE_01134]	Definition of API function ara::core::Optional::value_or
[SWS_CORE_01135]	Definition of API function ara::core::Optional::value_or
[SWS_CORE_01136]	Definition of API function ara::core::Optional::reset
[SWS_CORE_01138]	Definition of API function ara::core::make_optional
[SWS_CORE_01139]	Definition of API function ara::core::make_optional
[SWS_CORE_01140]	Definition of API function ara::core::make_optional
[SWS_CORE_01150]	Definition of API class ara::core::Optional< T & >
[SWS_CORE_01151]	Definition of API type ara::core::Optional< T & >::value_type
[SWS_CORE_01152]	Definition of API function ara::core::Optional< T & >::Optional
[SWS_CORE_01153]	Definition of API function ara::core::Optional< T & >::Optional
[SWS_CORE_01154]	Definition of API function ara::core::Optional< T & >::Optional
[SWS_CORE_01155]	Definition of API function ara::core::Optional< T & >::Optional
[SWS_CORE_01156]	Definition of API function ara::core::Optional< T & >::Optional
[SWS_CORE_01157]	Definition of API function ara::core::Optional< T & >::Optional
[SWS_CORE_01158]	Definition of API function ara::core::Optional< T & >::~Optional
[SWS_CORE_01159]	Definition of API function ara::core::Optional< T & >::operator=
[SWS_CORE_01160]	Definition of API function ara::core::Optional< T & >::operator=
[SWS_CORE_01161]	Definition of API function ara::core::Optional< T & >::operator=
[SWS_CORE_01162]	Definition of API function ara::core::Optional< T & >::operator=
[SWS_CORE_01163]	Definition of API function ara::core::Optional< T & >::operator=
[SWS_CORE_01164]	Definition of API function ara::core::Optional< T & >::emplace

▽

△

Number	Heading
[SWS_CORE_01165]	Definition of API function ara::core::Optional< T & >::reset
[SWS_CORE_01166]	Definition of API function ara::core::Optional< T & >::swap
[SWS_CORE_01167]	Definition of API function ara::core::Optional< T & >::operator->
[SWS_CORE_01168]	Definition of API function ara::core::Optional< T & >::operator->
[SWS_CORE_01169]	Definition of API function ara::core::Optional< T & >::operator*
[SWS_CORE_01170]	Definition of API function ara::core::Optional< T & >::operator*
[SWS_CORE_01171]	Definition of API function ara::core::Optional< T & >::operator bool
[SWS_CORE_01172]	Definition of API function ara::core::Optional< T & >::has_value
[SWS_CORE_01173]	Definition of API function ara::core::Optional< T & >::value
[SWS_CORE_01174]	Definition of API function ara::core::Optional< T & >::value
[SWS_CORE_01175]	Definition of API function ara::core::Optional< T & >::value_or
[SWS_CORE_01600]	Definition of API class ara::core::variant_size
[SWS_CORE_01602]	Definition of API class ara::core::variant_size< const T >
[SWS_CORE_01603]	Definition of API class ara::core::variant_size< volatile T >
[SWS_CORE_01604]	Definition of API class ara::core::variant_size< const volatile T >
[SWS_CORE_01605]	Definition of API class ara::core::variant_size< Variant< Types... > >
[SWS_CORE_01606]	Definition of API variable ara::core::variant_size_v
[SWS_CORE_01607]	Definition of API class ara::core::variant_alternative
[SWS_CORE_01608]	Definition of API class ara::core::variant_alternative< I, const T >
[SWS_CORE_01609]	Definition of API class ara::core::variant_alternative< I, volatile T >
[SWS_CORE_01610]	Definition of API class ara::core::variant_alternative< I, const volatile T >
[SWS_CORE_01611]	Definition of API class ara::core::variant_alternative< I, Variant< Types... > >
[SWS_CORE_01612]	Definition of API type ara::core::variant_alternative_t
[SWS_CORE_01613]	Definition of API function ara::core::get
[SWS_CORE_01614]	Definition of API function ara::core::get
[SWS_CORE_01615]	Definition of API function ara::core::get
[SWS_CORE_01616]	Definition of API function ara::core::get
[SWS_CORE_01617]	Definition of API function ara::core::get
[SWS_CORE_01618]	Definition of API function ara::core::get
[SWS_CORE_01619]	Definition of API function ara::core::get
[SWS_CORE_01620]	Definition of API function ara::core::get
[SWS_CORE_01621]	Definition of API function ara::core::get_if
[SWS_CORE_01622]	Definition of API function ara::core::get_if
[SWS_CORE_01623]	Definition of API function ara::core::get_if
[SWS_CORE_01624]	Definition of API function ara::core::get_if
[SWS_CORE_01626]	Definition of API function ara::core::holds_alternative
[SWS_CORE_01627]	Definition of API function ara::core::operator==
[SWS_CORE_01628]	Definition of API function ara::core::operator!=

▽

△

Number	Heading
[SWS_CORE_01629]	Definition of API function ara::core::operator<
[SWS_CORE_01630]	Definition of API function ara::core::operator>
[SWS_CORE_01631]	Definition of API function ara::core::operator<=
[SWS_CORE_01632]	Definition of API function ara::core::operator>=
[SWS_CORE_01633]	Definition of API function ara::core::visit
[SWS_CORE_01634]	Definition of API function ara::core::visit
[SWS_CORE_01640]	Definition of API class ara::core::Monostate
[SWS_CORE_01641]	Definition of API function ara::core::operator==
[SWS_CORE_01642]	Definition of API function ara::core::operator!=
[SWS_CORE_01643]	Definition of API function ara::core::operator<
[SWS_CORE_01644]	Definition of API function ara::core::operator>
[SWS_CORE_01645]	Definition of API function ara::core::operator<=
[SWS_CORE_01646]	Definition of API function ara::core::operator>=
[SWS_CORE_01649]	Definition of API function ara::core::Variant::Variant
[SWS_CORE_01650]	Definition of API function ara::core::Variant::Variant
[SWS_CORE_01651]	Definition of API function ara::core::Variant::Variant
[SWS_CORE_01652]	Definition of API function ara::core::Variant::Variant
[SWS_CORE_01653]	Definition of API function ara::core::Variant::Variant
[SWS_CORE_01654]	Definition of API function ara::core::Variant::Variant
[SWS_CORE_01655]	Definition of API function ara::core::Variant::Variant
[SWS_CORE_01656]	Definition of API function ara::core::Variant::Variant
[SWS_CORE_01657]	Definition of API function ara::core::Variant::~Variant
[SWS_CORE_01658]	Definition of API function ara::core::Variant::operator=
[SWS_CORE_01659]	Definition of API function ara::core::Variant::operator=
[SWS_CORE_01660]	Definition of API function ara::core::Variant::operator=
[SWS_CORE_01661]	Definition of API function ara::core::Variant::emplace
[SWS_CORE_01662]	Definition of API function ara::core::Variant::emplace
[SWS_CORE_01663]	Definition of API function ara::core::Variant::emplace
[SWS_CORE_01664]	Definition of API function ara::core::Variant::emplace
[SWS_CORE_01665]	Definition of API function ara::core::Variant::index
[SWS_CORE_01666]	Definition of API function ara::core::Variant::valueless_by_exception
[SWS_CORE_01667]	Definition of API function ara::core::Variant::swap
[SWS_CORE_01668]	Definition of API class std::hash<::ara::core::Variant< Types... > >
[SWS_CORE_01669]	Definition of API function std::hash<::ara::core::Variant< Types... > >::operator()
[SWS_CORE_01670]	Definition of API class std::hash<::ara::core::Monostate >
[SWS_CORE_01671]	Definition of API function std::hash<::ara::core::Monostate >::operator()
[SWS_CORE_02100]	Definition of API type ara::core::StringView::value_type
[SWS_CORE_02101]	Definition of API type ara::core::StringView::pointer

▽

△

Number	Heading
[SWS_CORE_02102]	Definition of API type ara::core::StringView::const_pointer
[SWS_CORE_02103]	Definition of API type ara::core::StringView::reference
[SWS_CORE_02104]	Definition of API type ara::core::StringView::const_reference
[SWS_CORE_02105]	Definition of API type ara::core::StringView::const_iterator
[SWS_CORE_02106]	Definition of API type ara::core::StringView::iterator
[SWS_CORE_02107]	Definition of API type ara::core::StringView::const_reverse_iterator
[SWS_CORE_02108]	Definition of API type ara::core::StringView::reverse_iterator
[SWS_CORE_02109]	Definition of API type ara::core::StringView::size_type
[SWS_CORE_02110]	Definition of API type ara::core::StringView::difference_type
[SWS_CORE_02111]	Definition of API variable ara::core::StringView::npos
[SWS_CORE_02112]	Definition of API function ara::core::StringView::StringView
[SWS_CORE_02113]	Definition of API function ara::core::StringView::StringView
[SWS_CORE_02114]	Definition of API function ara::core::StringView::StringView
[SWS_CORE_02115]	Definition of API function ara::core::StringView::StringView
[SWS_CORE_02116]	Definition of API function ara::core::StringView::operator=
[SWS_CORE_02117]	Definition of API function ara::core::StringView::StringView
[SWS_CORE_02118]	Definition of API function ara::core::StringView::operator=
[SWS_CORE_02119]	Definition of API function ara::core::StringView::~StringView
[SWS_CORE_02120]	Definition of API function ara::core::StringView::begin
[SWS_CORE_02121]	Definition of API function ara::core::StringView::end
[SWS_CORE_02122]	Definition of API function ara::core::StringView::cbegin
[SWS_CORE_02123]	Definition of API function ara::core::StringView::cend
[SWS_CORE_02124]	Definition of API function ara::core::StringView::rbegin
[SWS_CORE_02125]	Definition of API function ara::core::StringView::rend
[SWS_CORE_02126]	Definition of API function ara::core::StringView::crbegin
[SWS_CORE_02127]	Definition of API function ara::core::StringView::crend
[SWS_CORE_02128]	Definition of API function ara::core::StringView::size
[SWS_CORE_02129]	Definition of API function ara::core::StringView::length
[SWS_CORE_02130]	Definition of API function ara::core::StringView::max_size
[SWS_CORE_02131]	Definition of API function ara::core::StringView::empty
[SWS_CORE_02132]	Definition of API function ara::core::StringView::operator[]
[SWS_CORE_02133]	Definition of API function ara::core::StringView::at
[SWS_CORE_02134]	Definition of API function ara::core::StringView::front
[SWS_CORE_02135]	Definition of API function ara::core::StringView::back
[SWS_CORE_02136]	Definition of API function ara::core::StringView::data
[SWS_CORE_02137]	Definition of API function ara::core::StringView::remove_prefix
[SWS_CORE_02138]	Definition of API function ara::core::StringView::remove_suffix
[SWS_CORE_02139]	Definition of API function ara::core::StringView::swap

▽

△

Number	Heading
[SWS_CORE_02140]	Definition of API function ara::core::StringView::copy
[SWS_CORE_02141]	Definition of API function ara::core::StringView::substr
[SWS_CORE_02142]	Definition of API function ara::core::StringView::compare
[SWS_CORE_02143]	Definition of API function ara::core::StringView::compare
[SWS_CORE_02144]	Definition of API function ara::core::StringView::compare
[SWS_CORE_02145]	Definition of API function ara::core::StringView::compare
[SWS_CORE_02146]	Definition of API function ara::core::StringView::compare
[SWS_CORE_02147]	Definition of API function ara::core::StringView::compare
[SWS_CORE_02148]	Definition of API function ara::core::StringView::starts_with
[SWS_CORE_02149]	Definition of API function ara::core::StringView::starts_with
[SWS_CORE_02150]	Definition of API function ara::core::StringView::starts_with
[SWS_CORE_02151]	Definition of API function ara::core::StringView::ends_with
[SWS_CORE_02152]	Definition of API function ara::core::StringView::ends_with
[SWS_CORE_02153]	Definition of API function ara::core::StringView::ends_with
[SWS_CORE_02154]	Definition of API function ara::core::StringView::contains
[SWS_CORE_02155]	Definition of API function ara::core::StringView::contains
[SWS_CORE_02156]	Definition of API function ara::core::StringView::contains
[SWS_CORE_02157]	Definition of API function ara::core::StringView::find
[SWS_CORE_02158]	Definition of API function ara::core::StringView::find
[SWS_CORE_02159]	Definition of API function ara::core::StringView::find
[SWS_CORE_02160]	Definition of API function ara::core::StringView::find
[SWS_CORE_02161]	Definition of API function ara::core::StringView::rfind
[SWS_CORE_02162]	Definition of API function ara::core::StringView::rfind
[SWS_CORE_02163]	Definition of API function ara::core::StringView::rfind
[SWS_CORE_02164]	Definition of API function ara::core::StringView::rfind
[SWS_CORE_02165]	Definition of API function ara::core::StringView::find_first_of
[SWS_CORE_02166]	Definition of API function ara::core::StringView::find_first_of
[SWS_CORE_02167]	Definition of API function ara::core::StringView::find_first_of
[SWS_CORE_02168]	Definition of API function ara::core::StringView::find_first_of
[SWS_CORE_02169]	Definition of API function ara::core::StringView::find_last_of
[SWS_CORE_02170]	Definition of API function ara::core::StringView::find_last_of
[SWS_CORE_02171]	Definition of API function ara::core::StringView::find_last_of
[SWS_CORE_02172]	Definition of API function ara::core::StringView::find_last_of
[SWS_CORE_02173]	Definition of API function ara::core::StringView::find_first_not_of
[SWS_CORE_02174]	Definition of API function ara::core::StringView::find_first_not_of
[SWS_CORE_02175]	Definition of API function ara::core::StringView::find_first_not_of
[SWS_CORE_02176]	Definition of API function ara::core::StringView::find_first_not_of
[SWS_CORE_02177]	Definition of API function ara::core::StringView::find_last_not_of

▽

△

Number	Heading
[SWS_CORE_02178]	Definition of API function ara::core::StringView::find_last_not_of
[SWS_CORE_02179]	Definition of API function ara::core::StringView::find_last_not_of
[SWS_CORE_02180]	Definition of API function ara::core::StringView::find_last_not_of
[SWS_CORE_02181]	Definition of API function ara::core::operator==
[SWS_CORE_02182]	Definition of API function ara::core::operator!=
[SWS_CORE_02183]	Definition of API function ara::core::operator<
[SWS_CORE_02184]	Definition of API function ara::core::operator>
[SWS_CORE_02185]	Definition of API function ara::core::operator<=
[SWS_CORE_02186]	Definition of API function ara::core::operator>=
[SWS_CORE_02187]	Definition of API function ara::core::operator<<
[SWS_CORE_02188]	Definition of API function ara::core::literals::operator""_SV
[SWS_CORE_02189]	Definition of API class std::hash< ara::core::StringView >
[SWS_CORE_02190]	Definition of API function std::hash< ara::core::StringView >::operator()
[SWS_CORE_06566]	Definition of API function ara::core::MemoryResource::do_allocate
[SWS_CORE_06567]	Definition of API function ara::core::MemoryResource::do_deallocate
[SWS_CORE_06568]	Definition of API function ara::core::MemoryResource::do_is_equal
[SWS_CORE_06569]	Definition of API function ara::core::MonotonicBufferResource::do_allocate
[SWS_CORE_06570]	Definition of API function ara::core::MonotonicBufferResource::do_deallocate
[SWS_CORE_06571]	Definition of API function ara::core::MonotonicBufferResource::do_is_equal
[SWS_CORE_06572]	Definition of API type ara::core::PolymorphicAllocator::value_type
[SWS_CORE_06573]	Definition of API function ara::core::PolymorphicAllocator::select_on_container_copy_construction
[SWS_CORE_06574]	Definition of API function ara::core::MemoryResource::MemoryResource
[SWS_CORE_06575]	Definition of API function ara::core::MemoryResource::operator=
[SWS_CORE_06576]	Definition of API function ara::core::MonotonicBufferResource::MonotonicBufferResource
[SWS_CORE_06577]	Definition of API function ara::core::MonotonicBufferResource::operator=
[SWS_CORE_10003]	Definition of API function ara::core::Initialize
[SWS_CORE_10304]	Availability of `ara::core::ErrorDomain::ThrowAsException` and overriding functions
[SWS_CORE_12408]	DLT Logging of `Explicitly aborting an Operation`
[SWS_CORE_12409]	Usage of ARA API within AbortHandlers
[SWS_CORE_13018]	LogMessage FailedDefaultAllocation
[SWS_CORE_13019]	LogMessage AbortMessage
[SWS_CORE_13200]	AUTOSAR definition of a thread-safe member function
[SWS_CORE_13201]	AUTOSAR definition of multiple thread-safe member functions
[SWS_CORE_13202]	AUTOSAR definition of a thread-safe non-member function
[SWS_CORE_13203]	Behavior of a concurrently executed non-thread-safe member function
[SWS_CORE_13204]	Behavior of a concurrently executed non-thread-safe non-member function

▽

△

Number	Heading
[SWS_CORE_13205]	Behavior of concurrently executed functions on different objects
[SWS_CORE_13206]	AUTOSAR callable definition
[SWS_CORE_13207]	Behavior of thread-safe callable
[SWS_CORE_13208]	Behavior of non-thread-safe callable
[SWS_CORE_13209]	Behavior of conditionally thread-safe callable
[SWS_CORE_13210]	ARA API usage within a signal handler
[SWS_CORE_15006]	Command line argument injection in ara::core::Initialize
[SWS_CORE_90007]	Potentially throwing constructors
[SWS_CORE_90023]	AUTOSAR-standardized Functional Cluster Error Domain Identifiers
[SWS_CORE_90024]	AUTOSAR-standardized Functional Cluster Log and Trace settings
[SWS_CORE_90025]	AUTOSAR-standardized Functional Cluster names
[SWS_CORE_90026]	Framework initialization
[SWS_CORE_90027]	Clean deinitialization
[SWS_CORE_90028]	Re-try of initialization
[SWS_CORE_90029]	Double initialization
[SWS_CORE_90030]	Double de-initialization

Table E.14: Added Specification Items in R24-11

E.6.2 Changed Specification Items in R24-11

Number	Heading
[SWS_CORE_00003]	Handling of Non-Standardized Violations
[SWS_CORE_00005]	Handling of failed default allocations
[SWS_CORE_00135]	Definition of API function ara::core::ErrorDomain::ErrorDomain
[SWS_CORE_00136]	Definition of API function ara::core::ErrorDomain::~ErrorDomain
[SWS_CORE_00137]	Definition of API function ara::core::ErrorDomain::operator==
[SWS_CORE_00138]	Definition of API function ara::core::ErrorDomain::operator!=
[SWS_CORE_00151]	Definition of API function ara::core::ErrorDomain::Id
[SWS_CORE_00152]	Definition of API function ara::core::ErrorDomain::Name
[SWS_CORE_00153]	Definition of API function ara::core::ErrorDomain::Message
[SWS_CORE_00154]	Definition of API function ara::core::ErrorDomain::ThrowAsException
[SWS_CORE_00321]	Definition of API class ara::core::Future
[SWS_CORE_00322]	Definition of API function ara::core::Future::Future
[SWS_CORE_00323]	Definition of API function ara::core::Future::Future
[SWS_CORE_00325]	Definition of API function ara::core::Future::operator=
[SWS_CORE_00326]	Definition of API function ara::core::Future::get

▽

△

Number	Heading
[SWS_CORE_00327]	Definition of API function ara::core::Future::valid
[SWS_CORE_00328]	Definition of API function ara::core::Future::wait
[SWS_CORE_00329]	Definition of API function ara::core::Future::wait_for
[SWS_CORE_00330]	Definition of API function ara::core::Future::wait_until
[SWS_CORE_00331]	Definition of API function ara::core::Future::then
[SWS_CORE_00332]	Definition of API function ara::core::Future::is_ready
[SWS_CORE_00333]	Definition of API function ara::core::Future::~Future
[SWS_CORE_00336]	Definition of API function ara::core::Future::GetResult
[SWS_CORE_00337]	Definition of API function ara::core::Future::then
[SWS_CORE_00340]	Definition of API class ara::core::Promise
[SWS_CORE_00341]	Definition of API function ara::core::Promise::Promise
[SWS_CORE_00342]	Definition of API function ara::core::Promise::Promise
[SWS_CORE_00343]	Definition of API function ara::core::Promise::operator=
[SWS_CORE_00344]	Definition of API function ara::core::Promise::get_future
[SWS_CORE_00345]	Definition of API function ara::core::Promise::set_value
[SWS_CORE_00346]	Definition of API function ara::core::Promise::set_value
[SWS_CORE_00349]	Definition of API function ara::core::Promise::~Promise
[SWS_CORE_00352]	Definition of API function ara::core::Promise::swap
[SWS_CORE_00353]	Definition of API function ara::core::Promise::SetError
[SWS_CORE_00354]	Definition of API function ara::core::Promise::SetError
[SWS_CORE_00355]	Definition of API function ara::core::Promise::SetResult
[SWS_CORE_00356]	Definition of API function ara::core::Promise::SetResult
[SWS_CORE_00361]	Definition of API enum ara::core::FutureStatus
[SWS_CORE_00400]	Definition of API enum ara::core::FutureErrc
[SWS_CORE_00412]	Definition of API function ara::core::FutureException::FutureException
[SWS_CORE_00421]	Definition of API class ara::core::FutureErrorDomain
[SWS_CORE_00431]	Definition of API type ara::core::FutureErrorDomain::Errc
[SWS_CORE_00441]	Definition of API function ara::core::FutureErrorDomain::FutureErrorDomain
[SWS_CORE_00442]	Definition of API function ara::core::FutureErrorDomain::Name
[SWS_CORE_00443]	Definition of API function ara::core::FutureErrorDomain::Message
[SWS_CORE_00444]	Definition of API function ara::core::FutureErrorDomain::ThrowAsException
[SWS_CORE_00480]	Definition of API function ara::core::GetFutureErrorDomain
[SWS_CORE_00490]	Definition of API function ara::core::MakeErrorCode
[SWS_CORE_00512]	Definition of API function ara::core::ErrorCode::ErrorCode
[SWS_CORE_00513]	Definition of API function ara::core::ErrorCode::ErrorCode
[SWS_CORE_00514]	Definition of API function ara::core::ErrorCode::Value
[SWS_CORE_00515]	Definition of API function ara::core::ErrorCode::Domain
[SWS_CORE_00516]	Definition of API function ara::core::ErrorCode::SupportData
[SWS_CORE_00518]	Definition of API function ara::core::ErrorCode::Message

▽

△

Number	Heading
[SWS_CORE_00519]	Definition of API function ara::core::ErrorCode::ThrowAsException
[SWS_CORE_00571]	Definition of API function ara::core::operator==
[SWS_CORE_00572]	Definition of API function ara::core::operator!=
[SWS_CORE_00611]	Definition of API function ara::core::Exception::Exception
[SWS_CORE_00612]	Definition of API function ara::core::Exception::what
[SWS_CORE_00613]	Definition of API function ara::core::Exception::Error
[SWS_CORE_00614]	Definition of API function ara::core::Exception::operator=
[SWS_CORE_00615]	Definition of API function ara::core::Exception::Exception
[SWS_CORE_00616]	Definition of API function ara::core::Exception::operator=
[SWS_CORE_00617]	Definition of API function ara::core::Exception::~Exception
[SWS_CORE_00618]	Definition of API function ara::core::Exception::Exception
[SWS_CORE_00701]	Definition of API class ara::core::Result
[SWS_CORE_00721]	Definition of API function ara::core::Result::Result
[SWS_CORE_00722]	Definition of API function ara::core::Result::Result
[SWS_CORE_00723]	Definition of API function ara::core::Result::Result
[SWS_CORE_00724]	Definition of API function ara::core::Result::Result
[SWS_CORE_00725]	Definition of API function ara::core::Result::Result
[SWS_CORE_00726]	Definition of API function ara::core::Result::Result
[SWS_CORE_00727]	Definition of API function ara::core::Result::~Result
[SWS_CORE_00731]	Definition of API function ara::core::Result::FromValue
[SWS_CORE_00732]	Definition of API function ara::core::Result::FromValue
[SWS_CORE_00733]	Definition of API function ara::core::Result::FromValue
[SWS_CORE_00734]	Definition of API function ara::core::Result::FromError
[SWS_CORE_00735]	Definition of API function ara::core::Result::FromError
[SWS_CORE_00736]	Definition of API function ara::core::Result::FromError
[SWS_CORE_00741]	Definition of API function ara::core::Result::operator=
[SWS_CORE_00742]	Definition of API function ara::core::Result::operator=
[SWS_CORE_00743]	Definition of API function ara::core::Result::EmplaceValue
[SWS_CORE_00744]	Definition of API function ara::core::Result::EmplaceError
[SWS_CORE_00745]	Definition of API function ara::core::Result::Swap
[SWS_CORE_00751]	Definition of API function ara::core::Result::HasValue
[SWS_CORE_00752]	Definition of API function ara::core::Result::operator bool
[SWS_CORE_00753]	Definition of API function ara::core::Result::operator*
[SWS_CORE_00754]	Definition of API function ara::core::Result::operator->
[SWS_CORE_00755]	Definition of API function ara::core::Result::Value
[SWS_CORE_00756]	Definition of API function ara::core::Result::Value
[SWS_CORE_00757]	Definition of API function ara::core::Result::Error
[SWS_CORE_00758]	Definition of API function ara::core::Result::Error
[SWS_CORE_00759]	Definition of API function ara::core::Result::operator*

▽

△

Number	Heading
[SWS_CORE_00761]	Definition of API function ara::core::Result::ValueOr
[SWS_CORE_00762]	Definition of API function ara::core::Result::ValueOr
[SWS_CORE_00763]	Definition of API function ara::core::Result::ErrorOr
[SWS_CORE_00764]	Definition of API function ara::core::Result::ErrorOr
[SWS_CORE_00765]	Definition of API function ara::core::Result::CheckError
[SWS_CORE_00766]	Definition of API function ara::core::Result::ValueOrThrow
[SWS_CORE_00767]	Definition of API function ara::core::Result::Resolve
[SWS_CORE_00768]	Definition of API function ara::core::Result::Bind
[SWS_CORE_00769]	Definition of API function ara::core::Result::ValueOrThrow
[SWS_CORE_00770]	Definition of API function ara::core::Result::Ok
[SWS_CORE_00771]	Definition of API function ara::core::Result::Ok
[SWS_CORE_00772]	Definition of API function ara::core::Result::Err
[SWS_CORE_00773]	Definition of API function ara::core::Result::Err
[SWS_CORE_00774]	Definition of API function ara::core::Result::operator*
[SWS_CORE_00775]	Definition of API function ara::core::Result::Value
[SWS_CORE_00776]	Definition of API function ara::core::Result::Error
[SWS_CORE_00780]	Definition of API function ara::core::operator==
[SWS_CORE_00781]	Definition of API function ara::core::operator!=
[SWS_CORE_00782]	Definition of API function ara::core::operator==
[SWS_CORE_00783]	Definition of API function ara::core::operator==
[SWS_CORE_00784]	Definition of API function ara::core::operator!=
[SWS_CORE_00785]	Definition of API function ara::core::operator!=
[SWS_CORE_00786]	Definition of API function ara::core::operator==
[SWS_CORE_00787]	Definition of API function ara::core::operator==
[SWS_CORE_00788]	Definition of API function ara::core::operator!=
[SWS_CORE_00789]	Definition of API function ara::core::operator!=
[SWS_CORE_00796]	Definition of API function ara::core::swap
[SWS_CORE_00801]	Definition of API class ara::core::Result< void, E >
[SWS_CORE_00821]	Definition of API function ara::core::Result< void, E >::Result
[SWS_CORE_00823]	Definition of API function ara::core::Result< void, E >::Result
[SWS_CORE_00824]	Definition of API function ara::core::Result< void, E >::Result
[SWS_CORE_00825]	Definition of API function ara::core::Result< void, E >::Result
[SWS_CORE_00826]	Definition of API function ara::core::Result< void, E >::Result
[SWS_CORE_00827]	Definition of API function ara::core::Result< void, E >::~Result
[SWS_CORE_00831]	Definition of API function ara::core::Result< void, E >::FromValue
[SWS_CORE_00834]	Definition of API function ara::core::Result< void, E >::FromError
[SWS_CORE_00835]	Definition of API function ara::core::Result< void, E >::FromError
[SWS_CORE_00836]	Definition of API function ara::core::Result< void, E >::FromError
[SWS_CORE_00841]	Definition of API function ara::core::Result< void, E >::operator=

▽

△

Number	Heading
[SWS_CORE_00842]	Definition of API function ara::core::Result< void, E >::operator=
[SWS_CORE_00843]	Definition of API function ara::core::Result< void, E >::EmplaceValue
[SWS_CORE_00844]	Definition of API function ara::core::Result< void, E >::EmplaceError
[SWS_CORE_00845]	Definition of API function ara::core::Result< void, E >::Swap
[SWS_CORE_00851]	Definition of API function ara::core::Result< void, E >::HasValue
[SWS_CORE_00852]	Definition of API function ara::core::Result< void, E >::operator bool
[SWS_CORE_00853]	Definition of API function ara::core::Result< void, E >::operator*
[SWS_CORE_00855]	Definition of API function ara::core::Result< void, E >::Value
[SWS_CORE_00857]	Definition of API function ara::core::Result< void, E >::Error
[SWS_CORE_00858]	Definition of API function ara::core::Result< void, E >::Error
[SWS_CORE_00861]	Definition of API function ara::core::Result< void, E >::ValueOr
[SWS_CORE_00863]	Definition of API function ara::core::Result< void, E >::ErrorOr
[SWS_CORE_00864]	Definition of API function ara::core::Result< void, E >::ErrorOr
[SWS_CORE_00865]	Definition of API function ara::core::Result< void, E >::CheckError
[SWS_CORE_00866]	Definition of API function ara::core::Result< void, E >::ValueOrThrow
[SWS_CORE_00867]	Definition of API function ara::core::Result< void, E >::Resolve
[SWS_CORE_00868]	Definition of API function ara::core::Result< void, E >::Err
[SWS_CORE_00869]	Definition of API function ara::core::Result< void, E >::Err
[SWS_CORE_00870]	Definition of API function ara::core::Result< void, E >::Bind
[SWS_CORE_00876]	Definition of API function ara::core::Result< void, E >::Error
[SWS_CORE_01096]	Definition of API function ara::core::swap
[SWS_CORE_01241]	Definition of API function ara::core::Array::fill
[SWS_CORE_01242]	Definition of API function ara::core::Array::swap
[SWS_CORE_01250]	Definition of API function ara::core::Array::begin
[SWS_CORE_01251]	Definition of API function ara::core::Array::begin
[SWS_CORE_01252]	Definition of API function ara::core::Array::end
[SWS_CORE_01253]	Definition of API function ara::core::Array::end
[SWS_CORE_01254]	Definition of API function ara::core::Array::rbegin
[SWS_CORE_01255]	Definition of API function ara::core::Array::rbegin
[SWS_CORE_01256]	Definition of API function ara::core::Array::rend
[SWS_CORE_01257]	Definition of API function ara::core::Array::rend
[SWS_CORE_01258]	Definition of API function ara::core::Array::cbegin
[SWS_CORE_01259]	Definition of API function ara::core::Array::cend
[SWS_CORE_01260]	Definition of API function ara::core::Array::crbegin
[SWS_CORE_01261]	Definition of API function ara::core::Array::crend
[SWS_CORE_01262]	Definition of API function ara::core::Array::size
[SWS_CORE_01263]	Definition of API function ara::core::Array::max_size
[SWS_CORE_01264]	Definition of API function ara::core::Array::empty
[SWS_CORE_01265]	Definition of API function ara::core::Array::operator[]

▽

△

Number	Heading
[SWS_CORE_01266]	Definition of API function ara::core::Array::operator[]
[SWS_CORE_01267]	Definition of API function ara::core::Array::front
[SWS_CORE_01268]	Definition of API function ara::core::Array::front
[SWS_CORE_01269]	Definition of API function ara::core::Array::back
[SWS_CORE_01270]	Definition of API function ara::core::Array::back
[SWS_CORE_01271]	Definition of API function ara::core::Array::data
[SWS_CORE_01272]	Definition of API function ara::core::Array::data
[SWS_CORE_01273]	Definition of API function ara::core::Array::at
[SWS_CORE_01274]	Definition of API function ara::core::Array::at
[SWS_CORE_01282]	Definition of API function ara::core::get
[SWS_CORE_01283]	Definition of API function ara::core::get
[SWS_CORE_01284]	Definition of API function ara::core::get
[SWS_CORE_01290]	Definition of API function ara::core::operator==
[SWS_CORE_01291]	Definition of API function ara::core::operator!=
[SWS_CORE_01292]	Definition of API function ara::core::operator<
[SWS_CORE_01293]	Definition of API function ara::core::operator>
[SWS_CORE_01294]	Definition of API function ara::core::operator<=
[SWS_CORE_01295]	Definition of API function ara::core::operator>=
[SWS_CORE_01296]	Definition of API function ara::core::swap
[SWS_CORE_01390]	Definition of API function ara::core::operator==
[SWS_CORE_01391]	Definition of API function ara::core::operator!=
[SWS_CORE_01392]	Definition of API function ara::core::operator<
[SWS_CORE_01393]	Definition of API function ara::core::operator<=
[SWS_CORE_01394]	Definition of API function ara::core::operator>
[SWS_CORE_01395]	Definition of API function ara::core::operator>=
[SWS_CORE_01396]	Definition of API function ara::core::swap
[SWS_CORE_01496]	Definition of API function ara::core::swap
[SWS_CORE_01601]	Definition of API class ara::core::Variant
[SWS_CORE_01696]	Definition of API function ara::core::swap
[SWS_CORE_01941]	Definition of API function ara::core::Span::Span
[SWS_CORE_01942]	Definition of API function ara::core::Span::Span
[SWS_CORE_01943]	Definition of API function ara::core::Span::Span
[SWS_CORE_01944]	Definition of API function ara::core::Span::Span
[SWS_CORE_01945]	Definition of API function ara::core::Span::Span
[SWS_CORE_01946]	Definition of API function ara::core::Span::Span
[SWS_CORE_01947]	Definition of API function ara::core::Span::Span
[SWS_CORE_01948]	Definition of API function ara::core::Span::Span
[SWS_CORE_01949]	Definition of API function ara::core::Span::Span

▽

△

Number	Heading
[SWS_CORE_01950]	Definition of API function ara::core::Span::Span
[SWS_CORE_01951]	Definition of API function ara::core::Span::~Span
[SWS_CORE_01952]	Definition of API function ara::core::Span::operator=
[SWS_CORE_01953]	Definition of API function ara::core::Span::Span
[SWS_CORE_01954]	Definition of API function ara::core::Span::Span
[SWS_CORE_01959]	Definition of API function ara::core::Span::front
[SWS_CORE_01960]	Definition of API function ara::core::Span::back
[SWS_CORE_01961]	Definition of API function ara::core::Span::first
[SWS_CORE_01962]	Definition of API function ara::core::Span::first
[SWS_CORE_01963]	Definition of API function ara::core::Span::last
[SWS_CORE_01964]	Definition of API function ara::core::Span::last
[SWS_CORE_01965]	Definition of API function ara::core::Span::subspan
[SWS_CORE_01966]	Definition of API function ara::core::Span::subspan
[SWS_CORE_01967]	Definition of API function ara::core::Span::size
[SWS_CORE_01968]	Definition of API function ara::core::Span::size_bytes
[SWS_CORE_01969]	Definition of API function ara::core::Span::empty
[SWS_CORE_01970]	Definition of API function ara::core::Span::operator[]
[SWS_CORE_01971]	Definition of API function ara::core::Span::data
[SWS_CORE_01972]	Definition of API function ara::core::Span::begin
[SWS_CORE_01973]	Definition of API function ara::core::Span::end
[SWS_CORE_01974]	Definition of API function ara::core::Span::cbegin
[SWS_CORE_01975]	Definition of API function ara::core::Span::cend
[SWS_CORE_01976]	Definition of API function ara::core::Span::rbegin
[SWS_CORE_01977]	Definition of API function ara::core::Span::rend
[SWS_CORE_01978]	Definition of API function ara::core::Span::crbegin
[SWS_CORE_01979]	Definition of API function ara::core::Span::crend
[SWS_CORE_01980]	Definition of API function ara::core::as_bytes
[SWS_CORE_01981]	Definition of API function ara::core::as_writable_bytes
[SWS_CORE_01990]	Definition of API function ara::core::MakeSpan
[SWS_CORE_01991]	Definition of API function ara::core::MakeSpan
[SWS_CORE_01992]	Definition of API function ara::core::MakeSpan
[SWS_CORE_01993]	Definition of API function ara::core::MakeSpan
[SWS_CORE_01994]	Definition of API function ara::core::MakeSpan
[SWS_CORE_02001]	Definition of API class ara::core::StringView
[SWS_CORE_04012]	Definition of API function ara::core::in_place_t::in_place_t
[SWS_CORE_04022]	Definition of API function ara::core::in_place_type_t::in_place_type_t
[SWS_CORE_04032]	Definition of API function ara::core::in_place_index_t::in_place_index_t
[SWS_CORE_04110]	Definition of API function ara::core::data

▽

△

Number	Heading
[SWS_CORE_04111]	Definition of API function ara::core::data
[SWS_CORE_04112]	Definition of API function ara::core::data
[SWS_CORE_04113]	Definition of API function ara::core::data
[SWS_CORE_04120]	Definition of API function ara::core::size
[SWS_CORE_04121]	Definition of API function ara::core::size
[SWS_CORE_04130]	Definition of API function ara::core::empty
[SWS_CORE_04131]	Definition of API function ara::core::empty
[SWS_CORE_04132]	Definition of API function ara::core::empty
[SWS_CORE_04200]	Definition of API type ara::core::Byte
[SWS_CORE_05200]	Definition of API enum ara::core::CoreErrc
[SWS_CORE_05212]	Definition of API function ara::core::CoreException::CoreException
[SWS_CORE_05221]	Definition of API class ara::core::CoreErrorDomain
[SWS_CORE_05241]	Definition of API function ara::core::CoreErrorDomain::CoreErrorDomain
[SWS_CORE_05242]	Definition of API function ara::core::CoreErrorDomain::Name
[SWS_CORE_05243]	Definition of API function ara::core::CoreErrorDomain::Message
[SWS_CORE_05244]	Definition of API function ara::core::CoreErrorDomain::ThrowAsException
[SWS_CORE_05280]	Definition of API function ara::core::GetCoreErrorDomain
[SWS_CORE_05290]	Definition of API function ara::core::MakeErrorCode
[SWS_CORE_06221]	Definition of API class ara::core::Future< void, E >
[SWS_CORE_06222]	Definition of API function ara::core::Future< void, E >::Future
[SWS_CORE_06223]	Definition of API function ara::core::Future< void, E >::Future
[SWS_CORE_06225]	Definition of API function ara::core::Future< void, E >::operator=
[SWS_CORE_06226]	Definition of API function ara::core::Future< void, E >::get
[SWS_CORE_06227]	Definition of API function ara::core::Future< void, E >::valid
[SWS_CORE_06228]	Definition of API function ara::core::Future< void, E >::wait
[SWS_CORE_06229]	Definition of API function ara::core::Future< void, E >::wait_for
[SWS_CORE_06230]	Definition of API function ara::core::Future< void, E >::wait_until
[SWS_CORE_06231]	Definition of API function ara::core::Future< void, E >::then
[SWS_CORE_06232]	Definition of API function ara::core::Future< void, E >::is_ready
[SWS_CORE_06233]	Definition of API function ara::core::Future< void, E >::~Future
[SWS_CORE_06236]	Definition of API function ara::core::Future< void, E >::GetResult
[SWS_CORE_06237]	Definition of API function ara::core::Future< void, E >::then
[SWS_CORE_06340]	Definition of API class ara::core::Promise< void, E >
[SWS_CORE_06341]	Definition of API function ara::core::Promise< void, E >::Promise
[SWS_CORE_06342]	Definition of API function ara::core::Promise< void, E >::Promise
[SWS_CORE_06343]	Definition of API function ara::core::Promise< void, E >::operator=
[SWS_CORE_06344]	Definition of API function ara::core::Promise< void, E >::get_future
[SWS_CORE_06345]	Definition of API function ara::core::Promise< void, E >::set_value
[SWS_CORE_06349]	Definition of API function ara::core::Promise< void, E >::~Promise

▽

△

Number	Heading
[SWS_CORE_06352]	Definition of API function ara::core::Promise< void, E >::swap
[SWS_CORE_06353]	Definition of API function ara::core::Promise< void, E >::SetError
[SWS_CORE_06354]	Definition of API function ara::core::Promise< void, E >::SetError
[SWS_CORE_06355]	Definition of API function ara::core::Promise< void, E >::SetResult
[SWS_CORE_06356]	Definition of API function ara::core::Promise< void, E >::SetResult
[SWS_CORE_06432]	Definition of API function ara::core::SteadyClock::now
[SWS_CORE_06500]	Definition of API class ara::core::MemoryResource
[SWS_CORE_06501]	Definition of API function ara::core::MemoryResource::MemoryResource
[SWS_CORE_06502]	Definition of API function ara::core::MemoryResource::MemoryResource
[SWS_CORE_06503]	Definition of API function ara::core::MemoryResource::allocate
[SWS_CORE_06504]	Definition of API function ara::core::MemoryResource::deallocate
[SWS_CORE_06505]	Definition of API function ara::core::MemoryResource::is_equal
[SWS_CORE_06506]	Definition of API function ara::core::MemoryResource::~MemoryResource
[SWS_CORE_06507]	Definition of API function ara::core::MemoryResource::operator=
[SWS_CORE_06521]	Definition of API function ara::core::MonotonicBufferResource::Monotonic BufferResource
[SWS_CORE_06522]	Definition of API function ara::core::MonotonicBufferResource::Monotonic BufferResource
[SWS_CORE_06523]	Definition of API function ara::core::MonotonicBufferResource::Monotonic BufferResource
[SWS_CORE_06524]	Definition of API function ara::core::MonotonicBufferResource::Monotonic BufferResource
[SWS_CORE_06525]	Definition of API function ara::core::MonotonicBufferResource::Monotonic BufferResource
[SWS_CORE_06526]	Definition of API function ara::core::MonotonicBufferResource::Monotonic BufferResource
[SWS_CORE_06528]	Definition of API function ara::core::MonotonicBufferResource::~Monotonic BufferResource
[SWS_CORE_06530]	Definition of API function ara::core::MonotonicBufferResource::release
[SWS_CORE_06531]	Definition of API function ara::core::MonotonicBufferResource::upstream_resource
[SWS_CORE_06541]	Definition of API function ara::core::PolymorphicAllocator::Polymorphic Allocator
[SWS_CORE_06542]	Definition of API function ara::core::PolymorphicAllocator::Polymorphic Allocator
[SWS_CORE_06543]	Definition of API function ara::core::PolymorphicAllocator::Polymorphic Allocator
[SWS_CORE_06544]	Definition of API function ara::core::PolymorphicAllocator::Polymorphic Allocator
[SWS_CORE_06546]	Definition of API function ara::core::PolymorphicAllocator::Polymorphic Allocator
[SWS_CORE_06547]	Definition of API function ara::core::PolymorphicAllocator::allocate
[SWS_CORE_06548]	Definition of API function ara::core::PolymorphicAllocator::deallocate

▽

△

Number	Heading
[SWS_CORE_06549]	Definition of API function ara::core::PolymorphicAllocator::allocate_bytes
[SWS_CORE_06550]	Definition of API function ara::core::PolymorphicAllocator::deallocate_bytes
[SWS_CORE_06551]	Definition of API function ara::core::PolymorphicAllocator::allocate_object
[SWS_CORE_06552]	Definition of API function ara::core::PolymorphicAllocator::deallocate_object
[SWS_CORE_06553]	Definition of API function ara::core::PolymorphicAllocator::new_object
[SWS_CORE_06554]	Definition of API function ara::core::PolymorphicAllocator::delete_object
[SWS_CORE_06555]	Definition of API function ara::core::PolymorphicAllocator::construct
[SWS_CORE_06556]	Definition of API function ara::core::PolymorphicAllocator::destroy
[SWS_CORE_06557]	Definition of API function ara::core::PolymorphicAllocator::resource
[SWS_CORE_06560]	Definition of API function ara::core::operator==
[SWS_CORE_06561]	Definition of API function ara::core::operator==
[SWS_CORE_06562]	Definition of API function ara::core::NewDeleteResource
[SWS_CORE_06563]	Definition of API function ara::core::NullMemoryResource
[SWS_CORE_06564]	Definition of API function ara::core::SetDefaultResource
[SWS_CORE_06565]	Definition of API function ara::core::GetDefaultResource
[SWS_CORE_08021]	Definition of API function ara::core::InstanceSpecifier::InstanceSpecifier
[SWS_CORE_08022]	Definition of API function ara::core::InstanceSpecifier::InstanceSpecifier
[SWS_CORE_08023]	Definition of API function ara::core::InstanceSpecifier::InstanceSpecifier
[SWS_CORE_08024]	Definition of API function ara::core::InstanceSpecifier::operator=
[SWS_CORE_08025]	Definition of API function ara::core::InstanceSpecifier::operator=
[SWS_CORE_08029]	Definition of API function ara::core::InstanceSpecifier::~InstanceSpecifier
[SWS_CORE_08032]	Definition of API function ara::core::InstanceSpecifier::Create
[SWS_CORE_08041]	Definition of API function ara::core::InstanceSpecifier::ToString
[SWS_CORE_08042]	Definition of API function ara::core::InstanceSpecifier::operator==
[SWS_CORE_08043]	Definition of API function ara::core::InstanceSpecifier::operator==
[SWS_CORE_08044]	Definition of API function ara::core::InstanceSpecifier::operator!=
[SWS_CORE_08045]	Definition of API function ara::core::InstanceSpecifier::operator!=
[SWS_CORE_08046]	Definition of API function ara::core::InstanceSpecifier::operator<
[SWS_CORE_08081]	Definition of API function ara::core::operator==
[SWS_CORE_08082]	Definition of API function ara::core::operator!=
[SWS_CORE_10001]	Definition of API function ara::core::Initialize
[SWS_CORE_10002]	Definition of API function ara::core::Deinitialize
[SWS_CORE_10200]	Valid InstanceSpecifier representations - application interaction
[SWS_CORE_10203]	Valid InstanceSpecifier representations - functional cluster interaction
[SWS_CORE_10600]	Semantics of ara::core::Result
[SWS_CORE_10800]	Semantics of ara::core::Future and ara::core::Promise
[SWS_CORE_12403]	Standard Error Stream Logging of Explicitly aborting an Operation
[SWS_CORE_90001]	Include folder structure

▽

△

Number	Heading
[SWS_CORE_90005]	Custom declarations and definitions
[SWS_CORE_90021]	Pre-conditions for `ara::core::InstanceSpecifier`

Table E.15: Changed Specification Items in R24-11

E.6.3 Deleted Specification Items in R24-11

Number	Heading
[SWS_CORE_01030]	`value` member function overloads
[SWS_CORE_11950]	MemoryResource base behavior
[SWS_CORE_11951]	MemoryResource error behavior
[SWS_CORE_12402]	"Noreturn" property for Abort
[SWS_CORE_90006]	

Table E.16: Deleted Specification Items in R24-11