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	R23-11

Document Change History

Date	**Release**	**Changed by**	**Description**
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

Contents

1 Introduction

This document 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 relevant to Adaptive Core that are not included in the [1, AUTOSAR glossary].

Term	Description
Explicit Operation Abortion	Immediate abortion of 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

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] List of Adaptive Platform Functional Clusters
AUTOSAR_AP_TR_FunctionalClusterList

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

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

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

[7] Specification of Execution Management
AUTOSAR_AP_SWS_ExecutionManagement

[8] Explanation of ara::com API
AUTOSAR_AP_EXP_ARAComAPI

[9] N4659:Working Draft, Standard for ProgrammingLanguage C++
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4659.pdf

[10] N4820:Working Draft, Standard for Programming Language C++
http://www.open-std.org/JTC1/SC22/WG21/docs/papers/2019/n4820.pdf

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

4 Constraints and assumptions

4.1 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”.
- Chapter 7.2 (“Functional Specification”) describes some behavior informally that should rather be given as specification items.

4.2 Applicability to car domains

No restrictions to applicability.

5 Dependencies to other modules

This chapter provides an overview of the dependencies to other Functional Clusters in the AUTOSAR Adaptive Platform. Section 5.1 “Provided Interfaces” lists the interfaces provided by `Core` to other Functional Clusters. Section 5.2 “Required Interfaces” lists the interfaces required by `Core`.

A detailed technical architecture documentation of the 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

6 Requirements Tracing

The following tables reference the requirements specified in <CITATIONS_OF_CONTRIBUTED_DOCUMENTS> and links to the fulfillment of these. Please note that if column "Satisfied by" is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[RS_AP_00111]	The AUTOSAR Adaptive Platform shall support source code portability for AUTOSAR Adaptive applications.	[SWS_CORE_15005] [SWS_CORE_90001] [SWS_CORE_90002] [SWS_CORE_90003] [SWS_CORE_90004] [SWS_CORE_90005] [SWS_CORE_90006] [SWS_CORE_90021] [SWS_CORE_90022]
[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_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_00010] [SWS_CORE_00011] [SWS_CORE_00013] [SWS_CORE_00014] [SWS_CORE_00016] [SWS_CORE_00040] [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] [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] ▽

▽

△

Requirement	Description	Satisfied by
		△ [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_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] [SWS_CORE_00866] [SWS_CORE_00867] [SWS_CORE_00868] [SWS_CORE_00869] [SWS_CORE_00870] [SWS_CORE_00876] [SWS_CORE_01030] [SWS_CORE_01031] [SWS_CORE_01033] [SWS_CORE_01096] [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] ▽

▽

△

Requirement	Description	Satisfied by
		△ [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_01601] [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_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] ▽

▽

△

Requirement	Description	Satisfied by
		△ [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] [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_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] ▽

▽

△

Requirement	Description	Satisfied by
		△ [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_11950] [SWS_CORE_11951] [SWS_CORE_11952] [SWS_CORE_12000] [SWS_CORE_12200] [SWS_CORE_12402] [SWS_CORE_12403] [SWS_CORE_12404] [SWS_CORE_12405] [SWS_CORE_12406] [SWS_CORE_12407]
[RS_AP_00132]	noexcept behavior of API functions	[SWS_CORE_00050] [SWS_CORE_00051] [SWS_CORE_00052] [SWS_CORE_00053] [SWS_CORE_00054]
[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" in ara types.	[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_00020] [SWS_CORE_00021] [SWS_CORE_00022] [SWS_CORE_00023] [SWS_CORE_10600] [SWS_CORE_15002] [SWS_CORE_90021]
[RS_AP_00145]	Availability of special member functions.	[SWS_CORE_00617]
[RS_AP_00149]	Guidance on error handling.	[SWS_CORE_90021]
[RS_Main_00011]	Mechanisms for Reliable Systems	[SWS_CORE_10001] [SWS_CORE_10002] [SWS_CORE_15003] [SWS_CORE_15004]
[RS_Main_00150]	AUTOSAR shall support the deployment and reallocation of AUTOSAR Application Software	[SWS_CORE_08032]

▽

△

Requirement	Description	Satisfied by
[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: RequirementsTracing

7 Requirements Specification

7.1 General requirements for all Functional Clusters

The goal of this section is to define 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.

[SWS_CORE_90001] Include folder structure ⌈All #include directives in header files that refer to ARA libraries shall be written in the form

```
#include "ara/fc/header.h"
```

with “ara” as the first path element, “fc” being the remaining directory path of the implementation’s *installed* header file, starting with the Functional Cluster short name, and “header.h” being the filename of the header file.⌋*(RS_AP_00116, RS_AP_00111)*

The Functional Cluster short names are defined in [3].

Example: Execution Management (short name “exec”) provides class `ExecutionClient`, which can be accessed with:

```
#include "ara/exec/execution_client.h"
```

The "..." form of #include statements shall be used, due to the recommendation given in [4, the C++14 standard] section 16.2.7.

[SWS_CORE_90002] Prevent multiple inclusion of header file ⌈All public header files shall prevent multiple inclusion by using #include guards that are likely to be system-wide unique.⌋*(RS_AP_00111)*

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]{DRAFT} ⌈C/C++ preprocessor symbols that start with `ARA` are reserved for use by AUTOSAR.⌋*(RS_AP_00111)*

The Adaptive Platform generally avoids the use of C/C++ preprocessor macros. However, in case macros are introduced at some later point in time, any such macro will 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]{DRAFT} **Implementation-defined declaration classifiers** ⌈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.⌋*(RS_AP_00111)*
Note: The order of declarations may be freely chosen.

[4, The C++14 standard] defines in chapter 7.1 [dcl.spec] 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]{DRAFT} **Custom declarations and definitions** ⌈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.⌋*(RS_AP_00111)*

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_90006]{DRAFT} ⌈If a constructor in the `ara` framework is called with wrong or invalid `ara::core::InstanceSpecifier`, the Functional Cluster implementation shall treat this as a `Violation` with a standardized log message `"Invalid InstanceSpecifer >passed InstanceSpecifier< in ctor >ctor.shortname<"`.⌋*(RS_AP_00111)*

The rationale to treat this as a `Violation` is that this is seen as an integration error which anyway cannot be handled by the caller of the API. Aborting execution is in line with the strategy to fail early.

Any other error check within the constructors is defined within the respective SWS.

7.1.1 Initialize/Deinitialize

`ara::core::Initialize` allows a central initialization of all included shared libraries of the ARA framework. This could include static initializers 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_90021]{DRAFT} ⌈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 `Violation` with the message: "`Platform not initialized! The platform needs to be initialized before the execution of >constructor or function name<.`".⌋*(RS_AP_00111, RS_AP_00142, RS_AP_00149)*

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

Rationale: 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_90022]{DRAFT} ⌈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.⌋*(RS_AP_00111)*

Rationale: A check for deinitialization 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.

7.2 Functional Specification

This section describes the concepts that are introduced with this Functional Cluster. Particular emphasis is put on error handling.

7.2.1 Error handling

7.2.1.1 Types of unsuccessful operations

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, the following types of unsuccessful operations are distinguished within the Adaptive Platform:

[SWS_CORE_00020]{DRAFT} **Semantics of an Error** ⌈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.⌋*(RS_AP_00142)*

[SWS_CORE_00021]{DRAFT} **Semantics of a Violation** ⌈A `Violation` is the consequence of failed pre- or post-conditions of internal state of the application framework. They are the Adaptive Platform's analog to a failed assertion. A `Violation` is non-recoverable.⌋*(RS_AP_00142)*

[SWS_CORE_00022]{DRAFT} **Semantics of a Corruption** ⌈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.⌋*(RS_AP_00142)*

[SWS_CORE_00023]{DRAFT} **Semantics of a Failed Default Allocation** ⌈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.⌋*(RS_AP_00142)*

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`).

7.2.1.2 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.

7.2.1.3 Handling of unsuccessful operations in the Adaptive Platform

The types of unsuccessful operations defined in section 7.2.1.1 ("Types of unsuccessful operations") are to be treated in different ways.

[SWS_CORE_00002] Handling of Errors ⌈An Error shall be returned from the function as an instance of ara::core::Result or ara::core::Future.⌋*(RS_AP_00142, RS_AP_00139, RS_AP_00128)*

[SWS_CORE_00003] Handling of Violations ⌈If a Violation is detected, then the operation shall be terminated by either:

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

⌋*(RS_AP_00142)*

[SWS_CORE_00004] Handling of Corruptions ⌈If a Corruption is detected, it shall result in unsuccessful process termination, in an implementation-defined way.⌋*(RS_AP_00142)*
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.

[SWS_CORE_00005] Handling of failed default allocations ⌈A Failed Default Allocation shall be treated the same as a Violation.⌋*(RS_AP_00142)*
Note: An error of a custom allocator is not subject to this definition.

7.2.1.4 Facilities for Error Handling

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

7.2.1.4.1 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]{DRAFT} **Semantics of ErrorCode** ⌈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

⌋*(RS_AP_00119)*

`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]{DRAFT} **Comparison of ara::core::ErrorCode instances** ⌈Any comparison of two `ara::core::ErrorCode` instances shall consider only the following members:

- error code value
- error domain

⌋*(RS_AP_00119)*

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 ⌈Class `ara::core::ErrorCode` shall be a *literal type*, as defined in section 3.9-10 [basic.types] of [4, the C++14 standard].⌋*(RS_AP_00130)*

7.2.1.4.2 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 subclass. For normalized access to these associated types, type aliases with standardized names are defined within the ara::core::ErrorDomain subclass. This makes the ErrorDomain subclass the root of all data about errors.

[SWS_CORE_10303]{DRAFT} **Semantics of ErrorDomain** ⌈The type ara::core::ErrorDomain defines a context for a set of error conditions.⌋*(RS_AP_00119)*

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 occured errors between callee and caller (potentially located at different ECUs) is based on this Id.

[SWS_CORE_10401]{DRAFT} **Identity of ErrorDomains** ⌈Two instances of ara::core::ErrorDomain shall compare equal if and only if their unique identifiers are the same.⌋*(RS_AP_00119)*

Given this definition of identity of error domains, it usually makes sense to have only one single instance of each ara::core::ErrorDomain subclass. While new instances of these subclasses can be created by calling their constructors, the recommended way to gain access to these subclasses 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 subclass 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 subclasses directly; most access is done via ara::core::ErrorCode.

As ara::core::ErrorDomain subclasses 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_10400] ErrorDomain type properties ⌈Class ara::core::ErrorDomain and all its subclasses shall be *literal types*, as defined in section 3.9-10 [basic.types] of [4, the C++14 standard].⌋*(RS_AP_00130)*

7.2.1.4.3 Result

The ara::core::Result type follows the ValueOrError concept from the C++ proposal p0786 [5]. 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. However, ErrorType is defaulted to ara::core::ErrorCode, and it is expected that this assignment is kept throughout the Adaptive Platform.

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]{DRAFT} **Semantics of ara::core::Result** ⌈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 value *V*, where *V* may be any C++ type; or
- an error *E*, where *E* may be any C++ type; default is ara::core::ErrorCode.

⌋*(RS_AP_00119, RS_AP_00142, RS_AP_00128)*

Note: It is strongly recommended to use only ara::core::ErrorCode for template parameter *E*.

7.2.1.4.4 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 7.2.1.4.3, 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.

[SWS_CORE_10800]{DRAFT} **Semantics of ara::core::Future and ara::core::Promise** ⌈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 error type *E* which may be asynchronously retrieved in a thread-safe manner at a later point in time.⌋*(RS_AP_00138, RS_AP_00128)*

Note: It is strongly recommended to use only `ara::core::ErrorCode` for template parameter *E*.

7.2.1.5 Duality of ErrorCode and exceptions

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.

7.2.1.6 Exception hierarchy

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 subclass (which is described in section 7.2.1.4.2). This exception class also conforms to a standard naming scheme: <shortname> of `ApApplicationErrorDomain` plus “Exception” suffix (this makes it distinguishable from the `ErrorDomain` subclass itself). It is located in the same namespace as the corresponding `ErrorDomain` subclass.

7.2.1.7 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` subclass
- a non-member `ErrorDomain` subclass 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 ⌈The `ara::core::ErrorDomain` subclass 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.⌋*(RS_AP_00130)*
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 *shortname* of the `ApApplicationErrorDomain`.

7.2.1.7.1 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 ⌈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.⌋*(RS_AP_00130)*

[SWS_CORE_10901] Error condition enumeration naming ⌈Error domain error condition enumerations shall follow the naming scheme `<SN>Errc`, where `<SN>` is the shortname of the `ApApplicationErrorDomain`.⌋*(RS_AP_00130)*

[SWS_CORE_10902] Error condition enumeration contents ⌈Error domain error condition enumerations shall not contain any values that indicate success.⌋*(RS_AP_00130)*

[SWS_CORE_10903] Error condition enumeration numbers ⌈Error domain error condition enumerations shall keep the number 0 unassigned.⌋*(RS_AP_00130)*

7.2.1.7.2 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 ⌈Each error domain shall define an exception base type that is a subclass of `ara::core::Exception`.⌋*(RS_AP_00130)*

[SWS_CORE_10911] ErrorDomain exception base type naming ⌈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`.⌋*(RS_AP_00130)*

[SWS_CORE_10912]{DRAFT} **ErrorDomain exception type hierarchy** ⌈All additional exception types defined by a software module shall have the exception base type specified by [SWS_CORE_10910] as a base class.⌋*(RS_AP_00130)*

7.2.1.7.3 ErrorDomain subclass

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 subclass type ⌈Each error domain shall define a class type that derives publicly from ara::core::ErrorDomain.⌋*(RS_AP_00130)*

[SWS_CORE_10931] ErrorDomain subclass naming ⌈All subclasses of ara::core::ErrorDomain shall follow the naming scheme <SN>ErrorDomain, where <SN> is the shortname of the ApApplicationErrorDomain.⌋*(RS_AP_00130)*

[SWS_CORE_10932] ErrorDomain subclass non-extensibility ⌈All subclasses of ara::core::ErrorDomain shall be final.⌋*(RS_AP_00130, RS_AP_00140)*

[SWS_CORE_10933] ErrorDomain subclass Errc symbol ⌈All subclasses 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].⌋*(RS_AP_00130)*

[SWS_CORE_10934] ErrorDomain subclass Exception symbol ⌈All subclasses 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].⌋*(RS_AP_00130)*

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

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

[SWS_CORE_10950] ErrorDomain subclass member function property ⌈With the exception of ara::core::ErrorDomain::ThrowAsException, all public member functions of all ErrorDomain subclasses shall be noexcept.⌋*(RS_AP_00130)*

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

[SWS_CORE_10951] ErrorDomain subclass shortname retrieval ⌈The return value of an error domain's ara::core::ErrorDomain::Name member function shall be equal to the shortname of the ApApplicationErrorDomain.⌋*(RS_AP_00130)*

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 subclass unique identifier retrieval ⌈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].⌋*(RS_AP_00130)*

An ErrorDomain can “transform” an ErrorCode into an exception.

[SWS_CORE_10953] Throwing ErrorCodes as exceptions ⌈The type of an exception thrown by the ErrorDomain subclass’s implementation of ara::core::ErrorDomain::ThrowAsException shall derive from that ErrorDomain subclass’s Exception type alias defined by [SWS_CORE_10934].⌋*(RS_AP_00130)*

7.2.1.7.4 Non-member ErrorDomain subclass 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 subclass also shall be constexpr-constructible (see [SWS_CORE_10400]).

[SWS_CORE_10980] ErrorDomain subclass accessor function ⌈For all subclasses of ara::core::ErrorDomain, there shall be a non-member constexpr function that returns a reference-to-const to a singleton instance of it.⌋*(RS_AP_00130)*

[SWS_CORE_10981] ErrorDomain subclass accessor function naming ⌈All ara::core::ErrorDomain subclass accessor functions shall follow the naming scheme Get<SN>ErrorDomain, where <SN> is the shortname of the ApApplicationErrorDomain.⌋*(RS_AP_00130)*

[SWS_CORE_10982] ErrorDomain subclass accessor function ⌈All ara::core::ErrorDomain subclass accessor functions shall have a return type of const ErrorDomain&.⌋*(RS_AP_00130)*

7.2.1.7.5 Non-member MakeErrorCode overload

And finally, 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 subclass, and call the type-erased constructor of class ara::core::ErrorCode.

[SWS_CORE_10990] MakeErrorCode overload for new error domains ⌈For all subclasses 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 subclass’s error condition range.⌋*(RS_AP_00130)*

[SWS_CORE_10991] MakeErrorCode overload signature ⌈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`.⌋*(RS_AP_-00130)*

7.2.1.7.6 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
```

7.2.1.8 AUTOSAR error domains

The full range of unique error domain identifiers is partitioned into a range of AUTOSAR-specified IDs, another range of vendor-defined IDs, and another range of user-defined IDs.

User-defined IDs have their top-bit set to 0 and can use the remaining 63 bits to provide uniqueness. IDs with their top-bit set to 1 are reserved for AUTOSAR and stack vendor use.

[SWS_CORE_00010] Error domain identifier ⌈All error domains shall have a system-wide unique identifier that is represented as a 64-bit unsigned integer value.⌋*(RS_AP_-00130)*

[SWS_CORE_00011] AUTOSAR error domain range ⌈Error domain identifiers where bit #63 is set to 1 and bit #62 is set to 0 are reserved for AUTOSAR-defined error domains.⌋*(RS_AP_00130)*

[SWS_CORE_00016]{DRAFT} **Vendor-defined error domain range** ⌈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, and bits #15..#0 can be used by each vendor for error domain identifiers.⌋*(RS_AP_00130)*

[SWS_CORE_00013] The Future error domain ⌈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.⌋*(RS_AP_00130)*

[SWS_CORE_00014] The Core error domain ⌈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.⌋*(RS_AP_00130)*

7.2.2 Async signal safety

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

[6, 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.

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

7.2.3 Explicit Operation Abortion

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 `Explicit Operation Abortion`, which usually leads to an `Unexpected Termination` as defined by [7]. 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_12402]{DRAFT} **"Noreturn" property for Abort** ⌈The function `ara::core::Abort` shall not return to its caller.⌋*(RS_AP_00130)*

[SWS_CORE_12403]{DRAFT} **Logging of Explicit Operation Abortion** ⌈Calling `ara::core::Abort` shall result in a log message, which shall contain the string that has been passed to the function as argument, being output to the process's standard error stream.⌋*(RS_AP_00130)*

[SWS_CORE_12407]{DRAFT} **Thread-safety of Explicit Operation Abortion** ⌈While a call to `ara::core::Abort` is in progress, other calls to this function shall block the calling threads.⌋*(RS_AP_00130)*

`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]{DRAFT} **AbortHandler invocation** ⌈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.⌋*(RS_AP_00130)*

7.2.3.1 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]{DRAFT} **Final action without AbortHandler** ⌈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()`.⌋*(RS_AP_00130)*

[SWS_CORE_12406]{DRAFT} **Final action with returning AbortHandlers** ⌈If there are custom `ara::core::AbortHandler`s 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()`.⌋*(RS_AP_00130)*

7.2.3.2 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(SIGABORT)`.

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.

7.2.4 Advanced data types

7.2.4.1 AUTOSAR types

7.2.4.1.1 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 [8] sections 6.1 ("Instance Identifiers") and 9.4.4 ("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 ⌈In case of application interaction and thus in the presence of `PortPrototypes` the string representation of a valid `ara::core::InstanceSpecifier` consists of a "/"-separated list of model element `shortNames` starting from an `Executable` via the `RootSwComponentPrototype` and several `SwComponentPrototypes` to the respective `PortPrototype` to which the `ara::core::InstanceSpecifier` shall apply.⌋*(RS_AP_00130)*

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 ⌈In case of functional cluster interaction and thus in the absence of `PortPrototypes` the string representation of a valid `ara::core::InstanceSpecifier` consists of a "/"-separated list of model element `shortNames` starting from a top-level `ARPackage` via contained sub-packages to the respective mapping element that is derived from `FunctionalClusterInteractsWithFunctionalClusterMapping` (see TPS_MANI_03268 for further details).⌋*(RS_AP_00130)*

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

```
ARPackage.shortName/.../ARPackage.shortName
/FunctionalClusterInteractsWithFunctionalClusterMapping.shortName
```

[SWS_CORE_10201] Validation of meta-model paths ⌈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].⌋*(RS_AP_00130)*

[SWS_CORE_10202] Construction of InstanceSpecifier objects ⌈APIs for construction of `ara::core::InstanceSpecifier` objects shall be available in both potentially-throwing and non-throwing form.⌋*(RS_AP_00130)*

7.2.4.2 Types derived from the base C++ standard

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

Some types are already contained in [4, the C++14 standard]; 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]{DRAFT} **Errors originating from C++ standard classes** ⌈For the classes in ara::core specified below by means of the corresponding classes of the C++ standard, all functions that are specified by [4, the C++14 standard], [9, the C++17 standard], or [10, the draft C++20 standard] to throw any exceptions, are instead specified to be the cause of a `Violation` when they do so.⌋*(RS_AP_00130)*

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

7.2.4.2.1 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

[SWS_CORE_11200] Array base behavior ⌈`ara::core::Array` and all its member functions and supporting constructs shall behave identical to those of `std::array` in header `<array>` from [4, the C++14 standard], except for the differences specified in this document.⌋*(RS_AP_00130)*

7.2.4.2.2 Vector

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

[SWS_CORE_11300]{DRAFT} **Vector base behavior** ⌈ara::core::Vector and all its member functions and supporting constructs shall behave identical to those of std::vector in header <vector> from [4, the C++14 standard], except for the differences specified in this document.⌋*(RS_AP_00130)*

7.2.4.2.3 Map

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

[SWS_CORE_11400]{DRAFT} **Map base behavior** ⌈ara::core::Map and all its member functions and supporting constructs shall behave identical to those of std::map in header <map> from [4, the C++14 standard], except for the differences specified in this document.⌋*(RS_AP_00130)*

7.2.4.2.4 String and BasicString

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

[SWS_CORE_12000]{DRAFT} **String base behavior** ⌈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 [4, the C++14 standard], except for the differences specified in this document.⌋*(RS_AP_00130)*

7.2.4.2.5 SteadyClock

7.2.4.2.5.1 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.

7.2.4.2.5.2 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 undefinedness 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 ⌈Class `ara::core::SteadyClock` shall meet the requirements of *TrivialClock* from [4, the C++14 standard].⌋*(RS_AP_00130)*

[SWS_CORE_11801] Epoch of SteadyClock ⌈The `epoch` of `ara::core::SteadyClock` shall be the system start-up.⌋*(RS_AP_00130)*

7.2.4.3 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.

7.2.4.3.1 Optional

This section describes the `ara::core::Optional` type.

[SWS_CORE_11000]{DRAFT} **Optional base behavior** ⌈`ara::core::Optional` and all its member functions and supporting constructs shall behave identical to those of `std::optional` in header `<optional>` from [9, the C++17 standard], except for the differences specified in this document.⌋*(RS_AP_00130)*

Note: The `value()` function and the `bad_optional_access` exception defined in the C++ standard library are left out of this specification to provide an API that does not make use of exceptions. Use either `has_value` or `operator bool()` to check if the `ara::core::Optional` contains a value before accessing the value with e.g., `operator*`. 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_01030]{DRAFT} **`value` member function overloads** ⌈Contrary to the description in [9], no member functions with this name exist in `ara::core::Optional`.⌋*(RS_AP_00130)*

[SWS_CORE_01031]{DRAFT} **class bad_optional_access** ⌈No class named `bad_optional_access` is defined in the `ara::core` namespace.⌋*(RS_AP_00130)*

7.2.4.3.2 Variant

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

[SWS_CORE_11600]{DRAFT} **Variant base behavior** ⌈`ara::core::Variant` and all its member functions and supporting constructs shall behave identical to those of std::variant in header <variant> from [9, the C++17 standard], except for the differences specified in this document.⌋*(RS_AP_00130)*

7.2.4.3.3 StringView

This section describes the ara::core::StringView type that represents a read-only view over a contiguous sequence of characters whose storage is owned by another object.

[SWS_CORE_12200]{DRAFT} **StringView base behavior** ⌈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 [9, the C++17 standard], except for the differences specified in this document.⌋*(RS_AP_00130)*

7.2.4.3.4 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 C++20, with deviations mostly coming from the lack of C++20's "ranges" feature.

[SWS_CORE_11900]{DRAFT} **Span base behavior** ⌈ara::core::Span and all its member functions and supporting constructs shall behave identical to those of std::span in header <span> from [10, the draft C++20 standard], except for the differences specified in this document.⌋*(RS_AP_00130)*

7.2.4.3.5 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 [9, the C++17 standard] a conforming implementation, but also allow a class-based implementation with only C++14 means.

Unlike std::byte from [9, the C++17 standard], 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** ⌈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.⌋*(RS_AP_00130)*

[SWS_CORE_10101] Size of type ara::core::Byte ⌈The size (in bytes) of an instance of type `ara::core::Byte` (determined with `sizeof(ara::core::Byte)`) shall be 1.⌋*(RS_AP_00130)*

[SWS_CORE_10102] Value range of type ara::core::Byte ⌈The value of an instance of type `ara::core::Byte` shall be constrained to the range `[0..std::numeric_limits<unsigned char>::max()]`.⌋*(RS_AP_00130)*

[SWS_CORE_10103] Creation of ara::core::Byte instances ⌈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.⌋*(RS_AP_00130)*

[SWS_CORE_10104] Default-constructed ara::core::Byte instances ⌈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.⌋*(RS_AP_00130)*

[SWS_CORE_10105] Destructor of type ara::core::Byte ⌈The destructor of type `ara::core::Byte` shall be trivial.⌋*(RS_AP_00130)*

[SWS_CORE_10106] Implicit conversion from other types ⌈The type `ara::core::Byte` shall not be implicitly convertible from any other type.⌋*(RS_AP_00130)*

[SWS_CORE_10107] Implicit conversion to other types ⌈The type `ara::core::Byte` shall allow no implicit conversion to any other type, including `bool`.⌋*(RS_AP_-00130)*

[SWS_CORE_10108] Conversion to unsigned char ⌈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.⌋*(RS_AP_-00130)*

[SWS_CORE_10109] Equality comparison for ara::core::Byte ⌈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.⌋*(RS_AP_00130)*

[SWS_CORE_10110] Non-equality comparison for ara::core::Byte ⌈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.⌋*(RS_AP_00130)*

7.2.4.3.6 MemoryResource

`ara::core::MemoryResource` is an abstract interface to an unbounded set of classes (`ara::core::MonotonicBufferResource` and `ara::core::PolymorphicAllocator`) encapsulating memory resources. It is based on std::pmr::memory_resource from [9, the C++17 standard]/[10, the C++20 standard]

[SWS_CORE_11950]{DRAFT} **MemoryResource base behavior** ⌈ara::core::MemoryResource and all its member functions and supporting constructs (ara::core::MonotonicBufferResource and ara::core::PolymorphicAllocator) shall behave identical to those of std::pmr::memory_resource in header <memory_resource> from [10, the C++20 standard], except for the differences specified in this document.⌋*(RS_AP_00130)*

[SWS_CORE_11951]{DRAFT} **MemoryResource error behavior** ⌈ara::core::MemoryResource and all its member functions and supporting constructs (ara::core::MonotonicBufferResource and ara::core::PolymorphicAllocator) shall return a nullptr (if possible) in case of any error. Otherwise the error shall be silently ignored.⌋*(RS_AP_00130)*

Rationale for [SWS_CORE_11951]: Exceptions should be avoided.

Some [4, the C++14 standard] compilers support a backport of [[nodiscard]] but this is compiler specific.

[SWS_CORE_11952]{DRAFT} **Resolution of macro ARA_COMPILER_DEFINED_NODISCARD** ⌈The macro ARA_COMPILER_DEFINED_NODISCARD shall conditionally resolve to the C++ attribute [[nodiscard]], depending on whether this is supported by the compiler.⌋*(RS_AP_00130)*

7.2.5 Initialization and Shutdown

This section describes the global initialization and shutdown of the ARA framework. Before the framework is initialized, and after the it is deinitialized, 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, there exist initialization and shutdown functions 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_15003]{DRAFT} **Startup and initialization of ARA** ⌈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

⌋*(RS_Main_00011)*

[SWS_CORE_15004]{DRAFT} **Shutdown and de-initialization of ARA** ⌈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
- deallocation of dynamically allocated memory
- deallocation of other system resources

⌋*(RS_Main_00011)*

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 deinitialized. 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 deinitialization 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_15005]{DRAFT} ⌈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.⌋*(RS_AP_00111)*

[SWS_CORE_15002]{DRAFT} **Special ara::core types to be used independently of initialization** ⌈A small subset of `ara::core` types and functions shall be usable independently of initialization with `ara::core::Initialize` and deinitialization 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 subclasses, 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

⌋*(RS_AP_00142)*

The rationale for the exception for this subset is the intended use of these types and functions before initialization and after deinitialization. As well as that these types and functions are used as part of the initialization/deinitialization (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.

8 API specification

8.1 C++ language binding

All symbols described in this chapter reside within the namespace `ara::core`. All symbols have `public` visibility unless otherwise noted.

8.1.1 `ErrorDomain` data type

This section describes the `ara::core::ErrorDomain` type that constitutes a base class for error domain implementations.

[SWS_CORE_00110]{DRAFT} **Definition of API class ara::core::ErrorDomain** ⌈

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 class is a literal type, and subclasses are strongly advised to be literal types as well.

⌋*(RS_AP_00130)*

[SWS_CORE_00121]{DRAFT} **Definition of API type ara::core::ErrorDomain::IdType** ⌈

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 .

⌋*(RS_AP_00130)*

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

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 .

⌋*(RS_AP_00130)*

[SWS_CORE_00123] Definition of API type ara::core::ErrorDomain::SupportData Type ⌈

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 .

⌋*(RS_AP_00130)*

[SWS_CORE_00131]{DRAFT} **Definition of API function ara::core::ErrorDomain::ErrorDomain** ⌈

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

⌋*(RS_AP_00130)*

[SWS_CORE_00132]{DRAFT} **Definition of API function ara::core::ErrorDomain::ErrorDomain** ⌈

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

⌋*(RS_AP_00130)*

[SWS_CORE_00135]{DRAFT} **Definition of API function ara::core::ErrorDomain::ErrorDomain** ⌈

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

▽

△

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

⌋*(RS_AP_00130)*

[SWS_CORE_00136]{DRAFT} **Definition of API function ara::core::ErrorDomain::~ErrorDomain** ⌈

Kind:	function
Header file:	#include "ara/core/error_domain.h"
Scope:	class ara::core::ErrorDomain
Symbol:	~ErrorDomain()
Syntax:	`~ErrorDomain () noexcept=default;`
Exception Safety:	noexcept
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

⌋*(RS_AP_00130)*

[SWS_CORE_00133]{DRAFT} **Definition of API function ara::core::ErrorDomain::operator=** ⌈

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

⌋*(RS_AP_00130)*

[SWS_CORE_00134]{DRAFT} **Definition of API function ara::core::ErrorDomain::operator=** ⌈

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

⌋*(RS_AP_00130)*

[SWS_CORE_00137]{DRAFT} Definition of API function ara::core::ErrorDomain::operator== ⌈

Kind:	function	
Header file:	#include "ara/core/error_domain.h"	
Scope:	class ara::core::ErrorDomain	
Symbol:	operator==(const ErrorDomain &other)	
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:	noexcept	
Description:	Compare for equality with another ErrorDomain instance. Two ErrorDomain instances compare equal when their identifiers (returned by Id()) are equal.	

⌋*(RS_AP_00130)*

[SWS_CORE_00138]{DRAFT} Definition of API function ara::core::ErrorDomain::operator!= ⌈

Kind:	function	
Header file:	#include "ara/core/error_domain.h"	
Scope:	class ara::core::ErrorDomain	
Symbol:	operator!=(const ErrorDomain &other)	
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:	noexcept	
Description:	Compare for non-equality with another ErrorDomain instance.	

⌋*(RS_AP_00130)*

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

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

⌋*(RS_AP_00130)*

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

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

⌋*(RS_AP_00130)*

[SWS_CORE_00153]{DRAFT} Definition of API function ara::core::ErrorDomain::Message ⌈

Kind:	function	
Header file:	#include "ara/core/error_domain.h"	
Scope:	class ara::core::ErrorDomain	
Symbol:	Message(CodeType errorCode)	
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:	noexcept	
Description:	Return a textual representation of the given error code. It is a Violation if the errorCode did not originate from this error domain, and thus be subject to SWS_CORE_00003. The returned pointer remains owned by the ErrorDomain subclass and shall not be freed by clients.	

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/error_domain.h"	
Scope:	class ara::core::ErrorDomain	
Symbol:	ThrowAsException(const ErrorCode &errorCode)	
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
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.	

⌋*(RS_AP_00130)*

8.1.2 `ErrorCode` data type

This section describes the `ara::core::ErrorCode` type which holds a domain-specific error.

[SWS_CORE_00501]{DRAFT} **Definition of API class ara::core::ErrorCode** ⌈

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.

⌋*(RS_AP_00130, RS_AP_00140)*

[SWS_CORE_00512]{DRAFT} **Definition of API function ara::core::ErrorCode::ErrorCode** ⌈

Kind:	function	
Header file:	#include "ara/core/error_code.h"	
Scope:	class ara::core::ErrorCode	
Symbol:	ErrorCode(EnumT e, ErrorDomain::SupportDataType data=ErrorDomain::SupportDataType())	

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:	noexcept	
Description:	Construct a new ErrorCode instance with parameters. This constructor does not participate in overload resolution unless EnumT is an enum type.	

⌋*(RS_AP_00130)*

[SWS_CORE_00513]{DRAFT} **Definition of API function ara::core::ErrorCode::ErrorCode** ⌈

Kind:	function	
Header file:	#include "ara/core/error_code.h"	
Scope:	class ara::core::ErrorCode	
Symbol:	ErrorCode(ErrorDomain::CodeType value, const ErrorDomain &domain, ErrorDomain::SupportDataType data=ErrorDomain::SupportDataType())	
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:	noexcept	
Description:	Construct a new ErrorCode instance with parameters.	

⌋*(RS_AP_00130)*

[SWS_CORE_00514]{DRAFT} **Definition of API function ara::core::ErrorCode::Value** ⌈

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

⌋*(RS_AP_00130)*

[SWS_CORE_00515]{DRAFT} **Definition of API function ara::core::ErrorCode::Domain** ⌈

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

⌋*(RS_AP_00130)*

[SWS_CORE_00516]{DRAFT} **Definition of API function ara::core::ErrorCode::SupportData** ⌈

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

⌋*(RS_AP_00130)*

[SWS_CORE_00518]{DRAFT} **Definition of API function ara::core::ErrorCode::Message** ⌈

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

⌋*(RS_AP_00130)*

[SWS_CORE_00519]{DRAFT} **Definition of API function ara::core::ErrorCode::ThrowAsException** ⌈

Kind:	function	
Header file:	#include "ara/core/error_code.h"	
Scope:	class ara::core::ErrorCode	
Symbol:	ThrowAsException()	
Syntax:	`void ThrowAsException () const;`	
Return value:	None	
Exceptions:	<TYPE>	an exception of the type determined by the associated ErrorDomain as defined in [SWS_CORE_10953]
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)`.	

⌋*(RS_AP_00130)*

8.1.2.1 `ErrorCode` non-member operators

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

Kind:	function	
Header file:	#include "ara/core/error_code.h"	
Scope:	namespace ara::core	
Symbol:	operator==(const ErrorCode &lhs, const ErrorCode &rhs)	
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:	noexcept	

▽

△

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.

⌋(*RS_AP_00130*)

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

Kind:	function	
Header file:	#include "ara/core/error_code.h"	
Scope:	namespace ara::core	
Symbol:	operator!=(const ErrorCode &lhs, const ErrorCode &rhs)	
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:	noexcept	
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.	

⌋(*RS_AP_00130*)

8.1.3 `Exception` data type

This section describes the `ara::core::Exception` type that constitutes the base type for all exception types defined by the Adaptive Platform.

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

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 AUTOSAR exception types.

⌋(*RS_AP_00130*)

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

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

⌋*(RS_AP_00130)*

[SWS_CORE_00615]{DRAFT} **Definition of API function ara::core::Exception::Exception** ⌈

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

⌋*(RS_AP_00130)*

[SWS_CORE_00616]{DRAFT} **Definition of API function ara::core::Exception::operator=** ⌈

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

⌋*(RS_AP_00130)*

[SWS_CORE_00617]{DRAFT} **Definition of API function ara::core::Exception::~Exception** ⌈

Kind:	function
Header file:	#include "ara/core/exception.h"
Scope:	class ara::core::Exception
Symbol:	~Exception()

▽

△

Syntax:	`virtual ~Exception ()=default;`
Description:	Destructs the Exception object.

⌋*(RS_AP_00130, RS_AP_00145)*

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

Kind:	function	
Header file:	#include "ara/core/exception.h"	
Scope:	class ara::core::Exception	
Symbol:	what()	
Syntax:	`const char * what () const noexcept override;`	
Return value:	const char *	a null-terminated string
Exception Safety:	noexcept	
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.	

⌋*(RS_AP_00130)*

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

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

⌋*(RS_AP_00130)*

[SWS_CORE_00614]{DRAFT} Definition of API function ara::core::Exception::operator= ⌈

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

⌋*(RS_AP_00130)*

[SWS_CORE_00618]{DRAFT} **Definition of API function ara::core::Exception::Exception** ⌈

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

⌋*(RS_AP_00130)*

8.1.4 `Result` data type

This section describes the `ara::core::Result<T, E>` type (and its specialization for `T=void`) that contains a value of type T or an error of type E.

[SWS_CORE_00701]{DRAFT} **Definition of API class ara::core::Result** ⌈

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.	

⌋*(RS_AP_00130)*

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

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 .

⌋*(RS_AP_00130)*

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

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 .

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	Result(const T &t)	
Syntax:	`Result (const T &t);`	
Parameters (in):	t	the value to put into the Result
Exception Safety:	not exception safe	
Description:	Construct a new Result from the specified value (given as lvalue).	

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	Result(T &&t)	
Syntax:	`Result (T &&t);`	
Parameters (in):	t	the value to put into the Result
Exception Safety:	not exception safe	
Description:	Construct a new Result from the specified value (given as rvalue).	

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	Result(const E &e)	
Syntax:	`explicit Result (const E &e);`	
Parameters (in):	e	the error to put into the Result
Exception Safety:	not exception safe	
Description:	Construct a new Result from the specified error (given as lvalue).	

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	Result(E &&e)	
Syntax:	`explicit Result (E &&e);`	
Parameters (in):	e	the error to put into the Result
Exception Safety:	not exception safe	
Description:	Construct a new Result from the specified error (given as rvalue).	

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	Result(const Result &other)	
Syntax:	`Result (const Result &other);`	
Parameters (in):	other	the other instance
Exception Safety:	not exception safe	
Description:	Copy-construct a new Result from another instance.	

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	Result(Result &&other)	
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 noexcept	
Description:	Move-construct a new Result from another instance.	

⌋*(RS_AP_00130)*

[SWS_CORE_00727]{DRAFT} **Definition of API function ara::core::Result::~Result ⌈**

Kind:	function
Header file:	#include "ara/core/result.h"
Scope:	class ara::core::Result
Symbol:	~Result()
Syntax:	`~Result () noexcept;`
Exception Safety:	noexcept

▽

△

Description:	Destructor. This destructor is trivial if std::is_trivially_destructible<T>::value && std::is_trivially_destructible<E>::value is true.

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	FromValue(const T &t)	
Syntax:	`static Result FromValue (const T &t);`	
Parameters (in):	t	the value to put into the Result
Return value:	Result	a Result that contains the value t
Exception Safety:	not exception safe	
Description:	Build a new Result from the specified value (given as lvalue).	

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	FromValue(T &&t)	
Syntax:	`static Result FromValue (T &&t);`	
Parameters (in):	t	the value to put into the Result
Return value:	Result	a Result that contains the value t
Exception Safety:	not exception safe	
Description:	Build a new Result from the specified value (given as rvalue).	

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	FromValue(Args &&... args)	

Syntax:	`template <typename... Args>`
	`static Result FromValue (Args &&... args);`	

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:	not exception safe	

▽

△

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

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	FromError(const E &e)	
Syntax:	`static Result FromError (const E &e);`	
Parameters (in):	e	the error to put into the Result
Return value:	Result	a Result that contains the error e
Exception Safety:	not exception safe	
Description:	Build a new Result from the specified error (given as lvalue).	

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	FromError(E &&e)	
Syntax:	`static Result FromError (E &&e);`	
Parameters (in):	e	the error to put into the Result
Return value:	Result	a Result that contains the error e
Exception Safety:	not exception safe	
Description:	Build a new Result from the specified error (given as rvalue).	

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	FromError(Args &&... args)	

Syntax:	`template <typename... Args>`
	`static Result FromError (Args &&... args);`	

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:	not exception safe	

▽

△

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

⌋*(RS_AP_00130)*

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

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

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	operator=(Result &&other)	
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 noexcept	
Description:	Move-assign another Result to this instance.	

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	EmplaceValue(Args &&... args)	

Syntax:	`template <typename... Args>`
	`void EmplaceValue (Args &&... args);`	

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:	not exception safe	
Description:	Put a new value into this instance, constructed in-place from the given arguments.	

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	EmplaceError(Args &&... args)	
Syntax:	`template <typename... Args>` `void EmplaceError (Args &&... args);`	
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:	not exception safe	
Description:	Put a new error into this instance, constructed in-place from the given arguments.	

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	Swap(Result &other)	
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 noexcept	
Description:	Exchange the contents of this instance with those of other.	

⌋*(RS_AP_00130)*

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

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

▽

△

Exception Safety:	noexcept
Description:	Check whether *this contains a value.

⌋*(RS_AP_00130)*

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

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

⌋*(RS_AP_00130)*

[SWS_CORE_00753]{DRAFT} **Definition of API function ara::core::Result::operator* ⌈**

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	operator*()	
Syntax:	`const T & operator* () const &;`	
Return value:	const T &	a const_reference to the contained value
Exception Safety:	not exception safe	
Description:	Access the contained value. It shall be treated as a `Violation` if *this does not contain a value. The standardized log message is: "No value contained in this Result."	

⌋*(RS_AP_00130)*

[SWS_CORE_00774]{DRAFT} **Definition of API function ara::core::Result::operator* ⌈**

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	operator*()	
Syntax:	`T & operator* () &;`	
Return value:	T &	a reference to the contained value
Exception Safety:	not exception safe	
Description:	Access the contained value. It shall be treated as a `Violation` if *this does not contain a value. The standardized log message is: "No value contained in this Result."	

⌋*(RS_AP_00130)*

[SWS_CORE_00759]{DRAFT} **Definition of API function ara::core::Result::operator*** ⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	operator*()	
Syntax:	`T && operator* () &&;`	
Return value:	T &&	an rvalue reference to the contained value
Exception Safety:	not exception safe	
Description:	Access the contained value. It shall be treated as a `Violation` if *this does not contain a value. The standardized log message is: "No value contained in this Result."	

⌋*(RS_AP_00130)*

[SWS_CORE_00754]{DRAFT} **Definition of API function ara::core::Result::operator->** ⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	operator->()	
Syntax:	`const T * operator-> () const;`	
Return value:	const T *	a pointer to the contained value
Exception Safety:	not exception safe	
Description:	Access the contained value. It shall be treated as a `Violation` if *this does not contain a value. The standardized log message is: "No value contained in this Result."	

⌋*(RS_AP_00130)*

[SWS_CORE_00755]{DRAFT} **Definition of API function ara::core::Result::Value** ⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	Value()	
Syntax:	`const T & Value () const &;`	
Return value:	const T &	a const reference to the contained value
Exception Safety:	not exception safe	
Description:	Access the contained value. It shall be treated as a `Violation` if *this does not contain a value. The standardized log message is: "No value contained in this Result."	

⌋*(RS_AP_00130)*

[SWS_CORE_00775]{DRAFT} **Definition of API function ara::core::Result::Value** ⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	Value()	
Syntax:	`T & Value () &;`	
Return value:	T &	a reference to the contained value
Exception Safety:	not exception safe	
Description:	Access the contained value. It shall be treated as a `Violation` if *this does not contain a value. The standardized log message is: "No value contained in this Result."	

⌋*(RS_AP_00130)*

[SWS_CORE_00756]{DRAFT} **Definition of API function ara::core::Result::Value** ⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	Value()	
Syntax:	`T && Value () &&;`	
Return value:	T &&	an rvalue reference to the contained value
Exception Safety:	not exception safe	
Description:	Access the contained value. It shall be treated as a `Violation` if *this does not contain a value. The standardized log message is: "No value contained in this Result."	

⌋*(RS_AP_00130)*

[SWS_CORE_00757]{DRAFT} **Definition of API function ara::core::Result::Error** ⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	Error()	
Syntax:	`const E & Error () const &;`	
Return value:	const E &	a const reference to the contained error
Exception Safety:	not exception safe	
Description:	Access the contained error. It shall be treated as a `Violation` if *this does not contain an error. The standardized log message is: "No error contained in this Result."	

⌋*(RS_AP_00130)*

[SWS_CORE_00776]{DRAFT} Definition of API function ara::core::Result::Error ⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	Error()	
Syntax:	`E & Error () &;`	
Return value:	E &	a const reference to the contained error
Exception Safety:	not exception safe	
Description:	Access the contained error. It shall be treated as a `Violation` if *this does not contain an error. The standardized log message is: "No error contained in this Result."	

⌋*(RS_AP_00130)*

[SWS_CORE_00758]{DRAFT} Definition of API function ara::core::Result::Error ⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	Error()	
Syntax:	`E && Error () &&;`	
Return value:	E &&	an rvalue reference to the contained error
Exception Safety:	not exception safe	
Description:	Access the contained error. It shall be treated as a `Violation` if *this does not contain an error. The standardized log message is: "No error contained in this Result."	

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	Ok()	
Syntax:	`Optional< T > Ok () const &;`	
Return value:	Optional< T >	an Optional with the value, if present
Exception Safety:	not exception safe	
Description:	Return the contained value as an Optional.	

⌋*(RS_AP_00130)*

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

Kind:	function
Header file:	#include "ara/core/result.h"
Scope:	class ara::core::Result
Symbol:	Ok()

▽

△

Syntax:	`Optional< T > Ok () &&;`	
Return value:	Optional< T >	an Optional with the value, if present
Exception Safety:	not exception safe	
Description:	Return the contained value as an Optional.	

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	Err()	
Syntax:	`Optional< E > Err () const &;`	
Return value:	Optional< E >	an Optional with the error, if present
Exception Safety:	not exception safe	
Description:	Return the contained error as an Optional.	

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	Err()	
Syntax:	`Optional< E > Err () &&;`	
Return value:	Optional< E >	an Optional with the error, if present
Exception Safety:	not exception safe	
Description:	Return the contained error as an Optional.	

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	ValueOr(U &&defaultValue)	

Syntax:	`template <typename U>`
	`T ValueOr (U &&defaultValue) const &;`	

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:	not exception 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.	

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	ValueOr(U &&defaultValue)	

Syntax:	`template <typename U>`
	`T ValueOr (U &&defaultValue) &&;`	

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:	not exception 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.	

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	ErrorOr(G &&defaultError)	

Syntax:	`template <typename G>`
	`E ErrorOr (G &&defaultError) const &;`	

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:	not exception 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.	

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	ErrorOr(G &&defaultError)	

Syntax:	`template <typename G>`
	`E ErrorOr (G &&defaultError) &&;`	

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:	not exception 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.

⌋(*RS_AP_00130*)

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	CheckError(G &&error)	

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	
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().	

⌋(*RS_AP_00130*)

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	ValueOrThrow()	
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
Description:	Return the contained value or throw an exception. This function does not participate in overload resolution when the compiler toolchain does not support C++ exceptions.	

⌋(*RS_AP_00130*)

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	ValueOrThrow()	
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
Description:	Return the contained value or throw an exception. This function does not participate in overload resolution when the compiler toolchain does not support C++ exceptions.	

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	Resolve(F &&f)	

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	
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&);`	

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result	
Symbol:	Bind(F &&f)	

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	

▽

△

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.

⌋*(RS_AP_00130)*

8.1.4.1 `Result<void, E>` template specialization

This section defines the interface of the `ara::core::Result` template specialization where the type T is “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.

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

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.	

⌋*(RS_AP_00130)*

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

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 .

⌋*(RS_AP_00130)*

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

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 .

⌋*(RS_AP_00130)*

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

Kind:	function
Header file:	#include "ara/core/result.h"
Scope:	class ara::core::Result< void, E >
Symbol:	Result()
Syntax:	`Result () noexcept;`
Exception Safety:	noexcept
Description:	Construct a new Result with a "void" value.

⌋*(RS_AP_00130)*

[SWS_CORE_00823] Definition of API function ara::core::Result< void, E >::Result ⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result< void, E >	
Symbol:	Result(const E &e)	
Syntax:	`explicit Result (const E &e);`	
Parameters (in):	e	the error to put into the Result
Exception Safety:	not exception safe	
Description:	Construct a new Result from the specified error (given as lvalue).	

⌋*(RS_AP_00130)*

[SWS_CORE_00824] Definition of API function ara::core::Result< void, E >::Result ⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result< void, E >	
Symbol:	Result(E &&e)	
Syntax:	`explicit Result (E &&e);`	
Parameters (in):	e	the error to put into the Result
Exception Safety:	not exception safe	
Description:	Construct a new Result from the specified error (given as rvalue).	

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result< void, E >	
Symbol:	Result(const Result &other)	
Syntax:	`Result (const Result &other);`	
Parameters (in):	other	the other instance
Exception Safety:	not exception safe	
Description:	Copy-construct a new Result from another instance.	

⌋*(RS_AP_00130)*

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

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

⌋*(RS_AP_00130)*

[SWS_CORE_00827]{DRAFT} Definition of API function ara::core::Result< void, E >::~Result ⌈

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

▽

△

Symbol:	~Result()
Syntax:	`~Result () noexcept;`
Exception Safety:	noexcept
Description:	Destructor. This destructor is trivial if std::is_trivially_destructible<E>::value is true.

⌋*(RS_AP_00130)*

[SWS_CORE_00831] Definition of API function ara::core::Result< void, E >::FromValue ⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result< void, E >	
Symbol:	FromValue()	
Syntax:	`static Result FromValue () noexcept;`	
Return value:	Result	a Result that contains a "void" value
Exception Safety:	noexcept	
Description:	Build a new Result with "void" as value.	

⌋*(RS_AP_00130)*

[SWS_CORE_00834] Definition of API function ara::core::Result< void, E >::FromError ⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result< void, E >	
Symbol:	FromError(const E &e)	
Syntax:	`static Result FromError (const E &e);`	
Parameters (in):	e	the error to put into the Result
Return value:	Result	a Result that contains the error e
Exception Safety:	not exception safe	
Description:	Build a new Result from the specified error (given as lvalue).	

⌋*(RS_AP_00130)*

[SWS_CORE_00835] Definition of API function ara::core::Result< void, E >::FromError ⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result< void, E >	
Symbol:	FromError(E &&e)	
Syntax:	`static Result FromError (E &&e);`	
Parameters (in):	e	the error to put into the Result
Return value:	Result	a Result that contains the error e

▽

△

Exception Safety:	not exception safe
Description:	Build a new Result from the specified error (given as rvalue).

⌋*(RS_AP_00130)*

[SWS_CORE_00836] Definition of API function ara::core::Result< void, E >::From Error ⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result< void, E >	
Symbol:	FromError(Args &&... args)	

Syntax:	`template <typename... Args>`
	`static Result FromError (Args &&... args);`	

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:	not exception safe	
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	

⌋*(RS_AP_00130)*

[SWS_CORE_00841] Definition of API function ara::core::Result< void, E >::operator= ⌈

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

⌋*(RS_AP_00130)*

[SWS_CORE_00842] Definition of API function ara::core::Result< void, E >::operator= ⌈

Kind:	function
Header file:	#include "ara/core/result.h"
Scope:	class ara::core::Result< void, E >
Symbol:	operator=(Result &&other)

▽

△

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 noexcept	
Description:	Move-assign another Result to this instance.	

⌋*(RS_AP_00130)*

[SWS_CORE_00843] Definition of API function ara::core::Result< void, E >::EmplaceValue ⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result< void, E >	
Symbol:	EmplaceValue(Args &&... args)	

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:	noexcept	
Description:	Put a new value into this instance, constructed in-place from the given arguments.	

⌋*(RS_AP_00130)*

[SWS_CORE_00844] Definition of API function ara::core::Result< void, E >::EmplaceError ⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result< void, E >	
Symbol:	EmplaceError(Args &&... args)	

Syntax:	`template <typename... Args>`
	`void EmplaceError (Args &&... args);`	

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:	not exception safe	
Description:	Put a new error into this instance, constructed in-place from the given arguments.	

⌋*(RS_AP_00130)*

[SWS_CORE_00845] Definition of API function ara::core::Result< void, E >::Swap
⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result< void, E >	
Symbol:	Swap(Result &other)	
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 noexcept	
Description:	Exchange the contents of this instance with those of other.	

⌋*(RS_AP_00130)*

[SWS_CORE_00851] Definition of API function ara::core::Result< void, E >::HasValue ⌈

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

⌋*(RS_AP_00130)*

[SWS_CORE_00852] Definition of API function ara::core::Result< void, E >::operator bool ⌈

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

⌋*(RS_AP_00130)*

[SWS_CORE_00853] Definition of API function ara::core::Result< void, E >::operator* ⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result< void, E >	
Symbol:	operator*()	
Syntax:	`void operator* () const;`	
Return value:	None	
Exception Safety:	not exception safe	
Description:	Access the contained value. It shall be treated as a `Violation` if *this does not contain a value. The standardized log message is: "No value contained in this Result."	

⌋*(RS_AP_00130)*

[SWS_CORE_00855] Definition of API function ara::core::Result< void, E >::Value ⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result< void, E >	
Symbol:	Value()	
Syntax:	`void Value () const;`	
Return value:	None	
Exception Safety:	not exception safe	
Description:	This function only exists for helping with generic programming. It shall be treated as a `Violation` if *this does not contain a value. The standardized log message is: "No value contained in this Result."	

⌋*(RS_AP_00130)*

[SWS_CORE_00857] Definition of API function ara::core::Result< void, E >::Error ⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result< void, E >	
Symbol:	Error()	
Syntax:	`const E & Error () const &;`	
Return value:	const E &	a const reference to the contained error
Exception Safety:	not exception safe	
Description:	Access the contained error. It shall be treated as a `Violation` if *this does not contain an error. The standardized log message is: "No error contained in this Result."	

⌋*(RS_AP_00130)*

[SWS_CORE_00876]{DRAFT} **Definition of API function ara::core::Result< void, E >::Error** ⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result< void, E >	
Symbol:	Error()	
Syntax:	`E & Error () &;`	
Return value:	E &	a const reference to the contained error
Exception Safety:	not exception safe	
Description:	Access the contained error. It shall be treated as a `Violation` if *this does not contain an error. The standardized log message is: "No error contained in this Result."	

⌋*(RS_AP_00130)*

[SWS_CORE_00858] Definition of API function ara::core::Result< void, E >::Error ⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result< void, E >	
Symbol:	Error()	
Syntax:	`E && Error () &&;`	
Return value:	E &&	an rvalue reference to the contained error
Exception Safety:	not exception safe	
Description:	Access the contained error. It shall be treated as a `Violation` if *this does not contain an error. The standardized log message is: "No error contained in this Result."	

⌋*(RS_AP_00130)*

[SWS_CORE_00868] Definition of API function ara::core::Result< void, E >::Err ⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result< void, E >	
Symbol:	Err()	
Syntax:	`Optional< E > Err () const &;`	
Return value:	Optional< E >	an Optional with the error, if present
Exception Safety:	not exception safe	
Description:	Return the contained error as an Optional.	

⌋*(RS_AP_00130)*

[SWS_CORE_00869] Definition of API function ara::core::Result< void, E >::Err ⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result< void, E >	
Symbol:	Err()	
Syntax:	`Optional< E > Err () &&;`	
Return value:	Optional< E >	an Optional with the error, if present
Exception Safety:	not exception safe	
Description:	Return the contained error as an Optional.	

⌋*(RS_AP_00130)*

[SWS_CORE_00861] Definition of API function ara::core::Result< void, E >::ValueOr ⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result< void, E >	
Symbol:	ValueOr(U &&defaultValue)	

Syntax:	`template <typename U>`
	`void ValueOr (U &&defaultValue) const;`	

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:	not exception safe	
Description:	Do nothing. This function only exists for helping with generic programming.	

⌋*(RS_AP_00130)*

[SWS_CORE_00863] Definition of API function ara::core::Result< void, E >::ErrorOr ⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result< void, E >	
Symbol:	ErrorOr(G &&defaultError)	

Syntax:	`template <typename G>`
	`E ErrorOr (G &&defaultError) const &;`	

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:	not exception 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.	

⌋*(RS_AP_00130)*

[SWS_CORE_00864] Definition of API function ara::core::Result< void, E >::ErrorOr ⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result< void, E >	
Symbol:	ErrorOr(G &&defaultError)	

Syntax:	`template <typename G>`
	`E ErrorOr (G &&defaultError) &&;`	

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:	not exception 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.	

⌋*(RS_AP_00130)*

[SWS_CORE_00865] Definition of API function ara::core::Result< void, E >::CheckError ⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result< void, E >	
Symbol:	CheckError(G &&error)	

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	
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().	

⌋*(RS_AP_00130)*

[SWS_CORE_00866] Definition of API function ara::core::Result< void, E >::ValueOrThrow ⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result< void, E >	
Symbol:	ValueOrThrow()	
Syntax:	`void ValueOrThrow () const noexcept(false);`	
Return value:	None	
Exceptions:	<TYPE>	the exception type associated with the contained error

▽

△

Description:	Return the contained value or throw an exception. This function does not participate in overload resolution when the compiler toolchain does not support C++ exceptions.

⌋*(RS_AP_00130)*

[SWS_CORE_00867] Definition of API function ara::core::Result< void, E >::Resolve ⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result< void, E >	
Symbol:	Resolve(F &&f)	

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	
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.	

⌋*(RS_AP_00130)*

[SWS_CORE_00870] Definition of API function ara::core::Result< void, E >::Bind ⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	class ara::core::Result< void, E >	
Symbol:	Bind(F &&f)	

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	

▽

△

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.

⌋*(RS_AP_00130)*

8.1.4.2 Non-member function overloads

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	namespace ara::core	
Symbol:	operator==(const Result< T, E > &lhs, const Result< T, E > &rhs)	

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

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	namespace ara::core	
Symbol:	operator!=(const Result< T, E > &lhs, const Result< T, E > &rhs)	

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
Description:	Compare two Result instances for inequality. A Result that contains a value is unequal to every Result containing an error. A Result is equal to another Result only if both contain the same type, and the value of that type compares equal.

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	namespace ara::core	
Symbol:	operator==(const Result< T, E > &lhs, const T &rhs)	

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	
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.	

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	namespace ara::core	
Symbol:	operator==(const T &lhs, const Result< T, E > &rhs)	

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	
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.	

⌋*(RS_AP_00130)*

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

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

▽

△

Symbol:	operator!=(const Result< T, E > &lhs, const T &rhs)	
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	
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.	

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	namespace ara::core	
Symbol:	operator!=(const T &lhs, const Result< T, E > &rhs)	
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	
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.	

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	namespace ara::core	
Symbol:	operator==(const Result< T, E > &lhs, const E &rhs)	
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	
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.	

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	namespace ara::core	
Symbol:	operator==(const E &lhs, const Result< T, E > &rhs)	

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	
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.	

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	namespace ara::core	
Symbol:	operator!=(const Result< T, E > &lhs, const E &rhs)	

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	
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.	

⌋*(RS_AP_00130)*

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	namespace ara::core	
Symbol:	operator!=(const E &lhs, const Result< T, E > &rhs)	

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	

▽

△

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.

⌋(*RS_AP_00130*)

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

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	namespace ara::core	
Symbol:	swap(Result< T, E > &lhs, Result< T, E > &rhs)	

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 noexcept	
Description:	Swap the contents of the two given arguments.	

⌋(*RS_AP_00130*)

8.1.5 Core Error Domain

This section describes the `ara::core::CoreErrorDomain` type that derives from `ara::core::ErrorDomain` and contains the errors that can originate from within the CORE Functional Cluster.

8.1.5.1 CORE error codes

[SWS_CORE_05200] Definition of API enum ara::core::CoreErrc ⌈

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
	kInvalidMetaModel Shortname= 137	given string is not a valid model element shortname
	kInvalidMetaModelPath= 138	missing or invalid path to model element

▽

△

Description:	An enumeration that defines all errors of the CORE Functional Cluster.

⌋(*RS_AP_00130*)

8.1.5.2 `CoreException` type

[SWS_CORE_05211] Definition of API class ara::core::CoreException ⌈

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 CORE errors.

⌋(*RS_AP_00130*)

[SWS_CORE_05212] Definition of API function ara::core::CoreException::CoreException ⌈

Kind:	function	
Header file:	#include "ara/core/core_error_domain.h"	
Scope:	class ara::core::CoreException	
Symbol:	CoreException(ErrorCode err)	
Syntax:	`explicit CoreException (ErrorCode err) noexcept;`	
Parameters (in):	err	the ErrorCode
Exception Safety:	noexcept	
Description:	Construct a new CoreException from an ErrorCode.	

⌋(*RS_AP_00130*)

8.1.5.3 `CoreErrorDomain` type

[SWS_CORE_05221] Definition of API class ara::core::CoreErrorDomain ⌈

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:	0x8000'0000'0000'0014
Description:	An error domain for errors originating from the CORE Functional Cluster .

⌋(*RS_AP_00130*)

[SWS_CORE_05231] Definition of API type ara::core::CoreErrorDomain::Errc ⌈

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.

⌋(*RS_AP_00130*)

[SWS_CORE_05232] Definition of API type ara::core::CoreErrorDomain::Exception ⌈

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.

⌋(*RS_AP_00130*)

[SWS_CORE_05241] Definition of API function ara::core::CoreErrorDomain::CoreErrorDomain ⌈

Kind:	function
Header file:	#include "ara/core/core_error_domain.h"
Scope:	class ara::core::CoreErrorDomain
Symbol:	CoreErrorDomain()
Syntax:	`constexpr CoreErrorDomain () noexcept;`
Exception Safety:	noexcept
Description:	Default constructor.

⌋(*RS_AP_00130*)

[SWS_CORE_05242] Definition of API function ara::core::CoreErrorDomain::Name ⌈

Kind:	function
Header file:	#include "ara/core/core_error_domain.h"
Scope:	class ara::core::CoreErrorDomain
Symbol:	Name()

▽

△

Syntax:	`const char * Name () const noexcept override;`	
Return value:	const char *	"Core"
Exception Safety:	noexcept	
Description:	Return the "shortname" ApApplicationErrorDomain.SN of this error domain.	

⌋*(RS_AP_00130)*

[SWS_CORE_05243] Definition of API function ara::core::CoreErrorDomain::Message ⌈

Kind:	function	
Header file:	#include "ara/core/core_error_domain.h"	
Scope:	class ara::core::CoreErrorDomain	
Symbol:	Message(ErrorDomain::CodeType errorCode)	
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:	noexcept	
Description:	Translate an error code value into a text message.	

⌋*(RS_AP_00130)*

[SWS_CORE_05244] Definition of API function ara::core::CoreErrorDomain::ThrowAsException ⌈

Kind:	function	
Header file:	#include "ara/core/core_error_domain.h"	
Scope:	class ara::core::CoreErrorDomain	
Symbol:	ThrowAsException(const ErrorCode &errorCode)	
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
Description:	Throw the exception type corresponding to the given ErrorCode.	

⌋*(RS_AP_00130)*

8.1.5.4 `GetCoreErrorDomain` accessor function

[SWS_CORE_05280] Definition of API function ara::core::GetCoreErrorDomain ⌈

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

▽

△

Symbol:	GetCoreErrorDomain()	
Syntax:	`constexpr const ErrorDomain & GetCoreErrorDomain () noexcept;`	
Return value:	const ErrorDomain &	the CoreErrorDomain
Exception Safety:	noexcept	
Description:	Return a reference to the global CoreErrorDomain.	

⌋*(RS_AP_00130)*

8.1.5.5 `MakeErrorCode` overload for `CoreErrorDomain`

[SWS_CORE_05290] Definition of API function ara::core::MakeErrorCode ⌈

Kind:	function	
Header file:	#include "ara/core/core_error_domain.h"	
Scope:	namespace ara::core	
Symbol:	MakeErrorCode(CoreErrc code, ErrorDomain::SupportDataType data)	
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:	noexcept	
Description:	Create a new ErrorCode within CoreErrorDomain. This function is used internally by constructors of ErrorCode. It is usually not used directly by users.	

⌋*(RS_AP_00130)*

8.1.6 `Future` and `Promise` data types

This section describes the `Future` and `Promise` class templates used in `ara::core` to provide and retrieve the results of asynchronous method calls.

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 [4]. Whenever there is a mention of an experimental C++ item such as `std::experimental::future::is_ready`, the implied source material is [11].

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 [4], 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 "If the type of error E = ErrorCode the provider stores the ErrorCode broken_promise defined in [SWS_CORE_00400] in its `shared state`; Otherwise the behavior is implementation-defined. The provider should store an implementation-defined error that corresponds to broken_promise 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 [4, the C++14 standard] and the corresponding `std::experimental::` classes from [11], except for the deviations from the `std::` classes that result from the integration with `ara::core::Result`.

8.1.6.1 `future_errc` enumeration

[SWS_CORE_00400] Definition of API enum ara::core::future_errc ⌈

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:	future_errc	
Underlying type:	std::int32_t	
Syntax:	`enum class future_errc : std::int32_t {...};`	
Values:	broken_promise= 101	the asynchronous task abandoned its `shared state`
	no_state= 104	attempt to access Promise or Future without an associated `shared state`
Description:	Specifies the errors that can occur upon calling Future::get or Future::GetResult.	

⌋*(RS_AP_00130)*

8.1.6.2 `FutureException` type

[SWS_CORE_00411] Definition of API class ara::core::FutureException ⌈

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.

⌋*(RS_AP_00130)*

[SWS_CORE_00412] Definition of API function ara::core::FutureException::FutureException ⌈

Kind:	function	
Header file:	#include "ara/core/future_error_domain.h"	
Scope:	class ara::core::FutureException	
Symbol:	FutureException(ErrorCode err)	
Syntax:	`explicit FutureException (ErrorCode err) noexcept;`	
Parameters (in):	err	the ErrorCode
Exception Safety:	noexcept	
Description:	Construct a new FutureException from an ErrorCode.	

⌋*(RS_AP_00130)*

8.1.6.3 `FutureErrorDomain` type

[SWS_CORE_00421] Definition of API class ara::core::FutureErrorDomain ⌈

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:	0x8000'0000'0000'0013
Description:	Error domain for errors originating from classes Future and Promise.

⌋*(RS_AP_00130)*

[SWS_CORE_00431] Definition of API type ara::core::FutureErrorDomain::Errc ⌈

Kind:	type alias
Header file:	#include "ara/core/future_error_domain.h"
Scope:	class ara::core::FutureErrorDomain
Symbol:	Errc
Syntax:	`using Errc = future_errc;`
Description:	Alias for the error code value enumeration.

⌋*(RS_AP_00130)*

[SWS_CORE_00432] Definition of API type ara::core::FutureErrorDomain::Exception ⌈

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.

⌋*(RS_AP_00130)*

[SWS_CORE_00441] Definition of API function ara::core::FutureErrorDomain::FutureErrorDomain ⌈

Kind:	function
Header file:	#include "ara/core/future_error_domain.h"
Scope:	class ara::core::FutureErrorDomain
Symbol:	FutureErrorDomain()
Syntax:	`constexpr FutureErrorDomain () noexcept;`
Exception Safety:	noexcept
Description:	Default constructor.

⌋*(RS_AP_00130)*

[SWS_CORE_00442] Definition of API function ara::core::FutureErrorDomain::Name ⌈

Kind:	function	
Header file:	#include "ara/core/future_error_domain.h"	
Scope:	class ara::core::FutureErrorDomain	
Symbol:	Name()	
Syntax:	`const char * Name () const noexcept override;`	
Return value:	const char *	"Future"
Exception Safety:	noexcept	
Description:	Return the "shortname" ApApplicationErrorDomain.SN of this error domain.	

⌋*(RS_AP_00130)*

[SWS_CORE_00443] Definition of API function ara::core::FutureErrorDomain::Message ⌈

Kind:	function	
Header file:	#include "ara/core/future_error_domain.h"	
Scope:	class ara::core::FutureErrorDomain	
Symbol:	Message(ErrorDomain::CodeType errorCode)	
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:	noexcept	
Description:	Translate an error code value into a text message.	

⌋*(RS_AP_00130)*

[SWS_CORE_00444] Definition of API function ara::core::FutureErrorDomain::ThrowAsException ⌈

Kind:	function	
Header file:	#include "ara/core/future_error_domain.h"	
Scope:	class ara::core::FutureErrorDomain	
Symbol:	ThrowAsException(const ErrorCode &errorCode)	
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
Description:	Throw the exception type corresponding to the given ErrorCode.	

⌋*(RS_AP_00130)*

8.1.6.4 `FutureErrorDomain` accessor function

[SWS_CORE_00480] Definition of API function ara::core::GetFutureErrorDomain ⌈

Kind:	function	
Header file:	#include "ara/core/future_error_domain.h"	
Scope:	namespace ara::core	
Symbol:	GetFutureErrorDomain()	
Syntax:	`constexpr const ErrorDomain & GetFutureErrorDomain () noexcept;`	
Return value:	const ErrorDomain &	reference to the FutureErrorDomain instance
Exception Safety:	noexcept	
Description:	Obtain the reference to the single global FutureErrorDomain instance.	

⌋*(RS_AP_00130)*

8.1.6.5 `MakeErrorCode` overload for `FutureErrorDomain`

[SWS_CORE_00490] Definition of API function ara::core::MakeErrorCode ⌈

Kind:	function	
Header file:	#include "ara/core/future_error_domain.h"	
Scope:	namespace ara::core	
Symbol:	MakeErrorCode(future_errc code, ErrorDomain::SupportDataType data)	
Syntax:	`constexpr ErrorCode MakeErrorCode (future_errc code, ErrorDomain::SupportDataType data) noexcept;`	
Parameters (in):	code	an enumeration value from future_errc
	data	a vendor-defined supplementary value
Return value:	ErrorCode	the new ErrorCode instance
Exception Safety:	noexcept	
Description:	Create a new ErrorCode for FutureErrorDomain with the given support data type.	

⌋*(RS_AP_00130)*

8.1.6.6 `future_status` enumeration

[SWS_CORE_00361]{DRAFT} **Definition of API enum ara::core::future_status ⌈**

Kind:	enumeration	
Header file:	#include "ara/core/future.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	namespace ara::core	
Symbol:	future_status	
Underlying type:	std::uint8_t	
Syntax:	`enum class future_status : std::uint8_t {...};`	
Values:	ready	the `shared state` is ready
	timeout	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.	

⌋*(RS_AP_00130)*

8.1.6.7 `Future` data type

[SWS_CORE_00321]{DRAFT} **Definition of API class ara::core::Future** ⌈

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.	

⌋*(RS_AP_00130)*

[SWS_CORE_00322]{DRAFT} **Definition of API function ara::core::Future::Future** ⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	class ara::core::Future
Symbol:	Future()
Syntax:	`Future () noexcept=default;`
Exception Safety:	noexcept
Description:	Default constructor. This function shall behave the same as the corresponding std::future function.

⌋*(RS_AP_00130)*

[SWS_CORE_00334]{DRAFT} **Definition of API function ara::core::Future::Future** ⌈

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

⌋*(RS_AP_00130)*

[SWS_CORE_00323]{DRAFT} **Definition of API function ara::core::Future::Future** ⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	class ara::core::Future

▽

△

Symbol:	Future(Future &&other)	
Syntax:	`Future (Future &&other) noexcept;`	
Parameters (in):	other	the other instance
Exception Safety:	noexcept	
Description:	Move construct from another instance. This function shall behave the same as the corresponding std::future function.	

⌋*(RS_AP_00130)*

[SWS_CORE_00333]{DRAFT} **Definition of API function ara::core::Future::~Future** ⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	class ara::core::Future
Symbol:	~Future()
Syntax:	`~Future () noexcept;`
Exception Safety:	noexcept
Description:	Destructor for Future objects. Abandons any `shared state`.

⌋*(RS_AP_00130)*

[SWS_CORE_00335]{DRAFT} **Definition of API function ara::core::Future::operator=** ⌈

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

⌋*(RS_AP_00130)*

[SWS_CORE_00325]{DRAFT} **Definition of API function ara::core::Future::operator=** ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Future	
Symbol:	operator=(Future &&other)	
Syntax:	`Future & operator= (Future &&other) noexcept;`	
Parameters (in):	other	the other instance
Return value:	Future &	*this
Exception Safety:	noexcept	
Description:	Move assign from another instance. This function shall behave the same as the corresponding std::future function.	

⌋*(RS_AP_00130)*

[SWS_CORE_00326]{DRAFT} **Definition of API function ara::core::Future::get** ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Future	
Symbol:	get()	
Syntax:	`T get ();`	
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::Set Result or • the Promise was broken, meaning the `shared state` was abandoned by the corresponding Promise. Then if E=ErrorCode the error is broken_promise as defined in [SWS_CORE_00400], otherwise it is implementation-defined.
	FutureException	in case the Future is invalid. The contained ErrorCode is no_state
Description:	Get the value. This function shall behave the same as the corresponding std::future function. This function does not participate in overload resolution when the compiler toolchain does not support C++ exceptions.	

⌋*(RS_AP_00130)*

[SWS_CORE_00336]{DRAFT} **Definition of API function ara::core::Future::GetResult** ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Future	
Symbol:	GetResult()	
Syntax:	`Result< T, E > GetResult () noexcept;`	
Return value:	Result< T, E >	a Result 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::Set Result or • the Promise was broken, meaning the `shared state` was abandoned by the corresponding Promise. Then if E=ErrorCode the error is broken_promise as defined in [SWS_CORE_00400], otherwise it is implementation-defined.
Exception Safety:	noexcept	
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. It shall be treated as a `Violation` if the Future is invalid (valid returns false). The standardized log message is: "Calling GetResult() on an invalid Future is not allowed."	

⌋*(RS_AP_00130)*

[SWS_CORE_00327]{DRAFT} Definition of API function ara::core::Future::valid ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Future	
Symbol:	valid()	
Syntax:	`bool valid () const noexcept;`	
Return value:	bool	true if the Future is usable, false otherwise
Exception Safety:	noexcept	
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.	

⌋*(RS_AP_00130)*

[SWS_CORE_00328]{DRAFT} Definition of API function ara::core::Future::wait ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Future	
Symbol:	wait()	
Syntax:	`void wait () const;`	
Return value:	None	
Exception Safety:	not exception safe	
Description:	Wait for a value or an error to be available. This function shall behave the same as the corresponding std::future function. It shall be treated as a `Violation` if the Future is invalid (valid returns false). The standardized log message is: "Calling wait() on an invalid Future is not allowed."	

⌋*(RS_AP_00130)*

[SWS_CORE_00329]{DRAFT} Definition of API function ara::core::Future::wait_for ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Future	
Symbol:	wait_for(const std::chrono::duration< Rep, Period > &timeoutDuration)	

Syntax:	`template <typename Rep, typename Period>`
	`future_status wait_for (const std::chrono::duration< Rep, Period > &timeoutDuration) const;`	

Parameters (in):	timeoutDuration	maximal duration to wait for
Return value:	future_status	status that indicates whether the timeout hit or if a value is available
Exception Safety:	not exception safe	
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. It shall be treated as a `Violation` if the Future is invalid (valid returns false). The standardized log message is: "Calling wait_for() on an invalid Future is not allowed."	

⌋*(RS_AP_00130)*

[SWS_CORE_00330]{DRAFT} Definition of API function ara::core::Future::wait_until ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Future	
Symbol:	wait_until(const std::chrono::time_point< Clock, Duration > &deadline)	
Syntax:	`template <typename Clock, typename Duration>` `future_status wait_until (const std::chrono::time_point< Clock, Duration > &deadline) const;`	
Parameters (in):	deadline	latest point in time to wait
Return value:	future_status	status that indicates whether the time was reached or if a value is available
Exception Safety:	not exception safe	
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. It shall be treated as a `Violation` if the Future is invalid (valid returns false). The standardized log message is: "Calling wait_until() on an invalid Future is not allowed."	

⌋(*RS_AP_00130*)

[SWS_CORE_00331]{DRAFT} Definition of API function ara::core::Future::then ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Future	
Symbol:	then(F &&func)	
Syntax:	`template <typename F>` `auto then (F &&func) -> 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:	not exception safe	
Description:	Register a callable that gets called when the Future becomes ready. 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>. It shall be treated as a `Violation` if the Future is invalid (valid returns false). The standardized log message is: "Calling then() on an invalid Future is not allowed." The continuation shall not throw, except for the purpose of implementing a `Violation`. If the continuation throws this shall be treated as a `Violation` with the message "The continuation ▽	

▽

△

	△ given to Future::then threw an exception with the explanation: >explanatory string of the exception if available< " 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.

⌋*(RS_AP_00130)*

[SWS_CORE_00337]{DRAFT} **Definition of API function ara::core::Future::then** ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Future	
Symbol:	then(F &&func, ExecutorT &&executor)	

Syntax:	`template <typename F, typename ExecutorT>`
	`auto then (F &&func, ExecutorT &&executor) -> 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:	not exception safe	
Description:	Register a callable that gets called when the Future becomes ready. 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>. It shall be treated as a `Violation` if the Future is invalid (valid returns false). The standardized log message is: "Calling then() on an invalid Future is not allowed." The continuation shall not throw, except for the purpose of implementing a `Violation`. If the continuation throws this shall be treated as a `Violation` with the message "The continuation given to Future::then threw an exception with the explanation: >explanatory string of the exception if available< " 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.	

⌋*(RS_AP_00130)*

[SWS_CORE_00332]{DRAFT} **Definition of API function ara::core::Future::is_ready** ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Future	
Symbol:	is_ready()	
Syntax:	`bool is_ready () const;`	
Return value:	bool	true if the Future contains a value or an error, false otherwise
Exception Safety:	not exception safe	
Description:	Return whether the asynchronous operation has finished. If this function returns true, get(), GetResult() and the wait calls are guaranteed not to block. It shall be treated as a `Violation` if the Future is invalid (valid returns false). The standardized log message is: "Calling is_ready() on an invalid Future is not allowed."	

⌋*(RS_AP_00130)*

8.1.6.7.1 `Future<void, E>` template specialization

This section defines the interface of the `ara::core::Future<T,E>` template specialization where the type T is `void`.

[SWS_CORE_06221] Definition of API class ara::core::Future< void, E > ⌈

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.	

⌋*(RS_AP_00130)*

[SWS_CORE_06222] Definition of API function ara::core::Future< void, E >::Future ⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	class ara::core::Future< void, E >
Symbol:	Future()
Syntax:	`Future () noexcept;`
Exception Safety:	noexcept
Description:	Default constructor. This function shall behave the same as the corresponding std::future function.

⌋*(RS_AP_00130)*

[SWS_CORE_06234] Definition of API function ara::core::Future< void, E >::Future ⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	class ara::core::Future< void, E >
Symbol:	Future(const Future &other)
Syntax:	`Future (const Future &other)=delete;`
Description:	Copy constructor shall be disabled.

⌋*(RS_AP_00130)*

[SWS_CORE_06223] Definition of API function ara::core::Future< void, E >::Future ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Future< void, E >	
Symbol:	Future(Future &&other)	
Syntax:	`Future (Future &&other) noexcept;`	
Parameters (in):	other	the other instance
Exception Safety:	noexcept	
Description:	Move construct from another instance. This function shall behave the same as the corresponding std::future function.	

⌋*(RS_AP_00130)*

[SWS_CORE_06233]{DRAFT} **Definition of API function ara::core::Future< void, E >::~Future ⌈**

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	class ara::core::Future< void, E >
Symbol:	~Future()
Syntax:	`~Future () noexcept;`
Exception Safety:	noexcept
Description:	Destructor for Future objects. Abandons any `shared state`.

⌋*(RS_AP_00130)*

[SWS_CORE_06235] Definition of API function ara::core::Future< void, E >::operator= ⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	class ara::core::Future< void, E >
Symbol:	operator=(const Future &other)

▽

△

Syntax:	`Future & operator= (const Future &other)=delete;`
Description:	Copy assignment operator shall be disabled.

⌋*(RS_AP_00130)*

[SWS_CORE_06225] Definition of API function ara::core::Future< void, E >::operator= ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Future< void, E >	
Symbol:	operator=(Future &&other)	
Syntax:	`Future & operator= (Future &&other) noexcept;`	
Parameters (in):	other	the other instance
Return value:	Future &	*this
Exception Safety:	noexcept	
Description:	Move assign from another instance. This function shall behave the same as the corresponding std::future function.	

⌋*(RS_AP_00130)*

[SWS_CORE_06226] Definition of API function ara::core::Future< void, E >::get ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Future< void, E >	
Symbol:	get()	
Syntax:	`void get ();`	
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::Set Result or • the Promise was broken, meaning the `shared state` was abandoned by the corresponding Promise. Then if E=ErrorCode the error is broken_promise as defined in [SWS_CORE_00400], otherwise it is implementation-defined.
	FutureException	in case the Future is invalid. The contained ErrorCode is no_state
Description:	Get the value. This function shall behave the same as the corresponding std::future function. This function does not participate in overload resolution when the compiler toolchain does not support C++ exceptions.	

⌋*(RS_AP_00130)*

[SWS_CORE_06236] Definition of API function ara::core::Future< void, E >::GetResult ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Future< void, E >	
Symbol:	GetResult()	
Syntax:	`Result< void, E > GetResult () noexcept;`	
Return value:	Result< void, E >	a Result 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 if E=ErrorCode the error is broken_promise as defined in [SWS_CORE_00400], otherwise it is implementation-defined.
Exception Safety:	noexcept	
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. It shall be treated as a `Violation` if the Future is invalid (valid returns false). The standardized log message is: "Calling GetResult() on an invalid Future is not allowed."	

⌋*(RS_AP_00130)*

[SWS_CORE_06227] Definition of API function ara::core::Future< void, E >::valid ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Future< void, E >	
Symbol:	valid()	
Syntax:	`bool valid () const noexcept;`	
Return value:	bool	true if the Future is usable, false otherwise
Exception Safety:	noexcept	
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.	

⌋*(RS_AP_00130)*

[SWS_CORE_06228] Definition of API function ara::core::Future< void, E >::wait ⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	class ara::core::Future< void, E >
Symbol:	wait()
Syntax:	`void wait () const;`
Return value:	None
Exception Safety:	not exception safe

▽

△

Description:	Wait for a value or an error to be available. This function shall behave the same as the corresponding std::future function. It shall be treated as a `Violation` if the Future is invalid (valid returns false). The standardized log message is: "Calling wait() on an invalid Future is not allowed."

⌋*(RS_AP_00130)*

[SWS_CORE_06229] Definition of API function ara::core::Future< void, E >::wait_for ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Future< void, E >	
Symbol:	wait_for(const std::chrono::duration< Rep, Period > &timeoutDuration)	

Syntax:	`template <typename Rep, typename Period>`
	`future_status wait_for (const std::chrono::duration< Rep, Period > &timeoutDuration) const;`	

Parameters (in):	timeoutDuration	maximal duration to wait for
Return value:	future_status	status that indicates whether the timeout hit or if a value is available
Exception Safety:	not exception safe	
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. It shall be treated as a `Violation` if the Future is invalid (valid returns false). The standardized log message is: "Calling wait_for() on an invalid Future is not allowed."	

⌋*(RS_AP_00130)*

[SWS_CORE_06230] Definition of API function ara::core::Future< void, E >::wait_until ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Future< void, E >	
Symbol:	wait_until(const std::chrono::time_point< Clock, Duration > &deadline)	

Syntax:	`template <typename Clock, typename Duration>`
	`future_status wait_until (const std::chrono::time_point< Clock, Duration > &deadline) const;`	

Parameters (in):	deadline	latest point in time to wait
Return value:	future_status	status that indicates whether the time was reached or if a value is available
Exception Safety:	not exception safe	
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. It shall be treated as a `Violation` if the Future is invalid (valid returns false). The standardized log message is: "Calling wait_until() on an invalid Future is not allowed."	

⌋*(RS_AP_00130)*

[SWS_CORE_06231]{DRAFT} **Definition of API function ara::core::Future< void, E >::then** ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Future< void, E >	
Symbol:	then(F &&func)	

Syntax:	`template <typename F>`
	`auto then (F &&func) -> 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:	not exception safe	
Description:	Register a callable that gets called when the Future becomes ready. 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>. It shall be treated as a `Violation` if the Future is invalid (valid returns false). The standardized log message is: "Calling then() on an invalid Future is not allowed." The continuation shall not throw, except for the purpose of implementing a `Violation`. If the continuation throws this shall be treated as a `Violation` with the message "The continuation given to Future::then threw an exception with the explanation: >explanatory string of the exception if available< " 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.	

⌋*(RS_AP_00130)*

[SWS_CORE_06237]{DRAFT} **Definition of API function ara::core::Future< void, E >::then** ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Future< void, E >	
Symbol:	then(F &&func, ExecutorT &&executor)	

Syntax:	`template <typename F, typename ExecutorT>`
	`auto then (F &&func, ExecutorT &&executor) -> 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:	not exception safe	
Description:	Register a callable that gets called when the Future becomes ready. 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>. It shall be treated as a `Violation` if the Future is invalid (valid returns false). The standardized log message is: "Calling then() on an invalid Future is not allowed." The continuation shall not throw, except for the purpose of implementing a `Violation`. If the continuation throws this shall be treated as a `Violation` with the message "The continuation given to Future::then threw an exception with the explanation: >explanatory string of the exception if available< " 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.	

⌋*(RS_AP_00130)*

[SWS_CORE_06232] Definition of API function ara::core::Future< void, E >::is_ready ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Future< void, E >	
Symbol:	is_ready()	
Syntax:	`bool is_ready () const;`	
Return value:	bool	true if the Future contains a value or an error, false otherwise
Exception Safety:	not exception safe	
Description:	Return whether the asynchronous operation has finished. If this function returns true, get(), GetResult() and the wait calls are guaranteed not to block. It shall be treated as a `Violation` if the Future is invalid (valid returns false). The standardized log message is: "Calling is_ready() on an invalid Future is not allowed."	

⌋*(RS_AP_00130)*

8.1.6.8 `Promise` data type

[SWS_CORE_00340]{DRAFT} **Definition of API class ara::core::Promise** ⌈

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	

⌋*(RS_AP_00130)*

[SWS_CORE_00341]{DRAFT} **Definition of API function ara::core::Promise::Promise** ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Promise	
Symbol:	Promise()	
Syntax:	`Promise ();`	
Exception Safety:	not exception safe	
Description:	Default constructor. This function shall behave the same as the corresponding std::promise function.	

⌋*(RS_AP_00130)*

[SWS_CORE_00342]{DRAFT} **Definition of API function ara::core::Promise::Promise** ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Promise	
Symbol:	Promise(Promise &&other)	
Syntax:	`Promise (Promise &&other) noexcept;`	
Parameters (in):	other	the other instance
Exception Safety:	noexcept	
Description:	Move constructor. This function shall behave the same as the corresponding std::promise function.	

⌋*(RS_AP_00130)*

[SWS_CORE_00350]{DRAFT} **Definition of API function ara::core::Promise::Promise** ⌈

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

⌋*(RS_AP_00130)*

[SWS_CORE_00349]{DRAFT} **Definition of API function ara::core::Promise::~Promise** ⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	class ara::core::Promise
Symbol:	~Promise()
Syntax:	`~Promise () noexcept;`
Exception Safety:	noexcept
Description:	Destructor for Promise objects. Abandons any `shared state`.

⌋*(RS_AP_00130)*

[SWS_CORE_00343]{DRAFT} **Definition of API function ara::core::Promise::operator=** ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Promise	
Symbol:	operator=(Promise &&other)	
Syntax:	`Promise & operator= (Promise &&other) noexcept;`	
Parameters (in):	other	the other instance
Return value:	Promise &	*this
Exception Safety:	noexcept	
Description:	Move assignment. Abandons any `shared state` and then as if Promise(std::move(other)).swap(*this).	

⌋*(RS_AP_00130)*

[SWS_CORE_00351]{DRAFT} **Definition of API function ara::core::Promise::operator=** ⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	class ara::core::Promise
Symbol:	operator=(const Promise &)

▽

△

Syntax:	`Promise & operator= (const Promise &)=delete;`
Description:	Copy assignment operator shall be disabled.

⌋*(RS_AP_00130)*

[SWS_CORE_00352]{DRAFT} **Definition of API function ara::core::Promise::swap** ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Promise	
Symbol:	swap(Promise &other)	
Syntax:	`void swap (Promise &other) noexcept;`	
Parameters (in):	other	the other instance
Return value:	None	
Exception Safety:	noexcept	
Description:	Swap the contents of this instance with another one's. This function shall behave the same as the corresponding std::promise function.	

⌋*(RS_AP_00130)*

[SWS_CORE_00344]{DRAFT} **Definition of API function ara::core::Promise::get_future** ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Promise	
Symbol:	get_future()	
Syntax:	`Future< T, E > get_future ();`	
Return value:	Future< T, E >	a Future with the same `shared state` as *this
Exception Safety:	not exception safe	
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. It shall be treated as a `Violation` if the function is called more than once on the same `shared state`. The standardized log message is: "The Future was already retrieved. The Future cannot be retrieved again." It shall be treated as a `Violation` if *this has no `shared state`. The standardized log message is: "The Future associated with this Promise cannot be retrieved, since it has no shared state."	

⌋*(RS_AP_00130)*

[SWS_CORE_00345]{DRAFT} **Definition of API function ara::core::Promise::set_value** ⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	class ara::core::Promise

▽

△

Symbol:	set_value(const T &value)	
Syntax:	`void set_value (const T &value);`	
Parameters (in):	value	the value to store
Return value:	None	
Exception Safety:	not exception safe	
Description:	Copy a value into the `shared state` and make the `shared state` ready. It shall be treated as a `Violation` if the `shared state` already has a stored value or error. The standardized log message is: "The Promise is already satisfied. The value cannot be set again." It shall be treated as a `Violation` if *this has no `shared state`. The standardized log message is: "The value of this Promise cannot be set, since it has no shared state."	

⌋*(RS_AP_00130)*

[SWS_CORE_00346]{DRAFT} **Definition of API function ara::core::Promise::set_value** ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Promise	
Symbol:	set_value(T &&value)	
Syntax:	`void set_value (T &&value);`	
Parameters (in):	value	the value to store
Return value:	None	
Exception Safety:	not exception safe	
Description:	Move a value into the `shared state` and make the `shared state` ready. It shall be treated as a `Violation` if the `shared state` already has a stored value or error. The standardized log message is: "The Promise is already satisfied. The value cannot be set again." It shall be treated as a `Violation` if *this has no `shared state`. The standardized log message is: "The value of this Promise cannot be set, since it has no shared state."	

⌋*(RS_AP_00130)*

[SWS_CORE_00353]{DRAFT} **Definition of API function ara::core::Promise::SetError** ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Promise	
Symbol:	SetError(E &&error)	
Syntax:	`void SetError (E &&error);`	
Parameters (in):	error	the error to store
Return value:	None	
Exception Safety:	not exception safe	

▽

△

Description:	Move an error into the `shared state` and make the `shared state` ready. It shall be treated as a `Violation` if the `shared state` already has a stored value or error. The standardized log message is: "The Promise is already satisfied. The error cannot be set again." It shall be treated as a `Violation` if *this has no `shared state`. The standardized log message is: "The error of this Promise cannot be set, since it has no shared state."

⌋*(RS_AP_00130)*

[SWS_CORE_00354]{DRAFT} **Definition of API function ara::core::Promise::SetError** ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Promise	
Symbol:	SetError(const E &error)	
Syntax:	`void SetError (const E &error);`	
Parameters (in):	error	the error to store
Return value:	None	
Exception Safety:	not exception safe	
Description:	Copy an error into the `shared state` and make the `shared state` ready. It shall be treated as a `Violation` if the `shared state` already has a stored value or error. The standardized log message is: "The Promise is already satisfied. The error cannot be set again." It shall be treated as a `Violation` if *this has no `shared state`. The standardized log message is: "The error of this Promise cannot be set, since it has no shared state."	

⌋*(RS_AP_00130)*

[SWS_CORE_00355]{DRAFT} **Definition of API function ara::core::Promise::SetResult** ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Promise	
Symbol:	SetResult(const Result< T, E > &result)	
Syntax:	`void SetResult (const Result< T, E > &result);`	
Parameters (in):	result	the result to store
Return value:	None	
Exception Safety:	not exception safe	
Description:	Copy a Result into the `shared state` and make the `shared state` ready. It shall be treated as a `Violation` if the `shared state` already has a stored value or error. The standardized log message is: "The Promise is already satisfied. The result cannot be set again." It shall be treated as a `Violation` if *this has no `shared state`. The standardized log message is: "The result of this Promise cannot be set, since it has no shared state."	

⌋*(RS_AP_00130)*

[SWS_CORE_00356]{DRAFT} **Definition of API function ara::core::Promise::SetResult** ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Promise	
Symbol:	SetResult(Result< T, E > &&result)	
Syntax:	`void SetResult (Result< T, E > &&result);`	
Parameters (in):	result	the result to store
Return value:	None	
Exception Safety:	not exception safe	
Description:	Move a Result into the `shared state` and make the `shared state` ready. It shall be treated as a `Violation` if the `shared state` already has a stored value or error. The standardized log message is: "The Promise is already satisfied. The result cannot be set again." It shall be treated as a `Violation` if *this has no `shared state`. The standardized log message is: "The result of this Promise cannot be set, since it has no shared state."	

⌋*(RS_AP_00130)*

8.1.6.8.1 `Promise<void, E>` template specialization

This section defines the interface of the `ara::core::Promise<T,E>` template specialization where the type T is `void`.

[SWS_CORE_06340]{DRAFT} **Definition of API class ara::core::Promise< void, E >** ⌈

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.	

⌋*(RS_AP_00130)*

[SWS_CORE_06341]{DRAFT} **Definition of API function ara::core::Promise< void, E >::Promise** ⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	class ara::core::Promise< void, E >
Symbol:	Promise()
Syntax:	`Promise ();`

▽

△

Exception Safety:	not exception safe	
Description:	Default constructor. This function shall behave the same as the corresponding std::promise function.	

⌋*(RS_AP_00130)*

[SWS_CORE_06342]{DRAFT} **Definition of API function ara::core::Promise< void, E >::Promise** ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Promise< void, E >	
Symbol:	Promise(Promise &&other)	
Syntax:	`Promise (Promise &&other) noexcept;`	
Parameters (in):	other	the other instance
Exception Safety:	noexcept	
Description:	Move constructor. This function shall behave the same as the corresponding std::promise function.	

⌋*(RS_AP_00130)*

[SWS_CORE_06350]{DRAFT} **Definition of API function ara::core::Promise< void, E >::Promise** ⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	class ara::core::Promise< void, E >
Symbol:	Promise(const Promise &)
Syntax:	`Promise (const Promise &)=delete;`
Description:	Copy constructor shall be disabled.

⌋*(RS_AP_00130)*

[SWS_CORE_06349]{DRAFT} **Definition of API function ara::core::Promise< void, E >::~Promise** ⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	class ara::core::Promise< void, E >
Symbol:	~Promise()
Syntax:	`~Promise () noexcept;`
Exception Safety:	noexcept
Description:	Destructor for Promise objects. Abandons any `shared state`.

⌋*(RS_AP_00130)*

[SWS_CORE_06343]{DRAFT} Definition of API function ara::core::Promise< void, E >::operator= ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Promise< void, E >	
Symbol:	operator=(Promise &&other)	
Syntax:	`Promise & operator= (Promise &&other) noexcept;`	
Parameters (in):	other	the other instance
Return value:	Promise &	*this
Exception Safety:	noexcept	
Description:	Move assignment. Abandons any `shared state` and then as if Promise(std::move(other)).swap(*this).	

⌋*(RS_AP_00130)*

[SWS_CORE_06351]{DRAFT} Definition of API function ara::core::Promise< void, E >::operator= ⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	class ara::core::Promise< void, E >
Symbol:	operator=(const Promise &)
Syntax:	`Promise & operator= (const Promise &)=delete;`
Description:	Copy assignment operator shall be disabled.

⌋*(RS_AP_00130)*

[SWS_CORE_06352]{DRAFT} Definition of API function ara::core::Promise< void, E >::swap ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Promise< void, E >	
Symbol:	swap(Promise &other)	
Syntax:	`void swap (Promise &other) noexcept;`	
Parameters (in):	other	the other instance
Return value:	None	
Exception Safety:	noexcept	
Description:	Swap the contents of this instance with another one's. This function shall behave the same as the corresponding std::promise function.	

⌋*(RS_AP_00130)*

[SWS_CORE_06344]{DRAFT} **Definition of API function ara::core::Promise< void, E >::get_future** ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Promise< void, E >	
Symbol:	get_future()	
Syntax:	`Future< void, E > get_future ();`	
Return value:	Future< void, E >	a Future with the same `shared state` as *this
Exception Safety:	not exception safe	
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. It shall be treated as a `Violation` if the function is called more than once on the same `shared state`. The standardized log message is: "The Future was already retrieved. The Future cannot be retrieved again." It shall be treated as a `Violation` if *this has no `shared state`. The standardized log message is: "The Future associated with this Promise cannot be retrieved, since it has no shared state."	

⌋*(RS_AP_00130)*

[SWS_CORE_06345]{DRAFT} **Definition of API function ara::core::Promise< void, E >::set_value** ⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	class ara::core::Promise< void, E >
Symbol:	set_value()
Syntax:	`void set_value ();`
Return value:	None
Exception Safety:	not exception safe
Description:	Set the `shared state` value and make the `shared state` ready. It shall be treated as a `Violation` if the `shared state` already has a stored value or error. The standardized log message is: "The Promise is already satisfied. The value cannot be set again." It shall be treated as a `Violation` if *this has no `shared state`. The standardized log message is: "The value of this Promise cannot be set, since it has no shared state."

⌋*(RS_AP_00130)*

[SWS_CORE_06353]{DRAFT} **Definition of API function ara::core::Promise< void, E >::SetError** ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Promise< void, E >	
Symbol:	SetError(E &&error)	
Syntax:	`void SetError (E &&error);`	
Parameters (in):	error	the error to store

▽

△

Return value:	None
Exception Safety:	not exception safe
Description:	Move an error into the shared state and make the shared state ready. It shall be treated as a Violation if the shared state already has a stored value or error. The standardized log message is: "The Promise is already satisfied. The error cannot be set again." It shall be treated as a Violation if *this has no shared state. The standardized log message is: "The error of this Promise cannot be set, since it has no shared state."

⌋*(RS_AP_00130)*

[SWS_CORE_06354]{DRAFT} **Definition of API function ara::core::Promise< void, E >::SetError** ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Promise< void, E >	
Symbol:	SetError(const E &error)	
Syntax:	`void SetError (const E &error);`	
Parameters (in):	error	the error to store
Return value:	None	
Exception Safety:	not exception safe	
Description:	Copy an error into the shared state and make the shared state ready. It shall be treated as a Violation if the shared state already has a stored value or error. The standardized log message is: "The Promise is already satisfied. The error cannot be set again." It shall be treated as a Violation if *this has no shared state. The standardized log message is: "The error of this Promise cannot be set, since it has no shared state."	

⌋*(RS_AP_00130)*

[SWS_CORE_06355]{DRAFT} **Definition of API function ara::core::Promise< void, E >::SetResult** ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Promise< void, E >	
Symbol:	SetResult(const Result< void, E > &result)	
Syntax:	`void SetResult (const Result< void, E > &result);`	
Parameters (in):	result	the result to store
Return value:	None	
Exception Safety:	not exception safe	
Description:	Copy a Result into the shared state and make the shared state ready. It shall be treated as a Violation if the shared state already has a stored value or error. The standardized log message is: "The Promise is already satisfied. The result cannot be set again." It shall be treated as a Violation if *this has no shared state. The standardized log message is: "The result of this Promise cannot be set, since it has no shared state."	

⌋*(RS_AP_00130)*

[SWS_CORE_06356]{DRAFT} **Definition of API function ara::core::Promise< void, E >::SetResult** ⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	class ara::core::Promise< void, E >	
Symbol:	SetResult(Result< void, E > &&result)	
Syntax:	`void SetResult (Result< void, E > &&result);`	
Parameters (in):	result	the result to store
Return value:	None	
Exception Safety:	not exception safe	
Description:	Move a Result into the `shared state` and make the `shared state` ready. It shall be treated as a `Violation` if the `shared state` already has a stored value or error. The standardized log message is: "The Promise is already satisfied. The result cannot be set again." It shall be treated as a `Violation` if *this has no `shared state`. The standardized log message is: "The result of this Promise cannot be set, since it has no shared state."	

⌋(*RS_AP_00130*)

8.1.7 `Array` data type

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

8.1.7.1 Class Array

[SWS_CORE_01201] Definition of API class ara::core::Array ⌈

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.	

⌋(*RS_AP_00130*)

[SWS_CORE_01210] Definition of API type ara::core::Array::reference ⌈

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.

⌋*(RS_AP_00130)*

[SWS_CORE_01211] Definition of API type ara::core::Array::const_reference ⌈

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.

⌋*(RS_AP_00130)*

[SWS_CORE_01212] Definition of API type ara::core::Array::iterator ⌈

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.

⌋*(RS_AP_00130)*

[SWS_CORE_01213] Definition of API type ara::core::Array::const_iterator ⌈

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.

⌋*(RS_AP_00130)*

[SWS_CORE_01214] Definition of API type ara::core::Array::size_type ⌈

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.

⌋*(RS_AP_00130)*

[SWS_CORE_01215] Definition of API type ara::core::Array::difference_type ⌈

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.

⌋*(RS_AP_00130)*

[SWS_CORE_01216] Definition of API type ara::core::Array::value_type ⌈

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.

⌋*(RS_AP_00130)*

[SWS_CORE_01217] Definition of API type ara::core::Array::pointer ⌈

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.

⌋*(RS_AP_00130)*

[SWS_CORE_01218] Definition of API type ara::core::Array::const_pointer ⌈

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.

⌋*(RS_AP_00130)*

[SWS_CORE_01219] Definition of API type ara::core::Array::reverse_iterator ⌈

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.

⌋*(RS_AP_00130)*

[SWS_CORE_01220] Definition of API type ara::core::Array::const_reverse_iterator ⌈

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.

⌋*(RS_AP_00130)*

[SWS_CORE_01241] Definition of API function ara::core::Array::fill ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	class ara::core::Array	
Symbol:	fill(const T &u)	
Syntax:	`void fill (const T &u);`	
Parameters (in):	u	the value
Return value:	None	
Exception Safety:	not exception safe	
Description:	Assign the given value to all elements of this Array.	

⌋*(RS_AP_00130)*

[SWS_CORE_01242] Definition of API function ara::core::Array::swap ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	class ara::core::Array	
Symbol:	swap(Array< T, N > &other)	
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 noexcept	
Description:	Exchange the contents of this Array with those of other. The noexcept specification shall make use of ADL for the swap() call.	

⌋*(RS_AP_00130)*

[SWS_CORE_01250] Definition of API function ara::core::Array::begin ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	class ara::core::Array	
Symbol:	begin()	
Syntax:	`iterator begin () noexcept;`	
Return value:	iterator	the iterator
Exception Safety:	noexcept	
Description:	Return an iterator pointing to the first element of this Array.	

⌋*(RS_AP_00130)*

[SWS_CORE_01251] Definition of API function ara::core::Array::begin ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	class ara::core::Array	
Symbol:	begin()	
Syntax:	`const_iterator begin () const noexcept;`	
Return value:	const_iterator	the const_iterator
Exception Safety:	noexcept	
Description:	Return a const_iterator pointing to the first element of this Array.	

⌋*(RS_AP_00130)*

[SWS_CORE_01252] Definition of API function ara::core::Array::end ⌈

Kind:	function
Header file:	#include "ara/core/array.h"
Scope:	class ara::core::Array
Symbol:	end()
Syntax:	`iterator end () noexcept;`

▽

△

Return value:	iterator	the iterator
Exception Safety:	noexcept	
Description:	Return an iterator pointing past the last element of this Array.	

⌋*(RS_AP_00130)*

[SWS_CORE_01253] Definition of API function ara::core::Array::end ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	class ara::core::Array	
Symbol:	end()	
Syntax:	`const_iterator end () const noexcept;`	
Return value:	const_iterator	the const_iterator
Exception Safety:	noexcept	
Description:	Return a const_iterator pointing past the last element of this Array.	

⌋*(RS_AP_00130)*

[SWS_CORE_01254] Definition of API function ara::core::Array::rbegin ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	class ara::core::Array	
Symbol:	rbegin()	
Syntax:	`reverse_iterator rbegin () noexcept;`	
Return value:	reverse_iterator	the reverse_iterator
Exception Safety:	noexcept	
Description:	Return a reverse_iterator pointing to the last element of this Array.	

⌋*(RS_AP_00130)*

[SWS_CORE_01255] Definition of API function ara::core::Array::rbegin ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	class ara::core::Array	
Symbol:	rbegin()	
Syntax:	`const_reverse_iterator rbegin () const noexcept;`	
Return value:	const_reverse_iterator	the const_reverse_iterator
Exception Safety:	noexcept	
Description:	Return a const_reverse_iterator pointing to the last element of this Array.	

⌋*(RS_AP_00130)*

[SWS_CORE_01256] Definition of API function ara::core::Array::rend ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	class ara::core::Array	
Symbol:	rend()	
Syntax:	`reverse_iterator rend () noexcept;`	
Return value:	reverse_iterator	the reverse_iterator
Exception Safety:	noexcept	
Description:	Return a reverse_iterator pointing past the first element of this Array.	

⌋*(RS_AP_00130)*

[SWS_CORE_01257] Definition of API function ara::core::Array::rend ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	class ara::core::Array	
Symbol:	rend()	
Syntax:	`const_reverse_iterator rend () const noexcept;`	
Return value:	const_reverse_iterator	the const_reverse_iterator
Exception Safety:	noexcept	
Description:	Return a const_reverse_iterator pointing past the first element of this Array.	

⌋*(RS_AP_00130)*

[SWS_CORE_01258] Definition of API function ara::core::Array::cbegin ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	class ara::core::Array	
Symbol:	cbegin()	
Syntax:	`const_iterator cbegin () const noexcept;`	
Return value:	const_iterator	the const_iterator
Exception Safety:	noexcept	
Description:	Return a const_iterator pointing to the first element of this Array.	

⌋*(RS_AP_00130)*

[SWS_CORE_01259] Definition of API function ara::core::Array::cend ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	class ara::core::Array	
Symbol:	cend()	
Syntax:	`const_iterator cend () const noexcept;`	
Return value:	const_iterator	the const_iterator
Exception Safety:	noexcept	
Description:	Return a const_iterator pointing past the last element of this Array.	

⌋*(RS_AP_00130)*

[SWS_CORE_01260] Definition of API function ara::core::Array::crbegin ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	class ara::core::Array	
Symbol:	crbegin()	
Syntax:	`const_reverse_iterator crbegin () const noexcept;`	
Return value:	const_reverse_iterator	the const_reverse_iterator
Exception Safety:	noexcept	
Description:	Return a const_reverse_iterator pointing to the last element of this Array.	

⌋*(RS_AP_00130)*

[SWS_CORE_01261] Definition of API function ara::core::Array::crend ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	class ara::core::Array	
Symbol:	crend()	
Syntax:	`const_reverse_iterator crend () const noexcept;`	
Return value:	const_reverse_iterator	the const_reverse_iterator
Exception Safety:	noexcept	
Description:	Return a const_reverse_iterator pointing past the first element of this Array.	

⌋*(RS_AP_00130)*

[SWS_CORE_01262] Definition of API function ara::core::Array::size ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	class ara::core::Array	
Symbol:	size()	
Syntax:	`constexpr size_type size () const noexcept;`	
Return value:	size_type	N
Exception Safety:	noexcept	
Description:	Return the number of elements in this Array.	

⌋*(RS_AP_00130)*

[SWS_CORE_01263] Definition of API function ara::core::Array::max_size ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	class ara::core::Array	
Symbol:	max_size()	
Syntax:	`constexpr size_type max_size () const noexcept;`	
Return value:	size_type	N
Exception Safety:	noexcept	
Description:	Return the maximum number of elements supported by this Array.	

⌋*(RS_AP_00130)*

[SWS_CORE_01264] Definition of API function ara::core::Array::empty ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	class ara::core::Array	
Symbol:	empty()	
Syntax:	`constexpr bool empty () const noexcept;`	
Return value:	bool	true if this Array contains 0 elements, false otherwise
Exception Safety:	noexcept	
Description:	Return whether this Array is empty.	

⌋*(RS_AP_00130)*

[SWS_CORE_01265] Definition of API function ara::core::Array::operator[] ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	class ara::core::Array	
Symbol:	operator[](size_type n)	
Syntax:	`reference operator[] (size_type n);`	
Parameters (in):	n	the index into this Array
Return value:	reference	the reference
Exception Safety:	not exception 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.	

⌋*(RS_AP_00130)*

[SWS_CORE_01266] Definition of API function ara::core::Array::operator[] ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	class ara::core::Array	
Symbol:	operator[](size_type n)	
Syntax:	`constexpr const_reference operator[] (size_type n) const;`	
Parameters (in):	n	the index into this Array
Return value:	const_reference	the const_reference
Exception Safety:	not exception 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.	

⌋*(RS_AP_00130)*

[SWS_CORE_01273]{DRAFT} **Definition of API function ara::core::Array::at** ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	class ara::core::Array	
Symbol:	at(size_type n)	
Syntax:	`reference at (size_type n);`	
Parameters (in):	n	the index into this Array
Return value:	reference	the reference
Exception Safety:	not exception safe	
Description:	Return a reference to the n-th element of this Array, with bound checking. It shall be treated as a `Violation` if n is not within the range of the array. The standardized log message is: "Array access out of range: Tried to access >n< in array of size >N< "	

⌋*(RS_AP_00130)*

[SWS_CORE_01274]{DRAFT} **Definition of API function ara::core::Array::at** ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	class ara::core::Array	
Symbol:	at(size_type n)	
Syntax:	`constexpr const_reference at (size_type n) const;`	
Parameters (in):	n	the index into this Array
Return value:	const_reference	the const_reference
Exception Safety:	not exception safe	
Description:	Return a const_reference to the n-th element of this Array, with bound checking. It shall be treated as a `Violation` if n is not within the range of the array. The standardized log message is: "Array access out of range: Tried to access >n< in array of size >N< "	

⌋*(RS_AP_00130)*

[SWS_CORE_01267] Definition of API function ara::core::Array::front ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	class ara::core::Array	
Symbol:	front()	
Syntax:	`reference front ();`	
Return value:	reference	the reference
Exception Safety:	not exception safe	
Description:	Return a reference to the first element of this Array. The behavior of this function is undefined if the Array is empty.	

⌋*(RS_AP_00130)*

[SWS_CORE_01268] Definition of API function ara::core::Array::front ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	class ara::core::Array	
Symbol:	front()	
Syntax:	`constexpr const_reference front () const;`	
Return value:	const_reference	the reference
Exception Safety:	not exception safe	
Description:	Return a const_reference to the first element of this Array. The behavior of this function is undefined if the Array is empty.	

⌋*(RS_AP_00130)*

[SWS_CORE_01269] Definition of API function ara::core::Array::back ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	class ara::core::Array	
Symbol:	back()	
Syntax:	`reference back ();`	
Return value:	reference	the reference
Exception Safety:	not exception safe	
Description:	Return a reference to the last element of this Array. The behavior of this function is undefined if the Array is empty.	

⌋*(RS_AP_00130)*

[SWS_CORE_01270] Definition of API function ara::core::Array::back ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	class ara::core::Array	
Symbol:	back()	
Syntax:	`constexpr const_reference back () const;`	
Return value:	const_reference	the reference
Exception Safety:	not exception safe	
Description:	Return a const_reference to the last element of this Array. The behavior of this function is undefined if the Array is empty.	

⌋*(RS_AP_00130)*

[SWS_CORE_01271] Definition of API function ara::core::Array::data ⌈

Kind:	function
Header file:	#include "ara/core/array.h"
Scope:	class ara::core::Array
Symbol:	data()

▽

△

Syntax:	`pointer data () noexcept;`	
Return value:	pointer	the pointer
Exception Safety:	noexcept	
Description:	Return a pointer to the first element of this Array.	

⌋(*RS_AP_00130*)

[SWS_CORE_01272] Definition of API function ara::core::Array::data ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	class ara::core::Array	
Symbol:	data()	
Syntax:	`const_pointer data () const noexcept;`	
Return value:	const_pointer	the const_pointer
Exception Safety:	noexcept	
Description:	Return a const_pointer to the first element of this Array.	

⌋(*RS_AP_00130*)

8.1.7.2 Non-member functions

[SWS_CORE_01290] Definition of API function ara::core::operator== ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	namespace ara::core	
Symbol:	operator==(const Array< T, N > &lhs, const Array< T, N > &rhs)	

Syntax:	`template <typename T, std::size_t N>`
	`bool operator== (const Array< T, N > &lhs, const Array< T, N > &rhs);`	

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:	not exception safe	
Description:	Return true if the two Arrays have equal content.	

⌋(*RS_AP_00130*)

[SWS_CORE_01291] Definition of API function ara::core::operator!= ⌈

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

▽

△

Symbol:	operator!=(const Array< T, N > &lhs, const Array< T, N > &rhs)	

Syntax:	`template <typename T, std::size_t N>`
	`bool operator!= (const Array< T, N > &lhs, const Array< T, N > &rhs);`	

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:	not exception safe	
Description:	Return true if the two Arrays have non-equal content.	

⌋*(RS_AP_00130)*

[SWS_CORE_01292] Definition of API function ara::core::operator< ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	namespace ara::core	
Symbol:	operator<(const Array< T, N > &lhs, const Array< T, N > &rhs)	

Syntax:	`template <typename T, std::size_t N>`
	`bool operator< (const Array< T, N > &lhs, const Array< T, N > &rhs);`	

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:	not exception safe	
Description:	Return true if the contents of lhs are lexicographically less than the contents of rhs.	

⌋*(RS_AP_00130)*

[SWS_CORE_01293] Definition of API function ara::core::operator> ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	namespace ara::core	
Symbol:	operator>(const Array< T, N > &lhs, const Array< T, N > &rhs)	

Syntax:	`template <typename T, std::size_t N>`
	`bool operator> (const Array< T, N > &lhs, const Array< T, N > &rhs);`	

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:	not exception safe	
Description:	Return true if the contents of rhs are lexicographically less than the contents of lhs.	

⌋*(RS_AP_00130)*

[SWS_CORE_01294] Definition of API function ara::core::operator<= ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	namespace ara::core	
Symbol:	operator<=(const Array< T, N > &lhs, const Array< T, N > &rhs)	

Syntax:	`template <typename T, std::size_t N>`
	`bool operator<= (const Array< T, N > &lhs, const Array< T, N > &rhs);`	

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:	not exception safe	
Description:	Return true if the contents of lhs are lexicographically less than or equal to the contents of rhs.	

⌋*(RS_AP_00130)*

[SWS_CORE_01295] Definition of API function ara::core::operator>= ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	namespace ara::core	
Symbol:	operator>=(const Array< T, N > &lhs, const Array< T, N > &rhs)	

Syntax:	`template <typename T, std::size_t N>`
	`bool operator>= (const Array< T, N > &lhs, const Array< T, N > &rhs);`	

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:	not exception safe	
Description:	Return true if the contents of rhs are lexicographically less than or equal to the contents of lhs.	

⌋*(RS_AP_00130)*

[SWS_CORE_01296] Definition of API function ara::core::swap ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	namespace ara::core	
Symbol:	swap(Array< T, N > &lhs, Array< T, N > &rhs)	

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 noexcept
Description:	Overload of std::swap for ara::core::Array.

⌋*(RS_AP_00130)*

8.1.7.3 Tuple interface

These definitions implement the standard interface of tuple-like types for class `Array`.

The specializations of the `std::tuple_size` and `std::tuple_element` traits are put into the `std` namespace:

[SWS_CORE_01280] Definition of API class std::tuple_size< ara::core::Array< T, N > > ⌈

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 {...};`	

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>.	

⌋*(RS_AP_00130)*

[SWS_CORE_01281] Definition of API class std::tuple_element< I, ara::core::Array< T, N > > ⌈

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.

⌋*(RS_AP_00130)*

[SWS_CORE_01285] Definition of API type std::tuple_element< I, ara::core::Array< T, N > >::type ⌈

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.

⌋*(RS_AP_00130)*

The overloads of `std::get` are contained in the `ara::core` namespace; 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_01282] Definition of API function ara::core::get ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	namespace ara::core	
Symbol:	get(Array< T, N > &a)	

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:	noexcept	
Description:	Overload of std::get for an lvalue mutable ara::core::Array. The implementation shall flag the condition I >= N as a compile error.	

⌋*(RS_AP_00130)*

[SWS_CORE_01283] Definition of API function ara::core::get ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	namespace ara::core	
Symbol:	get(Array< T, N > &&a)	
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:	noexcept	
Description:	Overload of std::get for an rvalue ara::core::Array. The implementation shall flag the condition I >= N as a compile error.	

⌋*(RS_AP_00130)*

[SWS_CORE_01284] Definition of API function ara::core::get ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	namespace ara::core	
Symbol:	get(const Array< T, N > &a)	
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:	noexcept	
Description:	Overload of std::get for an lvalue const ara::core::Array. The implementation shall flag the condition I >= N as a compile error.	

⌋*(RS_AP_00130)*

8.1.8 `Vector` data type

This section describes the `ara::core::Vector` type that represents a container which can change in size.

[SWS_CORE_01301]{DRAFT} **Definition of API class ara::core::Vector** ⌈

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.	

⌋*(RS_AP_00130)*

[SWS_CORE_01390]{DRAFT} **Definition of API function ara::core::operator==** ⌈

Kind:	function	
Header file:	#include "ara/core/vector.h"	
Scope:	namespace ara::core	
Symbol:	operator==(const Vector< T, Allocator > &lhs, const Vector< T, Allocator > &rhs)	
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	
Description:	Return true if the two Vectors have equal content.	

⌋*(RS_AP_00130)*

[SWS_CORE_01391]{DRAFT} **Definition of API function ara::core::operator!=** ⌈

Kind:	function	
Header file:	#include "ara/core/vector.h"	
Scope:	namespace ara::core	
Symbol:	operator!=(const Vector< T, Allocator > &lhs, const Vector< T, Allocator > &rhs)	
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
Description:	Return true if the two Vectors have non-equal content.

⌋*(RS_AP_00130)*

[SWS_CORE_01392]{DRAFT} **Definition of API function ara::core::operator<** ⌈

Kind:	function	
Header file:	#include "ara/core/vector.h"	
Scope:	namespace ara::core	
Symbol:	operator<(const Vector< T, Allocator > &lhs, const Vector< T, Allocator > &rhs)	
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	
Description:	Return true if the contents of lhs are lexicographically less than the contents of rhs.	

⌋*(RS_AP_00130)*

[SWS_CORE_01393]{DRAFT} **Definition of API function ara::core::operator<=** ⌈

Kind:	function	
Header file:	#include "ara/core/vector.h"	
Scope:	namespace ara::core	
Symbol:	operator<=(const Vector< T, Allocator > &lhs, const Vector< T, Allocator > &rhs)	
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	
Description:	Return true if the contents of lhs are lexicographically less than or equal to the contents of rhs.	

⌋*(RS_AP_00130)*

[SWS_CORE_01394]{DRAFT} **Definition of API function ara::core::operator>** ⌈

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

▽

△

Symbol:	operator>(const Vector< T, Allocator > &lhs, const Vector< T, Allocator > &rhs)	
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	
Description:	Return true if the contents of rhs are lexicographically less than the contents of lhs.	

⌋*(RS_AP_00130)*

[SWS_CORE_01395]{DRAFT} **Definition of API function ara::core::operator>=** ⌈

Kind:	function	
Header file:	#include "ara/core/vector.h"	
Scope:	namespace ara::core	
Symbol:	operator>=(const Vector< T, Allocator > &lhs, const Vector< T, Allocator > &rhs)	
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	
Description:	Return true if the contents of rhs are lexicographically less than or equal to the contents of lhs.	

⌋*(RS_AP_00130)*

[SWS_CORE_01396]{DRAFT} **Definition of API function ara::core::swap** ⌈

Kind:	function	
Header file:	#include "ara/core/vector.h"	
Scope:	namespace ara::core	
Symbol:	swap(Vector< T, Allocator > &lhs, Vector< T, Allocator > &rhs)	
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	
Description:	Exchange the state of lhs with that of rhs.	

⌋*(RS_AP_00130)*

8.1.9 `Map` data type

This section describes the `ara::core::Map` type that represents a container which contains key-value pairs with unique keys.

[SWS_CORE_01400]{DRAFT} **Definition of API class ara::core::Map** ⌈

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.	

⌋*(RS_AP_00130)*

[SWS_CORE_01496]{DRAFT} **Definition of API function ara::core::swap** ⌈

Kind:	function	
Header file:	#include "ara/core/map.h"	
Scope:	namespace ara::core	
Symbol:	swap(Map< K, V, C, Allocator > &lhs, Map< K, V, C, Allocator > &rhs)	
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	
Description:	Exchange the state of lhs with that of rhs.	

⌋*(RS_AP_00130)*

8.1.10 `Optional` data type

This section describes the class template `ara::core::Optional` that provides access to optional record elements of a `Structure Implementation data type`. Whenever there is a mention of the standard C++17 item `std::optional`, the implied source material is [9, the C++17 standard].

The class template ara::core::Optional manages optional values, i.e. values that may or may not be present. The existence can be evaluated during both compile-time and runtime.
Note: Mandatory record elements are declared directly with the corresponding ImplementationDataType without using ara::core::Optional.

[SWS_CORE_01033]{DRAFT} **Definition of API class ara::core::Optional** ⌈

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 {...};`	

Template param:	typename T	the type of element in the container
Description:	A container with at most one element.	

⌋*(RS_AP_00130)*

[SWS_CORE_01096]{DRAFT} **Definition of API function ara::core::swap** ⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	namespace ara::core	
Symbol:	swap(Optional< T > &lhs, Optional< T > &rhs)	

Syntax:	`template <typename T>`
	`void swap (Optional< T > &lhs, Optional< T > &rhs);`	

Parameters (in):	lhs	the first Optional
	rhs	the second Optional
Return value:	None	
Exception Safety:	not exception safe	
Description:	Exchange the state of lhs with that of rhs.	

⌋*(RS_AP_00130)*

8.1.11 `Variant` data type

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

[SWS_CORE_01601]{DRAFT} **Definition of API class ara::core::Variant** ⌈

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 final {...};`	

Template param:	typename... Types	the types that the Variant is able to hold
Description:	A type-safe union.	

⌋*(RS_AP_00130)*

[SWS_CORE_01696]{DRAFT} **Definition of API function ara::core::swap** ⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	namespace ara::core	
Symbol:	swap(Variant< Types... > &lhs, Variant< Types... > &rhs)	

Syntax:	`template <typename... Types>`
	`void swap (Variant< Types... > &lhs, Variant< Types... > &rhs);`	

Parameters (in):	lhs	the first Variant
	rhs	the second Variant
Return value:	None	
Exception Safety:	not exception safe	
Description:	Exchange the state of lhs with that of rhs.	

⌋*(RS_AP_00130)*

8.1.12 `StringView` data type

This section describes the `ara::core::StringView` type that constitutes a read-only view over a contiguous sequence of characters, the storage of which is owned by another object.

[SWS_CORE_02001]{DRAFT} **Definition of API class ara::core::StringView** ⌈

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:	A read-only view over a contiguous sequence of characters whose storage is owned by another object.

⌋*(RS_AP_00130)*

8.1.13 String data types

This section describes the `ara::core::String` type and its complement `ara::core::BasicString` which both represent sequences of characters.

These types are closely modeled on `std::string` and `std::basic_string` respectively from [4, the C++14 standard], with a number of additions coming from [9, 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_03000]{DRAFT} **Definition of API class ara::core::BasicString** ⌈

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 to use for any memory allocations
Description:	BasicString type.	

⌋*(RS_AP_00130)*

[SWS_CORE_03012]{DRAFT} **Definition of API type ara::core::BasicString::size_type** ⌈

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:	Alias for the type of parameters that indicate a size of a number of values.

⌋*(RS_AP_00130)*

[SWS_CORE_03302]{DRAFT} **Definition of API function ara::core::BasicString::BasicString** ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	class ara::core::BasicString	
Symbol:	BasicString(StringView sv)	
Syntax:	`explicit BasicString (StringView sv);`	
Parameters (in):	sv	a StringView
Exception Safety:	not exception safe	
Description:	Constructor from StringView.	

⌋*(RS_AP_00130)*

[SWS_CORE_03303]{DRAFT} **Definition of API function ara::core::BasicString::BasicString** ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	class ara::core::BasicString	
Symbol:	BasicString(const T &t, size_type pos, size_type n, const Allocator &alloc=Allocator())	
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	
Description:	Constructor from implicit StringView.	

⌋*(RS_AP_00130)*

[SWS_CORE_03304]{DRAFT} **Definition of API function ara::core::BasicString::operator=** ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	class ara::core::BasicString	
Symbol:	operator=(StringView sv)	
Syntax:	`BasicString & operator= (StringView sv);`	
Parameters (in):	sv	the StringView
Return value:	BasicString &	*this
Exception Safety:	not exception safe	
Description:	Assignment operator from StringView.	

⌋*(RS_AP_00130)*

[SWS_CORE_03307]{DRAFT} **Definition of API function ara::core::BasicString::operator+=** ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	class ara::core::BasicString	
Symbol:	operator+=(StringView sv)	
Syntax:	`BasicString & operator+= (StringView sv);`	
Parameters (in):	sv	the StringView
Return value:	BasicString &	*this
Exception Safety:	not exception safe	
Description:	Concatenation operator from StringView.	

⌋*(RS_AP_00130)*

[SWS_CORE_03308]{DRAFT} **Definition of API function ara::core::BasicString::append** ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	class ara::core::BasicString	
Symbol:	append(StringView sv)	
Syntax:	`BasicString & append (StringView sv);`	
Parameters (in):	sv	the StringView
Return value:	BasicString &	*this
Exception Safety:	not exception safe	
Description:	Concatenation from StringView.	

⌋*(RS_AP_00130)*

[SWS_CORE_03309]{DRAFT} **Definition of API function ara::core::BasicString::append** ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	class ara::core::BasicString	
Symbol:	append(const T &t, size_type pos, size_type n=npos)	

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	
Description:	Concatenation from implicit StringView.	

⌋*(RS_AP_00130)*

[SWS_CORE_03305]{DRAFT} **Definition of API function ara::core::BasicString::assign** ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	class ara::core::BasicString	
Symbol:	assign(StringView sv)	
Syntax:	`BasicString & assign (StringView sv);`	
Parameters (in):	sv	the StringView
Return value:	BasicString &	*this
Exception Safety:	not exception safe	
Description:	Assignment from StringView.	

⌋*(RS_AP_00130)*

[SWS_CORE_03306]{DRAFT} **Definition of API function ara::core::BasicString::assign** ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	class ara::core::BasicString	
Symbol:	assign(const T &t, size_type pos, size_type n=npos)	
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	
Description:	Assignment from implicit StringView.	

⌋*(RS_AP_00130)*

[SWS_CORE_03310]{DRAFT} **Definition of API function ara::core::BasicString::insert** ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	class ara::core::BasicString	
Symbol:	insert(size_type pos, StringView sv)	
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	
Description:	Insertion of StringView.	

⌋*(RS_AP_00130)*

[SWS_CORE_03311]{DRAFT} **Definition of API function ara::core::BasicString::insert** ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	class ara::core::BasicString	
Symbol:	insert(size_type pos1, const T &t, size_type pos2, size_type n=npos)	
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	
Description:	Insertion of implicit StringView.	

⌋(*RS_AP_00130*)

[SWS_CORE_03312]{DRAFT} **Definition of API function ara::core::BasicString::replace** ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	class ara::core::BasicString	
Symbol:	replace(size_type pos1, size_type n1, StringView sv)	
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	
Description:	Replacement with StringView.	

⌋(*RS_AP_00130*)

[SWS_CORE_03313]{DRAFT} **Definition of API function ara::core::BasicString::replace** ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	class ara::core::BasicString	
Symbol:	replace(size_type pos1, size_type n1, const T &t, size_type pos2, size_type n2=npos)	
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	
Description:	Replacement with implicit StringView.	

⌋(*RS_AP_00130*)

[SWS_CORE_03314]{DRAFT} **Definition of API function ara::core::BasicString::replace** ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	class ara::core::BasicString	
Symbol:	replace(const_iterator i1, const_iterator i2, StringView sv)	
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	
Description:	Replacement of iterator range with StringView.	

⌋*(RS_AP_00130)*

[SWS_CORE_03301]{DRAFT} **Definition of API function ara::core::BasicString::operator StringView** ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	class ara::core::BasicString	
Symbol:	operator StringView()	
Syntax:	`operator StringView () const noexcept;`	
Return value:	StringView	a StringView
Exception Safety:	noexcept	
Description:	Implicit conversion to StringView.	

⌋*(RS_AP_00130)*

[SWS_CORE_03315]{DRAFT} **Definition of API function ara::core::BasicString::find** ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	class ara::core::BasicString	
Symbol:	find(StringView sv, size_type pos=0)	
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:	noexcept	
Description:	Find the first substring equal to the given StringView.	

⌋*(RS_AP_00130)*

[SWS_CORE_03316]{DRAFT} **Definition of API function ara::core::BasicString::rfind** ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	class ara::core::BasicString	
Symbol:	rfind(StringView sv, size_type pos=npos)	
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:	noexcept	
Description:	Find the last substring equal to the given StringView.	

⌋*(RS_AP_00130)*

[SWS_CORE_03317]{DRAFT} **Definition of API function ara::core::BasicString::find_first_of** ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	class ara::core::BasicString	
Symbol:	find_first_of(StringView sv, size_type pos=0)	
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:	noexcept	
Description:	Find the first character equal to one of the characters in the given StringView.	

⌋*(RS_AP_00130)*

[SWS_CORE_03318]{DRAFT} **Definition of API function ara::core::BasicString::find_last_of** ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	class ara::core::BasicString	
Symbol:	find_last_of(StringView sv, size_type pos=npos)	
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:	noexcept	
Description:	Find the last character equal to one of the characters in the given StringView.	

⌋*(RS_AP_00130)*

[SWS_CORE_03319]{DRAFT} **Definition of API function ara::core::BasicString::find_first_not_of** ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	class ara::core::BasicString	
Symbol:	find_first_not_of(StringView sv, size_type pos=0)	
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:	noexcept	
Description:	Find the first character that is not one of the characters in the given StringView.	

⌋*(RS_AP_00130)*

[SWS_CORE_03320]{DRAFT} **Definition of API function ara::core::BasicString::find_last_not_of** ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	class ara::core::BasicString	
Symbol:	find_last_not_of(StringView sv, size_type pos=npos)	
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:	noexcept	
Description:	Find the last character that is not one of the characters in the given StringView.	

⌋*(RS_AP_00130)*

[SWS_CORE_03321]{DRAFT} **Definition of API function ara::core::BasicString::compare** ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	class ara::core::BasicString	
Symbol:	compare(StringView sv)	
Syntax:	`int compare (StringView sv) const noexcept;`	
Parameters (in):	sv	the StringView
Return value:	int	as per description of std::string::compare
Exception Safety:	noexcept	
Description:	Compare with a StringView.	

⌋*(RS_AP_00130)*

[SWS_CORE_03322]{DRAFT} **Definition of API function ara::core::BasicString::compare** ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	class ara::core::BasicString	
Symbol:	compare(size_type pos1, size_type n1, StringView sv)	
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	
Description:	Compare with a StringView.	

⌋*(RS_AP_00130)*

[SWS_CORE_03323]{DRAFT} **Definition of API function ara::core::BasicString::compare** ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	class ara::core::BasicString	
Symbol:	compare(size_type pos1, size_type n1, const T &t, size_type pos2, size_type n2=npos)	
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	
Description:	Compare with an implicit StringView.	

⌋*(RS_AP_00130)*

[SWS_CORE_03296]{DRAFT} **Definition of API function ara::core::swap** ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	namespace ara::core	
Symbol:	swap(BasicString< Allocator > &lhs, BasicString< Allocator > &rhs)	
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	
Description:	Exchange the state of lhs with that of rhs.	

⌋*(RS_AP_00130)*

[SWS_CORE_03001]{DRAFT} **Definition of API type ara::core::String** ⌈

Kind:	type alias
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Symbol:	String
Syntax:	`using String = BasicString<>;`
Description:	String type.

⌋*(RS_AP_00130)*

8.1.14 `Span` data type

This section describes the `ara::core::Span` type that constitutes a view over a contiguous sequence of objects, the storage of which is owned by another object.

This specification is based on the draft standard of `std::span` in revision N4835 (section 22.7), but has been adapted in several ways:

- The type alias `Span::index_type` has been renamed into `Span::size_type`, following the P1872R0 proposal.
- Some compile-time checks are now being imposed on implementations, following the proposed resolution of LWG issue 3103.
- All symbols from section 22.7.3.8 (span.tuple) have been omitted, following the proposed resolution of LWG issue 3212.
- The `std::array`-based constructors have been made more flexible, following the proposed resolution of LWG issue 3255.
- Constructors have been added that take a `ara::core::Array`, with semantics that are the same as those of the constructors that take a `std::array`.
- A number of non-member `MakeSpan` factory function overloads have been added.

[SWS_CORE_01901]{DRAFT} **Definition of API variable ara::core::dynamic_extent** ⌈

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().

⌋*(RS_AP_00130)*

[SWS_CORE_01900]{DRAFT} **Definition of API class ara::core::Span** ⌈

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.	

⌋*(RS_AP_00130)*

[SWS_CORE_01911]{DRAFT} **Definition of API type ara::core::Span::element_type** ⌈

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.

⌋*(RS_AP_00130)*

[SWS_CORE_01912]{DRAFT} **Definition of API type ara::core::Span::value_type** ⌈

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.

⌋*(RS_AP_00130)*

[SWS_CORE_01921]{DRAFT} **Definition of API type ara::core::Span::size_type** ⌈

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.

⌋*(RS_AP_00130)*

[SWS_CORE_01914]{DRAFT} **Definition of API type ara::core::Span::difference_type** ⌈

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.

⌋*(RS_AP_00130)*

[SWS_CORE_01915]{DRAFT} **Definition of API type ara::core::Span::pointer** ⌈

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.

⌋*(RS_AP_00130)*

[SWS_CORE_01922]{DRAFT} **Definition of API type ara::core::Span::const_pointer** ⌈

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.

⌋*(RS_AP_00130)*

[SWS_CORE_01916]{DRAFT} **Definition of API type ara::core::Span::reference** ⌈

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.

⌋*(RS_AP_00130)*

[SWS_CORE_01923]{DRAFT} **Definition of API type ara::core::Span::const_reference** ⌈

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.

⌋*(RS_AP_00130)*

[SWS_CORE_01917]{DRAFT} **Definition of API type ara::core::Span::iterator** ⌈

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.

⌋*(RS_AP_00130)*

[SWS_CORE_01918]{DRAFT} **Definition of API type ara::core::Span::const_iterator** ⌈

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.

⌋*(RS_AP_00130)*

[SWS_CORE_01919]{DRAFT} **Definition of API type ara::core::Span::reverse_iterator** ⌈

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.

⌋*(RS_AP_00130)*

[SWS_CORE_01920]{DRAFT} **Definition of API type ara::core::Span::const_reverse_iterator** ⌈

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.

⌋*(RS_AP_00130)*

[SWS_CORE_01931]{DRAFT} **Definition of API variable ara::core::Span::extent** ⌈

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.

⌋*(RS_AP_00130)*

[SWS_CORE_01941]{DRAFT} **Definition of API function ara::core::Span::Span** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	Span()	
Syntax:	`constexpr Span () noexcept;`	
Exception Safety:	noexcept	
Description:	Default constructor. This constructor shall not participate in overload resolution unless (Extent == dynamic_extent || Extent == 0) is true.	

⌋*(RS_AP_00130)*

[SWS_CORE_01942]{DRAFT} **Definition of API function ara::core::Span::Span** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	Span(pointer ptr, size_type count)	
Syntax:	`constexpr Span (pointer ptr, size_type count);`	
Parameters (in):	ptr	the pointer
	count	the number of elements to take from ptr
Exception Safety:	not exception safe	
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.	

⌋*(RS_AP_00130)*

[SWS_CORE_01943]{DRAFT} **Definition of API function ara::core::Span::Span** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	Span(pointer firstElem, pointer lastElem)	
Syntax:	`constexpr Span (pointer firstElem, pointer lastElem);`	
Parameters (in):	firstElem	pointer to the first element
	lastElem	pointer to past the last element
Exception Safety:	not exception safe	
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 (last Elem - firstElem) shall be equal to extent.	

⌋*(RS_AP_00130)*

[SWS_CORE_01944]{DRAFT} **Definition of API function ara::core::Span::Span** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	Span(element_type(&arr)[N])	
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:	noexcept	
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(*)[].	

⌋*(RS_AP_00130)*

[SWS_CORE_01953]{DRAFT} **Definition of API function ara::core::Span::Span** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	Span(std::array< U, N > &arr)	
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:	noexcept	
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(*)[].	

⌋*(RS_AP_00130)*

[SWS_CORE_01954]{DRAFT} **Definition of API function ara::core::Span::Span** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	Span(const std::array< U, N > &arr)	
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:	noexcept
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(*)[].

⌋*(RS_AP_00130)*

[SWS_CORE_01945]{DRAFT} **Definition of API function ara::core::Span::Span** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	Span(Array< U, N > &arr)	

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:	noexcept	
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(*)[].	

⌋*(RS_AP_00130)*

[SWS_CORE_01946]{DRAFT} **Definition of API function ara::core::Span::Span** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	Span(const Array< U, N > &arr)	

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:	noexcept	
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(*)[].	

⌋*(RS_AP_00130)*

[SWS_CORE_01947]{DRAFT} **Definition of API function ara::core::Span::Span** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	Span(Container &cont)	
Syntax:	`template <typename Container>` `constexpr Span (Container &cont);`	
Template param:	Container	the type of container
Parameters (in):	cont	the container
Exception Safety:	not exception safe	
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(*)[].	

⌋*(RS_AP_00130)*

[SWS_CORE_01948]{DRAFT} **Definition of API function ara::core::Span::Span** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	Span(const Container &cont)	
Syntax:	`template <typename Container>` `constexpr Span (const Container &cont);`	
Template param:	Container	the type of container
Parameters (in):	cont	the container
Exception Safety:	not exception safe	
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(*)[].	

⌋*(RS_AP_00130)*

[SWS_CORE_01949]{DRAFT} **Definition of API function ara::core::Span::Span** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	Span(const Span &other)	
Syntax:	`constexpr Span (const Span &other) noexcept=default;`	
Parameters (in):	other	the other instance
Exception Safety:	noexcept	
Description:	Copy construct a new Span from another instance.	

⌋*(RS_AP_00130)*

[SWS_CORE_01950]{DRAFT} **Definition of API function ara::core::Span::Span** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	Span(const Span< U, N > &s)	

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:	noexcept	
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(*)[]	

⌋*(RS_AP_00130)*

[SWS_CORE_01951]{DRAFT} **Definition of API function ara::core::Span::~Span** ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	class ara::core::Span
Symbol:	~Span()
Syntax:	`~Span () noexcept=default;`
Exception Safety:	noexcept
Description:	Destructor.

⌋*(RS_AP_00130)*

[SWS_CORE_01952]{DRAFT} **Definition of API function ara::core::Span::operator=** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	operator=(const Span &other)	
Syntax:	`constexpr Span & operator= (const Span &other) noexcept=default;`	
Parameters (in):	other	the other instance
Return value:	Span &	*this
Exception Safety:	noexcept	
Description:	Copy assignment operator.	

⌋*(RS_AP_00130)*

[SWS_CORE_01961]{DRAFT} **Definition of API function ara::core::Span::first** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	first()	

Syntax:	`template <std::size_t Count>`
	`constexpr Span< element_type, Count > first () const;`	

Template param:	Count	the number of elements to take over
Return value:	Span< element_type, Count >	the subspan
Exception Safety:	not exception 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()).	

⌋*(RS_AP_00130)*

[SWS_CORE_01962]{DRAFT} **Definition of API function ara::core::Span::first** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	first(size_type count)	
Syntax:	`constexpr Span< element_type, dynamic_extent > first (size_type count) const;`	
Parameters (in):	count	the number of elements to take over
Return value:	Span< element_type, dynamic_extent >	the subspan
Exception Safety:	not exception safe	
Description:	Return a subspan containing only the first elements of this Span. The behavior of this function is undefined if (count > size()).	

⌋*(RS_AP_00130)*

[SWS_CORE_01963]{DRAFT} **Definition of API function ara::core::Span::last** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	last()	
Syntax:	`template <std::size_t Count>` `constexpr Span< element_type, Count > last () const;`	
Template param:	Count	the number of elements to take over
Return value:	Span< element_type, Count >	the subspan
Exception Safety:	not exception 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()).	

⌋*(RS_AP_00130)*

[SWS_CORE_01964]{DRAFT} **Definition of API function ara::core::Span::last** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	last(size_type count)	
Syntax:	`constexpr Span< element_type, dynamic_extent > last (size_type count) const;`	
Parameters (in):	count	the number of elements to take over
Return value:	Span< element_type, dynamic_extent >	the subspan
Exception Safety:	not exception safe	
Description:	Return a subspan containing only the last elements of this Span. The behavior of this function is undefined if (count > size()).	

⌋*(RS_AP_00130)*

[SWS_CORE_01965]{DRAFT} **Definition of API function ara::core::Span::subspan** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	subspan()	
Syntax:	`template <std::size_t Offset, std::size_t Count = dynamic_extent>` `constexpr auto subspan () const -> 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:	not exception 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.

⌋(*RS_AP_00130*)

[SWS_CORE_01966]{DRAFT} **Definition of API function ara::core::Span::subspan** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	subspan(size_type offset, size_type count=dynamic_extent)	
Syntax:	`constexpr Span< element_type, dynamic_extent > subspan (size_type offset, size_type count=dynamic_extent) const;`	
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:	not exception 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.	

⌋(*RS_AP_00130*)

[SWS_CORE_01967]{DRAFT} **Definition of API function ara::core::Span::size** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	size()	
Syntax:	`constexpr size_type size () const noexcept;`	
Return value:	size_type	the number of elements contained in this Span
Exception Safety:	noexcept	
Description:	Return the size of this Span.	

⌋(*RS_AP_00130*)

[SWS_CORE_01968]{DRAFT} Definition of API function ara::core::Span::size_bytes ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	size_bytes()	
Syntax:	`constexpr size_type size_bytes () const noexcept;`	
Return value:	size_type	the number of bytes covered by this Span
Exception Safety:	noexcept	
Description:	Return the size of this Span in bytes.	

⌋*(RS_AP_00130)*

[SWS_CORE_01969]{DRAFT} Definition of API function ara::core::Span::empty ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	empty()	
Syntax:	`constexpr bool empty () const noexcept;`	
Return value:	bool	true if this Span contains 0 elements, false otherwise
Exception Safety:	noexcept	
Description:	Return whether this Span is empty.	

⌋*(RS_AP_00130)*

[SWS_CORE_01970]{DRAFT} Definition of API function ara::core::Span::operator[] ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	operator[](size_type idx)	
Syntax:	`constexpr reference operator[] (size_type idx) const;`	
Parameters (in):	idx	the index into this Span
Return value:	reference	the reference
Exception Safety:	not exception safe	
Description:	Return a reference to the n-th element of this Span.	

⌋*(RS_AP_00130)*

[SWS_CORE_01959]{DRAFT} Definition of API function ara::core::Span::front ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	class ara::core::Span
Symbol:	front()

▽

△

Syntax:	`constexpr reference front () const;`	
Return value:	reference	the reference
Exception Safety:	not exception safe	
Description:	Return a reference to the first element of this Span. The behavior of this function is undefined if empty() is true.	

⌋*(RS_AP_00130)*

[SWS_CORE_01960]{DRAFT} **Definition of API function ara::core::Span::back** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	back()	
Syntax:	`constexpr reference back () const;`	
Return value:	reference	the reference
Exception Safety:	not exception safe	
Description:	Return a reference to the last element of this Span. The behavior of this function is undefined if empty() is true.	

⌋*(RS_AP_00130)*

[SWS_CORE_01971]{DRAFT} **Definition of API function ara::core::Span::data** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	data()	
Syntax:	`constexpr pointer data () const noexcept;`	
Return value:	pointer	the pointer
Exception Safety:	noexcept	
Description:	Return a pointer to the start of the memory block covered by this Span.	

⌋*(RS_AP_00130)*

[SWS_CORE_01972]{DRAFT} **Definition of API function ara::core::Span::begin** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	begin()	
Syntax:	`constexpr iterator begin () const noexcept;`	
Return value:	iterator	the iterator
Exception Safety:	noexcept	
Description:	Return an iterator pointing to the first element of this Span.	

⌋*(RS_AP_00130)*

[SWS_CORE_01973]{DRAFT} **Definition of API function ara::core::Span::end** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	end()	
Syntax:	`constexpr iterator end () const noexcept;`	
Return value:	iterator	the iterator
Exception Safety:	noexcept	
Description:	Return an iterator pointing past the last element of this Span.	

⌋*(RS_AP_00130)*

[SWS_CORE_01974]{DRAFT} **Definition of API function ara::core::Span::cbegin** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	cbegin()	
Syntax:	`constexpr const_iterator cbegin () const noexcept;`	
Return value:	const_iterator	the const_iterator
Exception Safety:	noexcept	
Description:	Return a const_iterator pointing to the first element of this Span.	

⌋*(RS_AP_00130)*

[SWS_CORE_01975]{DRAFT} **Definition of API function ara::core::Span::cend** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	cend()	
Syntax:	`constexpr const_iterator cend () const noexcept;`	
Return value:	const_iterator	the const_iterator
Exception Safety:	noexcept	
Description:	Return a const_iterator pointing past the last element of this Span.	

⌋*(RS_AP_00130)*

[SWS_CORE_01976]{DRAFT} **Definition of API function ara::core::Span::rbegin** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	rbegin()	
Syntax:	`constexpr reverse_iterator rbegin () const noexcept;`	
Return value:	reverse_iterator	the reverse_iterator

▽

△

Exception Safety:	noexcept
Description:	Return a reverse_iterator pointing to the last element of this Span.

⌋*(RS_AP_00130)*

[SWS_CORE_01977]{DRAFT} **Definition of API function ara::core::Span::rend** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	rend()	
Syntax:	`constexpr reverse_iterator rend () const noexcept;`	
Return value:	reverse_iterator	the reverse_iterator
Exception Safety:	noexcept	
Description:	Return a reverse_iterator pointing past the first element of this Span.	

⌋*(RS_AP_00130)*

[SWS_CORE_01978]{DRAFT} **Definition of API function ara::core::Span::crbegin** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	crbegin()	
Syntax:	`constexpr const_reverse_iterator crbegin () const noexcept;`	
Return value:	const_reverse_iterator	the const_reverse_iterator
Exception Safety:	noexcept	
Description:	Return a const_reverse_iterator pointing to the last element of this Span.	

⌋*(RS_AP_00130)*

[SWS_CORE_01979]{DRAFT} **Definition of API function ara::core::Span::crend** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	class ara::core::Span	
Symbol:	crend()	
Syntax:	`constexpr const_reverse_iterator crend () const noexcept;`	
Return value:	const_reverse_iterator	the reverse_iterator
Exception Safety:	noexcept	
Description:	Return a const_reverse_iterator pointing past the first element of this Span.	

⌋*(RS_AP_00130)*

Some non-member factory functions for `ara::core::Span` allow to create instances without explicitly mentioning the template parameter type – this type is being deduced from the functions' arguments:

[SWS_CORE_01990]{DRAFT} **Definition of API function ara::core::MakeSpan** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	namespace ara::core	
Symbol:	MakeSpan(T *ptr, typename Span< T >::size_type count)	

Syntax:	`template <typename T>`
	`constexpr Span< T > MakeSpan (T *ptr, typename Span< T >::size_type count);`	

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:	not exception safe	
Description:	Create a new Span from the given pointer and size.	

⌋*(RS_AP_00130)*

[SWS_CORE_01991]{DRAFT} **Definition of API function ara::core::MakeSpan** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	namespace ara::core	
Symbol:	MakeSpan(T *firstElem, T *lastElem)	

Syntax:	`template <typename T>`
	`constexpr Span< T > MakeSpan (T *firstElem, T *lastElem);`	

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:	not exception safe	
Description:	Create a new Span from the open range between [firstElem, lastElem).	

⌋*(RS_AP_00130)*

[SWS_CORE_01992]{DRAFT} **Definition of API function ara::core::MakeSpan** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	namespace ara::core	
Symbol:	MakeSpan(T(&arr)[N])	

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:	noexcept	
Description:	Create a new Span from the given raw array.	

⌋*(RS_AP_00130)*

[SWS_CORE_01993]{DRAFT} **Definition of API function ara::core::MakeSpan** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	namespace ara::core	
Symbol:	MakeSpan(Container &cont)	
Syntax:	`template <typename Container>` `constexpr Span< typename Container::value_type > MakeSpan (Container &cont);`	
Template param:	Container	the type of container
Parameters (in):	cont	the container
Return value:	Span< typename Container::value_type >	the new Span
Exception Safety:	not exception safe	
Description:	Create a new Span from the given container.	

⌋*(RS_AP_00130)*

[SWS_CORE_01994]{DRAFT} **Definition of API function ara::core::MakeSpan** ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	namespace ara::core	
Symbol:	MakeSpan(const Container &cont)	
Syntax:	`template <typename Container>` `constexpr Span< typename Container::value_type const > MakeSpan (const Container &cont);`	
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:	not exception safe	
Description:	Create a new Span from the given const container.	

⌋*(RS_AP_00130)*

These non-member functions allow to “convert” a `Span<T>` into a `Span<Byte>`, thereby gaining access to the in-memory representation of the object referenced by a `Span` instance.

Unlike `std::byte` from [9, the C++17 standard], it is implementation-defined whether `ara::core::Byte` can be used for type aliasing without triggering Undefined Behavior. This may also affect `ara::core::as_bytes` and `ara::core::as_writable_bytes` in particular. Implementations usually provide a way to make this safe by loosening the aliasing restrictions of the C++ compiler.

[SWS_CORE_01980]{DRAFT} Definition of API function ara::core::as_bytes ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	namespace ara::core	
Symbol:	as_bytes(Span< ElementType, Extent > s)	
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:	noexcept	
Description:	Return a read-only Span<Byte> over the object representation of the input Span<T>	

⌋*(RS_AP_00130)*

[SWS_CORE_01981]{DRAFT} Definition of API function ara::core::as_writable_bytes ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	namespace ara::core	
Symbol:	as_writable_bytes(Span< ElementType, Extent > s)	
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:	noexcept	
Description:	Return a writable Span<Byte> over the object representation of the input Span<T>	

⌋*(RS_AP_00130)*

8.1.15 `SteadyClock` data type

[SWS_CORE_06401] Definition of API class ara::core::SteadyClock ⌈

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.

⌋*(RS_AP_00130)*

[SWS_CORE_06412] Definition of API type ara::core::SteadyClock::rep ⌈

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 .

⌋*(RS_AP_00130)*

[SWS_CORE_06413] Definition of API type ara::core::SteadyClock::period ⌈

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 .

⌋*(RS_AP_00130)*

[SWS_CORE_06411] Definition of API type ara::core::SteadyClock::duration ⌈

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>

⌋*(RS_AP_00130)*

[SWS_CORE_06414] Definition of API type ara::core::SteadyClock::time_point ⌈

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>

⌋*(RS_AP_00130)*

[SWS_CORE_06431] Definition of API variable ara::core::SteadyClock::is_steady ⌈

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

⌋*(RS_AP_00130)*

[SWS_CORE_06432] Definition of API function ara::core::SteadyClock::now ⌈

Kind:	function	
Header file:	#include "ara/core/steady_clock.h"	
Scope:	class ara::core::SteadyClock	
Symbol:	now()	
Syntax:	`static time_point now () noexcept;`	
Return value:	time_point	a time_point
Exception Safety:	noexcept	
Description:	Return a time_point representing the current value of the clock.	

⌋*(RS_AP_00130)*

8.1.16 `InstanceSpecifier` data type

This section defines the `ara::core::InstanceSpecifier` type that describes the path to a meta model element.

[SWS_CORE_08001] Definition of API class ara::core::InstanceSpecifier ⌈

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:	class representing an AUTOSAR Instance Specifier, which is basically an AUTOSAR shortname-path wrapper.

⌋*(RS_AP_00140, RS_Main_00320)*

[SWS_CORE_08021] Definition of API function ara::core::InstanceSpecifier::InstanceSpecifier ⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	class ara::core::InstanceSpecifier	
Symbol:	InstanceSpecifier(StringView metaModelIdentifier)	
Syntax:	`explicit InstanceSpecifier (StringView metaModelIdentifier);`	
Parameters (in):	metaModelIdentifier	string representation of a valid InstanceSpecifier, according to the syntax rules given by SWS_CORE_10200 and SWS_CORE_10203.
Exceptions:	CoreException	in case the given metaModelIdentifier is not a valid meta-model identifier/short name path.
Description:	throwing ctor from meta-model string	

⌋*(RS_Main_00320)*

[SWS_CORE_08022] Definition of API function ara::core::InstanceSpecifier::InstanceSpecifier ⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	class ara::core::InstanceSpecifier	
Symbol:	InstanceSpecifier(const InstanceSpecifier &other)	
Syntax:	`InstanceSpecifier (const InstanceSpecifier &other);`	
Parameters (in):	other	the other instance
Exception Safety:	not exception safe	
Description:	Copy constructor.	

⌋*(RS_Main_00320)*

[SWS_CORE_08023] Definition of API function ara::core::InstanceSpecifier::InstanceSpecifier ⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	class ara::core::InstanceSpecifier	
Symbol:	InstanceSpecifier(InstanceSpecifier &&other)	
Syntax:	`InstanceSpecifier (InstanceSpecifier &&other) noexcept;`	
Parameters (in):	other	the other instance
Exception Safety:	noexcept	
Description:	Move constructor.	

⌋*(RS_Main_00320)*

[SWS_CORE_08024] Definition of API function ara::core::InstanceSpecifier::operator= ⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	class ara::core::InstanceSpecifier	
Symbol:	operator=(const InstanceSpecifier &other)	
Syntax:	`InstanceSpecifier & operator= (const InstanceSpecifier &other);`	
Parameters (in):	other	the other instance
Return value:	InstanceSpecifier &	*this
Exception Safety:	not exception safe	
Description:	Copy assignment operator.	

⌋*(RS_Main_00320)*

[SWS_CORE_08025] Definition of API function ara::core::InstanceSpecifier::operator= ⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	class ara::core::InstanceSpecifier	
Symbol:	operator=(InstanceSpecifier &&other)	
Syntax:	`InstanceSpecifier & operator= (InstanceSpecifier &&other);`	
Parameters (in):	other	the other instance
Return value:	InstanceSpecifier &	*this
Exception Safety:	not exception safe	
Description:	Move assignment operator.	

⌋*(RS_Main_00320)*

[SWS_CORE_08029] Definition of API function ara::core::InstanceSpecifier::~InstanceSpecifier ⌈

Kind:	function
Header file:	#include "ara/core/instance_specifier.h"
Scope:	class ara::core::InstanceSpecifier
Symbol:	~InstanceSpecifier()
Syntax:	`~InstanceSpecifier () noexcept;`
Exception Safety:	noexcept
Description:	Destructor.

⌋*(RS_AP_00134, RS_Main_00320)*

[SWS_CORE_08032] Definition of API function ara::core::InstanceSpecifier::Create ⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	class ara::core::InstanceSpecifier	
Symbol:	Create(StringView metaModelIdentifier)	
Syntax:	`static Result< InstanceSpecifier > Create (StringView metaModel Identifier) noexcept;`	
Parameters (in):	metaModelIdentifier	string representation of a valid InstanceSpecifier, according to the syntax rules given by SWS_CORE_10200 and SWS_ CORE_10203.
Return value:	Result< Instance Specifier >	a Result, containing either a syntactically valid InstanceSpecifier, or an ErrorCode
Exception Safety:	noexcept	
Errors:	CoreErrc::kInvalidMeta ModelShortname	if any of the path elements of metaModelIdentifier is missing or contains invalid characters
	CoreErrc::kInvalidMeta ModelPath	if the metaModelIdentifier is not a valid path to a model element
Description:	Create a new instance of this class.	

⌋*(RS_Main_00150, RS_AP_00137, RS_AP_00136)*

[SWS_CORE_08042] Definition of API function ara::core::InstanceSpecifier::operator== ⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	class ara::core::InstanceSpecifier	
Symbol:	operator==(const InstanceSpecifier &other)	
Syntax:	`bool operator== (const InstanceSpecifier &other) const noexcept;`	
Parameters (in):	other	InstanceSpecifier instance to compare this one with.
Return value:	bool	true in case both InstanceSpecifiers are denoting exactly the same model element, false otherwise.
Exception Safety:	noexcept	
Description:	eq operator to compare with other InstanceSpecifier instance.	

⌋*(RS_Main_00320)*

[SWS_CORE_08043] Definition of API function ara::core::InstanceSpecifier::operator== ⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	class ara::core::InstanceSpecifier	
Symbol:	operator==(StringView other)	
Syntax:	`bool operator== (StringView other) const noexcept;`	
Parameters (in):	other	string representation to compare this one with.
Return value:	bool	true in case this InstanceSpecifier is denoting exactly the same model element as other, false otherwise.

▽

△

Exception Safety:	noexcept
Description:	eq operator to compare with other InstanceSpecifier instance.

⌋(*RS_Main_00320*)

[SWS_CORE_08044] Definition of API function ara::core::InstanceSpecifier::operator!= ⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	class ara::core::InstanceSpecifier	
Symbol:	operator!=(const InstanceSpecifier &other)	
Syntax:	`bool operator!= (const InstanceSpecifier &other) const noexcept;`	
Parameters (in):	other	InstanceSpecifier instance to compare this one with.
Return value:	bool	false in case both InstanceSpecifiers are denoting exactly the same model element, true otherwise.
Exception Safety:	noexcept	
Description:	uneq operator to compare with other InstanceSpecifier instance.	

⌋(*RS_Main_00320*)

[SWS_CORE_08045] Definition of API function ara::core::InstanceSpecifier::operator!= ⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	class ara::core::InstanceSpecifier	
Symbol:	operator!=(StringView other)	
Syntax:	`bool operator!= (StringView other) const noexcept;`	
Parameters (in):	other	string representation to compare this one with.
Return value:	bool	false in case this InstanceSpecifier is denoting exactly the same model element as other, true otherwise.
Exception Safety:	noexcept	
Description:	uneq operator to compare with other InstanceSpecifier string representation.	

⌋(*RS_Main_00320*)

[SWS_CORE_08046] Definition of API function ara::core::InstanceSpecifier::operator< ⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	class ara::core::InstanceSpecifier	
Symbol:	operator<(const InstanceSpecifier &other)	
Syntax:	`bool operator< (const InstanceSpecifier &other) const noexcept;`	
Parameters (in):	other	InstanceSpecifier instance to compare this one with.
Return value:	bool	true in case this InstanceSpecifier is lexically lower than other, false otherwise.

▽

△

Exception Safety:	noexcept
Description:	lower than operator to compare with other InstanceSpecifier for ordering purposes (f.i. when collecting identifiers in maps).

⌋*(RS_Main_00320)*

[SWS_CORE_08041] Definition of API function ara::core::InstanceSpecifier::ToString ⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	class ara::core::InstanceSpecifier	
Symbol:	ToString()	
Syntax:	`StringView ToString () const noexcept;`	
Return value:	StringView	stringified form of InstanceSpecifier. Lifetime of the underlying string is only guaranteed for the lifetime of the underlying string of the StringView passed to the constructor.
Exception Safety:	noexcept	
Description:	method to return the stringified form of InstanceSpecifier	

⌋*(RS_Main_00320)*

[SWS_CORE_08081] Definition of API function ara::core::operator== ⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	namespace ara::core	
Symbol:	operator==(StringView lhs, const InstanceSpecifier &rhs)	
Syntax:	`bool operator== (StringView lhs, const InstanceSpecifier &rhs) noexcept;`	
Parameters (in):	lhs	stringified form of a InstanceSpecifier
	rhs	an InstanceSpecifier
Return value:	bool	true in case rhs string representation equals lhs
Exception Safety:	noexcept	
Description:	Non-member function operator== to allow StringView on lhs.	

⌋*(RS_Main_00320)*

[SWS_CORE_08082] Definition of API function ara::core::operator!= ⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	namespace ara::core	
Symbol:	operator!=(StringView lhs, const InstanceSpecifier &rhs)	
Syntax:	`bool operator!= (StringView lhs, const InstanceSpecifier &rhs) noexcept;`	
Parameters (in):	lhs	stringified form of a InstanceSpecifier
	rhs	an InstanceSpecifier
Return value:	bool	true in case rhs string representation not equals lhs

▽

△

Exception Safety:	noexcept
Description:	Non-member function operator!= to allow StringView on lhs.

⌋*(RS_Main_00320)*

8.1.17 Polymorphic Memory Resources

[SWS_CORE_06561]{DRAFT} **Definition of API function ara::core::operator==** ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	namespace ara::core	
Symbol:	operator==(const MemoryResource &a, const MemoryResource &b)	
Syntax:	`bool operator== (const MemoryResource &a, const MemoryResource &b) noexcept;`	
Parameters (in):	a	left side of the comparision
	b	right side of the camparision
Return value:	bool	true if the two instances compare equal, false otherwise
Exception Safety:	noexcept	
Description:	This function behaves the same as the corresponding std::pmr::operator== function.	

⌋*(RS_AP_00130)*

[SWS_CORE_06560]{DRAFT} **Definition of API function ara::core::operator==** ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	namespace ara::core	
Symbol:	operator==(const PolymorphicAllocator< T1 > &a, const PolymorphicAllocator< T2 > &b)	
Syntax:	`template <class T1, class T2>` `bool operator== (const PolymorphicAllocator< T1 > &a, const PolymorphicAllocator< T2 > &b) noexcept;`	
Parameters (in):	a	left side of the comparision
	b	right side of the camparision
Return value:	bool	true if the two instances compare equal, false otherwise
Exception Safety:	noexcept	
Description:	This function behaves the same as the corresponding std::pmr::operator== function.	

⌋*(RS_AP_00130)*

[SWS_CORE_06562]{DRAFT} **Definition of API function ara::core::NewDeleteResource** ⌈

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

▽

△

Symbol:	NewDeleteResource()	
Syntax:	`MemoryResource * NewDeleteResource () noexcept;`	
Return value:	MemoryResource *	a pointer to a MemoryResource that uses the global operator new and operator delete to allocate memory.
Exception Safety:	noexcept	
Description:	This function behaves the same as the corresponding std::pmr function.	

⌋*(RS_AP_00130)*

[SWS_CORE_06563]{DRAFT} **Definition of API function ara::core::NullMemoryResource** ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	namespace ara::core	
Symbol:	NullMemoryResource()	
Syntax:	`MemoryResource * NullMemoryResource () noexcept;`	
Return value:	MemoryResource *	–
Exception Safety:	noexcept	
Description:	This function behaves the same as the corresponding std::pmr function.	

⌋*(RS_AP_00130)*

[SWS_CORE_06564]{DRAFT} **Definition of API function ara::core::SetDefaultResource** ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	namespace ara::core	
Symbol:	SetDefaultResource(MemoryResource *r)	
Syntax:	`MemoryResource * SetDefaultResource (MemoryResource *r) noexcept;`	
DIRECTION NOT DEFINED	r	–
Return value:	MemoryResource *	–
Exception Safety:	noexcept	
Description:	This function behaves the same as the corresponding std::pmr function.	

⌋*(RS_AP_00130)*

[SWS_CORE_06565]{DRAFT} **Definition of API function ara::core::GetDefaultResource** ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	namespace ara::core	
Symbol:	GetDefaultResource()	
Syntax:	`MemoryResource * GetDefaultResource () noexcept;`	
Return value:	MemoryResource *	–

▽

△

Exception Safety:	noexcept
Description:	This function behaves the same as the corresponding std::pmr function.

⌋*(RS_AP_00130)*

8.1.17.1 `MemoryResource` data type

[SWS_CORE_06500]{DRAFT} **Definition of API class ara::core::MemoryResource**
⌈

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:	Provides ara::core specific MemoryResources derived from std::pmr::memory_resource.

⌋*(RS_AP_00130)*

[SWS_CORE_06501]{DRAFT} **Definition of API function ara::core::MemoryResource::MemoryResource** ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	class ara::core::MemoryResource
Symbol:	MemoryResource()
Syntax:	`MemoryResource ()=default;`
Description:	Default constructor. This function behaves the same as the corresponding std::pmr::memory_resource function.

⌋*(RS_AP_00130)*

[SWS_CORE_06502]{DRAFT} **Definition of API function ara::core::MemoryResource::MemoryResource** ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	class ara::core::MemoryResource	
Symbol:	MemoryResource(const MemoryResource &other)	
Syntax:	`MemoryResource (const MemoryResource &other)=default;`	
Parameters (in):	other	the other instance
Description:	Default copy constructor. This function behaves the same as the corresponding std::pmr::memory_resource function.	

⌋*(RS_AP_00130)*

[SWS_CORE_06506]{DRAFT} **Definition of API function ara::core::MemoryResource::~MemoryResource** ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	class ara::core::MemoryResource
Symbol:	~MemoryResource()
Syntax:	`virtual ~MemoryResource ();`
Exception Safety:	not exception safe
Description:	destructor This function behaves the same as the corresponding std::pmr::memory_resource function.

⌋*(RS_AP_00130)*

[SWS_CORE_06507]{DRAFT} **Definition of API function ara::core::MemoryResource::operator=** ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	class ara::core::MemoryResource	
Symbol:	operator=(const MemoryResource &other)	
Syntax:	`MemoryResource & operator= (const MemoryResource &other)=default;`	
Parameters (in):	other	the other instance
Return value:	MemoryResource &	–
Description:	Copy assignment operator. This function behaves the same as the corresponding std::pmr::memory_resource function.	

⌋*(RS_AP_00130)*

[SWS_CORE_06503]{DRAFT} **Definition of API function ara::core::MemoryResource::allocate** ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	class ara::core::MemoryResource	
Symbol:	allocate(std::size_t bytes, std::size_t alignment=max_align)	
Syntax:	`ARA_COMPILER_DEFINED_NODISCARD void * allocate (std::size_t bytes, std::size_t alignment=max_align) noexcept;`	
Parameters (in):	bytes	size of at bytes to be at least allocated
	alignment	defined the alignement of the allocated memory
Return value:	ARA_COMPILER_DEFINED_NODISCARD void *	allocated storage with a size of at least bytes bytes, aligned to the specified alignment; or nullptr.
Exception Safety:	noexcept	
Description:	Allocates storage. This function behaves the same as the corresponding std::pmr::memory_resource function. Any error like failed allocation is indicated by a returned nullptr.	

⌋*(RS_AP_00130)*

[SWS_CORE_06504]{DRAFT} **Definition of API function ara::core::MemoryResource::deallocate** ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	class ara::core::MemoryResource	
Symbol:	deallocate(void *p, std::size_t bytes, std::size_t alignment=max_align)	
Syntax:	`void deallocate (void *p, std::size_t bytes, std::size_t alignment=max_align) noexcept;`	
Parameters (in):	p	points to the storage to be deallocated
	bytes	size of at bytes to be dellocated
	alignment	defined the alignement of the allocated memory
Return value:	None	
Exception Safety:	noexcept	
Description:	Deallocates storage. This function behaves the same as the corresponding std::pmr::memory_resource function. Any error is silently ignored.	

⌋*(RS_AP_00130)*

[SWS_CORE_06505]{DRAFT} **Definition of API function ara::core::MemoryResource::is_equal** ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	class ara::core::MemoryResource	
Symbol:	is_equal(const MemoryResource &other)	
Syntax:	`bool is_equal (const MemoryResource &other) const noexcept;`	
Parameters (in):	other	points to the storage to be compared
Return value:	bool	true if the instances compare equal, false otherwise
Exception Safety:	noexcept	
Description:	compare for equality with another memory_resource This function behaves the same as the corresponding std::pmr::memory_resource function.	

⌋*(RS_AP_00130)*

8.1.17.2 `MonotonicBufferResource` data type

[SWS_CORE_06520]{DRAFT} **Definition of API class ara::core::MonotonicBuffer Resource** ⌈

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:	Provides ara::core specific MonotonicBufferResource derived from std::pmr::monotonic_buffer_resource.
Visibility:	private

⌋*(RS_AP_00130)*

[SWS_CORE_06521]{DRAFT} **Definition of API function ara::core::MonotonicBufferResource::MonotonicBufferResource** ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	class ara::core::MonotonicBufferResource	
Symbol:	MonotonicBufferResource(MemoryResource *upstream)	
Syntax:	`explicit MonotonicBufferResource (MemoryResource *upstream) noexcept;`	
DIRECTION NOT DEFINED	upstream	–
Exception Safety:	noexcept	
Description:	constructor This function behaves the same as the corresponding std::pmr::monotonic_buffer_resource function.	

⌋*(RS_AP_00130)*

[SWS_CORE_06522]{DRAFT} **Definition of API function ara::core::MonotonicBufferResource::MonotonicBufferResource** ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	class ara::core::MonotonicBufferResource	
Symbol:	MonotonicBufferResource(std::size_t initial_size, MemoryResource *upstream)	
Syntax:	`MonotonicBufferResource (std::size_t initial_size, MemoryResource *upstream) noexcept;`	
DIRECTION NOT DEFINED	initial_size	–
	upstream	–
Exception Safety:	noexcept	
Description:	constructor This function behaves the same as the corresponding std::pmr::monotonic_buffer_resource function.	

⌋*(RS_AP_00130)*

[SWS_CORE_06523]{DRAFT} **Definition of API function ara::core::MonotonicBufferResource::MonotonicBufferResource** ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	class ara::core::MonotonicBufferResource

▽

△

Symbol:	MonotonicBufferResource(void *buffer, std::size_t buffer_size, MemoryResource *upstream)	
Syntax:	`MonotonicBufferResource (void *buffer, std::size_t buffer_size, MemoryResource *upstream) noexcept;`	
DIRECTION NOT DEFINED	buffer	–
	buffer_size	–
	upstream	–
Exception Safety:	noexcept	
Description:	constructor This function behaves the same as the corresponding std::pmr::monotonic_buffer_resource function.	

⌋(*RS_AP_00130*)

[SWS_CORE_06524]{DRAFT} **Definition of API function ara::core::MonotonicBufferResource::MonotonicBufferResource** ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	class ara::core::MonotonicBufferResource
Symbol:	MonotonicBufferResource()
Syntax:	`MonotonicBufferResource ();`
Exception Safety:	not exception safe
Description:	constructor This function behaves the same as the corresponding std::pmr::monotonic_buffer_resource function.

⌋(*RS_AP_00130*)

[SWS_CORE_06525]{DRAFT} **Definition of API function ara::core::MonotonicBufferResource::MonotonicBufferResource** ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	class ara::core::MonotonicBufferResource	
Symbol:	MonotonicBufferResource(std::size_t initial_size)	
Syntax:	`explicit MonotonicBufferResource (std::size_t initial_size);`	
DIRECTION NOT DEFINED	initial_size	–
Exception Safety:	not exception safe	
Description:	constructor This function behaves the same as the corresponding std::pmr::monotonic_buffer_resource function.	

⌋(*RS_AP_00130*)

[SWS_CORE_06526]{DRAFT} **Definition of API function ara::core::MonotonicBufferResource::MonotonicBufferResource** ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	class ara::core::MonotonicBufferResource	
Symbol:	MonotonicBufferResource(void *buffer, std::size_t buffer_size)	
Syntax:	`MonotonicBufferResource (void *buffer, std::size_t buffer_size);`	
DIRECTION NOT DEFINED	buffer	–
	buffer_size	–
Exception Safety:	not exception safe	
Description:	Default constructor. This function behaves the same as the corresponding std::pmr::monotonic_buffer_resource function.	

⌋*(RS_AP_00130)*

[SWS_CORE_06527]{DRAFT} **Definition of API function ara::core::MonotonicBufferResource::MonotonicBufferResource** ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	class ara::core::MonotonicBufferResource
Symbol:	MonotonicBufferResource(const MonotonicBufferResource &)
Syntax:	`MonotonicBufferResource (const MonotonicBufferResource &)=delete;`
Description:	Deleted copy constructor. This function behaves the same as the corresponding std::pmr::monotonic_buffer_resource function.

⌋*(RS_AP_00130)*

[SWS_CORE_06528]{DRAFT} **Definition of API function ara::core::MonotonicBufferResource::~MonotonicBufferResource** ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	class ara::core::MonotonicBufferResource
Symbol:	~MonotonicBufferResource()
Syntax:	`virtual ~MonotonicBufferResource ();`
Exception Safety:	not exception safe
Description:	Deconstructor. This function behaves the same as the corresponding std::pmr::monotonic_buffer_resource function.

⌋*(RS_AP_00130)*

[SWS_CORE_06529]{DRAFT} **Definition of API function ara::core::MonotonicBufferResource::operator=** ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	class ara::core::MonotonicBufferResource
Symbol:	operator=(const MonotonicBufferResource &)
Syntax:	`MonotonicBufferResource & operator= (const MonotonicBufferResource &)=delete;`
Description:	Deleted copy operator. This function behaves the same as the corresponding std::pmr::monotonic_buffer_resource function.

⌋*(RS_AP_00130)*

[SWS_CORE_06530]{DRAFT} **Definition of API function ara::core::MonotonicBufferResource::release** ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	class ara::core::MonotonicBufferResource
Symbol:	release()
Syntax:	`void release () noexcept;`
Return value:	None
Exception Safety:	noexcept
Description:	This function behaves the same as the corresponding std::pmr::monotonic_buffer_resource function.

⌋*(RS_AP_00130)*

[SWS_CORE_06531]{DRAFT} **Definition of API function ara::core::MonotonicBufferResource::upstream_resource** ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	class ara::core::MonotonicBufferResource	
Symbol:	upstream_resource()	
Syntax:	`MemoryResource * upstream_resource () const noexcept;`	
Return value:	MemoryResource *	–
Exception Safety:	noexcept	
Description:	This function behaves the same as the corresponding std::pmr::monotonic_buffer_resource function.	

⌋*(RS_AP_00130)*

8.1.17.3 `PolymorphicAllocator` data type

[SWS_CORE_06540]{DRAFT} **Definition of API class ara::core::PolymorphicAllocator** ⌈

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	the type of values
Description:	Provides ara::core specific PolymorphicAllocator derived from std::pmr::polymorphic_allocator.	

⌋*(RS_AP_00130)*

[SWS_CORE_06541]{DRAFT} **Definition of API function ara::core::PolymorphicAllocator::PolymorphicAllocator** ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	class ara::core::PolymorphicAllocator
Symbol:	PolymorphicAllocator()
Syntax:	`PolymorphicAllocator () noexcept;`
Exception Safety:	noexcept
Description:	This function behaves the same as the corresponding std::pmr::polymorphic_allocator function.

⌋*(RS_AP_00130)*

[SWS_CORE_06542]{DRAFT} **Definition of API function ara::core::PolymorphicAllocator::PolymorphicAllocator** ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	class ara::core::PolymorphicAllocator	
Symbol:	PolymorphicAllocator(MemoryResource *r)	
Syntax:	`PolymorphicAllocator (MemoryResource *r) noexcept;`	
DIRECTION NOT DEFINED	r	–
Exception Safety:	noexcept	
Description:	This function behaves the same as the corresponding std::pmr::polymorphic_allocator function.	

⌋*(RS_AP_00130)*

[SWS_CORE_06543]{DRAFT} **Definition of API function ara::core::PolymorphicAllocator::PolymorphicAllocator** ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	class ara::core::PolymorphicAllocator	
Symbol:	PolymorphicAllocator(const PolymorphicAllocator &other)	
Syntax:	`PolymorphicAllocator (const PolymorphicAllocator &other)=default;`	
DIRECTION NOT DEFINED	other	–
Description:	This function behaves the same as the corresponding std::pmr::polymorphic_allocator function.	

⌋*(RS_AP_00130)*

[SWS_CORE_06544]{DRAFT} **Definition of API function ara::core::PolymorphicAllocator::PolymorphicAllocator** ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	class ara::core::PolymorphicAllocator	
Symbol:	PolymorphicAllocator(const PolymorphicAllocator< U > &other)	

Syntax:	`template <class U>`
	`PolymorphicAllocator (const PolymorphicAllocator< U > &other)`
	`noexcept;`	

DIRECTION NOT DEFINED	other	–
Exception Safety:	noexcept	
Description:	This function behaves the same as the corresponding std::pmr::polymorphic_allocator function.	

⌋*(RS_AP_00130)*

[SWS_CORE_06546]{DRAFT} **Definition of API function ara::core::PolymorphicAllocator::PolymorphicAllocator** ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	class ara::core::PolymorphicAllocator	
Symbol:	PolymorphicAllocator(PolymorphicAllocator &&other)	
Syntax:	`PolymorphicAllocator (PolymorphicAllocator &&other) noexcept;`	
DIRECTION NOT DEFINED	other	–
Exception Safety:	noexcept	
Description:	This function behaves the same as the corresponding std::pmr::polymorphic_allocator function.	

⌋*(RS_AP_00130)*

[SWS_CORE_06547]{DRAFT} **Definition of API function ara::core::Polymorphic Allocator::allocate** ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	class ara::core::PolymorphicAllocator	
Symbol:	allocate(std::size_t n)	
Syntax:	`ARA_COMPILER_DEFINED_NODISCARD Tp * allocate (std::size_t n) noexcept;`	
DIRECTION NOT DEFINED	n	–
Return value:	ARA_COMPILER_ DEFINED_NODISCARD Tp *	–
Exception Safety:	noexcept	
Description:	This function behaves the same as the corresponding std::pmr::polymorphic_allocator function.	

⌋*(RS_AP_00130)*

[SWS_CORE_06549]{DRAFT} **Definition of API function ara::core::Polymorphic Allocator::allocate_bytes** ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	class ara::core::PolymorphicAllocator	
Symbol:	allocate_bytes(std::size_t nbytes, std::size_t alignment=alignof(std::max_align_t))	
Syntax:	`ARA_COMPILER_DEFINED_NODISCARD void * allocate_bytes (std::size_t nbytes, std::size_t alignment=alignof(std::max_align_t)) noexcept;`	
DIRECTION NOT DEFINED	nbytes	–
	alignment	–
Return value:	ARA_COMPILER_ DEFINED_NODISCARD void *	–
Exception Safety:	noexcept	
Description:	This function behaves the same as the corresponding std::pmr::polymorphic_allocator function.	

⌋*(RS_AP_00130)*

[SWS_CORE_06551]{DRAFT} **Definition of API function ara::core::Polymorphic Allocator::allocate_object** ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	class ara::core::PolymorphicAllocator	
Symbol:	allocate_object(std::size_t n=1)	
Syntax:	`template <class T>` `ARA_COMPILER_DEFINED_NODISCARD T * allocate_object (std::size_t n=1) noexcept;`	
DIRECTION NOT DEFINED	n	–

▽

△

Return value:	ARA_COMPILER_ DEFINED_NODISCARD T *	–
Exception Safety:	noexcept	
Description:	This function behaves the same as the corresponding std::pmr::polymorphic_allocator function.	

⌋*(RS_AP_00130)*

[SWS_CORE_06555]{DRAFT} **Definition of API function ara::core::Polymorphic Allocator::construct** ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	class ara::core::PolymorphicAllocator	
Symbol:	construct(T *p, Args &&... args)	
Syntax:	`template <class T, class... Args>` `void construct (T *p, Args &&... args);`	
DIRECTION NOT DEFINED	p	–
	args	–
Return value:	None	
Exception Safety:	not exception safe	
Description:	This function behaves the same as the corresponding std::pmr::polymorphic_allocator function.	

⌋*(RS_AP_00130)*

[SWS_CORE_06548]{DRAFT} **Definition of API function ara::core::Polymorphic Allocator::deallocate** ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	class ara::core::PolymorphicAllocator	
Symbol:	deallocate(Tp *p, std::size_t n)	
Syntax:	`void deallocate (Tp *p, std::size_t n) noexcept;`	
DIRECTION NOT DEFINED	p	–
	n	–
Return value:	None	
Exception Safety:	noexcept	
Description:	This function behaves the same as the corresponding std::pmr::polymorphic_allocator function.	

⌋*(RS_AP_00130)*

[SWS_CORE_06550]{DRAFT} **Definition of API function ara::core::Polymorphic Allocator::deallocate_bytes** ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	class ara::core::PolymorphicAllocator
Symbol:	deallocate_bytes(void *p, std::size_t nbytes, std::size_t alignment=alignof(std::max_align_t))

▽

△

Syntax:	`void deallocate_bytes (void *p, std::size_t nbytes, std::size_t alignment=alignof(std::max_align_t)) noexcept;`	
DIRECTION NOT DEFINED	p	–
	nbytes	–
	alignment	–
Return value:	None	
Exception Safety:	noexcept	
Description:	This function behaves the same as the corresponding std::pmr::polymorphic_allocator function.	

⌋*(RS_AP_00130)*

[SWS_CORE_06552]{DRAFT} **Definition of API function ara::core::PolymorphicAllocator::deallocate_object** ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	class ara::core::PolymorphicAllocator	
Symbol:	deallocate_object(T *p, std::size_t n=1)	
Syntax:	`template <class T>` `void deallocate_object (T *p, std::size_t n=1) noexcept;`	
DIRECTION NOT DEFINED	p	–
	n	–
Return value:	None	
Exception Safety:	noexcept	
Description:	This function behaves the same as the corresponding std::pmr::polymorphic_allocator function.	

⌋*(RS_AP_00130)*

[SWS_CORE_06554]{DRAFT} **Definition of API function ara::core::PolymorphicAllocator::delete_object** ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	class ara::core::PolymorphicAllocator	
Symbol:	delete_object(T *p)	
Syntax:	`template <class T>` `void delete_object (T *p) noexcept;`	
DIRECTION NOT DEFINED	p	–
Return value:	None	
Exception Safety:	noexcept	
Description:	This function behaves the same as the corresponding std::pmr::polymorphic_allocator function.	

⌋*(RS_AP_00130)*

[SWS_CORE_06556]{DRAFT} **Definition of API function ara::core::PolymorphicAllocator::destroy** ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	class ara::core::PolymorphicAllocator	
Symbol:	destroy(T *p)	
Syntax:	`template <class T>` `void destroy (T *p);`	
DIRECTION NOT DEFINED	p	–
Return value:	None	
Exception Safety:	not exception safe	
Description:	This function behaves the same as the corresponding std::pmr::polymorphic_allocator function.	

⌋*(RS_AP_00130)*

[SWS_CORE_06553]{DRAFT} **Definition of API function ara::core::PolymorphicAllocator::new_object** ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	class ara::core::PolymorphicAllocator	
Symbol:	new_object(CtorArgs &&... ctor_args)	
Syntax:	`template <class T, class... CtorArgs>` `ARA_COMPILER_DEFINED_NODISCARD T * new_object (CtorArgs &&... ctor_args) noexcept;`	
DIRECTION NOT DEFINED	ctor_args	–
Return value:	ARA_COMPILER_DEFINED_NODISCARD T *	–
Exception Safety:	noexcept	
Description:	This function behaves the same as the corresponding std::pmr::polymorphic_allocator function.	

⌋*(RS_AP_00130)*

[SWS_CORE_06545]{DRAFT} **Definition of API function ara::core::PolymorphicAllocator::operator=** ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	class ara::core::PolymorphicAllocator
Symbol:	operator=(const PolymorphicAllocator &)
Syntax:	`PolymorphicAllocator & operator= (const PolymorphicAllocator &)=delete;`
Description:	This function behaves the same as the corresponding std::pmr::polymorphic_allocator function.

⌋*(RS_AP_00130)*

[SWS_CORE_06557]{DRAFT} **Definition of API function ara::core::PolymorphicAllocator::resource** ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	class ara::core::PolymorphicAllocator	
Symbol:	resource()	
Syntax:	`MemoryResource * resource () const noexcept;`	
Return value:	MemoryResource *	–
Exception Safety:	noexcept	
Description:	This function behaves the same as the corresponding std::pmr::polymorphic_allocator function.	

⌋*(RS_AP_00130)*

8.1.18 Generic helpers

8.1.18.1 ara::core::Byte

The exact setup of this type is implementation-defined; the specifications in section 7.2.4.3.5 ("Byte") define the expected behavior.

[SWS_CORE_04200] Definition of API type ara::core::Byte ⌈

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.

⌋*(RS_AP_00130)*

8.1.18.2 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 [9]. 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 [9, the C++17 standard].

8.1.18.2.1 `in_place_t` tag

[SWS_CORE_04011] Definition of API class ara::core::in_place_t ⌈

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.

⌋*(RS_AP_00130)*

[SWS_CORE_04012] Definition of API function ara::core::in_place_t::in_place_t ⌈

Kind:	function
Header file:	#include "ara/core/utility.h"
Scope:	struct ara::core::in_place_t
Symbol:	in_place_t()
Syntax:	`explicit in_place_t ()=default;`
Description:	Default constructor.

⌋*(RS_AP_00130)*

[SWS_CORE_04013] Definition of API variable ara::core::in_place ⌈

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.

⌋*(RS_AP_00130)*

8.1.18.2.2 `in_place_type_t` tag

[SWS_CORE_04021] Definition of API class ara::core::in_place_type_t ⌈

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.	

⌋*(RS_AP_00130)*

[SWS_CORE_04022] Definition of API function ara::core::in_place_type_t::in_place_type_t ⌈

Kind:	function
Header file:	#include "ara/core/utility.h"
Scope:	struct ara::core::in_place_type_t
Symbol:	in_place_type_t()
Syntax:	`explicit in_place_type_t ()=default;`
Description:	Default constructor.

⌋*(RS_AP_00130)*

[SWS_CORE_04023] Definition of API variable ara::core::in_place_type ⌈

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.	

⌋*(RS_AP_00130)*

8.1.18.2.3 `in_place_index_t` tag

[SWS_CORE_04031] Definition of API class ara::core::in_place_index_t ⌈

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.	

⌋*(RS_AP_00130)*

[SWS_CORE_04032] Definition of API function ara::core::in_place_index_t::in_place_index_t ⌈

Kind:	function
Header file:	#include "ara/core/utility.h"
Scope:	struct ara::core::in_place_index_t
Symbol:	in_place_index_t()
Syntax:	`explicit in_place_index_t ()=default;`
Description:	Default constructor.

⌋*(RS_AP_00130)*

[SWS_CORE_04033] Definition of API variable ara::core::in_place_index ⌈

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.	

⌋*(RS_AP_00130)*

8.1.18.3 Non-member container access

These 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 [9].

[SWS_CORE_04110] Definition of API function ara::core::data ⌈

Kind:	function	
Header file:	#include "ara/core/utility.h"	
Scope:	namespace ara::core	
Symbol:	data(Container &c)	
Syntax:	`template <typename Container>` `constexpr auto data (Container &c) -> 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:	not exception safe	
Description:	Return a pointer to the block of memory that contains the elements of a container.	

⌋*(RS_AP_00130)*

[SWS_CORE_04111] Definition of API function ara::core::data ⌈

Kind:	function	
Header file:	#include "ara/core/utility.h"	
Scope:	namespace ara::core	
Symbol:	data(const Container &c)	
Syntax:	`template <typename Container>` `constexpr auto data (const Container &c) -> 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:	not exception safe	
Description:	Return a const_pointer to the block of memory that contains the elements of a container.	

⌋*(RS_AP_00130)*

[SWS_CORE_04112] Definition of API function ara::core::data ⌈

Kind:	function	
Header file:	#include "ara/core/utility.h"	
Scope:	namespace ara::core	
Symbol:	data(T(&array)[N])	
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:	noexcept	
Description:	Return a pointer to the block of memory that contains the elements of a raw array.	

⌋*(RS_AP_00130)*

[SWS_CORE_04113] Definition of API function ara::core::data ⌈

Kind:	function	
Header file:	#include "ara/core/utility.h"	
Scope:	namespace ara::core	
Symbol:	data(std::initializer_list< E > il)	
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:	noexcept	
Description:	Return a pointer to the block of memory that contains the elements of a std::initializer_list.	

⌋*(RS_AP_00130)*

[SWS_CORE_04120] Definition of API function ara::core::size ⌈

Kind:	function	
Header file:	#include "ara/core/utility.h"	
Scope:	namespace ara::core	
Symbol:	size(const Container &c)	
Syntax:	`template <typename Container>` `constexpr auto size (const Container &c) -> decltype(c.size());`	
Template param:	Container	a type with a data() method
Parameters (in):	c	an instance of Container
Return value:	decltype(c.size())	the size of the container
Exception Safety:	not exception safe	
Description:	Return the size of a container.	

⌋*(RS_AP_00130)*

[SWS_CORE_04121] Definition of API function ara::core::size ⌈

Kind:	function	
Header file:	#include "ara/core/utility.h"	
Scope:	namespace ara::core	
Symbol:	size(const T(&array)[N])	
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:	noexcept	
Description:	Return the size of a raw array.	

⌋*(RS_AP_00130)*

[SWS_CORE_04130] Definition of API function ara::core::empty ⌈

Kind:	function	
Header file:	#include "ara/core/utility.h"	
Scope:	namespace ara::core	
Symbol:	empty(const Container &c)	
Syntax:	`template <typename Container>` `constexpr auto empty (const Container &c) -> 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:	not exception safe	
Description:	Return whether the given container is empty.	

⌋*(RS_AP_00130)*

[SWS_CORE_04131] Definition of API function ara::core::empty ⌈

Kind:	function	
Header file:	#include "ara/core/utility.h"	
Scope:	namespace ara::core	
Symbol:	empty(const T(&array)[N])	
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:	noexcept	
Description:	Return whether the given raw array is empty. As raw arrays cannot have zero elements in C++, this function always returns false.	

⌋*(RS_AP_00130)*

[SWS_CORE_04132] Definition of API function ara::core::empty ⌈

Kind:	function	
Header file:	#include "ara/core/utility.h"	
Scope:	namespace ara::core	
Symbol:	empty(std::initializer_list< E > il)	
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:	noexcept	
Description:	Return whether the given std::initializer_list is empty.	

⌋*(RS_AP_00130)*

8.1.19 Initialization and Shutdown

This section describes the non-member initialization and shutdown functions that initialize resp. deinitialize data structures and threads of the AUTOSAR Runtime for Adaptive Applications.

[SWS_CORE_10001]{DRAFT} **Definition of API function ara::core::Initialize** ⌈

Kind:	function	
Header file:	#include "ara/core/initialization.h"	
Scope:	namespace ara::core	
Symbol:	Initialize()	
Syntax:	`Result< void > Initialize () noexcept;`	
Return value:	Result< void >	a Result with an error code, in case an error occurred
Exception Safety:	noexcept	
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.	

⌋*(RS_Main_00011)*

[SWS_CORE_10002]{DRAFT} **Definition of API function ara::core::Deinitialize** ⌈

Kind:	function	
Header file:	#include "ara/core/initialization.h"	
Scope:	namespace ara::core	
Symbol:	Deinitialize()	
Syntax:	`Result< void > Deinitialize () noexcept;`	
Return value:	Result< void >	a Result with an error code, in case an error occurred
Exception Safety:	noexcept	
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.	

⌋*(RS_Main_00011)*

8.1.20 Abnormal process termination

This section describes the APIs that constitute the explicit abnormal termination facility.

[SWS_CORE_00053]{DRAFT} **Definition of API function ara::core::AbortHandler Prototype** ⌈

Kind:	function	
Header file:	#include "ara/core/abort.h"	
Scope:	namespace ara::core	
Symbol:	AbortHandlerPrototype()	
Syntax:	`void AbortHandlerPrototype () noexcept;`	
Return value:	None	
Exception Safety:	noexcept	
Description:	A function declaration with the correct prototype for 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.	

⌋*(RS_AP_00132)*

[SWS_CORE_00050] Definition of API type ara::core::AbortHandler ⌈

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 SetAbortHandler().

⌋*(RS_AP_00132)*

[SWS_CORE_00051] Definition of API function ara::core::SetAbortHandler ⌈

Kind:	function	
Header file:	#include "ara/core/abort.h"	
Scope:	namespace ara::core	
Symbol:	SetAbortHandler(AbortHandler handler)	
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:	noexcept	
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.	

⌋*(RS_AP_00132)*

[SWS_CORE_00054]{DRAFT} **Definition of API function ara::core::AddAbortHandler** ⌈

Kind:	function	
Header file:	#include "ara/core/abort.h"	
Scope:	namespace ara::core	
Symbol:	AddAbortHandler(AbortHandler handler)	
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:	noexcept	
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.	

⌋*(RS_AP_00132)*

[SWS_CORE_00052] Definition of API function ara::core::Abort ⌈

Kind:	function	
Header file:	#include "ara/core/abort.h"	
Scope:	namespace ara::core	
Symbol:	Abort(const Args &... args)	

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:	noexcept	
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::String View.	

⌋*(RS_AP_00127, RS_AP_00132, RS_AP_00136)*

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*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
namespace (ordered)	SymbolProps	*	aggr	This aggregation defines the namespace of the Ap ApplicationErrorDomain
value	PositiveUnlimitedInteger	0..1	attr	This attribute identifies the error category.

Table A.1: ApApplicationErrorDomain

Class	**ImplementationDataType**			
Package	M2::AUTOSARTemplates::CommonStructure::ImplementationDataTypes			
Note	Describes a reusable data type on the implementation level. This will typically correspond to a typedef in C-code. **Tags:** atp.recommendedPackage=ImplementationDataTypes			
Base	*ARElement*, *ARObject*, *AbstractImplementationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
dynamicArray SizeProfile	String	0..1	attr	Specifies the profile which the array will follow in case this data type is a variable size array.
isStructWith Optional Element	Boolean	0..1	attr	This attribute is only valid if the attribute category is set to STRUCTURE. If set to true, this attribute indicates that the ImplementationDataType has been created with the intention to define at least one element of the structure as optional.
subElement (ordered)	ImplementationData TypeElement	*	aggr	Specifies an element of an array, struct, or union data type. The aggregation of ImplementionDataTypeElement is subject to variability with the purpose to support the conditional existence of elements inside a Implementation DataType representing a structure. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=subElement.shortName, sub Element.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**ImplementationDataType**			
symbolProps	SymbolProps	0..1	aggr	This represents the SymbolProps for the Implementation DataType. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=symbolProps.shortName
typeEmitter	NameToken	0..1	attr	This attribute is used to control which part of the AUTOSAR toolchain is supposed to trigger data type definitions.

Table A.2: ImplementationDataType

B Interfaces to other Functional Clusters (informative)

B.1 Overview

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 informative guidelines how the interaction between Functional Clusters looks like, by clustering the relevant requirements of this document to describe Inter-Functional Cluster (IFC) interfaces. In addition, the standardized public interfaces which are accessible by user space applications (see chapter 8 ("API specification")) can also be used for interaction between Functional Clusters.

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 the interfaces are up to the platform provider. Additional interfaces, parameters and return values can be added.

B.2 Interface Tables

B.2.1 Functional Cluster initialization

`ara::core::Initialize` and `ara::core::Deinitialize` initialize and deinitialize 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 deinitialization.

C History of Specification Items

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.

C.1 Specification Item History of this document compared to AUTOSAR R22-11.

C.1.1 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
[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::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_06527]	Definition of API function ara::core::MonotonicBufferResource::Monotonic BufferResource

▽

△

Number	Heading
[SWS_CORE_06528]	Definition of API function ara::core::MonotonicBufferResource::~Monotonic BufferResource
[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::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_06545]	Definition of API function ara::core::PolymorphicAllocator::operator=
[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
[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_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 C.1: Added Specification Items in R23-11

C.1.2 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
[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

▽

△

Number	Heading
[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*
[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[]

▽

△

Number	Heading
[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
[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 C.2: Changed Specification Items in R23-11

C.1.3 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]	

▽

△

Number	Heading
[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]	
[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 C.3: Deleted Specification Items in R23-11

C.2 Specification Item History of this document compared to AUTOSAR R21-11.

C.2.1 Added Specification Items in R22-11

Number	Heading
[SWS_CORE_00054]	
[SWS_CORE_00615]	
[SWS_CORE_00616]	
[SWS_CORE_00617]	

▽

△

Number	Heading
[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 C.4: Added Specification Items in R22-11

C.2.2 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]	

▽

△

Number	Heading
[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]	
[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]	

▽

△

Number	Heading
[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]	
[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 C.5: Changed Specification Items in R22-11

C.2.3 Deleted Specification Items in R22-11

none

C.3 Specification Item History of this document compared to AUTOSAR R20-11.

C.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 C.6: Added Specification Items in R21-11

C.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 C.7: Changed Specification Items in R21-11

C.3.3 Deleted Specification Items in R21-11

Number	Heading
[SWS_CORE_01913]	

Table C.8: Deleted Specification Items in R21-11

C.4 Specification Item History of this document compared to AUTOSAR R19-11.

C.4.1 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]	

▽

△

Number	Heading
[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_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]	

▽

△

Number	Heading
[SWS_CORE_01981]	
[SWS_CORE_04023]	
[SWS_CORE_04033]	
[SWS_CORE_06237]	
[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_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

▽

△

Number	Heading
[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
[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 C.9: Added Specification Items in R20-11

C.4.2 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]	

▽

△

Number	Heading
[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_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]	

▽

△

Number	Heading
[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_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]	

▽

△

Number	Heading
[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]	
[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]	

▽

△

Number	Heading
[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_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]	

▽

△

Number	Heading
[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]	
[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]	

▽

△

Number	Heading
[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`
[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]	

▽

△

Number	Heading
[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]	
[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 C.10: Changed Specification Items in R20-11

C.4.3 Deleted Specification Items in R20-11

none

C.5 Specification Item History of this document compared to AUTOSAR R19-03.

C.5.1 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]	
[SWS_CORE_00331]	
[SWS_CORE_00332]	

▽

△

Number	Heading
[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]	
[SWS_CORE_00723]	
[SWS_CORE_00724]	

▽

△

Number	Heading
[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]	
[SWS_CORE_00786]	
[SWS_CORE_00787]	

▽

△

Number	Heading
[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]	
[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_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]	
[SWS_CORE_04131]	
[SWS_CORE_04132]	

▽

△

Number	Heading
[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]	
[SWS_CORE_06352]	
[SWS_CORE_06353]	

▽

△

Number	Heading
[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 C.11: Added Specification Items in R19-11

C.5.2 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
[SWS_CORE_03296]	`swap` overload for `BasicString`
[SWS_CORE_03301]	Implicit conversion to `StringView`

▽

△

Number	Heading
[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 C.12: Changed Specification Items in R19-11

C.5.3 Deleted Specification Items in R19-11

Number	Heading
[SWS_CORE_00001]	Handling of Fatal Errors
[SWS_CORE_00012]	The POSIX error domain

Table C.13: Deleted Specification Items in R19-11