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	R25-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2025-11-27	R25-11	AUTOSAR Release Management	• Add full specification of ara::core::String, ara::core::Map, ara::core::Vector, and ara::core::Bitset • Document undefined and implementation-defined behavior • Harmonize Violations across data types • Adapt some APIs to C++17's enhanced language and library capabilities • Standardize Functional Clusters' Error Domain identifiers along with their Log and Trace settings • Remove standardized stderr output for Violations • Introduce various minor extensions, corrections, and clarifications to the C++ data-type specifications

△

2024-11-27	R24-11	AUTOSAR Release Management	• Extend specification of ara::core::MemoryResource and derived classes • Add full specification of ara::core::Optional, ara::core::Variant, and ara::core::StringView • Add T& specializations of ara::core::Optional and ara::core::Result • Specify exception safety and thread safety of ARA APIs • Mandate ara::core::ErrorCode as ErrorType of Result and Future/Promise • Various extensions and fixes to the C++ data types • Adapt the document to the new template with a generated Chapter 8 • Refine specification about platform initialization • Enable ara::core::Initialize() take command line arguments • Refine Violation specification of ARA APIs with standardized Violation messages
2023-11-23	R23-11	AUTOSAR Release Management	• Add specification of ara::core::MemoryResource • Remove specification of ara::core::ScaleLinearAndTexttable • Refine specification about platform initialization • Refine specification of Future, and Promise with regards to error handling • Extend Array specification with accessor functions performing checked access

▽

▽

△

			△ • Make undefined behavior explicit by mandating Violations across various C++ data types • Rework of chapter 5 with dependencies to other modules
2022-11-24	R22-11	AUTOSAR Release Management	• Extend ara::core::Abort to allow multiple arguments • Add support for registering multiple AbortHandlers • Merge header files of ara::core::Future and ara::core:Promise into a single one • Add full specification of ara::core::String and ara::core::BasicString • Forbid user extensions of standardized AUTOSAR namespaces
2021-11-25	R21-11	AUTOSAR Release Management	• Add spec items for error handling definitions • Add specifications for ScaleLinearAndTexttable, taken over from SWS_CommunicationManagement • Refine scope of ara::core::Initialize • Adapt some APIs to C++14's enhanced capabilities • Align Span with std::span from the C++20 standard • Reduce requirements imposed on handling Violations • Rename document into "Adaptive Platform Core"

▽

△

2020-11-30	R20-11	AUTOSAR Release Management	• Add specifications about “Explicit Operation Abortion” • Add specification about reserved symbol prefixes • Add specification of class SteadyClock • Add section about async signal safety of ARA APIs • Extend error domain scope with vendor-defined error domains • Add specifications about defining own error domains • Various extensions and fixes to the C++ data types • Incorporate contents of SWS_General • Rename document into “Adaptive Core”
2019-11-28	R19-11	AUTOSAR Release Management	• Rework error handling definitions • Add specifications of BasicString and Byte, and add overloads and template specializations for ErrorCode, Result, Future, and Promise • Add bits about validity of InstanceSpecifier arguments, and rework the specification of its construction mechanism • Rework ErrorCode to get rid of “User Message” and make “SupportDataType” implementation-defined • Replace PosixErrorDomain with CoreErrorDomain • Rename FutureErrorDomain accessor function • Changed Document Status from Final to published
2019-03-29	19-03	AUTOSAR Release Management	• Add specification of the template specialization Result<void, E>

▽

△

2018-10-31	18-10	AUTOSAR Release Management	• Add chapter 2 with acronyms • Add chapter 4 with limitations of the current specifications • Add chapter 5 with dependencies to other modules • Add chapter 7 • Add classes representing the approach to error handling to chapter 8 • Adapt classes Future and Promise to the error handling approach • Add global functions for initialization and shutdown of the framework • Add class InstanceSpecifier to chapter 8 • Add more types and functions from the C++ standard
2018-03-29	18-03	AUTOSAR Release Management	• Initial Release

Disclaimer

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

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

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

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

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

1 Introduction and functional overview 48
2 Acronyms and Abbreviations 49
3 Related documentation 50
3.1 Input documents & related standards and norms . . . 50
4 Constraints and assumptions 52
4.1 Known limitations . . . 52
5 Dependencies to other Functional Clusters 53
5.1 Provided Interfaces . . . 53
5.2 Required Interfaces . . . 53
5.3 Functional Cluster initialization . . . 53
6 Requirements Tracing 55
7 Functional specification 60
7.1 General requirements for all Functional Clusters . . . 60
7.1.1 Functional Cluster names . . . 60
7.1.2 Functional Cluster header files . . . 61
7.1.3 ARA coding guidelines . . . 62
7.1.4 `Undefined behavior` . . . 63
7.1.5 `Implementation-defined behavior` . . . 63
7.1.6 `Vendor-specific behavior` . . . 63
7.1.7 Initialization/De-initialization . . . 63
7.1.7.1 Common aspects . . . 63
7.1.7.2 Initialization . . . 64
7.1.7.3 De-initialization . . . 65
7.1.7.4 Sequence of Initialization and De-initialization . . . 66
7.1.8 Logging and Tracing . . . 67
7.1.9 Thread safety . . . 68
7.1.10 Async signal safety . . . 71
7.2 Error handling . . . 71
7.2.1 Traditional error handling in C and C++ . . . 71
7.2.2 Violation . . . 72
7.2.3 Corruption . . . 74
7.2.4 Failed default allocation . . . 74
7.2.5 Error . . . 75
7.2.5.1 ErrorCode . . . 75
7.2.5.2 Error Domain . . . 77
7.2.5.3 Usages of ErrorDomain . . . 77
7.2.5.3.1 Error domain Identifiers . . . 79
7.2.5.3.2 Vendor error domains . . . 81

7.2.5.3.3 Creating new error domains . . . 81
7.2.5.3.3.1 ErrorDomain sub-class . . . 82
7.2.5.3.3.2 Non-member ErrorDomain sub-class accessor function 84
7.2.5.3.3.3 Non-member MakeErrorCode overload . . . 84
7.2.5.3.3.4 Error condition enumeration . . . 85
7.2.5.3.3.5 Exception base class . . . 86
7.2.5.3.3.6 Error Domain C++ pseudo code example . . . 86
7.2.5.4 Result . . . 87
7.2.5.5 Future and Promise . . . 88
7.2.5.6 Exceptions . . . 89
7.2.5.7 Duality of ErrorCode and Exception . . . 90
7.3 Explicitly aborting an Operation . . . 90
7.3.1 AbortHandler . . . 92
7.3.2 SIGABRT handler . . . 93
7.4 Core data types . . . 93
7.4.1 Data types devised by AUTOSAR . . . 93
7.4.1.1 ara::core::InstanceSpecifier . . . 93
7.4.1.2 ara::core::Executor . . . 95
7.4.2 Data types derived from the C++ standard . . . 95
7.4.2.1 Containers library . . . 96
7.4.2.1.1 ara::core::Array . . . 96
7.4.2.1.2 ara::core::Vector . . . 96
7.4.2.1.3 ara::core::Map . . . 96
7.4.2.1.4 ara::core::Span . . . 96
7.4.2.2 General utilities library . . . 97
7.4.2.2.1 ara::core::Bitset . . . 97
7.4.2.2.2 ara::core::Optional . . . 97
7.4.2.2.2.1 ara::core::Optional<T&> . . . 97
7.4.2.2.3 ara::core::Variant . . . 98
7.4.2.2.4 ara::core::MemoryResource . . . 98
7.4.2.2.5 In-place disambiguation tags . . . 98
7.4.2.2.6 Non-member container access . . . 98
7.4.2.3 Strings library . . . 98
7.4.2.3.1 ara::core::String . . . 98
7.4.2.3.2 ara::core::StringView . . . 98
7.4.2.4 Language support library . . . 99
7.4.2.4.1 ara::core::Byte . . . 99
7.4.2.5 Time library . . . 101
7.4.2.5.1 ara::core::SteadyClock . . . 101
7.4.2.5.1.1 Definitions of terms . . . 101
7.4.2.5.2 Clocks in the Adaptive Platform . . . 101
7.5 Functional cluster life-cycle . . . 102
7.5.1 Startup . . . 102

7.5.2 Shutdown . . . 102
7.6 Reporting . . . 103
7.6.1 Security Events . . . 103
7.6.2 Log Messages . . . 103
7.6.2.1 AbortMessage . . . 103
7.6.2.2 FailedDefaultAllocation . . . 103
7.6.3 Violation Messages . . . 104
7.6.3.1 AraNotInitializedViolation . . . 104
7.6.3.2 FutureContinuationHasThrownViolation . . . 105
7.6.3.3 FutureInvalidViolation . . . 105
7.6.3.4 InstanceSpecifierAlreadyInUseViolation . . . 106
7.6.3.5 InstanceSpecifierMappingIntegrityViolation . . . 106
7.6.3.6 InsufficientPermissionsViolation . . . 107
7.6.3.7 IsoStdBadAllocViolation . . . 107
7.6.3.8 IsoStdBadOptionalAccessViolation . . . 108
7.6.3.9 IsoStdBadVariantAccessViolation . . . 108
7.6.3.10 IsoStdFutureErrcFutureAlreadyRetrievedViolation 109
7.6.3.11 IsoStdFutureErrcNoStateViolation . . . 109
7.6.3.12 IsoStdFutureErrcPromiseAlreadySatisfiedViolation . . . 110
7.6.3.13 IsoStdInvalidArgumentViolation . . . 110
7.6.3.14 IsoStdLengthErrorViolation . . . 111
7.6.3.15 IsoStdOutOfRangeViolation . . . 111
7.6.3.16 IsoStdOverflowErrorViolation . . . 112
7.6.3.17 PortInterfaceMappingViolation . . . 112
7.6.3.18 ProcessMappingViolation . . . 113
7.6.3.19 PromiseNoSharedStateToSetViolation . . . 113
7.6.3.20 ResultWithNoErrorViolation . . . 114
7.6.3.21 ResultWithNoValueViolation . . . 114
7.6.3.22 WrongDomainOfAnErrorCodeViolation . . . 115
7.6.4 Production Errors . . . 115
8 API specification 116
8.1 Header: ara/core/array.h . . . 117
8.1.1 Non-Member Functions . . . 117
8.1.1.1 Other . . . 117
8.1.1.1.1 get(Array<T, N>&) . . . 117
8.1.1.1.2 get(const Array<T, N>&) . . . 117
8.1.1.1.3 get(const Array<T, N>&&) . . . 118
8.1.1.1.4 get(Array<T, N>&&) . . . 118
8.1.1.1.5 operator!= . . . 119
8.1.1.1.6 operator< . . . 119
8.1.1.1.7 operator<= . . . 119
8.1.1.1.8 operator== . . . 120

8.1.1.1.9 operator> . . . 120
8.1.1.1.10 operator>= . . . 120
8.1.1.1.11 swap . . . 121
8.1.2 Class: Array . . . 121
8.1.2.1 Public Member Types . . . 121
8.1.2.1.1 Type Alias: const_iterator . . . 121
8.1.2.1.2 Type Alias: const_pointer . . . 122
8.1.2.1.3 Type Alias: const_reference . . . 122
8.1.2.1.4 Type Alias: const_reverse_iterator . . . 122
8.1.2.1.5 Type Alias: difference_type . . . 123
8.1.2.1.6 Type Alias: iterator . . . 123
8.1.2.1.7 Type Alias: pointer . . . 123
8.1.2.1.8 Type Alias: reference . . . 124
8.1.2.1.9 Type Alias: reverse_iterator . . . 124
8.1.2.1.10 Type Alias: size_type . . . 124
8.1.2.1.11 Type Alias: value_type . . . 125
8.1.2.2 Public Member Functions . . . 125
8.1.2.2.1 Member Functions . . . 125
8.1.2.2.1.1 fill . . . 125
8.1.2.2.1.2 swap . . . 125
8.1.2.2.2 Capacity . . . 126
8.1.2.2.2.1 empty . . . 126
8.1.2.2.2.2 max_size . . . 126
8.1.2.2.2.3 size . . . 126
8.1.2.2.3 Element access . . . 127
8.1.2.2.3.1 at(size_type) . . . 127
8.1.2.2.3.2 at(size_type) const . . . 127
8.1.2.2.3.3 back() . . . 127
8.1.2.2.3.4 back() const . . . 128
8.1.2.2.3.5 data() const . . . 128
8.1.2.2.3.6 data() . . . 128
8.1.2.2.3.7 front() const . . . 129
8.1.2.2.3.8 front() . . . 129
8.1.2.2.3.9 operator[](size_type) const . . . 129
8.1.2.2.3.10 operator[](size_type) . . . 130
8.1.2.2.4 Iterators . . . 130
8.1.2.2.4.1 begin() const . . . 130
8.1.2.2.4.2 begin() . . . 130
8.1.2.2.4.3 cbegin() const . . . 131
8.1.2.2.4.4 cend() const . . . 131
8.1.2.2.4.5 crbegin() const . . . 131
8.1.2.2.4.6 crend() const . . . 132
8.1.2.2.4.7 end() . . . 132

8.1.2.2.4.8 end() const . . . 132
8.1.2.2.4.9 rbegin() const . . . 133
8.1.2.2.4.10 rbegin() . . . 133
8.1.2.2.4.11 rend() . . . 133
8.1.2.2.4.12 rend() const . . . 134
8.1.3 Struct: tuple_element . . . 134
8.1.3.1 Public Member Types . . . 134
8.1.3.1.1 Type Alias: type . . . 134
8.1.4 Struct: tuple_size . . . 135
8.2 Header: ara/core/bitset.h . . . 135
8.2.1 Non-Member Functions . . . 136
8.2.1.1 Other . . . 136
8.2.1.1.1 operator& . . . 136
8.2.1.1.2 operator<< . . . 136
8.2.1.1.3 operator>> . . . 136
8.2.1.1.4 operator^ . . . 137
8.2.1.1.5 operator| . . . 137
8.2.2 Class: Bitset . . . 137
8.2.2.1 Public Member Functions . . . 138
8.2.2.1.1 Special Member Functions . . . 138
8.2.2.1.1.1 Move Constructor . . . 138
8.2.2.1.1.2 Default Constructor . . . 138
8.2.2.1.1.3 Copy Constructor . . . 138
8.2.2.1.1.4 Move Assignment Operator . . . 139
8.2.2.1.1.5 Copy Assignment Operator . . . 139
8.2.2.1.1.6 Destructor . . . 139
8.2.2.1.2 Constructors . . . 140
8.2.2.1.2.1 Bitset(const BasicString<Allocator>&, typename BasicString<Allocator>::size_type, typename BasicString<Allocator>::size_type, CharT, CharT) . . . 140
8.2.2.1.2.2 Bitset(unsigned long long) . . . 140
8.2.2.1.2.3 Bitset(const CharT *, typename BasicString<CharT>::size_type, CharT, CharT) . . . 141
8.2.2.1.3 Capacity . . . 141
8.2.2.1.3.1 size . . . 141
8.2.2.1.4 Conversions . . . 141
8.2.2.1.4.1 to_string . . . 141
8.2.2.1.4.2 to_uint16t . . . 142
8.2.2.1.4.3 to_uint32t . . . 142
8.2.2.1.4.4 to_uint64t . . . 143
8.2.2.1.4.5 to_uint8t . . . 143
8.2.2.1.4.6 to_ullong . . . 144
8.2.2.1.4.7 to_ulong . . . 144
8.2.2.1.5 Element access . . . 145

8.2.2.1.5.1 all . . . 145
8.2.2.1.5.2 any . . . 145
8.2.2.1.5.3 count . . . 145
8.2.2.1.5.4 none . . . 146
8.2.2.1.5.5 operator[](std::size_t) . . . 146
8.2.2.1.5.6 operator[](std::size_t) const . . . 146
8.2.2.1.5.7 test . . . 147
8.2.2.1.6 Modifiers . . . 147
8.2.2.1.6.1 flip(std::size_t) . . . 147
8.2.2.1.6.2 flip() . . . 147
8.2.2.1.6.3 operator&= . . . 148
8.2.2.1.6.4 operator<< . . . 148
8.2.2.1.6.5 operator<<= . . . 148
8.2.2.1.6.6 operator>> . . . 149
8.2.2.1.6.7 operator>>= . . . 149
8.2.2.1.6.8 operator^= . . . 149
8.2.2.1.6.9 operator|= . . . 150
8.2.2.1.6.10 operator~ . . . 150
8.2.2.1.6.11 reset(std::size_t) . . . 150
8.2.2.1.6.12 reset() . . . 151
8.2.2.1.6.13 set(std::size_t, bool) . . . 151
8.2.2.1.6.14 set() . . . 151
8.2.2.1.7 Other . . . 152
8.2.2.1.7.1 operator== . . . 152
8.2.3 Class: reference . . . 152
8.2.3.1 Public Member Functions . . . 152
8.2.3.1.1 Special Member Functions . . . 152
8.2.3.1.1.1 Copy Assignment Operator . . . 152
8.2.3.1.1.2 Move Assignment Operator . . . 153
8.2.3.1.1.3 Move Constructor . . . 153
8.2.3.1.1.4 Copy Constructor . . . 153
8.2.3.1.1.5 Destructor . . . 154
8.2.3.1.2 Member Functions . . . 154
8.2.3.1.2.1 flip . . . 154
8.2.3.1.2.2 operator bool . . . 154
8.2.3.1.2.3 operator= . . . 155
8.2.3.1.2.4 operator~ . . . 155
8.2.3.2 Private Member Functions . . . 155
8.2.3.2.1 Special Member Functions . . . 155
8.2.3.2.1.1 Default Constructor . . . 155
8.2.4 Struct: hash . . . 156
8.3 Header: ara/core/map.h . . . 156
8.3.1 Non-Member Functions . . . 157

8.3.1.1 Other . . . 157
8.3.1.1.1 swap . . . 157
8.3.2 Class: Map . . . 157
8.3.2.1 Public Member Types . . . 158
8.3.2.1.1 Type Alias: allocator_type . . . 158
8.3.2.1.2 Type Alias: const_iterator . . . 158
8.3.2.1.3 Type Alias: const_pointer . . . 158
8.3.2.1.4 Type Alias: const_reference . . . 159
8.3.2.1.5 Type Alias: const_reverse_iterator . . . 159
8.3.2.1.6 Type Alias: difference_type . . . 159
8.3.2.1.7 Type Alias: insert_return_type . . . 160
8.3.2.1.8 Type Alias: iterator . . . 160
8.3.2.1.9 Type Alias: key_compare . . . 160
8.3.2.1.10 Type Alias: key_type . . . 161
8.3.2.1.11 Type Alias: mapped_type . . . 161
8.3.2.1.12 Type Alias: node_type . . . 161
8.3.2.1.13 Type Alias: pointer . . . 162
8.3.2.1.14 Type Alias: reference . . . 162
8.3.2.1.15 Type Alias: reverse_iterator . . . 162
8.3.2.1.16 Type Alias: size_type . . . 163
8.3.2.1.17 Type Alias: value_type . . . 163
8.3.2.2 Public Member Functions . . . 163
8.3.2.2.1 Special Member Functions . . . 163
8.3.2.2.1.1 Map(Map&&) . . . 163
8.3.2.2.1.2 Map() . . . 164
8.3.2.2.1.3 Map(const Map&) . . . 164
8.3.2.2.1.4 operator=(const Map&) . . . 164
8.3.2.2.1.5 operator=(Map&&) . . . 165
8.3.2.2.1.6 Destructor . . . 165
8.3.2.2.2 Constructors . . . 165
8.3.2.2.2.1 Map(std::initializer_list<value_type>, const Compare&, const Allocator&) . . . 165
8.3.2.2.2.2 Map(const Compare&, const Allocator&) . . . 166
8.3.2.2.2.3 Map(InputIterator, InputIterator, const Compare&, const Allocator&) . . . 166
8.3.2.2.2.4 Map(const Allocator&) . . . 166
8.3.2.2.2.5 Map(const Map&, const Allocator&) . . . 167
8.3.2.2.2.6 Map(Map&&, const Allocator&) . . . 167
8.3.2.2.2.7 Map(std::initializer_list<value_type>, const Allocator&) 167
8.3.2.2.2.8 Map(InputIterator, InputIterator, const Allocator&) . . . 168
8.3.2.2.3 Capacity . . . 168
8.3.2.2.3.1 empty . . . 168
8.3.2.2.3.2 max_size . . . 168
8.3.2.2.3.3 size . . . 169

8.3.2.2.4 Element access . . . 169
8.3.2.2.4.1 at(const key_type&) . . . 169
8.3.2.2.4.2 at(const key_type&) const . . . 169
8.3.2.2.4.3 operator[](const key_type&) . . . 170
8.3.2.2.4.4 operator[](key_type&&) . . . 170
8.3.2.2.5 Iterators . . . 170
8.3.2.2.5.1 begin() . . . 170
8.3.2.2.5.2 begin() const . . . 171
8.3.2.2.5.3 cbegin . . . 171
8.3.2.2.5.4 cend . . . 171
8.3.2.2.5.5 crbegin . . . 172
8.3.2.2.5.6 crend . . . 172
8.3.2.2.5.7 end() const . . . 172
8.3.2.2.5.8 end() . . . 173
8.3.2.2.5.9 rbegin() const . . . 173
8.3.2.2.5.10 rbegin() . . . 173
8.3.2.2.5.11 rend() . . . 174
8.3.2.2.5.12 rend() const . . . 174
8.3.2.2.6 Lookup . . . 174
8.3.2.2.6.1 contains(const key_type&) const . . . 174
8.3.2.2.6.2 contains(const K&) const . . . 175
8.3.2.2.6.3 count(const K&) const . . . 175
8.3.2.2.6.4 count(const key_type&) const . . . 175
8.3.2.2.6.5 equal_range(const K&) const . . . 176
8.3.2.2.6.6 equal_range(const K&) . . . 176
8.3.2.2.6.7 equal_range(const key_type&) . . . 176
8.3.2.2.6.8 equal_range(const key_type&) const . . . 177
8.3.2.2.6.9 find(const key_type&) . . . 177
8.3.2.2.6.10 find(const K&) const . . . 177
8.3.2.2.6.11 find(const K&) . . . 178
8.3.2.2.6.12 find(const key_type&) const . . . 178
8.3.2.2.6.13 lower_bound(const key_type&) . . . 178
8.3.2.2.6.14 lower_bound(const K&) . . . 179
8.3.2.2.6.15 lower_bound(const K&) const . . . 179
8.3.2.2.6.16 lower_bound(const key_type&) const . . . 179
8.3.2.2.6.17 upper_bound(const K&) . . . 180
8.3.2.2.6.18 upper_bound(const key_type&) const . . . 180
8.3.2.2.6.19 upper_bound(const key_type&) . . . 180
8.3.2.2.6.20 upper_bound(const K&) const . . . 181
8.3.2.2.7 Modifiers . . . 181
8.3.2.2.7.1 clear . . . 181
8.3.2.2.7.2 emplace(Args ...&&) . . . 181
8.3.2.2.7.3 emplace_hint(const_iterator, Args ...&&) . . . 182

8.3.2.2.7.4 erase(const_iterator, const_iterator) . . . 182
8.3.2.2.7.5 erase(const_iterator) . . . 182
8.3.2.2.7.6 erase(const key_type&) . . . 183
8.3.2.2.7.7 erase(iterator) . . . 183
8.3.2.2.7.8 extract(const_iterator) . . . 183
8.3.2.2.7.9 extract(const key_type&) . . . 184
8.3.2.2.7.10 insert(std::initializer_list<value_type>) . . . 184
8.3.2.2.7.11 insert(const_iterator, P&&) . . . 184
8.3.2.2.7.12 insert(const_iterator, node_type&&) . . . 185
8.3.2.2.7.13 insert(const_iterator, value_type&&) . . . 185
8.3.2.2.7.14 insert(const_iterator, const value_type&) . . . 185
8.3.2.2.7.15 insert(P&&) . . . 186
8.3.2.2.7.16 insert(InputIterator, InputIterator) . . . 186
8.3.2.2.7.17 insert(value_type&&) . . . 186
8.3.2.2.7.18 insert(node_type&&) . . . 187
8.3.2.2.7.19 insert(const value_type&) . . . 187
8.3.2.2.7.20 insert_or_assign(const_iterator, key_type&&, M&&) . . . 187
8.3.2.2.7.21 insert_or_assign(const key_type&, M&&) . . . 188
8.3.2.2.7.22 insert_or_assign(key_type&&, M&&) . . . 188
8.3.2.2.7.23 insert_or_assign(const_iterator, const key_type&, M&&) 188
8.3.2.2.7.24 merge(Map<Key, T, C2, Allocator>&) . . . 189
8.3.2.2.7.25 merge(Map<Key, T, C2, Allocator>&&) . . . 189
8.3.2.2.7.26 swap . . . 189
8.3.2.2.7.27 try_emplace(key_type&&, Args ...&&) . . . 190
8.3.2.2.7.28 try_emplace(const key_type&, Args ...&&) . . . 190
8.3.2.2.7.29 try_emplace(const_iterator, const key_type&, Args ...&&) . . . 190
8.3.2.2.7.30 try_emplace(const_iterator, key_type&&, Args ...&&) . . . 191
8.3.2.2.8 Observers . . . 191
8.3.2.2.8.1 key_comp . . . 191
8.3.2.2.8.2 value_comp . . . 191
8.3.2.2.9 Operator= . . . 192
8.3.2.2.9.1 operator= . . . 192
8.3.2.2.10 Other . . . 192
8.3.2.2.10.1 get_allocator . . . 192
8.3.3 Class: value_compare . . . 192
8.3.3.1 Protected Member Variables . . . 193
8.3.3.1.1 comp . . . 193
8.3.3.2 Public Member Functions . . . 193
8.3.3.2.1 Member Functions . . . 193
8.3.3.2.1.1 operator() . . . 193
8.3.3.3 Protected Member Functions . . . 194
8.3.3.3.1 Constructors . . . 194

8.3.3.3.1.1 value_compare . . . 194
8.4 Header: ara/core/memory_resource.h . . . 194
8.4.1 Non-Member Functions . . . 195
8.4.1.1 Other . . . 195
8.4.1.1.1 GetDefaultResource . . . 195
8.4.1.1.2 NewDeleteResource . . . 195
8.4.1.1.3 NullMemoryResource . . . 196
8.4.1.1.4 SetDefaultResource . . . 196
8.4.1.1.5 operator== . . . 197
8.4.1.1.6 operator== . . . 197
8.4.2 Class: MemoryResource . . . 197
8.4.2.1 Public Member Functions . . . 198
8.4.2.1.1 Special Member Functions . . . 198
8.4.2.1.1.1 Default Constructor . . . 198
8.4.2.1.1.2 Move Constructor . . . 198
8.4.2.1.1.3 Copy Constructor . . . 199
8.4.2.1.1.4 Move Assignment Operator . . . 199
8.4.2.1.1.5 Copy Assignment Operator . . . 200
8.4.2.1.1.6 Destructor . . . 200
8.4.2.1.2 Member Functions . . . 200
8.4.2.1.2.1 allocate . . . 200
8.4.2.1.2.2 deallocate . . . 201
8.4.2.1.2.3 is_equal . . . 201
8.4.2.2 Private Member Functions . . . 202
8.4.2.2.1 Member Functions . . . 202
8.4.2.2.1.1 do_allocate . . . 202
8.4.2.2.1.2 do_deallocate . . . 202
8.4.2.2.1.3 do_is_equal . . . 203
8.4.3 Class: MonotonicBufferResource . . . 203
8.4.3.1 Public Member Functions . . . 204
8.4.3.1.1 Special Member Functions . . . 204
8.4.3.1.1.1 Move Constructor . . . 204
8.4.3.1.1.2 Default Constructor . . . 204
8.4.3.1.1.3 Copy Constructor . . . 205
8.4.3.1.1.4 Copy Assignment Operator . . . 205
8.4.3.1.1.5 Move Assignment Operator . . . 205
8.4.3.1.1.6 Destructor . . . 206
8.4.3.1.2 Constructors . . . 206
8.4.3.1.2.1 MonotonicBufferResource(std::size_t) . . . 206
8.4.3.1.2.2 MonotonicBufferResource(void *, std::size_t, MemoryResource *) . . . 206
8.4.3.1.2.3 MonotonicBufferResource(MemoryResource *) . . . 207

8.4.3.1.2.4 MonotonicBufferResource(std::size_t, MemoryResource *) . . . 207
8.4.3.1.2.5 MonotonicBufferResource . . . 208
8.4.3.1.3 Member Functions . . . 208
8.4.3.1.3.1 release . . . 208
8.4.3.1.3.2 upstream_resource . . . 208
8.4.3.2 Protected Member Functions . . . 209
8.4.3.2.1 Member Functions . . . 209
8.4.3.2.1.1 do_allocate . . . 209
8.4.3.2.1.2 do_deallocate . . . 209
8.4.3.2.1.3 do_is_equal . . . 210
8.4.4 Class: PolymorphicAllocator . . . 210
8.4.4.1 Public Member Types . . . 211
8.4.4.1.1 Type Alias: value_type . . . 211
8.4.4.2 Public Member Functions . . . 211
8.4.4.2.1 Special Member Functions . . . 211
8.4.4.2.1.1 Copy Constructor . . . 211
8.4.4.2.1.2 Default Constructor . . . 211
8.4.4.2.1.3 Copy Constructor . . . 212
8.4.4.2.1.4 Move Constructor . . . 212
8.4.4.2.1.5 Move Assignment Operator . . . 213
8.4.4.2.1.6 Copy Assignment Operator . . . 213
8.4.4.2.2 Constructors . . . 213
8.4.4.2.2.1 PolymorphicAllocator . . . 213
8.4.4.2.3 Member Functions . . . 214
8.4.4.2.3.1 allocate . . . 214
8.4.4.2.3.2 allocate_bytes . . . 214
8.4.4.2.3.3 allocate_object . . . 215
8.4.4.2.3.4 construct . . . 215
8.4.4.2.3.5 deallocate . . . 215
8.4.4.2.3.6 deallocate_bytes . . . 216
8.4.4.2.3.7 deallocate_object . . . 216
8.4.4.2.3.8 delete_object . . . 217
8.4.4.2.3.9 destroy . . . 217
8.4.4.2.3.10 new_object . . . 217
8.4.4.2.3.11 resource . . . 218
8.4.4.2.3.12 select_on_container_copy_construction . . . 218
8.4.5 Struct: PoolOptions . . . 219
8.4.5.1 Public Member Variables . . . 219
8.4.5.1.1 largest_required_pool_block . . . 219
8.4.5.1.2 max_blocks_per_chunk . . . 219
8.4.6 Class: SynchronizedPoolResource . . . 220
8.4.6.1 Public Member Functions . . . 220

8.4.6.1.1 Special Member Functions . . . 220
8.4.6.1.1.1 Move Constructor . . . 220
8.4.6.1.1.2 Default Constructor . . . 221
8.4.6.1.1.3 Copy Constructor . . . 221
8.4.6.1.1.4 Move Assignment Operator . . . 221
8.4.6.1.1.5 Copy Assignment Operator . . . 222
8.4.6.1.1.6 Destructor . . . 222
8.4.6.1.2 Constructors . . . 222
8.4.6.1.2.1 SynchronizedPoolResource . . . 222
8.4.6.1.2.2 SynchronizedPoolResource . . . 223
8.4.6.1.2.3 SynchronizedPoolResource . . . 223
8.4.6.1.3 Member Functions . . . 223
8.4.6.1.3.1 options . . . 223
8.4.6.1.3.2 release . . . 224
8.4.6.1.3.3 upstream_resource . . . 224
8.4.6.2 Protected Member Functions . . . 224
8.4.6.2.1 Member Functions . . . 224
8.4.6.2.1.1 do_allocate . . . 224
8.4.6.2.1.2 do_deallocate . . . 225
8.4.6.2.1.3 do_is_equal . . . 225
8.4.7 Class: UnsynchronizedPoolResource . . . 226
8.4.7.1 Public Member Functions . . . 226
8.4.7.1.1 Special Member Functions . . . 226
8.4.7.1.1.1 Move Constructor . . . 226
8.4.7.1.1.2 Default Constructor . . . 227
8.4.7.1.1.3 Copy Constructor . . . 227
8.4.7.1.1.4 Copy Assignment Operator . . . 227
8.4.7.1.1.5 Move Assignment Operator . . . 228
8.4.7.1.1.6 Destructor . . . 228
8.4.7.1.2 Constructors . . . 228
8.4.7.1.2.1 UnsynchronizedPoolResource . . . 228
8.4.7.1.2.2 UnsynchronizedPoolResource . . . 229
8.4.7.1.2.3 UnsynchronizedPoolResource . . . 229
8.4.7.1.3 Member Functions . . . 230
8.4.7.1.3.1 options . . . 230
8.4.7.1.3.2 release . . . 230
8.4.7.1.3.3 upstream_resource . . . 230
8.4.7.2 Protected Member Functions . . . 231
8.4.7.2.1 Member Functions . . . 231
8.4.7.2.1.1 do_allocate . . . 231
8.4.7.2.1.2 do_deallocate . . . 231
8.4.7.2.1.3 do_is_equal . . . 232
8.5 Header: ara/core/optional.h . . . 232

8.5.1 Global Variables . . . 233
8.5.1.1 nullopt . . . 233
8.5.2 Non-Member Functions . . . 233
8.5.2.1 Other . . . 233
8.5.2.1.1 make_optional(Args ...&&) . . . 233
8.5.2.1.2 make_optional(std::initializer_list<U>, Args ...&&) . . . 234
8.5.2.1.3 make_optional(T&&) . . . 234
8.5.2.1.4 operator!=(const Optional<T>&, nullopt_t) . . . 234
8.5.2.1.5 operator!=(nullopt_t, const Optional<T>&) . . . 235
8.5.2.1.6 operator!=(const Optional<T>&, const Optional<U>&) . . . 235
8.5.2.1.7 operator!=(const Optional<T>&, const U&) . . . 235
8.5.2.1.8 operator!=(const U&, const Optional<T>&) . . . 236
8.5.2.1.9 operator<(const Optional<T>&, nullopt_t) . . . 236
8.5.2.1.10 operator<(const U&, const Optional<T>&) . . . 236
8.5.2.1.11 operator<(const Optional<T>&, const Optional<U>&) . . . 237
8.5.2.1.12 operator<(const Optional<T>&, const U&) . . . 237
8.5.2.1.13 operator<(nullopt_t, const Optional<T>&) . . . 237
8.5.2.1.14 operator<=(const Optional<T>&, nullopt_t) . . . 238
8.5.2.1.15 operator<=(nullopt_t, const Optional<T>&) . . . 238
8.5.2.1.16 operator<=(const Optional<T>&, const U&) . . . 238
8.5.2.1.17 operator<=(const U&, const Optional<T>&) . . . 239
8.5.2.1.18 operator<=(const Optional<T>&, const Optional<U>&) . . . 239
8.5.2.1.19 operator==(nullopt_t, const Optional<T>&) . . . 239
8.5.2.1.20 operator==(const Optional<T>&, nullopt_t) . . . 240
8.5.2.1.21 operator==(const Optional<T>&, const Optional<U>&) . . . 240
8.5.2.1.22 operator==(const Optional<T>&, const U&) . . . 240
8.5.2.1.23 operator==(const U&, const Optional<T>&) . . . 241
8.5.2.1.24 operator>(const Optional<T>&, const Optional<U>&) . . . 241
8.5.2.1.25 operator>(const Optional<T>&, const U&) . . . 241
8.5.2.1.26 operator>(const U&, const Optional<T>&) . . . 242
8.5.2.1.27 operator>(const Optional<T>&, nullopt_t) . . . 242
8.5.2.1.28 operator>(nullopt_t, const Optional<T>&) . . . 242
8.5.2.1.29 operator>=(const Optional<T>&, const U&) . . . 243
8.5.2.1.30 operator>=(nullopt_t, const Optional<T>&) . . . 243
8.5.2.1.31 operator>=(const Optional<T>&, const Optional<U>&) . . . 243
8.5.2.1.32 operator>=(const Optional<T>&, nullopt_t) . . . 244
8.5.2.1.33 operator>=(const U&, const Optional<T>&) . . . 244
8.5.2.1.34 swap . . . 244
8.5.3 Class: Optional . . . 245
8.5.3.1 Public Member Types . . . 245
8.5.3.1.1 Type Alias: value_type . . . 245
8.5.3.2 Public Member Functions . . . 246
8.5.3.2.1 Special Member Functions . . . 246

8.5.3.2.1.1 Copy Constructor . . . 246
8.5.3.2.1.2 Copy Constructor . . . 246
8.5.3.2.1.3 Move Constructor . . . 247
8.5.3.2.1.4 Default Constructor . . . 247
8.5.3.2.1.5 Move Constructor . . . 247
8.5.3.2.1.6 Copy Assignment Operator . . . 248
8.5.3.2.1.7 Move Assignment Operator . . . 248
8.5.3.2.1.8 Move Assignment Operator . . . 249
8.5.3.2.1.9 Destructor . . . 249
8.5.3.2.2 Constructors . . . 250
8.5.3.2.2.1 Optional(nullopt_t) . . . 250
8.5.3.2.2.2 Optional(in_place_t, Args&&) . . . 250
8.5.3.2.2.3 Optional(in_place_t, std::initializer_list<U>, Args&&) . . . 250
8.5.3.2.2.4 Optional(U&&) . . . 251
8.5.3.2.3 Modifiers . . . 251
8.5.3.2.3.1 emplace(Args&&) . . . 251
8.5.3.2.3.2 emplace(std::initializer_list<U>, Args&&) . . . 252
8.5.3.2.3.3 reset . . . 252
8.5.3.2.3.4 swap . . . 253
8.5.3.2.4 Observers . . . 253
8.5.3.2.4.1 has_value . . . 253
8.5.3.2.4.2 operator bool . . . 253
8.5.3.2.4.3 operator*() const && . . . 254
8.5.3.2.4.4 operator*() & . . . 254
8.5.3.2.4.5 operator*() && . . . 255
8.5.3.2.4.6 operator*() const & . . . 255
8.5.3.2.4.7 operator-> . . . 256
8.5.3.2.4.8 operator->() const . . . 256
8.5.3.2.4.9 value() const & . . . 257
8.5.3.2.4.10 value() & . . . 257
8.5.3.2.4.11 value() && . . . 258
8.5.3.2.4.12 value() const && . . . 258
8.5.3.2.4.13 value_or(U&&) && . . . 259
8.5.3.2.4.14 value_or(U&&) const & . . . 259
8.5.3.2.5 Operator= . . . 259
8.5.3.2.5.1 operator=(U&&) & . . . 259
8.5.3.2.5.2 operator=(const Optional<U>&) & . . . 260
8.5.3.2.5.3 operator= . . . 260
8.5.4 Class: Optional<T &> . . . 261
8.5.4.1 Public Member Types . . . 261
8.5.4.1.1 Type Alias: value_type . . . 261
8.5.4.2 Public Member Functions . . . 262
8.5.4.2.1 Special Member Functions . . . 262

8.5.4.2.1.1 Default Constructor . . . 262
8.5.4.2.1.2 Copy Constructor . . . 262
8.5.4.2.1.3 Move Constructor . . . 263
8.5.4.2.1.4 Copy Assignment Operator . . . 263
8.5.4.2.1.5 Move Assignment Operator . . . 263
8.5.4.2.1.6 Copy Assignment Operator . . . 264
8.5.4.2.1.7 Destructor . . . 264
8.5.4.2.2 Constructors . . . 265
8.5.4.2.2.1 Optional(U&&) . . . 265
8.5.4.2.2.2 Optional(Optional<U> const&) . . . 265
8.5.4.2.2.3 Optional . . . 266
8.5.4.2.3 Modifiers . . . 266
8.5.4.2.3.1 emplace . . . 266
8.5.4.2.3.2 reset . . . 267
8.5.4.2.3.3 swap . . . 267
8.5.4.2.4 Observers . . . 267
8.5.4.2.4.1 has_value . . . 267
8.5.4.2.4.2 operator bool . . . 268
8.5.4.2.4.3 operator*() const . . . 268
8.5.4.2.4.4 operator* . . . 269
8.5.4.2.4.5 operator-> . . . 269
8.5.4.2.4.6 operator->() const . . . 270
8.5.4.2.4.7 value() const . . . 270
8.5.4.2.4.8 value . . . 270
8.5.4.2.4.9 value_or . . . 271
8.5.4.2.5 Operator= . . . 271
8.5.4.2.5.1 operator=(nullopt_t) & . . . 271
8.5.4.2.5.2 operator=(U&&) & . . . 272
8.5.5 Struct: nullopt_t . . . 272
8.5.6 Struct: hash . . . 272
8.5.6.1 Public Member Functions . . . 273
8.5.6.1.1 Member Functions . . . 273
8.5.6.1.1.1 operator() . . . 273
8.6 Header: ara/core/span.h . . . 273
8.6.1 Global Variables . . . 274
8.6.1.1 dynamic_extent . . . 274
8.6.2 Non-Member Functions . . . 274
8.6.2.1 Other . . . 274
8.6.2.1.1 MakeSpan(T *, typename Span<T>::size_type) . . . 274
8.6.2.1.2 MakeSpan(T *, T *) . . . 275
8.6.2.1.3 MakeSpan(T()&) . . . 275
8.6.2.1.4 MakeSpan(Container&) . . . 276
8.6.2.1.5 MakeSpan(const Container&) . . . 276

8.6.2.1.6 as_bytes . . . 277
8.6.2.1.7 as_writable_bytes . . . 277
8.6.3 Class: Span . . . 277
8.6.3.1 Public Member Types . . . 278
8.6.3.1.1 Type Alias: const_iterator . . . 278
8.6.3.1.2 Type Alias: const_pointer . . . 278
8.6.3.1.3 Type Alias: const_reference . . . 278
8.6.3.1.4 Type Alias: const_reverse_iterator . . . 279
8.6.3.1.5 Type Alias: difference_type . . . 279
8.6.3.1.6 Type Alias: element_type . . . 279
8.6.3.1.7 Type Alias: iterator . . . 280
8.6.3.1.8 Type Alias: pointer . . . 280
8.6.3.1.9 Type Alias: reference . . . 280
8.6.3.1.10 Type Alias: reverse_iterator . . . 281
8.6.3.1.11 Type Alias: size_type . . . 281
8.6.3.1.12 Type Alias: value_type . . . 281
8.6.3.2 Public Member Variables . . . 282
8.6.3.2.1 extent . . . 282
8.6.3.3 Public Member Functions . . . 282
8.6.3.3.1 Special Member Functions . . . 282
8.6.3.3.1.1 Span(const Span<U, N>&) . . . 282
8.6.3.3.1.2 Default Constructor . . . 282
8.6.3.3.1.3 Copy Constructor . . . 283
8.6.3.3.1.4 Copy Assignment Operator . . . 283
8.6.3.3.1.5 Destructor . . . 283
8.6.3.3.2 Constructors . . . 284
8.6.3.3.2.1 Span(pointer, pointer) . . . 284
8.6.3.3.2.2 Span(element_type()&) . . . 284
8.6.3.3.2.3 Span(ara::core::Array<U, N>&) . . . 284
8.6.3.3.2.4 Span(const ara::core::Array<U, N>&) . . . 285
8.6.3.3.2.5 Span(Container&) . . . 285
8.6.3.3.2.6 Span(const Container&) . . . 285
8.6.3.3.2.7 Span(pointer, size_type) . . . 286
8.6.3.3.3 Element access . . . 286
8.6.3.3.3.1 back() const . . . 286
8.6.3.3.3.2 data() const . . . 286
8.6.3.3.3.3 front() const . . . 287
8.6.3.3.3.4 operator[](size_type) const . . . 287
8.6.3.3.4 Iterators . . . 287
8.6.3.3.4.1 begin . . . 287
8.6.3.3.4.2 cbegin . . . 288
8.6.3.3.4.3 cend . . . 288
8.6.3.3.4.4 crbegin . . . 288

8.6.3.3.4.5 crend . . . 289
8.6.3.3.4.6 end . . . 289
8.6.3.3.4.7 rbegin . . . 289
8.6.3.3.4.8 rend . . . 290
8.6.3.3.5 Observers . . . 290
8.6.3.3.5.1 empty . . . 290
8.6.3.3.5.2 size . . . 290
8.6.3.3.5.3 size_bytes . . . 291
8.6.3.3.6 Subviews . . . 291
8.6.3.3.6.1 first(size_type) const . . . 291
8.6.3.3.6.2 first() const . . . 291
8.6.3.3.6.3 last(size_type) const . . . 292
8.6.3.3.6.4 last() const . . . 292
8.6.3.3.6.5 subspan() const . . . 292
8.6.3.3.6.6 subspan(size_type, size_type) const . . . 293
8.7 Header: ara/core/string_view.h . . . 293
8.7.1 Non-Member Functions . . . 294
8.7.1.1 Other . . . 294
8.7.1.1.1 operator!= . . . 294
8.7.1.1.2 operator""_SV . . . 294
8.7.1.1.3 operator< . . . 295
8.7.1.1.4 operator<< . . . 295
8.7.1.1.5 operator<= . . . 295
8.7.1.1.6 operator== . . . 296
8.7.1.1.7 operator> . . . 296
8.7.1.1.8 operator>= . . . 296
8.7.2 Class: StringView . . . 297
8.7.2.1 Public Member Types . . . 297
8.7.2.1.1 Type Alias: const_iterator . . . 297
8.7.2.1.2 Type Alias: const_pointer . . . 298
8.7.2.1.3 Type Alias: const_reference . . . 298
8.7.2.1.4 Type Alias: const_reverse_iterator . . . 298
8.7.2.1.5 Type Alias: difference_type . . . 299
8.7.2.1.6 Type Alias: iterator . . . 299
8.7.2.1.7 Type Alias: pointer . . . 300
8.7.2.1.8 Type Alias: reference . . . 300
8.7.2.1.9 Type Alias: reverse_iterator . . . 300
8.7.2.1.10 Type Alias: size_type . . . 301
8.7.2.1.11 Type Alias: value_type . . . 301
8.7.2.2 Public Member Variables . . . 302
8.7.2.2.1 npos . . . 302
8.7.2.3 Public Member Functions . . . 302
8.7.2.3.1 Special Member Functions . . . 302

8.7.2.3.1.1 Default Constructor . . . 302
8.7.2.3.1.2 Copy Constructor . . . 303
8.7.2.3.1.3 Move Constructor . . . 303
8.7.2.3.1.4 Copy Assignment Operator . . . 303
8.7.2.3.1.5 Move Assignment Operator . . . 304
8.7.2.3.1.6 Destructor . . . 304
8.7.2.3.2 Constructors . . . 305
8.7.2.3.2.1 StringView(const char *, size_type) . . . 305
8.7.2.3.2.2 StringView(const char *) . . . 305
8.7.2.3.3 Capacity . . . 306
8.7.2.3.3.1 empty . . . 306
8.7.2.3.3.2 length . . . 306
8.7.2.3.3.3 max_size . . . 306
8.7.2.3.3.4 size . . . 307
8.7.2.3.4 Element access . . . 307
8.7.2.3.4.1 at . . . 307
8.7.2.3.4.2 back . . . 308
8.7.2.3.4.3 data . . . 308
8.7.2.3.4.4 front . . . 308
8.7.2.3.4.5 operator[] . . . 309
8.7.2.3.4.6 remove_prefix . . . 309
8.7.2.3.4.7 remove_suffix . . . 310
8.7.2.3.4.8 swap . . . 310
8.7.2.3.5 Iterators . . . 311
8.7.2.3.5.1 begin() const . . . 311
8.7.2.3.5.2 cbegin() const . . . 311
8.7.2.3.5.3 cend() const . . . 311
8.7.2.3.5.4 crbegin() const . . . 312
8.7.2.3.5.5 crend() const . . . 312
8.7.2.3.5.6 end() const . . . 313
8.7.2.3.5.7 rbegin() const . . . 313
8.7.2.3.5.8 rend() const . . . 313
8.7.2.3.6 Operations . . . 314
8.7.2.3.6.1 compare(size_type, size_type, StringView) const . . . 314
8.7.2.3.6.2 compare(StringView) const . . . 314
8.7.2.3.6.3 compare(const char *) const . . . 315
8.7.2.3.6.4 compare(size_type, size_type, const char *) const . . . 315
8.7.2.3.6.5 compare(size_type, size_type, StringView, size_type, size_type) const . . . 316
8.7.2.3.6.6 compare(size_type, size_type, const char *, size_type) const . . . 316
8.7.2.3.6.7 contains(const char *) const . . . 317
8.7.2.3.6.8 contains(StringView) const . . . 317
8.7.2.3.6.9 contains(char) const . . . 317

8.7.2.3.6.10 copy . . . 318
8.7.2.3.6.11 ends_with(char) const . . . 318
8.7.2.3.6.12 ends_with(StringView) const . . . 318
8.7.2.3.6.13 ends_with(const char *) const . . . 319
8.7.2.3.6.14 find(char, size_type) const . . . 319
8.7.2.3.6.15 find(StringView, size_type) const . . . 320
8.7.2.3.6.16 find(const char *, size_type, size_type) const . . . 320
8.7.2.3.6.17 find(const char *, size_type) const . . . 320
8.7.2.3.6.18 find_first_not_of(char, size_type) const . . . 321
8.7.2.3.6.19 find_first_not_of(const char *, size_type) const . . . 321
8.7.2.3.6.20 find_first_not_of(const char *, size_type, size_type) const . . . 322
8.7.2.3.6.21 find_first_not_of(StringView, size_type) const . . . 322
8.7.2.3.6.22 find_first_of(const char *, size_type) const . . . 323
8.7.2.3.6.23 find_first_of(const char *, size_type, size_type) const . 323
8.7.2.3.6.24 find_first_of(StringView, size_type) const . . . 324
8.7.2.3.6.25 find_first_of(char, size_type) const . . . 324
8.7.2.3.6.26 find_last_not_of(const char *, size_type) const . . . 324
8.7.2.3.6.27 find_last_not_of(const char *, size_type, size_type) const . . . 325
8.7.2.3.6.28 find_last_not_of(StringView, size_type) const . . . 325
8.7.2.3.6.29 find_last_not_of(char, size_type) const . . . 326
8.7.2.3.6.30 find_last_of(const char *, size_type, size_type) const . 326
8.7.2.3.6.31 find_last_of(char, size_type) const . . . 327
8.7.2.3.6.32 find_last_of(StringView, size_type) const . . . 327
8.7.2.3.6.33 find_last_of(const char *, size_type) const . . . 327
8.7.2.3.6.34 rfind(char, size_type) const . . . 328
8.7.2.3.6.35 rfind(StringView, size_type) const . . . 328
8.7.2.3.6.36 rfind(const char *, size_type) const . . . 329
8.7.2.3.6.37 rfind(const char *, size_type, size_type) const . . . 329
8.7.2.3.6.38 starts_with(char) const . . . 329
8.7.2.3.6.39 starts_with(const char *) const . . . 330
8.7.2.3.6.40 starts_with(StringView) const . . . 330
8.7.2.3.6.41 substr . . . 331
8.7.3 Struct: hash . . . 331
8.7.3.1 Public Member Functions . . . 332
8.7.3.1.1 Member Functions . . . 332
8.7.3.1.1.1 operator() . . . 332
8.8 Header: ara/core/string.h . . . 332
8.8.1 Non-Member Types . . . 333
8.8.1.1 Type Alias: String . . . 333
8.8.2 Non-Member Functions . . . 333
8.8.2.1 Other . . . 333

8.8.2.1.1 getline(std::basic_istream<charT, traits>&&, BasicString<Allocator>&, charT) . . . 333
8.8.2.1.2 getline(std::basic_istream<charT, traits>&, BasicString<Allocator>&, charT) . . . 334
8.8.2.1.3 getline(std::basic_istream<charT, traits>&&, BasicString<Allocator>&) . . . 334
8.8.2.1.4 getline(std::basic_istream<charT, traits>&, BasicString<Allocator>&) . . . 334
8.8.2.1.5 operator!=(const charT *, const BasicString<Allocator>&) . 335
8.8.2.1.6 operator!=(const BasicString<Allocator>&, const charT *) . 335
8.8.2.1.7 operator!=(const BasicString<Allocator>&, const BasicString<Allocator>&) . . . 336
8.8.2.1.8 operator+(charT, const BasicString<Allocator>&) . . . 336
8.8.2.1.9 operator+(charT, BasicString<Allocator>&&) . . . 337
8.8.2.1.10 operator+(const charT *, BasicString<Allocator>&&) . . . 337
8.8.2.1.11 operator+(BasicString<Allocator>&&, const charT *) . . . 337
8.8.2.1.12 operator+(const BasicString<Allocator>&, charT) . . . 338
8.8.2.1.13 operator+(BasicString<Allocator>&&, charT) . . . 338
8.8.2.1.14 operator+(const charT *, const BasicString<Allocator>&) . 339
8.8.2.1.15 operator+(BasicString<Allocator>&&, BasicString<Allocator>&&) . . . 339
8.8.2.1.16 operator+(const BasicString<Allocator>&, BasicString<Allocator>&&) . . . 339
8.8.2.1.17 operator+(BasicString<Allocator>&&, const BasicString<Allocator>&) . . . 340
8.8.2.1.18 operator+(const BasicString<Allocator>&, const BasicString<Allocator>&) . . . 340
8.8.2.1.19 operator+(const BasicString<Allocator>&, const charT *) . 341
8.8.2.1.20 operator<(const charT *, const BasicString<Allocator>&) . 341
8.8.2.1.21 operator<(const BasicString<Allocator>&, const BasicString<Allocator>&) . . . 342
8.8.2.1.22 operator<(const BasicString<Allocator>&, const charT *) . 342
8.8.2.1.23 operator<<(std::basic_ostream<charT, traits>&, const BasicString<Allocator>&) . . . 342
8.8.2.1.24 operator<=(const BasicString<Allocator>&, const charT *) . 343
8.8.2.1.25 operator<=(const BasicString<Allocator>&, const BasicString<Allocator>&) . . . 343
8.8.2.1.26 operator<=(const charT *, const BasicString<Allocator>&) . 344
8.8.2.1.27 operator==(const BasicString<Allocator>&, const BasicString<Allocator>&) . . . 344
8.8.2.1.28 operator==(const BasicString<Allocator>&, const charT *) . 345
8.8.2.1.29 operator==(const charT *, const BasicString<Allocator>&) . 345
8.8.2.1.30 operator>(const BasicString<Allocator>&, const BasicString<Allocator>&) . . . 346

8.8.2.1.31 operator>(const BasicString<Allocator>&, const charT *) . 346
8.8.2.1.32 operator>(const charT *, const BasicString<Allocator>&) . 347
8.8.2.1.33 operator>=(const BasicString<Allocator>&, const BasicString<Allocator>&) . 347
8.8.2.1.34 operator>=(const charT *, const BasicString<Allocator>&) . 348
8.8.2.1.35 operator>=(const BasicString<Allocator>&, const charT *) . 348
8.8.2.1.36 operator>>(std::basic_istream<charT, traits>&, BasicString<Allocator>&) . 348
8.8.2.1.37 stod(const ara::core::String&, size_t *) . 349
8.8.2.1.38 stof(const ara::core::String&, size_t *) . 349
8.8.2.1.39 stoi(const ara::core::String&, size_t *, int) . 350
8.8.2.1.40 stol(const ara::core::String&, size_t *, int) . 350
8.8.2.1.41 stold(const ara::core::String&, size_t *) . 350
8.8.2.1.42 stoll(const ara::core::String&, size_t *, int) . 351
8.8.2.1.43 stoul(const ara::core::String&, size_t *, int) . 351
8.8.2.1.44 stoull(const ara::core::String&, size_t *, int) . 351
8.8.2.1.45 swap(BasicString<Allocator>&, BasicString<Allocator>&) . 352
8.8.2.1.46 to_string(unsigned long long) . 352
8.8.2.1.47 to_string(float) . 352
8.8.2.1.48 to_string(double) . 353
8.8.2.1.49 to_string(long long) . 353
8.8.2.1.50 to_string(unsigned long) . 353
8.8.2.1.51 to_string(long) . 354
8.8.2.1.52 to_string(unsigned) . 354
8.8.2.1.53 to_string(int) . 354
8.8.2.1.54 to_string(long double) . 355
8.8.3 Class: BasicString . 355
8.8.3.1 Public Member Types . 355
8.8.3.1.1 Type Alias: allocator_type . 355
8.8.3.1.2 Type Alias: const_iterator . 356
8.8.3.1.3 Type Alias: const_pointer . 356
8.8.3.1.4 Type Alias: const_reference . 357
8.8.3.1.5 Type Alias: const_reverse_iterator . 357
8.8.3.1.6 Type Alias: difference_type . 357
8.8.3.1.7 Type Alias: iterator . 358
8.8.3.1.8 Type Alias: pointer . 358
8.8.3.1.9 Type Alias: reference . 358
8.8.3.1.10 Type Alias: reverse_iterator . 359
8.8.3.1.11 Type Alias: size_type . 359
8.8.3.1.12 Type Alias: traits_type . 359
8.8.3.1.13 Type Alias: value_type . 360
8.8.3.2 Public Member Variables . 360
8.8.3.2.1 npos . 360
8.8.3.3 Public Member Functions . 360

8.8.3.3.1 Special Member Functions . . . 360
8.8.3.3.1.1 BasicString(BasicString&&) . . . 360
8.8.3.3.1.2 BasicString() . . . 361
8.8.3.3.1.3 BasicString(const BasicString&) . . . 361
8.8.3.3.1.4 operator=(const BasicString&) . . . 362
8.8.3.3.1.5 operator=(BasicString&&) . . . 362
8.8.3.3.1.6 Destructor . . . 362
8.8.3.3.2 Constructors . . . 363
8.8.3.3.2.1 BasicString(const BasicString&, const Allocator&) . . . 363
8.8.3.3.2.2 BasicString(std::initializer_list<value_type>, const Allocator&) . . . 363
8.8.3.3.2.3 BasicString(const Allocator&) . . . 364
8.8.3.3.2.4 BasicString(const BasicString&, size_type, const Allocator&) . . . 364
8.8.3.3.2.5 BasicString(const BasicString&, size_type, size_type, const Allocator&) . . . 364
8.8.3.3.2.6 BasicString(const T&, size_type, size_type, const Allocator&) . . . 365
8.8.3.3.2.7 BasicString(StringView, const Allocator&) . . . 365
8.8.3.3.2.8 BasicString(const value_type *, size_type, const Allocator&) . . . 365
8.8.3.3.2.9 BasicString(const value_type *, const Allocator&) . . . 366
8.8.3.3.2.10 BasicString(size_type, value_type, const Allocator&) . . . 366
8.8.3.3.2.11 BasicString(InputIterator, InputIterator, const Allocator&) . . . 367
8.8.3.3.2.12 BasicString(BasicString&&, const Allocator&) . . . 367
8.8.3.3.3 Capacity . . . 367
8.8.3.3.3.1 capacity . . . 367
8.8.3.3.3.2 clear . . . 368
8.8.3.3.3.3 empty . . . 368
8.8.3.3.3.4 length . . . 368
8.8.3.3.3.5 max_size . . . 369
8.8.3.3.3.6 reserve . . . 369
8.8.3.3.3.7 resize(size_type, value_type) . . . 369
8.8.3.3.3.8 resize(size_type) . . . 370
8.8.3.3.3.9 shrink_to_fit . . . 370
8.8.3.3.3.10 size . . . 370
8.8.3.3.4 Element access . . . 371
8.8.3.3.4.1 at(size_type) . . . 371
8.8.3.3.4.2 at(size_type) const . . . 371
8.8.3.3.4.3 back() const . . . 371
8.8.3.3.4.4 back() . . . 372
8.8.3.3.4.5 c_str . . . 372
8.8.3.3.4.6 data . . . 372

8.8.3.3.4.7 data . . . 373
8.8.3.3.4.8 front() const . . . 373
8.8.3.3.4.9 front() . . . 373
8.8.3.3.4.10 operator StringView . . . 374
8.8.3.3.4.11 operator[](size_type) const . . . 374
8.8.3.3.4.12 operator[](size_type) . . . 374
8.8.3.3.5 Iterators . . . 375
8.8.3.3.5.1 begin() const . . . 375
8.8.3.3.5.2 begin() . . . 375
8.8.3.3.5.3 cbegin . . . 375
8.8.3.3.5.4 cend . . . 376
8.8.3.3.5.5 crbegin . . . 376
8.8.3.3.5.6 crend . . . 376
8.8.3.3.5.7 end() . . . 377
8.8.3.3.5.8 end() const . . . 377
8.8.3.3.5.9 rbegin() . . . 377
8.8.3.3.5.10 rbegin() const . . . 378
8.8.3.3.5.11 rend() const . . . 378
8.8.3.3.5.12 rend() . . . 378
8.8.3.3.6 Modifiers . . . 379
8.8.3.3.6.1 append(const BasicString&, size_type, size_type) . . . 379
8.8.3.3.6.2 append(StringView) . . . 379
8.8.3.3.6.3 append(const BasicString&) . . . 379
8.8.3.3.6.4 append(const T&, size_type, size_type) . . . 380
8.8.3.3.6.5 append(const value_type *, size_type) . . . 380
8.8.3.3.6.6 append(const value_type *) . . . 380
8.8.3.3.6.7 append(size_type, value_type) . . . 381
8.8.3.3.6.8 append(InputIterator, InputIterator) . . . 381
8.8.3.3.6.9 append(std::initializer_list<value_type>) . . . 381
8.8.3.3.6.10 copy . . . 382
8.8.3.3.6.11 erase(const_iterator, const_iterator) . . . 382
8.8.3.3.6.12 erase(const_iterator) . . . 382
8.8.3.3.6.13 erase(size_type, size_type) . . . 383
8.8.3.3.6.14 insert(const_iterator, std::initializer_list<value_type>) . . . 383
8.8.3.3.6.15 insert(size_type, const BasicString&, size_type, size_type) . . . 384
8.8.3.3.6.16 insert(size_type, const BasicString&) . . . 384
8.8.3.3.6.17 insert(const_iterator, InputIterator, InputIterator) . . . 384
8.8.3.3.6.18 insert(const_iterator, size_type, value_type) . . . 385
8.8.3.3.6.19 insert(const_iterator, value_type) . . . 385
8.8.3.3.6.20 insert(size_type, size_type, value_type) . . . 385
8.8.3.3.6.21 insert(size_type, const T&, size_type, size_type) . . . 386
8.8.3.3.6.22 insert(size_type, StringView) . . . 386

8.8.3.3.6.23 insert(size_type, const value_type *, size_type) 386
8.8.3.3.6.24 insert(size_type, const value_type *) 387
8.8.3.3.6.25 operator+=(const value_type *) 387
8.8.3.3.6.26 operator+=(value_type) 388
8.8.3.3.6.27 operator+=(const BasicString&) 388
8.8.3.3.6.28 operator+=(std::initializer_list<value_type>) 388
8.8.3.3.6.29 operator+=(StringView) 389
8.8.3.3.6.30 pop_back 389
8.8.3.3.6.31 push_back(value_type) 389
8.8.3.3.6.32 replace(size_type, size_type, size_type, value_type) . 390
8.8.3.3.6.33 replace(size_type, size_type, const value_type *, size_type) 390
8.8.3.3.6.34 replace(const_iterator, const_iterator, const BasicString&) 390
8.8.3.3.6.35 replace(size_type, size_type, StringView) 391
8.8.3.3.6.36 replace(size_type, size_type, const BasicString&, size_type, size_type) 391
8.8.3.3.6.37 replace(size_type, size_type, const BasicString&) . . . 392
8.8.3.3.6.38 replace(const_iterator, const_iterator, const value_type *) 392
8.8.3.3.6.39 replace(const_iterator, const_iterator, StringView) . . . 392
8.8.3.3.6.40 replace(size_type, size_type, const T&, size_type, size_type) 393
8.8.3.3.6.41 replace(const_iterator, const_iterator, const value_type *, size_type) 393
8.8.3.3.6.42 replace(const_iterator, const_iterator, size_type, value_type) 393
8.8.3.3.6.43 replace(const_iterator, const_iterator, InputIterator, InputIterator) 394
8.8.3.3.6.44 replace(const_iterator, const_iterator, std::initializer_list<value_type>) 394
8.8.3.3.6.45 replace(size_type, size_type, const value_type *) . . . 394
8.8.3.3.6.46 swap 395
8.8.3.3.7 Operations 395
8.8.3.3.7.1 compare(const BasicString&) const 395
8.8.3.3.7.2 compare(size_type, size_type, const value_type *, size_type) const 396
8.8.3.3.7.3 compare(size_type, size_type, const BasicString&) const 396
8.8.3.3.7.4 compare(size_type, size_type, StringView) const . . . 396
8.8.3.3.7.5 compare(size_type, size_type, const BasicString&, size_type, size_type) const 397
8.8.3.3.7.6 compare(const value_type *) const 397

8.8.3.3.7.7 compare(size_type, size_type, const value_type *) const . . . 398
8.8.3.3.7.8 compare(size_type, size_type, const T&, size_type, size_type) const . . . 398
8.8.3.3.7.9 compare(StringView) const . . . 398
8.8.3.3.7.10 substr(size_type, size_type) const . . . 399
8.8.3.3.8 Operator= . . . 399
8.8.3.3.8.1 operator=(std::initializer_list<value_type>) . . . 399
8.8.3.3.8.2 operator=(value_type) . . . 400
8.8.3.3.8.3 operator=(StringView) . . . 400
8.8.3.3.8.4 operator=(const value_type *) . . . 400
8.8.3.3.9 Other . . . 401
8.8.3.3.9.1 assign(const BasicString&) . . . 401
8.8.3.3.9.2 assign(BasicString&&) . . . 401
8.8.3.3.9.3 assign(const BasicString&, size_type, size_type) . . . 401
8.8.3.3.9.4 assign(StringView) . . . 402
8.8.3.3.9.5 assign(const T&, size_type, size_type) . . . 402
8.8.3.3.9.6 assign(const value_type *, size_type) . . . 402
8.8.3.3.9.7 assign(const value_type *) . . . 403
8.8.3.3.9.8 assign(InputIterator, InputIterator) . . . 403
8.8.3.3.9.9 assign(std::initializer_list<value_type>) . . . 403
8.8.3.3.9.10 assign(size_type, value_type) . . . 404
8.8.3.3.9.11 get_allocator() const . . . 404
8.8.3.3.10 Search . . . 405
8.8.3.3.10.1 find(StringView, size_type) const . . . 405
8.8.3.3.10.2 find(const BasicString&, size_type) const . . . 405
8.8.3.3.10.3 find(const value_type *, size_type) const . . . 405
8.8.3.3.10.4 find(value_type, size_type) const . . . 406
8.8.3.3.10.5 find(const value_type *, size_type, size_type) const . . . 406
8.8.3.3.10.6 find_first_not_of(value_type, size_type) const . . . 406
8.8.3.3.10.7 find_first_not_of(const value_type *, size_type, size_type) const . . . 407
8.8.3.3.10.8 find_first_not_of(const BasicString&, size_type) const 407
8.8.3.3.10.9 find_first_not_of(StringView, size_type) const . . . 407
8.8.3.3.10.10 find_first_not_of(const value_type *, size_type) const . 408
8.8.3.3.10.11 find_first_of(const value_type *, size_type) const . . . 408
8.8.3.3.10.12 find_first_of(StringView, size_type) const . . . 408
8.8.3.3.10.13 find_first_of(value_type, size_type) const . . . 409
8.8.3.3.10.14 find_first_of(const BasicString&, size_type) const . . . 409
8.8.3.3.10.15 find_first_of(const value_type *, size_type, size_type) const . . . 409
8.8.3.3.10.16 find_last_not_of(const value_type *, size_type, size_type) const . . . 410
8.8.3.3.10.17 find_last_not_of(value_type, size_type) const . . . 410

8.8.3.3.10.18 find_last_not_of(const BasicString&, size_type) const 410
8.8.3.3.10.19 find_last_not_of(StringView, size_type) const 411
8.8.3.3.10.20 find_last_not_of(const value_type *, size_type) const . 411
8.8.3.3.10.21 find_last_of(const BasicString&, size_type) const . . . 411
8.8.3.3.10.22 find_last_of(const value_type *, size_type, size_type) const 412
8.8.3.3.10.23 find_last_of(const value_type *, size_type) const . . . 412
8.8.3.3.10.24 find_last_of(value_type, size_type) const 412
8.8.3.3.10.25 find_last_of(StringView, size_type) const 413
8.8.3.3.10.26 rfind(const value_type *, size_type, size_type) const . 413
8.8.3.3.10.27 rfind(const value_type *, size_type) const 413
8.8.3.3.10.28 rfind(const BasicString&, size_type) const 414
8.8.3.3.10.29 rfind(value_type, size_type) const 414
8.8.3.3.10.30 rfind(StringView, size_type) const 414
8.8.4 Struct: hash 415
8.9 Header: ara/core/variant.h 415
8.9.1 Non-Member Types 416
8.9.1.1 Type Alias: variant_alternative_t 416
8.9.2 Global Variables 416
8.9.2.1 variant_size_v 416
8.9.3 Non-Member Functions 417
8.9.3.1 Other 417
8.9.3.1.1 get(Variant<Types...>&) 417
8.9.3.1.2 get(Variant<Types...>&&) 417
8.9.3.1.3 get(const Variant<Types...>&) 418
8.9.3.1.4 get(const Variant<Types...>&&) 418
8.9.3.1.5 get(Variant<Types...>&) 419
8.9.3.1.6 get(Variant<Types...>&&) 419
8.9.3.1.7 get(const Variant<Types...>&) 420
8.9.3.1.8 get(const Variant<Types...>&&) 420
8.9.3.1.9 get_if(Variant<Types...> *) 421
8.9.3.1.10 get_if(const Variant<Types...> *) 421
8.9.3.1.11 get_if(Variant<Types...> *) 421
8.9.3.1.12 get_if(const Variant<Types...> *) 422
8.9.3.1.13 holds_alternative 422
8.9.3.1.14 operator!=(Monostate, Monostate) 423
8.9.3.1.15 operator!=(const Variant<Types...>&, const Variant<Types...>&) 423
8.9.3.1.16 operator<(const Variant<Types...>&, const Variant<Types...>&) 423
8.9.3.1.17 operator<(Monostate, Monostate) 424
8.9.3.1.18 operator<=(Monostate, Monostate) 424
8.9.3.1.19 operator<=(const Variant<Types...>&, const Variant<Types...>&) 425

8.9.3.1.20 operator==(Monostate, Monostate) . . . 425
8.9.3.1.21 operator==(const Variant<Types...>&, const Variant<Types...>&) . . . 425
8.9.3.1.22 operator>(Monostate, Monostate) . . . 426
8.9.3.1.23 operator>(const Variant<Types...>&, const Variant<Types...>&) . . . 426
8.9.3.1.24 operator>=(Monostate, Monostate) . . . 427
8.9.3.1.25 operator>=(const Variant<Types...>&, const Variant<Types...>&) . . . 427
8.9.3.1.26 swap . . . 427
8.9.3.1.27 visit(Visitor&&, Variants ...&&) . . . 428
8.9.3.1.28 visit(Visitor&&, Variants ...&&) . . . 428
8.9.4 Struct: Monostate . . . 429
8.9.5 Class: Variant . . . 429
8.9.5.1 Public Member Functions . . . 430
8.9.5.1.1 Special Member Functions . . . 430
8.9.5.1.1.1 Move Constructor . . . 430
8.9.5.1.1.2 Default Constructor . . . 430
8.9.5.1.1.3 Copy Constructor . . . 431
8.9.5.1.1.4 Move Assignment Operator . . . 431
8.9.5.1.1.5 Copy Assignment Operator . . . 432
8.9.5.1.1.6 Destructor . . . 432
8.9.5.1.2 Constructors . . . 433
8.9.5.1.2.1 Variant(in_place_type_t<T>, Args ...&&) . . . 433
8.9.5.1.2.2 Variant(in_place_type_t<T>, std::initializer_list<U>, Args ...&&) . . . 433
8.9.5.1.2.3 Variant(in_place_index_t<I>, Args ...&&) . . . 434
8.9.5.1.2.4 Variant(in_place_index_t<I>, std::initializer_list<U>, Args ...&&) . . . 434
8.9.5.1.2.5 Variant(T&&) . . . 435
8.9.5.1.3 Member Functions . . . 435
8.9.5.1.3.1 emplace(Args ...&&) . . . 435
8.9.5.1.3.2 emplace(std::initializer_list<U>, Args ...&&) . . . 436
8.9.5.1.4 Modifiers . . . 436
8.9.5.1.4.1 emplace(std::initializer_list<U>, Args ...&&) . . . 436
8.9.5.1.4.2 emplace(Args ...&&) . . . 437
8.9.5.1.4.3 swap . . . 437
8.9.5.1.5 Observers . . . 438
8.9.5.1.5.1 index . . . 438
8.9.5.1.5.2 valueless_by_exception . . . 438
8.9.5.1.6 Operator= . . . 438
8.9.5.1.6.1 operator= . . . 438
8.9.6 Struct: variant_alternative . . . 439
8.9.7 Struct: variant_alternative<I, Variant<Types...>> . . . 439

8.9.8 Struct: variant_alternative<I, const T> . . . 440
8.9.9 Struct: variant_alternative<I, const volatile T> . . . 440
8.9.10 Struct: variant_alternative<I, volatile T> . . . 441
8.9.11 Struct: variant_size . . . 441
8.9.12 Struct: variant_size<Variant<Types...>> . . . 442
8.9.13 Struct: variant_size<const T> . . . 442
8.9.14 Struct: variant_size<const volatile T> . . . 443
8.9.15 Struct: variant_size<volatile T> . . . 443
8.9.16 Struct: Monostate> . . . 443
8.9.16.1 Public Member Functions . . . 444
8.9.16.1.1 Member Functions . . . 444
8.9.16.1.1.1 operator() . . . 444
8.9.17 Struct: Variant<Types...>> . . . 444
8.9.17.1 Public Member Functions . . . 445
8.9.17.1.1 Member Functions . . . 445
8.9.17.1.1.1 operator() . . . 445
8.10 Header: ara/core/vector.h . . . 445
8.10.1 Non-Member Functions . . . 446
8.10.1.1 Other . . . 446
8.10.1.1.1 operator!= . . . 446
8.10.1.1.2 operator< . . . 446
8.10.1.1.3 operator<= . . . 447
8.10.1.1.4 operator== . . . 447
8.10.1.1.5 operator> . . . 447
8.10.1.1.6 operator>= . . . 448
8.10.1.1.7 swap . . . 448
8.10.2 Class: Vector . . . 449
8.10.2.1 Public Member Types . . . 449
8.10.2.1.1 Type Alias: allocator_type . . . 449
8.10.2.1.2 Type Alias: const_iterator . . . 449
8.10.2.1.3 Type Alias: const_pointer . . . 450
8.10.2.1.4 Type Alias: const_reference . . . 450
8.10.2.1.5 Type Alias: const_reverse_iterator . . . 451
8.10.2.1.6 Type Alias: difference_type . . . 451
8.10.2.1.7 Type Alias: iterator . . . 451
8.10.2.1.8 Type Alias: pointer . . . 452
8.10.2.1.9 Type Alias: reference . . . 452
8.10.2.1.10 Type Alias: reverse_iterator . . . 452
8.10.2.1.11 Type Alias: size_type . . . 453
8.10.2.1.12 Type Alias: value_type . . . 453
8.10.2.2 Public Member Functions . . . 453
8.10.2.2.1 Special Member Functions . . . 453
8.10.2.2.1.1 Move Constructor . . . 453

8.10.2.2.1.2 Default Constructor . . . 454
8.10.2.2.1.3 Copy Constructor . . . 454
8.10.2.2.1.4 Copy Assignment Operator . . . 455
8.10.2.2.1.5 Move Assignment Operator . . . 455
8.10.2.2.1.6 Destructor . . . 455
8.10.2.2.2 Constructors . . . 456
8.10.2.2.2.1 Vector(Vector&&, const Allocator&) . . . 456
8.10.2.2.2.2 Vector(const Allocator&) . . . 456
8.10.2.2.2.3 Vector(size_type, const Allocator&) . . . 456
8.10.2.2.2.4 Vector(size_type, const T&, const Allocator&) . . . 457
8.10.2.2.2.5 Vector(InputIterator, InputIterator, const Allocator&) . . . 457
8.10.2.2.2.6 Vector(const Vector&, const Allocator&) . . . 457
8.10.2.2.2.7 Vector(std::initializer_list<T>, const Allocator&) . . . 458
8.10.2.2.3 Capacity . . . 458
8.10.2.2.3.1 capacity . . . 458
8.10.2.2.3.2 empty . . . 458
8.10.2.2.3.3 max_size . . . 459
8.10.2.2.3.4 reserve . . . 459
8.10.2.2.3.5 shrink_to_fit . . . 459
8.10.2.2.3.6 size . . . 460
8.10.2.2.4 Element access . . . 460
8.10.2.2.4.1 at(size_type) . . . 460
8.10.2.2.4.2 at(size_type) const . . . 460
8.10.2.2.4.3 back() . . . 461
8.10.2.2.4.4 back() const . . . 461
8.10.2.2.4.5 data() const . . . 461
8.10.2.2.4.6 data() . . . 462
8.10.2.2.4.7 front() . . . 462
8.10.2.2.4.8 front() const . . . 462
8.10.2.2.4.9 operator[](size_type) . . . 463
8.10.2.2.4.10 operator[](size_type) const . . . 463
8.10.2.2.5 Iterators . . . 463
8.10.2.2.5.1 begin() const . . . 463
8.10.2.2.5.2 begin() . . . 464
8.10.2.2.5.3 cbegin . . . 464
8.10.2.2.5.4 cend . . . 464
8.10.2.2.5.5 crbegin . . . 465
8.10.2.2.5.6 crend . . . 465
8.10.2.2.5.7 end() . . . 465
8.10.2.2.5.8 end() const . . . 466
8.10.2.2.5.9 rbegin() const . . . 466
8.10.2.2.5.10 rbegin() . . . 466
8.10.2.2.5.11 rend() const . . . 467

8.10.2.2.5.12 rend() . . . 467
8.10.2.2.6 Modifiers . . . 467
8.10.2.2.6.1 clear . . . 467
8.10.2.2.6.2 emplace . . . 468
8.10.2.2.6.3 emplace_back . . . 468
8.10.2.2.6.4 erase(const_iterator) . . . 468
8.10.2.2.6.5 erase(const_iterator, const_iterator) . . . 469
8.10.2.2.6.6 insert(const_iterator, InputIterator, InputIterator) . . . 469
8.10.2.2.6.7 insert(const_iterator, std::initializer_list<T>) . . . 469
8.10.2.2.6.8 insert(const_iterator, T&&) . . . 470
8.10.2.2.6.9 insert(const_iterator, const T&) . . . 470
8.10.2.2.6.10 insert(const_iterator, size_type, const T&) . . . 470
8.10.2.2.6.11 pop_back . . . 471
8.10.2.2.6.12 push_back(const T&) . . . 471
8.10.2.2.6.13 push_back(T&&) . . . 471
8.10.2.2.6.14 resize(size_type, const T&) . . . 472
8.10.2.2.6.15 resize(size_type) . . . 472
8.10.2.2.6.16 swap . . . 472
8.10.2.2.7 Operator= . . . 473
8.10.2.2.7.1 operator=(std::initializer_list<T>) . . . 473
8.10.2.2.8 Other . . . 473
8.10.2.2.8.1 assign(InputIterator, InputIterator) . . . 473
8.10.2.2.8.2 assign(std::initializer_list<T>) . . . 473
8.10.2.2.8.3 assign(size_type, const T&) . . . 474
8.10.2.2.8.4 get_allocator . . . 474
8.10.3 Struct: hash . . . 474
8.10.3.1 Public Member Functions . . . 475
8.10.3.1.1 Member Functions . . . 475
8.10.3.1.1.1 operator() . . . 475
8.11 Header: ara/core/abort.h . . . 475
8.11.1 Non-Member Types . . . 475
8.11.1.1 Type Alias: AbortHandler . . . 475
8.11.2 Non-Member Functions . . . 476
8.11.2.1 Other . . . 476
8.11.2.1.1 Abort . . . 476
8.11.2.1.2 AbortHandlerPrototype . . . 476
8.11.2.1.3 AddAbortHandler . . . 477
8.11.2.1.4 SetAbortHandler . . . 477
8.12 Header: ara/core/exception.h . . . 478
8.12.1 Class: Exception . . . 478
8.12.1.1 Public Member Functions . . . 478
8.12.1.1.1 Special Member Functions . . . 478
8.12.1.1.1.1 Move Constructor . . . 478

8.12.1.1.1.2 Move Assignment Operator . . . 479
8.12.1.1.1.3 Destructor . . . 479
8.12.1.1.2 Constructors . . . 479
8.12.1.1.2.1 Exception . . . 479
8.12.1.1.3 Member Functions . . . 480
8.12.1.1.3.1 Error . . . 480
8.12.1.1.3.2 what . . . 480
8.12.1.2 Protected Member Functions . . . 481
8.12.1.2.1 Special Member Functions . . . 481
8.12.1.2.1.1 Copy Constructor . . . 481
8.12.1.2.1.2 Copy Assignment Operator . . . 481
8.13 Header: ara/core/error_domain.h . . . 482
8.13.1 Class: ErrorDomain . . . 482
8.13.1.1 Public Member Types . . . 482
8.13.1.1.1 Type Alias: CodeType . . . 482
8.13.1.1.2 Type Alias: IdType . . . 483
8.13.1.1.3 Type Alias: SupportDataType . . . 483
8.13.1.2 Public Member Functions . . . 484
8.13.1.2.1 Special Member Functions . . . 484
8.13.1.2.1.1 Copy Constructor . . . 484
8.13.1.2.1.2 Move Constructor . . . 484
8.13.1.2.1.3 Move Assignment Operator . . . 485
8.13.1.2.1.4 Copy Assignment Operator . . . 485
8.13.1.2.2 Member Functions . . . 485
8.13.1.2.2.1 Id . . . 485
8.13.1.2.2.2 Message . . . 486
8.13.1.2.2.3 Name . . . 486
8.13.1.2.2.4 ThrowAsException . . . 487
8.13.1.2.2.5 operator!= . . . 487
8.13.1.2.2.6 operator== . . . 488
8.13.1.3 Protected Member Functions . . . 489
8.13.1.3.1 Special Member Functions . . . 489
8.13.1.3.1.1 Destructor . . . 489
8.13.1.3.2 Constructors . . . 489
8.13.1.3.2.1 ErrorDomain . . . 489
8.14 Header: ara/core/error_code.h . . . 490
8.14.1 Non-Member Functions . . . 490
8.14.1.1 Other . . . 490
8.14.1.1.1 operator!= . . . 490
8.14.1.1.2 operator== . . . 490
8.14.2 Class: ErrorCode . . . 491
8.14.2.1 Public Member Functions . . . 491
8.14.2.1.1 Constructors . . . 491

8.14.2.1.1.1 ErrorCode . . . 491
8.14.2.1.1.2 ErrorCode . . . 492
8.14.2.1.2 Member Functions . . . 492
8.14.2.1.2.1 Domain . . . 492
8.14.2.1.2.2 Message . . . 493
8.14.2.1.2.3 SupportData . . . 493
8.14.2.1.2.4 ThrowAsException . . . 494
8.14.2.1.2.5 Value . . . 494
8.15 Header: ara/core/core_error_domain.h . . . 495
8.15.1 Non-Member Types . . . 495
8.15.1.1 Enumeration: CoreErrc . . . 495
8.15.2 Non-Member Functions . . . 495
8.15.2.1 Other . . . 495
8.15.2.1.1 GetCoreErrorDomain . . . 495
8.15.2.1.2 MakeErrorCode . . . 496
8.15.3 Class: CoreErrorDomain . . . 496
8.15.3.1 Public Member Types . . . 497
8.15.3.1.1 Type Alias: Errc . . . 497
8.15.3.1.2 Type Alias: Exception . . . 497
8.15.3.2 Public Member Functions . . . 498
8.15.3.2.1 Special Member Functions . . . 498
8.15.3.2.1.1 Default Constructor . . . 498
8.15.3.2.2 Member Functions . . . 498
8.15.3.2.2.1 Message . . . 498
8.15.3.2.2.2 Name . . . 499
8.15.3.2.2.3 ThrowAsException . . . 499
8.15.4 Class: CoreException . . . 500
8.15.4.1 Public Member Functions . . . 500
8.15.4.1.1 Constructors . . . 500
8.15.4.1.1.1 CoreException . . . 500
8.16 Header: ara/core/future.h . . . 501
8.16.1 Non-Member Types . . . 501
8.16.1.1 Enumeration: FutureStatus . . . 501
8.16.2 Class: Future . . . 501
8.16.2.1 Public Member Functions . . . 502
8.16.2.1.1 Special Member Functions . . . 502
8.16.2.1.1.1 Copy Constructor . . . 502
8.16.2.1.1.2 Default Constructor . . . 502
8.16.2.1.1.3 Move Constructor . . . 503
8.16.2.1.1.4 Copy Assignment Operator . . . 503
8.16.2.1.1.5 Move Assignment Operator . . . 503
8.16.2.1.1.6 Destructor . . . 504
8.16.2.1.2 Member Functions . . . 504

8.16.2.1.2.1 GetResult . . . 504
8.16.2.1.2.2 get . . . 505
8.16.2.1.2.3 is_ready . . . 506
8.16.2.1.2.4 then(F&&, ExecutorT&&) . . . 506
8.16.2.1.2.5 then(F&&) . . . 508
8.16.2.1.2.6 valid . . . 509
8.16.2.1.2.7 wait . . . 509
8.16.2.1.2.8 wait_for . . . 510
8.16.2.1.2.9 wait_until . . . 510
8.16.3 Class: Future<void, E> . . . 511
8.16.3.1 Public Member Functions . . . 511
8.16.3.1.1 Special Member Functions . . . 511
8.16.3.1.1.1 Copy Constructor . . . 511
8.16.3.1.1.2 Default Constructor . . . 512
8.16.3.1.1.3 Move Constructor . . . 512
8.16.3.1.1.4 Copy Assignment Operator . . . 513
8.16.3.1.1.5 Move Assignment Operator . . . 513
8.16.3.1.1.6 Destructor . . . 514
8.16.3.1.2 Member Functions . . . 514
8.16.3.1.2.1 GetResult . . . 514
8.16.3.1.2.2 get . . . 515
8.16.3.1.2.3 is_ready . . . 516
8.16.3.1.2.4 then(F&&, ExecutorT&&) . . . 516
8.16.3.1.2.5 then(F&&) . . . 517
8.16.3.1.2.6 valid . . . 518
8.16.3.1.2.7 wait . . . 519
8.16.3.1.2.8 wait_for . . . 519
8.16.3.1.2.9 wait_until . . . 520
8.16.4 Class: Promise . . . 520
8.16.4.1 Public Member Functions . . . 521
8.16.4.1.1 Special Member Functions . . . 521
8.16.4.1.1.1 Copy Constructor . . . 521
8.16.4.1.1.2 Default Constructor . . . 521
8.16.4.1.1.3 Move Constructor . . . 521
8.16.4.1.1.4 Copy Assignment Operator . . . 522
8.16.4.1.1.5 Move Assignment Operator . . . 522
8.16.4.1.1.6 Destructor . . . 523
8.16.4.1.2 Member Functions . . . 523
8.16.4.1.2.1 SetError(E&&) . . . 523
8.16.4.1.2.2 SetError(const E&) . . . 524
8.16.4.1.2.3 SetResult(Result<T, E>&&) . . . 524
8.16.4.1.2.4 SetResult(const Result<T, E>&) . . . 525
8.16.4.1.2.5 get_future . . . 525

8.16.4.1.2.6 set_value(const T&) . . . 526
8.16.4.1.2.7 set_value(T&&) . . . 526
8.16.4.1.2.8 swap . . . 527
8.16.5 Class: Promise<void, E> . . . 527
8.16.5.1 Public Member Functions . . . 528
8.16.5.1.1 Special Member Functions . . . 528
8.16.5.1.1.1 Move Constructor . . . 528
8.16.5.1.1.2 Default Constructor . . . 528
8.16.5.1.1.3 Copy Constructor . . . 529
8.16.5.1.1.4 Copy Assignment Operator . . . 529
8.16.5.1.1.5 Move Assignment Operator . . . 530
8.16.5.1.1.6 Destructor . . . 530
8.16.5.1.2 Member Functions . . . 531
8.16.5.1.2.1 SetError(E&&) . . . 531
8.16.5.1.2.2 SetError(const E&) . . . 531
8.16.5.1.2.3 SetResult(Result<void, E>&&) . . . 532
8.16.5.1.2.4 SetResult(const Result<void, E>&) . . . 533
8.16.5.1.2.5 get_future . . . 533
8.16.5.1.2.6 set_value() . . . 534
8.16.5.1.2.7 swap . . . 534
8.17 Header: ara/core/future_error_domain.h . . . 535
8.17.1 Non-Member Types . . . 535
8.17.1.1 Enumeration: FutureErrc . . . 535
8.17.2 Non-Member Functions . . . 536
8.17.2.1 Other . . . 536
8.17.2.1.1 GetFutureErrorDomain . . . 536
8.17.2.1.2 MakeErrorCode . . . 536
8.17.3 Class: FutureErrorDomain . . . 537
8.17.3.1 Public Member Types . . . 537
8.17.3.1.1 Type Alias: Errc . . . 537
8.17.3.1.2 Type Alias: Exception . . . 538
8.17.3.2 Public Member Functions . . . 538
8.17.3.2.1 Special Member Functions . . . 538
8.17.3.2.1.1 Default Constructor . . . 538
8.17.3.2.2 Member Functions . . . 539
8.17.3.2.2.1 Message . . . 539
8.17.3.2.2.2 Name . . . 539
8.17.3.2.2.3 ThrowAsException . . . 540
8.17.4 Class: FutureException . . . 540
8.17.4.1 Public Member Functions . . . 541
8.17.4.1.1 Constructors . . . 541
8.17.4.1.1.1 FutureException . . . 541
8.18 Header: ara/core/steady_clock.h . . . 541

8.18.1 Class: SteadyClock . . . 541
8.18.1.1 Public Member Types . . . 542
8.18.1.1.1 Type Alias: duration . . . 542
8.18.1.1.2 Type Alias: period . . . 542
8.18.1.1.3 Type Alias: rep . . . 542
8.18.1.1.4 Type Alias: time_point . . . 543
8.18.1.2 Public Member Variables . . . 543
8.18.1.2.1 is_steady . . . 543
8.18.1.3 Public Member Functions . . . 544
8.18.1.3.1 Member Functions . . . 544
8.18.1.3.1.1 now . . . 544
8.19 Header: ara/core/utility.h . . . 544
8.19.1 Non-Member Types . . . 544
8.19.1.1 Type Alias: Byte . . . 544
8.19.2 Global Variables . . . 545
8.19.2.1 in_place . . . 545
8.19.2.2 in_place_index . . . 545
8.19.2.3 in_place_type . . . 546
8.19.3 Non-Member Functions . . . 546
8.19.3.1 Other . . . 546
8.19.3.1.1 data(T()&) . . . 546
8.19.3.1.2 data(std::initializer_list<E>) . . . 547
8.19.3.1.3 data(Container&) . . . 547
8.19.3.1.4 data(const Container&) . . . 548
8.19.3.1.5 empty(std::initializer_list<E>) . . . 548
8.19.3.1.6 empty(const T()&) . . . 549
8.19.3.1.7 empty(const Container&) . . . 549
8.19.3.1.8 size(const T()&) . . . 550
8.19.3.1.9 size(const Container&) . . . 550
8.19.4 Struct: in_place_index_t . . . 551
8.19.4.1 Public Member Functions . . . 551
8.19.4.1.1 Special Member Functions . . . 551
8.19.4.1.1.1 Default Constructor . . . 551
8.19.5 Struct: in_place_t . . . 552
8.19.5.1 Public Member Functions . . . 552
8.19.5.1.1 Special Member Functions . . . 552
8.19.5.1.1.1 Default Constructor . . . 552
8.19.6 Struct: in_place_type_t . . . 553
8.19.6.1 Public Member Functions . . . 553
8.19.6.1.1 Special Member Functions . . . 553
8.19.6.1.1.1 Default Constructor . . . 553
8.19.7 Struct: is_nothrow_swappable . . . 554
8.20 Header: ara/core/executor.h . . . 554

8.20.1 Class: Executor . . . 554
8.20.1.1 Public Member Functions . . . 555
8.20.1.1.1 Special Member Functions . . . 555
8.20.1.1.1.1 Copy Constructor . . . 555
8.20.1.1.1.2 Move Constructor . . . 555
8.20.1.1.1.3 Copy Assignment Operator . . . 556
8.20.1.1.1.4 Move Assignment Operator . . . 556
8.20.1.1.1.5 Destructor . . . 557
8.20.1.1.2 Member Functions . . . 557
8.20.1.1.2.1 execute . . . 557
8.20.1.1.2.2 oneway_execute . . . 558
8.21 Header: ara/core/initialization.h . . . 558
8.21.1 Non-Member Functions . . . 558
8.21.1.1 Other . . . 558
8.21.1.1.1 Deinitialize . . . 558
8.21.1.1.2 Initialize(int&, char **) . . . 559
8.22 Header: ara/core/instance_specifier.h . . . 560
8.22.1 Non-Member Functions . . . 560
8.22.1.1 Other . . . 560
8.22.1.1.1 operator!= . . . 560
8.22.1.1.2 operator== . . . 560
8.22.2 Class: InstanceSpecifier . . . 561
8.22.2.1 Public Member Functions . . . 561
8.22.2.1.1 Special Member Functions . . . 561
8.22.2.1.1.1 Move Constructor . . . 561
8.22.2.1.1.2 Copy Constructor . . . 562
8.22.2.1.1.3 operator=(const InstanceSpecifier&) . . . 562
8.22.2.1.1.4 operator=(InstanceSpecifier&&) . . . 563
8.22.2.1.1.5 Destructor . . . 563
8.22.2.1.2 Constructors . . . 564
8.22.2.1.2.1 InstanceSpecifier . . . 564
8.22.2.1.3 Member Functions . . . 564
8.22.2.1.3.1 ToString . . . 564
8.22.2.1.3.2 operator!=(ara::core::StringView) const . . . 565
8.22.2.1.3.3 operator!=(const InstanceSpecifier&) const . . . 565
8.22.2.1.3.4 operator< . . . 566
8.22.2.1.3.5 operator==(const InstanceSpecifier&) const . . . 566
8.22.2.1.3.6 operator==(ara::core::StringView) const . . . 567
8.22.2.1.4 Named Constructors . . . 567
8.22.2.1.4.1 Create . . . 567
8.23 Header: ara/core/result.h . . . 568
8.23.1 Non-Member Functions . . . 568
8.23.1.1 Other . . . 568

8.23.1.1.1 operator!=(const Result<T, E>&, const E&) . . . 568
8.23.1.1.2 operator!=(const Result<T, E>&, const Result<T, E>&) . . . 568
8.23.1.1.3 operator!=(const T&, const Result<T, E>&) . . . 569
8.23.1.1.4 operator!=(const E&, const Result<T, E>&) . . . 570
8.23.1.1.5 operator!=(const Result<T, E>&, const T&) . . . 570
8.23.1.1.6 operator==(const Result<T, E>&, const T&) . . . 571
8.23.1.1.7 operator==(const Result<T, E>&, const E&) . . . 571
8.23.1.1.8 operator==(const E&, const Result<T, E>&) . . . 572
8.23.1.1.9 operator==(const Result<T, E>&, const Result<T, E>&) . . . 572
8.23.1.1.10 operator==(const T&, const Result<T, E>&) . . . 573
8.23.1.1.11 swap . . . 573
8.23.2 Class: Result . . . 574
8.23.2.1 Public Member Types . . . 574
8.23.2.1.1 Type Alias: error_type . . . 574
8.23.2.1.2 Type Alias: value_type . . . 575
8.23.2.2 Public Member Functions . . . 575
8.23.2.2.1 Special Member Functions . . . 575
8.23.2.2.1.1 Copy Constructor . . . 575
8.23.2.2.1.2 Move Constructor . . . 576
8.23.2.2.1.3 Move Assignment Operator . . . 576
8.23.2.2.1.4 Copy Assignment Operator . . . 577
8.23.2.2.1.5 Destructor . . . 577
8.23.2.2.2 Constructors . . . 578
8.23.2.2.2.1 Result(E&&) . . . 578
8.23.2.2.2.2 Result(T&&) . . . 578
8.23.2.2.2.3 Result(const T&) . . . 579
8.23.2.2.2.4 Result(const E&) . . . 579
8.23.2.2.3 Member Functions . . . 580
8.23.2.2.3.1 Bind . . . 580
8.23.2.2.3.2 CheckError . . . 580
8.23.2.2.3.3 EmplaceError . . . 581
8.23.2.2.3.4 EmplaceValue . . . 581
8.23.2.2.3.5 Err() && . . . 582
8.23.2.2.3.6 Err() const & . . . 582
8.23.2.2.3.7 Error() && . . . 583
8.23.2.2.3.8 Error() & . . . 583
8.23.2.2.3.9 Error() const & . . . 584
8.23.2.2.3.10 ErrorOr(G&&) && . . . 584
8.23.2.2.3.11 ErrorOr(G&&) const & . . . 585
8.23.2.2.3.12 HasValue . . . 585
8.23.2.2.3.13 Ok() const & . . . 586
8.23.2.2.3.14 Ok() && . . . 586
8.23.2.2.3.15 Resolve . . . 587

8.23.2.2.3.16 Swap . . . 587
8.23.2.2.3.17 Value() && . . . 588
8.23.2.2.3.18 Value() & . . . 588
8.23.2.2.3.19 Value() const & . . . 589
8.23.2.2.3.20 ValueOr(U&&) const & . . . 589
8.23.2.2.3.21 ValueOr(U&&) && . . . 590
8.23.2.2.3.22 ValueOrThrow() & . . . 590
8.23.2.2.3.23 ValueOrThrow() const & . . . 591
8.23.2.2.3.24 ValueOrThrow() && . . . 591
8.23.2.2.3.25 operator bool . . . 592
8.23.2.2.3.26 operator*() & . . . 592
8.23.2.2.3.27 operator*() && . . . 593
8.23.2.2.3.28 operator*() const & . . . 593
8.23.2.2.3.29 operator->() const . . . 594
8.23.2.2.3.30 operator->() . . . 594
8.23.2.2.4 Named Constructors . . . 595
8.23.2.2.4.1 FromError(Args ...&&) . . . 595
8.23.2.2.4.2 FromError(const E&) . . . 595
8.23.2.2.4.3 FromError(E&&) . . . 596
8.23.2.2.4.4 FromValue(Args ...&&) . . . 596
8.23.2.2.4.5 FromValue(T&&) . . . 597
8.23.2.2.4.6 FromValue(const T&) . . . 597
8.23.3 Class: Result<T &, E> . . . 598
8.23.3.1 Public Member Types . . . 598
8.23.3.1.1 Type Alias: error_type . . . 598
8.23.3.1.2 Type Alias: value_type . . . 599
8.23.3.2 Public Member Functions . . . 599
8.23.3.2.1 Special Member Functions . . . 599
8.23.3.2.1.1 Move Constructor . . . 599
8.23.3.2.1.2 Copy Constructor . . . 600
8.23.3.2.1.3 Copy Assignment Operator . . . 600
8.23.3.2.1.4 Move Assignment Operator . . . 601
8.23.3.2.1.5 Destructor . . . 601
8.23.3.2.2 Constructors . . . 602
8.23.3.2.2.1 Result(const E&) . . . 602
8.23.3.2.2.2 Result(E&&) . . . 602
8.23.3.2.2.3 Result . . . 603
8.23.3.2.3 Member Functions . . . 603
8.23.3.2.3.1 Bind . . . 603
8.23.3.2.3.2 CheckError . . . 604
8.23.3.2.3.3 EmplaceError . . . 605
8.23.3.2.3.4 Err() const & . . . 605
8.23.3.2.3.5 Err() && . . . 606

8.23.3.2.3.6 Error() & . . . 606
8.23.3.2.3.7 Error() && . . . 607
8.23.3.2.3.8 Error() const & . . . 607
8.23.3.2.3.9 ErrorOr(G&&) && . . . 608
8.23.3.2.3.10 ErrorOr(G&&) const & . . . 608
8.23.3.2.3.11 HasValue . . . 609
8.23.3.2.3.12 Ok() const . . . 609
8.23.3.2.3.13 Ok() . . . 610
8.23.3.2.3.14 Resolve(F&&) const . . . 610
8.23.3.2.3.15 Resolve(F&&) const . . . 611
8.23.3.2.3.16 Swap . . . 611
8.23.3.2.3.17 Value() & . . . 612
8.23.3.2.3.18 Value() const & . . . 612
8.23.3.2.3.19 ValueOr(U&&) && . . . 613
8.23.3.2.3.20 ValueOr(U&&) const & . . . 613
8.23.3.2.3.21 ValueOrThrow() const . . . 614
8.23.3.2.3.22 ValueOrThrow() . . . 614
8.23.3.2.3.23 operator bool . . . 615
8.23.3.2.3.24 operator*() & . . . 615
8.23.3.2.3.25 operator*() const & . . . 616
8.23.3.2.3.26 operator->() const . . . 616
8.23.3.2.3.27 operator->() . . . 617
8.23.3.2.4 Named Constructors . . . 617
8.23.3.2.4.1 FromError . . . 617
8.23.3.2.4.2 FromError(const E&) . . . 618
8.23.3.2.4.3 FromError(E&&) . . . 618
8.23.3.2.4.4 FromValue . . . 619
8.23.4 Class: Result<void, E> . . . 619
8.23.4.1 Public Member Types . . . 620
8.23.4.1.1 Type Alias: error_type . . . 620
8.23.4.1.2 Type Alias: value_type . . . 620
8.23.4.2 Public Member Functions . . . 621
8.23.4.2.1 Special Member Functions . . . 621
8.23.4.2.1.1 Copy Constructor . . . 621
8.23.4.2.1.2 Move Constructor . . . 621
8.23.4.2.1.3 Default Constructor . . . 622
8.23.4.2.1.4 Move Assignment Operator . . . 622
8.23.4.2.1.5 Copy Assignment Operator . . . 623
8.23.4.2.1.6 Destructor . . . 623
8.23.4.2.2 Constructors . . . 624
8.23.4.2.2.1 Result(E&&) . . . 624
8.23.4.2.2.2 Result(const E&) . . . 624
8.23.4.2.3 Member Functions . . . 625

8.23.4.2.3.1 Bind . . . 625
8.23.4.2.3.2 CheckError . . . 625
8.23.4.2.3.3 EmplaceError . . . 626
8.23.4.2.3.4 EmplaceValue . . . 626
8.23.4.2.3.5 Err() const & . . . 627
8.23.4.2.3.6 Err() && . . . 627
8.23.4.2.3.7 Error() && . . . 628
8.23.4.2.3.8 Error() & . . . 628
8.23.4.2.3.9 Error() const & . . . 629
8.23.4.2.3.10 ErrorOr(G&&) && . . . 629
8.23.4.2.3.11 ErrorOr(G&&) const & . . . 630
8.23.4.2.3.12 HasValue . . . 630
8.23.4.2.3.13 Resolve . . . 631
8.23.4.2.3.14 Swap . . . 631
8.23.4.2.3.15 Value . . . 632
8.23.4.2.3.16 ValueOr . . . 632
8.23.4.2.3.17 ValueOrThrow() const . . . 633
8.23.4.2.3.18 operator bool . . . 633
8.23.4.2.3.19 operator*() const . . . 634
8.23.4.2.4 Named Constructors . . . 634
8.23.4.2.4.1 FromError . . . 634
8.23.4.2.4.2 FromError(const E&) . . . 635
8.23.4.2.4.3 FromError(E&&) . . . 635
8.23.4.2.4.4 FromValue . . . 636
9 Service Interfaces 637
10 Configuration 638
10.1 Default Values . . . 638
10.2 Semantic Constraints . . . 638
A Mentioned Manifest Elements 639
B Demands and constraints on Base Software (normative) 646
C Platform Extension Interfaces (normative) 647
D Implementation-defined and undefined behavior (informative) 648
D.1 Implementation-defined behavior (IDB) . . . 648
D.2 Undefined behavior (UB) . . . 649
E Not implemented requirements 651
F Change History 652
F.1 Change History of this document according to AUTOSAR Release R19-11 652
F.1.1 Added Specification Items in R19-11 . . . 652
F.1.2 Changed Specification Items in R19-11 . . . 659
F.1.3 Deleted Specification Items in R19-11 . . . 659

F.2 Change History of this document according to AUTOSAR Release R20-11 660
F.2.1 Added Specification Items in R20-11 660
F.2.2 Changed Specification Items in R20-11 663
F.2.3 Deleted Specification Items in R20-11 671
F.3 Change History of this document according to AUTOSAR Release R21-11 671
F.3.1 Added Specification Items in R21-11 671
F.3.2 Changed Specification Items in R21-11 673
F.3.3 Deleted Specification Items in R21-11 683
F.4 Change History of this document according to AUTOSAR Release R22-11 684
F.4.1 Added Specification Items in R22-11 684
F.4.2 Changed Specification Items in R22-11 684
F.4.3 Deleted Specification Items in R22-11 687
F.5 Change History of this document according to AUTOSAR Release R23-11 687
F.5.1 Added Specification Items in R23-11 687
F.5.2 Changed Specification Items in R23-11 689
F.5.3 Deleted Specification Items in R23-11 692
F.6 Change History of this document according to AUTOSAR Release R24-11 693
F.6.1 Added Specification Items in R24-11 693
F.6.2 Changed Specification Items in R24-11 703
F.6.3 Deleted Specification Items in R24-11 713
F.7 Change History of this document according to AUTOSAR Release R25-11 713
F.7.1 Added Specification Items in R25-11 713
F.7.2 Changed Specification Items in R25-11 727
F.7.3 Deleted Specification Items in R25-11 740

1 Introduction and functional overview

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

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

2 Acronyms and Abbreviations

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

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

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

3 Related documentation

3.1 Input documents & related standards and norms

[1] Glossary
AUTOSAR_FO_TR_Glossary

[2] Explanation of Adaptive Platform Software Architecture
AUTOSAR_AP_EXP_SWArchitecture

[3] General Requirements specific to Adaptive Platform
AUTOSAR_AP_RS_General

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

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

[6] Explanation of Adaptive Platform Design
AUTOSAR_AP_EXP_PlatformDesign

[7] Specification of Log and Trace
AUTOSAR_AP_SWS_LogAndTrace

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

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

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

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

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

[13] Specification of Execution Management
AUTOSAR_AP_SWS_ExecutionManagement

[14] Explanation of ara::com API
AUTOSAR_AP_EXP_ARAComAPI

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

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

[17] Specification of Manifest
AUTOSAR_AP_TPS_ManifestSpecification

4 Constraints and assumptions

4.1 Known limitations

- The specification of some data types (Array, Map, Optional, String, StringView, Variant) mentions “supporting constructs”, but lacks a precise scope definition of this term.
- The specification of some data types (Map, Vector, String) is lacking a comprehensive definition of memory allocation behavior; it currently only describes it as “implementation-defined”. The interacton with ara::core::MemoryResource and derived types is not defined.
- Chapter 7 (“Functional specification”) describes some behavior informally that should rather be given as specification items.
- Some parts of the chapter 8 (“API specification”) (SynchronizedPoolResource, UnsynchronizedPoolResource, Executor, String, some constructors) have not been assigned a thread safety category as specified in the chapter 7.1.9 (“Thread safety”).

5 Dependencies to other Functional Clusters

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

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

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

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

5.1 Provided Interfaces

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

Interface	*Functional Cluster*	*Purpose*
No provided interfaces		

Table 5.1: Interfaces provided to other Functional Clusters

5.2 Required Interfaces

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

Functional Cluster	*Interface*	*Purpose*
No required interfaces		

Table 5.2: Interfaces required from other Functional Clusters

5.3 Functional Cluster initialization

`ara::core::Initialize` and `ara::core::Deinitialize` initialize and 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 de-initialization.

6 Requirements Tracing

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

Requirement	Description	Satisfied by
[RS_AP_00111]	Source Code Portability Support	[SWS_CORE_00055] [SWS_CORE_15005] [SWS_CORE_90001] [SWS_CORE_90002] [SWS_CORE_90003] [SWS_CORE_90004] [SWS_CORE_90005] [SWS_CORE_90007] [SWS_CORE_90022]
[RS_AP_00115]	Public namespaces	[SWS_CORE_90025]
[RS_AP_00116]	Header file name	[SWS_CORE_90001]
[RS_AP_00119]	Return values / application errors	[SWS_CORE_10301] [SWS_CORE_10302] [SWS_CORE_10303] [SWS_CORE_10304] [SWS_CORE_10401] [SWS_CORE_10600]
[RS_AP_00120]	Method and Function names	[SWS_CORE_08041] [SWS_CORE_08042]
[RS_AP_00127]	Usage of ara::core types	[SWS_CORE_00052]
[RS_AP_00128]	Error reporting	[SWS_CORE_00002] [SWS_CORE_10600] [SWS_CORE_10800]
[RS_AP_00130]	AUTOSAR Adaptive Platform shall represent a rich and modern programming environment	[SWS_CORE_00008] [SWS_CORE_00009] [SWS_CORE_00010] [SWS_CORE_00011] [SWS_CORE_00013] [SWS_CORE_00014] [SWS_CORE_00015] [SWS_CORE_00016] [SWS_CORE_00017] [SWS_CORE_00018] [SWS_CORE_00019] [SWS_CORE_00024] [SWS_CORE_00025] [SWS_CORE_00026] [SWS_CORE_00029] [SWS_CORE_00040] [SWS_CORE_00045] [SWS_CORE_00048] [SWS_CORE_00049] [SWS_CORE_00069] [SWS_CORE_00070] [SWS_CORE_00094] [SWS_CORE_00095] [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] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SWS_CORE_00442] [SWS_CORE_00443] [SWS_CORE_00444] [SWS_CORE_00480] [SWS_CORE_00490] [SWS_CORE_00501] [SWS_CORE_00512] [SWS_CORE_00513] [SWS_CORE_00514] [SWS_CORE_00515] [SWS_CORE_00516] [SWS_CORE_00518] [SWS_CORE_00519] [SWS_CORE_00571] [SWS_CORE_00572] [SWS_CORE_00601] [SWS_CORE_00611] [SWS_CORE_00612] [SWS_CORE_00613] [SWS_CORE_00614] [SWS_CORE_00615] [SWS_CORE_00616] [SWS_CORE_00617] [SWS_CORE_00618] [SWS_CORE_00701] [SWS_CORE_00711] [SWS_CORE_00712] [SWS_CORE_00721] [SWS_CORE_00722] [SWS_CORE_00723] [SWS_CORE_00724] [SWS_CORE_00725] [SWS_CORE_00726] [SWS_CORE_00727] [SWS_CORE_00731] [SWS_CORE_00732] [SWS_CORE_00733] [SWS_CORE_00734] [SWS_CORE_00735] [SWS_CORE_00736] [SWS_CORE_00741] [SWS_CORE_00742] [SWS_CORE_00743] [SWS_CORE_00744] [SWS_CORE_00745] [SWS_CORE_00751] [SWS_CORE_00752] [SWS_CORE_00753] [SWS_CORE_00754] [SWS_CORE_00755] [SWS_CORE_00756] [SWS_CORE_00757] [SWS_CORE_00758] [SWS_CORE_00759] [SWS_CORE_00761] [SWS_CORE_00762] [SWS_CORE_00763] [SWS_CORE_00764] [SWS_CORE_00765] [SWS_CORE_00766] [SWS_CORE_00767] [SWS_CORE_00768] [SWS_CORE_00769] [SWS_CORE_00770] [SWS_CORE_00771] [SWS_CORE_00772] [SWS_CORE_00773] [SWS_CORE_00774] [SWS_CORE_00775] [SWS_CORE_00776] [SWS_CORE_00777] [SWS_CORE_00778] [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_00901] [SWS_CORE_00902] [SWS_CORE_00903] [SWS_CORE_00904] [SWS_CORE_00905] [SWS_CORE_00906] [SWS_CORE_00907] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SWS_CORE_00908] [SWS_CORE_00909] [SWS_CORE_00910] [SWS_CORE_00911] [SWS_CORE_00912] [SWS_CORE_00913] [SWS_CORE_00914] [SWS_CORE_00915] [SWS_CORE_00916] [SWS_CORE_00917] [SWS_CORE_00918] [SWS_CORE_00919] [SWS_CORE_00920] [SWS_CORE_00921] [SWS_CORE_00922] [SWS_CORE_00923] [SWS_CORE_00924] [SWS_CORE_00925] [SWS_CORE_00926] [SWS_CORE_00927] [SWS_CORE_00928] [SWS_CORE_00929] [SWS_CORE_00930] [SWS_CORE_00931] [SWS_CORE_00932] [SWS_CORE_00933] [SWS_CORE_00934] [SWS_CORE_00935] [SWS_CORE_00936] [SWS_CORE_00937] [SWS_CORE_00938] [SWS_CORE_00939] [SWS_CORE_00940] [SWS_CORE_00941] [SWS_CORE_00949] [SWS_CORE_01032] [SWS_CORE_01034] [SWS_CORE_01141] [SWS_CORE_01200] [SWS_CORE_01300] [SWS_CORE_01401] [SWS_CORE_01601] [SWS_CORE_01672] [SWS_CORE_01700] [SWS_CORE_01706] [SWS_CORE_01707] [SWS_CORE_01712] [SWS_CORE_01714] [SWS_CORE_01715] [SWS_CORE_01716] [SWS_CORE_01717] [SWS_CORE_01718] [SWS_CORE_01736] [SWS_CORE_01737] [SWS_CORE_01738] [SWS_CORE_01739] [SWS_CORE_01902] [SWS_CORE_01990] [SWS_CORE_01991] [SWS_CORE_01992] [SWS_CORE_01993] [SWS_CORE_01994] [SWS_CORE_02002] [SWS_CORE_03002] [SWS_CORE_04011] [SWS_CORE_04012] [SWS_CORE_04013] [SWS_CORE_04021] [SWS_CORE_04022] [SWS_CORE_04023] [SWS_CORE_04031] [SWS_CORE_04032] [SWS_CORE_04033] [SWS_CORE_04110] [SWS_CORE_04111] [SWS_CORE_04112] [SWS_CORE_04113] [SWS_CORE_04120] [SWS_CORE_04121] [SWS_CORE_04130] [SWS_CORE_04131] [SWS_CORE_04132] [SWS_CORE_04200] [SWS_CORE_05200] [SWS_CORE_05211] [SWS_CORE_05212] [SWS_CORE_05221] [SWS_CORE_05231] [SWS_CORE_05232] [SWS_CORE_05241] [SWS_CORE_05242] [SWS_CORE_05243] [SWS_CORE_05244] [SWS_CORE_05280] [SWS_CORE_05290] [SWS_CORE_06221] [SWS_CORE_06222] [SWS_CORE_06223] [SWS_CORE_06225] [SWS_CORE_06226] [SWS_CORE_06227] [SWS_CORE_06228] [SWS_CORE_06229] [SWS_CORE_06230] [SWS_CORE_06231] [SWS_CORE_06232] [SWS_CORE_06233] [SWS_CORE_06234] [SWS_CORE_06235] [SWS_CORE_06236] [SWS_CORE_06237] [SWS_CORE_06340] [SWS_CORE_06341] [SWS_CORE_06342] [SWS_CORE_06343] [SWS_CORE_06344] [SWS_CORE_06345] [SWS_CORE_06349] [SWS_CORE_06350] [SWS_CORE_06351] [SWS_CORE_06352] [SWS_CORE_06353] [SWS_CORE_06354] [SWS_CORE_06355] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SWS_CORE_06356] [SWS_CORE_06401] [SWS_CORE_06411] [SWS_CORE_06412] [SWS_CORE_06413] [SWS_CORE_06414] [SWS_CORE_06431] [SWS_CORE_06432] [SWS_CORE_06500] [SWS_CORE_06502] [SWS_CORE_06507] [SWS_CORE_06520] [SWS_CORE_06540] [SWS_CORE_06560] [SWS_CORE_06561] [SWS_CORE_06562] [SWS_CORE_06563] [SWS_CORE_06564] [SWS_CORE_06565] [SWS_CORE_06578] [SWS_CORE_08043] [SWS_CORE_08044] [SWS_CORE_08045] [SWS_CORE_08046] [SWS_CORE_08081] [SWS_CORE_08082] [SWS_CORE_10100] [SWS_CORE_10101] [SWS_CORE_10102] [SWS_CORE_10103] [SWS_CORE_10104] [SWS_CORE_10105] [SWS_CORE_10106] [SWS_CORE_10107] [SWS_CORE_10108] [SWS_CORE_10109] [SWS_CORE_10110] [SWS_CORE_10200] [SWS_CORE_10201] [SWS_CORE_10202] [SWS_CORE_10203] [SWS_CORE_10300] [SWS_CORE_10400] [SWS_CORE_10900] [SWS_CORE_10901] [SWS_CORE_10902] [SWS_CORE_10903] [SWS_CORE_10910] [SWS_CORE_10911] [SWS_CORE_10912] [SWS_CORE_10930] [SWS_CORE_10931] [SWS_CORE_10932] [SWS_CORE_10933] [SWS_CORE_10934] [SWS_CORE_10950] [SWS_CORE_10951] [SWS_CORE_10953] [SWS_CORE_10980] [SWS_CORE_10981] [SWS_CORE_10982] [SWS_CORE_10990] [SWS_CORE_10991] [SWS_CORE_10999] [SWS_CORE_11800] [SWS_CORE_11801] [SWS_CORE_12403] [SWS_CORE_12404] [SWS_CORE_12405] [SWS_CORE_12406] [SWS_CORE_12407] [SWS_CORE_12408] [SWS_CORE_12409] [SWS_CORE_13206] [SWS_CORE_90023] [SWS_CORE_90024]
[RS_AP_00134]	noexcept behavior of class destructors	[SWS_CORE_08029]
[RS_AP_00136]	Usage of string types	[SWS_CORE_00052] [SWS_CORE_08032]
[RS_AP_00137]	Connecting run-time interface with model	[SWS_CORE_08032]
[RS_AP_00138]	Return type of asynchronous function calls	[SWS_CORE_10800]
[RS_AP_00139]	Return type of synchronous function calls	[SWS_CORE_00002]
[RS_AP_00140]	Usage of "final specifier"	[SWS_CORE_00501] [SWS_CORE_08001] [SWS_CORE_10932]

▽

△

Requirement	Description	Satisfied by
[RS_AP_00142]	Handling of unsuccessful operations	[SWS_CORE_00002] [SWS_CORE_00003] [SWS_CORE_00004] [SWS_CORE_00005] [SWS_CORE_00006] [SWS_CORE_00007] [SWS_CORE_00020] [SWS_CORE_00021] [SWS_CORE_00022] [SWS_CORE_00023] [SWS_CORE_00090] [SWS_CORE_00092] [SWS_CORE_10600] [SWS_CORE_13000] [SWS_CORE_13001] [SWS_CORE_13003] [SWS_CORE_13004] [SWS_CORE_13005] [SWS_CORE_13006] [SWS_CORE_13007] [SWS_CORE_13008] [SWS_CORE_13009] [SWS_CORE_13010] [SWS_CORE_13011] [SWS_CORE_13012] [SWS_CORE_13013] [SWS_CORE_13014] [SWS_CORE_13015] [SWS_CORE_13016] [SWS_CORE_13017] [SWS_CORE_13018] [SWS_CORE_13019] [SWS_CORE_13020] [SWS_CORE_13021] [SWS_CORE_13022] [SWS_CORE_13023] [SWS_CORE_13024] [SWS_CORE_15002] [SWS_CORE_90026] [SWS_CORE_90027] [SWS_CORE_90028] [SWS_CORE_90029] [SWS_CORE_90030]
[RS_AP_00145]	Availability of special member functions	[SWS_CORE_00043] [SWS_CORE_00044] [SWS_CORE_00066] [SWS_CORE_00617] [SWS_CORE_01314] [SWS_CORE_01315] [SWS_CORE_01316] [SWS_CORE_01317] [SWS_CORE_01424] [SWS_CORE_01425] [SWS_CORE_01426] [SWS_CORE_01427] [SWS_CORE_01437] [SWS_CORE_02115] [SWS_CORE_02116] [SWS_CORE_02117] [SWS_CORE_02118] [SWS_CORE_02119] [SWS_CORE_03020] [SWS_CORE_03021] [SWS_CORE_03022] [SWS_CORE_03023] [SWS_CORE_03038] [SWS_CORE_06546] [SWS_CORE_06574] [SWS_CORE_06575] [SWS_CORE_06576] [SWS_CORE_06577] [SWS_CORE_08021] [SWS_CORE_08022] [SWS_CORE_08023] [SWS_CORE_08024] [SWS_CORE_08025] [SWS_CORE_08029]
[RS_AP_00155]	Avoidance of cluster-specific initialization functions	[SWS_CORE_10002] [SWS_CORE_10003] [SWS_CORE_15003] [SWS_CORE_15004] [SWS_CORE_15006]
[RS_AP_00159]	usage of "noexcept" specifier	[SWS_CORE_00050] [SWS_CORE_00051] [SWS_CORE_00052] [SWS_CORE_00053] [SWS_CORE_00054]
[RS_AP_00163]	Reentrancy of Functions	[SWS_CORE_13210]
[RS_AP_00164]	Thread-safety of Functions	[SWS_CORE_13200] [SWS_CORE_13201] [SWS_CORE_13202] [SWS_CORE_13203] [SWS_CORE_13204] [SWS_CORE_13207] [SWS_CORE_13211] [SWS_CORE_13212] [SWS_CORE_13213] [SWS_CORE_13214]
[RS_AP_00165]	Concurrent Use of Different Objects	[SWS_CORE_13205]

Table 6.1: Requirements Tracing

7 Functional specification

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

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

7.1 General requirements for all Functional Clusters

7.1.1 Functional Cluster names

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

[SWS_CORE_90025] AUTOSAR-standardized Functional Cluster names

Status: DRAFT
Upstream requirements: RS_AP_00115

⌈

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

▽

△

Name	Description	C++ namespace
vucm	Vehicle Update and Configuration Management	ara::vucm

⌋

7.1.2 Functional Cluster header files

[SWS_CORE_90001] Include folder structure

Upstream requirements: RS_AP_00116, RS_AP_00111

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

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

⌋

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

The "..." form of `#include` statements shall be used, due to the recommendation given in [4, C++14] `[cpp.include] Par. 7`.

[SWS_CORE_90002] Prevent multiple inclusion of header file

Upstream requirements: RS_AP_00111

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

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

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

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

7.1.3 ARA coding guidelines

[SWS_CORE_90003] Reservation of ARA preprocessor macros

Upstream requirements: RS_AP_00111

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

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

[SWS_CORE_90004] Consistency of Implementation-defined declaration classifiers

Upstream requirements: RS_AP_00111

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

Note: The order of declarations may be freely chosen.

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

[SWS_CORE_90005] Custom declarations and definitions

Upstream requirements: RS_AP_00111

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

The Adaptive Platform is designed for source code portability. Wherefore any conformant implementation of the Adaptive Platform allows a successful compilation and linking of an Adaptive Application that uses ARA only as specified in the standard. No changes to the source code, and no conditional compilation constructs will be necessary for this if the application only uses constructs from the designated minimum C++ language version. The implementation may provide proprietary, non-ARA interfaces, as long as they are not contradicting the AP standard.

[SWS_CORE_90007] Potentially throwing constructors

Upstream requirements: RS_AP_00111

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

See ArcDecision in FO_EXP_SWArchitecturalDecisions [5] *Potentially throwing constructors*.

7.1.4 Undefined behavior

[SWS_CORE_90031] Undefined behavior in AUTOSAR APIs ⌈

- Generally: as per [defns.undefined] in [4]
- Additionally: Where known, UB is documented by AUTOSAR. The effects of UB are not documented and no guarantees are given about how the API shall execute during UB. Nevertheless, documented UB in AUTOSAR is not an error.

⌋

7.1.5 Implementation-defined behavior

[SWS_CORE_90032] Implementation-defined behavior in AUTOSAR APIs ⌈

- Generally: as per [defns.impl.defined] in [4]
- Additionally: The precise behavior of IDB in an API is unknown and undocumented by AUTOSAR. The effects are deferred by AUTOSAR to the (out-of-scope) implementation where they shall be well-documented.

⌋

7.1.6 Vendor-specific behavior

[SWS_CORE_90033] Vendor-specific behavior in AUTOSAR APIs ⌈Vendor-specific behavior is equivalent to Implementation-defined behavior ([SWS_CORE_90032]) since any "vendor-specifics" are implicitly "implementation-defined".⌋

7.1.7 Initialization/De-initialization

7.1.7.1 Common aspects

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

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

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

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

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

Upstream requirements: RS_AP_00142

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

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

⌋

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

7.1.7.2 Initialization

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

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

[SWS_CORE_15003] Startup and initialization of ARA

Upstream requirements: RS_AP_00155

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

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

⌋

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

Upstream requirements: RS_AP_00155

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

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

Upstream requirements: RS_AP_00111

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

7.1.7.3 De-initialization

[SWS_CORE_15004] Shutdown and de-initialization of ARA

Upstream requirements: RS_AP_00155

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

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

⌋

An error returned by `ara::core::Deinitialize` is the only way for the ARA to report an error that is guaranteed to be available, e.g. in case `ara::log` has already been de-initialized. The user is not expected to be able to recover from such an error. However, the user may have a project-specific way of recording errors during

de-initialization without `ara::log`. A typical error case to be reported here is that the user is still holding some resource from the ARA.

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

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

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

Upstream requirements: RS_AP_00111

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

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

7.1.7.4 Sequence of Initialization and De-initialization

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

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

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

[SWS_CORE_90026] Framework initialization

Upstream requirements: RS_AP_00142

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

[SWS_CORE_90027] Clean deinitialization

Upstream requirements: RS_AP_00142

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

[SWS_CORE_90028] Re-try of initialization

Upstream requirements: RS_AP_00142

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

[SWS_CORE_90029] Double initialization

Upstream requirements: RS_AP_00142

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

[SWS_CORE_90030] Double de-initialization

Upstream requirements: RS_AP_00142

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

7.1.8 Logging and Tracing

The Adaptive Platform Foundation provides libraries that are integrated into adaptive application processes (see [6] "Figure: 'Overview Log and Trace'" and [7] "Figure: 'Architecture overview'"). Each function cluster has its own "Context ID" to distinguish these ARA libraries from the application logging and from each other. Further information on Context ID can be found in [7] in chapter: "Context ID".

[SWS_CORE_00055] Functional Cluster Log and Trace messages

Upstream requirements: RS_AP_00111

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

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

⌋

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

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

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

⌋

7.1.9 Thread safety

The [4, the C++14 standard] does not provide explicit criteria for defining thread-safety in functions, it is therefore necessary to provide an AUTOSAR definition to set the criteria during evaluation of whether an AUTOSAR function shall be thread-safe. The definitions are deliberately scoped to AUTOSAR APIs, and do not apply to non-AUTOSAR APIs where no guarantees are given.

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

Upstream requirements: RS_AP_00164

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

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

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

Upstream requirements: RS_AP_00164

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

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

Upstream requirements: RS_AP_00164

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

[SWS_CORE_13211] AUTOSAR definition of a thread-safe static member function

Upstream requirements: RS_AP_00164

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

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

Upstream requirements: RS_AP_00164

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

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

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

Upstream requirements: RS_AP_00164

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

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

[SWS_CORE_13205] Behavior of concurrently executed functions on different instances

Upstream requirements: RS_AP_00165

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

[SWS_CORE_13206] AUTOSAR `callable` definition

Upstream requirements: RS_AP_00130

⌈The `callable` related definitions in `[func.def]` in [4] shall apply to AUTOSAR APIs when used.⌋

[SWS_CORE_13207] Behavior of thread-safe `callable`

Upstream requirements: RS_AP_00164

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

[SWS_CORE_13212] Definition of `functions implemented by the user of the API`

Upstream requirements: RS_AP_00164

⌈`Functions implemented by the user of the API` are functions where the signature is defined by the AUTOSAR standard and that are implemented by the user of the API. These are either:

- `callable`s that are passed as callbacks by the user of the API
- pure virtual functions that are implemented by the user of the API

⌋

[SWS_CORE_13213] Usage of thread-safe user-implemented functions

Upstream requirements: RS_AP_00164

⌈The `ara` may assume that `functions implemented by the user of the API` that are defined to be thread-safe in the AUTOSAR standard can be called concurrently to themselves. This also applies to conditionally thread-safe user-implemented functions if their condition determines that they are thread-safe.⌋

[SWS_CORE_13214] Usage of non-thread-safe user-implemented functions

Upstream requirements: RS_AP_00164

⌈The `ara` shall call `functions implemented by the user of the API` that are defined to be non-thread-safe in the AUTOSAR standard non-concurrently (i.e. sequentially) to themselves. This also applies to conditionally thread-safe user-implemented functions if their condition determines that they are non-thread-safe.⌋

An example for the callback is the callback `ara::tsync::SynchronizedTimeBaseNotifier` [SWS_TS_14221]. It is defined to be non-thread-safe. That means the implementer of the function - who is the user of the time sync API - may implement that callback to be non-thread-safe. As a consequence, when that callback is registered via `ara::tsync::SynchronizedTimeBaseConsumer::RegisterTimeLeapNotifier(SynchronizedTimeBaseNotifier notifier)` [SWS_TS_01014], the functional cluster implementation needs to make sure that this callback is only called sequentially ([SWS_CORE_13214]).

If the callback would be specified to be thread-safe then the user needs to make sure that it is implemented that way. In that case the functional cluster implementation can call that callback concurrently ([SWS_CORE_13213]).

The same also applies to pure virtual functions that are implemented by the user of the API and that are then called by the functional cluster implementation. This is a common situation when the Skeleton pattern is used (see also *Offer/StopOffer Pattern* in [2]). For example `ara::phm::RecoveryAction::RecoveryHandler` [SWS_PHM_01142] is defined to be thread-safe. That means when the user of the API implements that function it is expected that this implementation is thread-safe. The Platform Health Management functional cluster implementation may call this function concurrently.

7.1.10 Async signal safety

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

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

[SWS_CORE_13210] ARA API usage within a signal handler

Upstream requirements: RS_AP_00163

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

7.2 Error handling

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

7.2.1 Traditional error handling in C and C++

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

Error codes in C come in several flavors:

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

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

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

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

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

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

The second one is even more critical in the context of developing safety-critical software. The specification of C++ exceptions pose significant problems for C++ compiler vendors that want their products be certified for development of safety-critical software. In fact, ASIL-certified C++ compilers generally do not support exceptions at all. One particular problem with exceptions is that exception handling, as specified for C++, implies the use of dynamic memory allocation, which generally has non-predictable or even unbounded execution time. This makes exceptions currently unsuitable for development of certain safety-critical software in the automotive industry.

7.2.2 Violation

[SWS_CORE_00021] Semantics of a Violation

Upstream requirements: RS_AP_00142

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

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

[SWS_CORE_00090] Handling of `Standardized Violations`

Upstream requirements: RS_AP_00142

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

- explicitly terminating the process abnormally via a call to `Abort()`

In this case a message to support debugging shall be created according to the DLT message defined for the concrete `Violation`. It shall be output in a string format to the process' standard error stream. Additionally, it should be logged into the log sinks with the "Context Identifier" ([SWS_CORE_90024]) of either:

- the functional cluster that detected the `Violation` for the affected process, or
- the `Execution Management`

Which Log and Trace Context Identifier ([SWS_CORE_90024]) is chosen is implementation-defined.⌋

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

Implementations will have `Violations` that are specific to their implementation approach. Thus it is expected that they will need to detect `Violations` that are not and cannot be standardized.

[SWS_CORE_00003] Handling of `Non-Standardized Violations`

Upstream requirements: RS_AP_00142

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

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

In both cases a message to support debugging should be created.⌋

See also: [5] Messages for unrecoverable errors.

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

Note: `ara::core::Abort` is ideal for implementing the two previous requirements. In contrast to that, an interface to create a `Violation` implemented as an exception and associated logs is not standardized.

[SWS_CORE_00006] Handling of exception-based Violations

Upstream requirements: RS_AP_00142

⌈If a `Violation` is realized using an `Exception` (that is not a sub-class of `ara::core::Exception`), and the `noexcept` specifier condition ([4] `[except.spec]`) of the function in which it is raised, evaluates to `true`: the C++ runtime will invoke `std::terminate()` automatically.⌋

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

7.2.3 Corruption

[SWS_CORE_00022] Semantics of a Corruption

Upstream requirements: RS_AP_00142

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

[SWS_CORE_00004] Handling of Corruptions

Upstream requirements: RS_AP_00142

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

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

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

7.2.4 Failed default allocation

[SWS_CORE_00023] Semantics of a Failed Default Allocation

Upstream requirements: RS_AP_00142

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

[SWS_CORE_00005] Handling of failed default allocations

Upstream requirements: RS_AP_00142

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

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

[SWS_CORE_00007] Handling of exception-based Failed Default Allocations

Upstream requirements: RS_AP_00142

⌈If a `Failed Default Allocation` is realized using an `Exception` (that is not a sub-class of `ara::core::Exception`), and the `noexcept` specifier condition ([4] `[except.spec]`) of the function in which it is raised, evaluates to `true`: the C++ runtime will invoke `std::terminate()` automatically.⌋

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

7.2.5 Error

[SWS_CORE_00020] Semantics of an Error

Upstream requirements: RS_AP_00142

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

[SWS_CORE_00002] Handling of Errors

Upstream requirements: RS_AP_00142, RS_AP_00139, RS_AP_00128

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

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

7.2.5.1 ErrorCode

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

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

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

[SWS_CORE_10302] Semantics of ErrorCode

Upstream requirements: RS_AP_00119

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

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

⌋

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

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

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

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

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

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

Upstream requirements: RS_AP_00119

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

- error code value
- error domain

⌋

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

[SWS_CORE_10300] ErrorCode type properties

Upstream requirements: RS_AP_00130

⌈Class `ara::core::ErrorCode` shall be a *literal type*, as defined in [4, C++14] `[basic.types] Par. 10.`⌋

7.2.5.2 Error Domain

An AUTOSAR `ara::core::ErrorDomain` is the abstract base class for holding category specific error information in AUTOSAR. This class is loosely based on `std::error_category`, but differs significantly from it.

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

[SWS_CORE_10303] Semantics of ErrorDomain

Upstream requirements: RS_AP_00119

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

7.2.5.3 Usages of ErrorDomain

ErrorDomains may be sub-classed in the following scenarios:

- functional clusters: AUTOSAR functional clusters or platform extensions APIs are sub-classes of `ara::core::ErrorDomain`. These are directly standardized by AUTOSAR.

- applications: AUTOSAR applications may derive ErrorDomains from `ApApplicationErrorDomain`. These sub-classes of `ara::core::ErrorDomain` may be generated PortInterface language binding rule
- implementation-defined: ErrorDomains ([SWS_CORE_90032]) / vendor-specific ([SWS_CORE_90033]):

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

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

[SWS_CORE_10401] Identity of ErrorDomains

Upstream requirements: RS_AP_00119

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

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

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

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

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

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

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

Upstream requirements: RS_AP_00119

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

[SWS_CORE_10400] ErrorDomain type properties

Upstream requirements: RS_AP_00130

⌈Class `ara::core::ErrorDomain` and all its sub-classes shall be *literal types*, as defined in [4, C++14] `[basic.types] Par. 10`.⌋

7.2.5.3.1 Error domain Identifiers

Each error domain has a numeric 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.

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

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

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

[SWS_CORE_90023] AUTOSAR-standardized Functional Cluster Error Domain Identifiers

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Name	*C++ Type*	*Error Domain Identifier*	*Error Domain Name*
`core`	`ara::core::FutureErrorDomain`	0x8000'0000'0000'0013	CoreFuture
`core`	`ara::core::CoreErrorDomain`	0x8000'0000'0000'0014	Core
`per`	`ara::per::PerErrorDomain`	0x8000'0000'0000'0101	Per
`rper`	`ara::rper::RemotePerErrorDomain`	0x8000'0000'0000'0102	RemotePer

▽

△

Name	C++ Type	Error Domain Identifier	Error Domain Name
exec	ara::exec::ExecErrorDomain	0x8000'0000'0000'0202	Exec
sm	ara::sm::SmErrorDomain	0x8000'0000'0000'0301	Sm
diag	ara::diag::DiagErrorDomain	0x8000'0000'0000'0401	Diag
diag	ara::diag::DiagReportingErrorDomain	0x8000'0000'0000'0402	DiagReporting
diag	ara::diag::DiagOfferErrorDomain	0x8000'0000'0000'0403	DiagOffer
diag	ara::diag::DiagUdsNrcErrorDomain	0x8000'0000'0000'0411	DiagUdsNrc
diag	ara::diag::DiagSovdErrorDomain	0x8000'0000'0000'0412	DiagSovd
fw	ara::fw::FwErrorDomain	0x8000'0000'0000'0501	Fw
phm	ara::phm::PhmErrorDomain	0x8000'0000'0000'0602	Phm
ucm	ara::ucm::UcmErrorDomain	0x8000'0000'0000'0701	Ucm
crypto	ara::crypto::CryptoErrorDomain	0x8000'0000'0000'0801	Crypto
tsync	ara::tsync::TsyncErrorDomain	0x8000'0000'0000'0901	Tsync
tsync	apext::tsync::TsyncFvmErrorDomain	0x8000'0000'0000'0911	TsyncFvm
nm	ara::nm::NmErrorDomain	0x8000'0000'0000'1001	Nm
idsm	ara::idsm::IdsmErrorDomain	0x8000'0000'0000'1101	Idsm
com	ara::com::ComErrorDomain	0x8000'0000'0000'1267	Com
com	ara::com::e2e::ComE2eErrorDomain	0x8000'0000'0000'1268	ComE2e
com	apext::com::secoc::ComSecOcFvmErrorDomain	0x8000'0000'0000'1271	ComSecOcFvm
rds	ara::rds::RdsErrorDomain	0x8000'0000'0000'1280	Rds
vucm	ara::vucm::VucmErrorDomain	0x8000'0000'0000'1301	Vucm
shwa	ara::shwa::ShwaErrorDomain	0x8000'0000'0000'1302	Shwa

⌋

[SWS_CORE_00010] Error domain identifier

Upstream requirements: RS_AP_00130

⌈All sub-classes of error domains specified in the standard shall implement the `ErrorDomain::Id()` function which shall return the system wide unique error domain identifier of the `ara::core::ErrorDomain` from [SWS_CORE_90023].⌋

[SWS_CORE_00011] AUTOSAR error domain range

Upstream requirements: RS_AP_00130

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

[SWS_CORE_00016] Vendor-specific error domain range

Upstream requirements: RS_AP_00130

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

[SWS_CORE_00094] Error domain name

Upstream requirements: RS_AP_00130

⌈All sub-classes of error domains specified in the standard shall implement the `ErrorDomain::Name()` function, which shall return the system-wide unique name of the `ara::core::ErrorDomain` from [SWS_CORE_90023].⌋

[SWS_CORE_00013] The Future error domain

Upstream requirements: RS_AP_00130

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

[SWS_CORE_00014] The Core error domain

Upstream requirements: RS_AP_00130

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

7.2.5.3.2 Vendor error domains

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

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

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

[SWS_CORE_00092] Vendor-specific errors

Upstream requirements: RS_AP_00142

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

7.2.5.3.3 Creating new error domains

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

An error domain consists of:

- an error condition enumeration

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

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

[SWS_CORE_10999] Custom error domain scope

Upstream requirements: RS_AP_00130

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

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

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

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

7.2.5.3.3.1 ErrorDomain sub-class

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

[SWS_CORE_10930] ErrorDomain sub-class type

Upstream requirements: RS_AP_00130

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

[SWS_CORE_10931] ErrorDomain sub-class naming

Upstream requirements: RS_AP_00130

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

[SWS_CORE_00095] Error domain message

Upstream requirements: RS_AP_00130

⌈All sub-classes of `ara::core::ErrorDomain` shall implement the `ErrorDomain::Message()` function, which shall return the description text, for any given error code in its domain.⌋

[SWS_CORE_10932] ErrorDomain sub-class non-extensibility

Upstream requirements: RS_AP_00130, RS_AP_00140

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

[SWS_CORE_10933] ErrorDomain sub-class Errc symbol

Upstream requirements: RS_AP_00130

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

[SWS_CORE_10934] ErrorDomain sub-class Exception symbol

Upstream requirements: RS_AP_00130

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

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

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

[SWS_CORE_10950] ErrorDomain sub-class member function property

Upstream requirements: RS_AP_00130

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

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

[SWS_CORE_10951] ErrorDomain sub-class shortname retrieval

Upstream requirements: RS_AP_00130

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

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

[SWS_CORE_10953] Throwing ErrorCodes as exceptions

Upstream requirements: RS_AP_00130

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

7.2.5.3.3.2 Non-member ErrorDomain sub-class accessor function

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

[SWS_CORE_10980] ErrorDomain sub-class accessor function

Upstream requirements: RS_AP_00130

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

[SWS_CORE_10981] ErrorDomain sub-class accessor function naming

Upstream requirements: RS_AP_00130

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

[SWS_CORE_10982] ErrorDomain sub-class accessor function

Upstream requirements: RS_AP_00130

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

7.2.5.3.3.3 Non-member MakeErrorCode overload

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

[SWS_CORE_10990] MakeErrorCode overload for new error domains

Upstream requirements: RS_AP_00130

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

[SWS_CORE_10991] MakeErrorCode overload signature

Upstream requirements: RS_AP_00130

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

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

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

7.2.5.3.3.4 Error condition enumeration

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

[SWS_CORE_10900] Error condition enumeration type

Upstream requirements: RS_AP_00130

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

[SWS_CORE_10901] Error condition enumeration naming

Upstream requirements: RS_AP_00130

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

[SWS_CORE_10902] Error condition enumeration contents

Upstream requirements: RS_AP_00130

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

[SWS_CORE_10903] Error condition enumeration numbers

Upstream requirements: RS_AP_00130

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

7.2.5.3.3.5 Exception base class

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

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

[SWS_CORE_10910] ErrorDomain exception base type

Upstream requirements: RS_AP_00130

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

[SWS_CORE_10911] ErrorDomain exception base type naming

Upstream requirements: RS_AP_00130

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

[SWS_CORE_10912] ErrorDomain exception type hierarchy

Upstream requirements: RS_AP_00130

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

7.2.5.3.3.6 Error Domain C++ pseudo code example

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

```
namespace my
{

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

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

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

```

```
21      constexpr <SN>ErrorDomain() noexcept;
22
23      const char* Name() const noexcept override;
24      const char* Message(ara::core::ErrorDomain::CodeType errorCode)
           const noexcept override;
25      void ThrowAsException(const ara::core::ErrorCode& errorCode) const
           noexcept(false) override;
26  };
27
28  constexpr const ara::core::ErrorDomain& Get<SN>ErrorDomain() noexcept;
29
30  constexpr ara::core::ErrorCode MakeErrorCode(<SN>Errc code, ara::core::
       ErrorDomain::SupportDataType data) noexcept;
31
32  }  // namespace my
```

7.2.5.4 Result

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

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

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

Upstream requirements: RS_AP_00119, RS_AP_00142, RS_AP_00128

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

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

⌋

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

Upstream requirements: RS_AP_00130

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

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

Upstream requirements: RS_AP_00130

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

See also the "rebind" approach in 7.4.2.2.2.1

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

- `operator->`
- `Bind`

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

7.2.5.5 Future and Promise

`ara::core::Future` and its companion class `ara::core::Promise` are closely modeled on `std::future` and `std::promise`, but have been adapted to interoperate with `ara::core::Result`. Similar to `ara::core::Result` described in section 7.2.5.4, the class `ara::core::Future` either contains a value, or an error (the `Future` first has to be in “ready” state, though).

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

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

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

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

Whenever there is a mention of a standard C++14 item (class, class template, enum or function) such as `std::future` or `std::promise`, the implied source material is [4]. Whenever there is a mention of an experimental C++ item such as `std::experimental::future::is_ready`, the implied source material is [12].

Futures are technically referred to as "asynchronous return objects", and Promises are referred to as "asynchronous providers". Their interaction is made possible by a `shared state`. The `shared state` concept is described in [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 "stores the ErrorCode `kBrokenPromise` defined in [SWS_CORE_00400] in its `shared state`; and then"

Class `ara::core::Future` and `ara::core::Promise` are closely modeled on `std::future` and `std::promise`. Consequently, the behavior of `ara::core::Future` and `ara::core::Promise` is expected to be same as that of `std::future` and `std::promise` from [4, the C++14 standard] and the corresponding `std::experimental::` classes from [12], except for the deviations from the `std::` classes that result from the integration with `ara::core::Result`.

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

Upstream requirements: RS_AP_00138, RS_AP_00128

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

7.2.5.6 Exceptions

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

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

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

7.2.5.7 Duality of ErrorCode and Exception

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

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

7.3 Explicitly aborting an Operation

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

Calling `ara::core::Abort` will result in an `Explicitly aborting an Operation`, which usually leads to an `Abnormal Termination` as defined by [13]. This section defines the behavior of this mechanism.

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

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

Upstream requirements: RS_AP_00130

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

[SWS_CORE_12408] DLT Logging of `Explicitly aborting an Operation`

Upstream requirements: RS_AP_00130

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

- For calls that resulted from a `Violation` or `Failed Default Allocation` the logging behavior as defined in [SWS_CORE_00090], [SWS_CORE_00003], and [SWS_CORE_00005] applies.
- For calls from application code an `AbortMessage` ([SWS_CORE_13019]) with the *message* argument equal to the log message that was output to the standard error stream according to [SWS_CORE_12403] and the context id of ara::core shall be used.

⌋

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

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

Upstream requirements: RS_AP_00130

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

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

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

[SWS_CORE_12404] AbortHandler invocation

Upstream requirements: RS_AP_00130

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

7.3.1 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 `Abnormal Termination` is properly seen by Execution Management.

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

[SWS_CORE_12405] Final action without AbortHandler

Upstream requirements: RS_AP_00130

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

[SWS_CORE_12406] Final action with returning AbortHandlers

Upstream requirements: RS_AP_00130

⌈If there are custom `ara::core::AbortHandlers` that have been installed with `ara::core::SetAbortHandler` or `ara::core::AddAbortHandler` and

all of them return, then the implementation of `ara::core::Abort` shall call `std::abort()`.⌋

[SWS_CORE_12409] Usage of ARA API within AbortHandlers

Upstream requirements: RS_AP_00130

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

7.3.2 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(SIGABRT)`.

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

7.4 Core data types

7.4.1 Data types devised by AUTOSAR

AUTOSAR has devised a suite of data types commonly used across all functional clusters.

7.4.1.1 `ara::core::InstanceSpecifier`

An `ara::core::InstanceSpecifier` is used to bridge the divide between a modeled `PortPrototype` instance (in the context of a model) and the API class which implements the language binding within the context of source code.

A detailed description and background can be found in [14] Chapter 4.8.1 *"Instance Identifiers and Instance Specifiers"* and [14] Chapter 7.4.5 *"Usage of meta-model identifiers within ara::com based application code"*.

An `ara::core::InstanceSpecifier` is a type wrapper for a string representation of a valid meta-model path to a `PortPrototype` in a model. The `ara::core::InstanceSpecifier` may be constructed either:

- via named constructor `Create()` which provides an exception-less error handling solution by returning an `ara::core::ErrorCode`, or
- via a typical constructor, which may raise a exception in an error situation

[SWS_CORE_10200] Valid InstanceSpecifier representations - application interaction

Upstream requirements: RS_AP_00130

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

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

[SWS_CORE_10203] Valid InstanceSpecifier representations - functional cluster interaction

Upstream requirements: RS_AP_00130

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

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

[SWS_CORE_10201] Validation of meta-model paths

Upstream requirements: RS_AP_00130

⌈The construction mechanisms of class `ara::core::InstanceSpecifier` shall reject meta-model paths that are syntactically invalid according to the syntax rules defined in [SWS_CORE_10200] or [SWS_CORE_10203].⌋

[SWS_CORE_10202] Construction of InstanceSpecifier objects

Upstream requirements: RS_AP_00130

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

7.4.1.2 `ara::core::Executor`

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

7.4.2 Data types derived from the C++ standard

In addition to AUTOSAR-devised data types, `ara::core` has derived implementations of a sub-set of data types from `ISO-14882`. There are several rationales for this:

1. to eliminate non-deterministic/undefined behavior ([SWS_CORE_90031])
2. to apply implementation-defined ([SWS_CORE_90032]) / vendor-specific behavior ([SWS_CORE_90033])
3. to apply AUTOSAR specific extensions (e.g. additional functions, type aliases)
4. to apply `final`-ity to an otherwise extensible `class` or `struct`
5. to apply exception-less API pattern: `ara::core::Result`
6. to apply fail-early mechanisms: `Violation`s
7. to provide forward implementations of a C++ API in a `ISO-14882` version, downstream of the current support language version ([RS_AP_00114])

General aspects:

The precise version of `ISO-14882` upon which the respective header is based, is specified in the respective header in 8.

All deviations to ISO-14882 derived APIs are listed explicitly in the API tables' description.

Deviations:

[SWS_CORE_00040] Errors originating from C++ standard classes

Upstream requirements: RS_AP_00130

⌈For the classes in `ara::core` specified below by means of the corresponding classes of the C++ standard, all functions that are specified by [4, C++14], [11, C++17], or [15, C++20] to throw any exceptions, are instead specified to be the cause of a `Violation` when they do so.⌋

7.4.2.1 Containers library

7.4.2.1.1 ara::core::Array

Sequence container: based on `<array>`; ISO-14882 `[array.syn]`. API specified in 8.1.

The overloaded `get()` methods 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 `std::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);
```

7.4.2.1.2 ara::core::Vector

Sequence container: based on `<vector>`; ISO-14882 `[vector.syn]`. API specified in 8.10.

7.4.2.1.3 ara::core::Map

Associative container: `<map>`; ISO-14882 `[associative.map.syn]`. API specified in 8.3.

7.4.2.1.4 ara::core::Span

View: `<span>`; `ISO-14882 [span.syn]`. API specified in 8.6.

7.4.2.2 General utilities library

7.4.2.2.1 ara::core::Bitset

Fixed-size sequences of bits. API specified in 8.2.

7.4.2.2.2 ara::core::Optional

Optional objects: <optional>; ISO-14882 [optional.syn]. API specified in 8.5.

The existence of the contained value can be evaluated during both compile-time (via constexpr specifiers) and runtime. There are two usages:

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

7.4.2.2.2.1 ara::core::Optional<T&>

[SWS_CORE_01032] Optional references

Status: DRAFT
Upstream requirements: RS_AP_00130

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

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

```
1 int x = 42;
2 Optional<int&> opt = x;  // "bind" opt to x
3 int y = 43;
4 opt = y;  // "assign-through" or "rebind"?
```

the resulting program state could be:

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

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

[SWS_CORE_01034] Assignment behavior of Optional references

Status: DRAFT
Upstream requirements: RS_AP_00130

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

7.4.2.2.3 ara::core::Variant

Variants: <variant>; ISO-14882 [variant.syn]. API specified in 8.9.

7.4.2.2.4 ara::core::MemoryResource

Memory resources: <memory_resource>; ISO-14882 [mem.res.syn]. API specified in 8.4.

7.4.2.2.5 In-place disambiguation tags

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

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

7.4.2.2.6 Non-member container access

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

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

7.4.2.3 Strings library

7.4.2.3.1 ara::core::String

String classes: <string>; ISO-14882 [string.syn]. API specified in 8.1.

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

7.4.2.3.2 ara::core::StringView

String view classes: <string_view>; ISO-14882 [string.view.syn]. API specified in 8.7.

It is the programmer's responsibility to ensure that `ara::core::StringView` does not outlive the pointed-to character array. As the UTF-8 encoding is used throughout the Adaptive Platform, only the `char` type is supported for `ara::core::StringView`.

7.4.2.4 Language support library

7.4.2.4.1 `ara::core::Byte`

API specified in 8.19.

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

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

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

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

Upstream requirements: RS_AP_00130

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

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

Upstream requirements: RS_AP_00130

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

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

Upstream requirements: RS_AP_00130

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

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

Upstream requirements: RS_AP_00130

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

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

Upstream requirements: RS_AP_00130

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

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

Upstream requirements: RS_AP_00130

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

[SWS_CORE_10106] Implicit conversion from other types

Upstream requirements: RS_AP_00130

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

[SWS_CORE_10107] Implicit conversion to other types

Upstream requirements: RS_AP_00130

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

[SWS_CORE_10108] Conversion to unsigned char

Upstream requirements: RS_AP_00130

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

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

Upstream requirements: RS_AP_00130

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

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

Upstream requirements: RS_AP_00130

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

7.4.2.5 Time library

7.4.2.5.1 `ara::core::SteadyClock`

7.4.2.5.1.1 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.

7.4.2.5.1.2 Clocks in the Adaptive Platform

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

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

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

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

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

[SWS_CORE_11800] SteadyClock type requirements

Upstream requirements: RS_AP_00130

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

[SWS_CORE_11801] Epoch of SteadyClock

Upstream requirements: RS_AP_00130

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

7.5 Functional cluster life-cycle

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

7.5.1 Startup

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

7.5.2 Shutdown

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

7.6 Reporting

7.6.1 Security Events

This functional cluster does not define any security events.

7.6.2 Log Messages

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

7.6.2.1 AbortMessage

[SWS_CORE_13019] LogMessage AbortMessage

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

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

⌋

7.6.2.2 FailedDefaultAllocation

[SWS_CORE_13018] LogMessage FailedDefaultAllocation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

Dlt-Message	FailedDefaultAllocation		
Description	Sent in case the affected process was terminated because of a failed default allocation		
MessageId	0x80001000		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator	uint8 [encoding UTF-8]	

▽

△

location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}	uint8 [encoding UTF-8]	
message	Additional message that can describe the cause of the failed default allocation in more detail	uint8 [encoding UTF-8]	

⌋

7.6.3 Violation Messages

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

7.6.3.1 `AraNotInitializedViolation`

[SWS_CORE_13007] ViolationMessage AraNotInitializedViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

Dlt-Message	AraNotInitializedViolation		
Description	Violation message that is sent in case a constructor or function checks for an initialized ara and identifies that ara is not initialized.		
MessageId	0x80001ff8		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator.	uint8 [encoding UTF-8]	
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	
instanceSpecifier	InstanceSpecifier used to try to create the object. (only present in case this violation is created by a function/constructor that has an InstanceSpecifier as an argument)	uint8 [encoding UTF-8]	
functionName	Name of the function/constructor that was called that checks for an initialized ara.	uint8 [encoding UTF-8]	

⌋

7.6.3.2 FutureContinuationHasThrownViolation

[SWS_CORE_13012] ViolationMessage FutureContinuationHasThrownViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

DIt-Message	FutureContinuationHasThrownViolation		
Description	Violation message that is sent in case the continuation given to Future::then threw an exception.		
MessageId	0x80001ff3		
MessageType Info	DLT_LOG_FATAL		
DIt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator.	uint8 [encoding UTF-8]	
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	
exception Explanatory String	Explanatory string of the exception if available.	uint8 [encoding UTF-8]	

⌋

7.6.3.3 FutureInvalidViolation

[SWS_CORE_13011] ViolationMessage FutureInvalidViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

DIt-Message	FutureInvalidViolation		
Description	Violation message that is sent in case a prohibited function call was performed on an invalid Future object.		
MessageId	0x80001ff4		
MessageType Info	DLT_LOG_FATAL		
DIt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator.	uint8 [encoding UTF-8]	
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	
methodName	Future class method name where the violation was detected.	uint8 [encoding UTF-8]	

⌋

7.6.3.4 InstanceSpecifierAlreadyInUseViolation

[SWS_CORE_13006] ViolationMessage InstanceSpecifierAlreadyInUseViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

Dlt-Message	InstanceSpecifierAlreadyInUseViolation		
Description	Violation message that is sent in case a constructor in the ara framework was called with an Instance Specifier already in use in this process.		
MessageId	0x80001ff9		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator.	uint8 [encoding UTF-8]	
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	
instanceSpecifier	InstanceSpecifier used to try to create the object.	uint8 [encoding UTF-8]	
className	Name of the class that was instantiated.	uint8 [encoding UTF-8]	

⌋

7.6.3.5 InstanceSpecifierMappingIntegrityViolation

[SWS_CORE_13003] ViolationMessage InstanceSpecifierMappingIntegrityViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

Dlt-Message	InstanceSpecifierMappingIntegrityViolation		
Description	InstanceSpecifier either cannot be resolved in the model in the context of your executable, or it refers to a model element other than a PortPrototype.		
MessageId	0x80001ffc		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator.	uint8 [encoding UTF-8]	
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	
instanceSpecifier	InstanceSpecifier used to try to create the object.	uint8 [encoding UTF-8]	
className	Name of the class that was instantiated.	uint8 [encoding UTF-8]	

⌋

7.6.3.6 InsufficientPermissionsViolation

[SWS_CORE_13000] ViolationMessage InsufficientPermissionsViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

Dlt-Message	InsufficientPermissionsViolation		
Description	Sent in case the caller had insufficient permissions for the requested operation.		
MessageId	0x80001fff		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator.	uint8 [encoding UTF-8]	
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	
message	Additional message that describes the cause of the access violation.	uint8 [encoding UTF-8]	

⌋

7.6.3.7 IsoStdBadAllocViolation

[SWS_CORE_13021] ViolationMessage IsoStdBadAllocViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

Dlt-Message	IsoStdBadAllocViolation		
Description	Description as per `std::bad_alloc`		
MessageId	0x80001fec		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator.	uint8 [encoding UTF-8]	
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	

⌋

7.6.3.8 IsoStdBadOptionalAccessViolation

[SWS_CORE_13020] ViolationMessage IsoStdBadOptionalAccessViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

Dlt-Message	IsoStdBadOptionalAccessViolation		
Description	Description as per `std::bad_optional_access`.		
MessageId	0x80001fed		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator.	uint8 [encoding UTF-8]	
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	

⌋

7.6.3.9 IsoStdBadVariantAccessViolation

[SWS_CORE_13001] ViolationMessage IsoStdBadVariantAccessViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

Dlt-Message	IsoStdBadVariantAccessViolation		
Description	Description as per `std::bad_variant_access`.		
MessageId	0x80001ffe		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator.	uint8 [encoding UTF-8]	
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	

⌋

7.6.3.10 IsoStdFutureErrcFutureAlreadyRetrievedViolation

[SWS_CORE_13013] ViolationMessage IsoStdFutureErrcFutureAlreadyRetrievedViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

Dlt-Message	IsoStdFutureErrcFutureAlreadyRetrievedViolation		
Description	Description as per `std::future_errc::future_already_retrieved`.		
MessageId	0x80001ff2		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator.	uint8 [encoding UTF-8]	
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	

⌋

7.6.3.11 IsoStdFutureErrcNoStateViolation

[SWS_CORE_13014] ViolationMessage IsoStdFutureErrcNoStateViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

Dlt-Message	IsoStdFutureErrcNoStateViolation		
Description	Description as per `std::future_errc::no_state`.		
MessageId	0x80001ff1		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator.	uint8 [encoding UTF-8]	
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	

⌋

7.6.3.12 IsoStdFutureErrcPromiseAlreadySatisfiedViolation

[SWS_CORE_13015] ViolationMessage IsoStdFutureErrcPromiseAlreadySatisfiedViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

Dlt-Message	IsoStdFutureErrcPromiseAlreadySatisfiedViolation		
Description	Description as per `std::future_errc::promise_already_satisfied`.		
MessageId	0x80001ff0		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator.	uint8 [encoding UTF-8]	
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	
targetField	Target field to set in the Promise setter method call where the violation was detected. Allowed values are: "value", "error", "result".	uint8 [encoding UTF-8]	

⌋

7.6.3.13 IsoStdInvalidArgumentViolation

[SWS_CORE_13022] ViolationMessage IsoStdInvalidArgumentViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

Dlt-Message	IsoStdInvalidArgumentViolation		
Description	Description as per `std::invalid_argument`		
MessageId	0x80001feb		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator.	uint8 [encoding UTF-8]	
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	

⌋

7.6.3.14 IsoStdLengthErrorViolation

[SWS_CORE_13024] ViolationMessage IsoStdLengthErrorViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

DIt-Message	IsoStdLengthErrorViolation		
Description	Description as per `std::length_error`		
MessageId	0x80001fe9		
MessageType Info	DLT_LOG_FATAL		
DIt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator	uint8 [encoding UTF-8]	
location	An implementation-defined identifier of the location where the violation was detected, for example {Class Name}:{MethodName}, {filename}:{linenumber}, etc.	uint8 [encoding UTF-8]	

⌋

7.6.3.15 IsoStdOutOfRangeViolation

[SWS_CORE_13017] ViolationMessage IsoStdOutOfRangeViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

DIt-Message	IsoStdOutOfRangeViolation		
Description	Description as per `std::out_of_range`.		
MessageId	0x80001fee		
MessageType Info	DLT_LOG_FATAL		
DIt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator.	uint8 [encoding UTF-8]	
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	
indexValue	Index value passed as input parameter	uint8 [encoding UTF-8]	

⌋

7.6.3.16 IsoStdOverflowErrorViolation

[SWS_CORE_13023] ViolationMessage IsoStdOverflowErrorViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

DIt-Message	IsoStdOverflowErrorViolation		
Description	Description as per `std::overflow_error`		
MessageId	0x80001fea		
MessageType Info	DLT_LOG_FATAL		
DIt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator	uint8 [encoding UTF-8]	
location	An implementation-defined identifier of the location where the violation was detected, for example {Class Name}:{MethodName}, {filename}:{linenumber}, etc.	uint8 [encoding UTF-8]	

⌋

7.6.3.17 PortInterfaceMappingViolation

[SWS_CORE_13004] ViolationMessage PortInterfaceMappingViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

DIt-Message	PortInterfaceMappingViolation		
Description	The type of mapping does not match the expected type of PortInterface: {portInterfaceTypeName} referenced by a {mappingTypeName}.		
MessageId	0x80001ffb		
MessageType Info	DLT_LOG_FATAL		
DIt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator.	uint8 [encoding UTF-8]	
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	
instanceSpecifier	InstanceSpecifier used to try to create the object.	uint8 [encoding UTF-8]	
className	Name of the class that was instantiated.	uint8 [encoding UTF-8]	

⌋

7.6.3.18 ProcessMappingViolation

[SWS_CORE_13005] ViolationMessage ProcessMappingViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

DIt-Message	ProcessMappingViolation		
Description	Matching InstanceRef exists, but no matching (modelled) Process found that matches the (runtime) process.		
MessageId	0x80001ffa		
MessageType Info	DLT_LOG_FATAL		
DIt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator.	uint8 [encoding UTF-8]	
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	
instanceSpecifier	InstanceSpecifier used to try to create the object.	uint8 [encoding UTF-8]	
className	Name of the class that was instantiated.	uint8 [encoding UTF-8]	

⌋

7.6.3.19 PromiseNoSharedStateToSetViolation

[SWS_CORE_13016] ViolationMessage PromiseNoSharedStateToSetViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

DIt-Message	PromiseNoSharedStateToSetViolation		
Description	Violation message that is sent in case the Promise had no shared state when a function was called that expected the shared state to exist.		
MessageId	0x80001fef		
MessageType Info	DLT_LOG_FATAL		
DIt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator.	uint8 [encoding UTF-8]	
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	
targetField	Target field to set in the Promise setter method call where the violation was detected. Allowed values are: "value", "error", "result".	uint8 [encoding UTF-8]	

⌋

7.6.3.20 ResultWithNoErrorViolation

[SWS_CORE_13010] ViolationMessage ResultWithNoErrorViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

Dlt-Message	ResultWithNoErrorViolation		
Description	Violation message that is sent in case error of a Result is accessed while there is no error contained in this Result object.		
MessageId	0x80001ff5		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator.	uint8 [encoding UTF-8]	
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	

⌋

7.6.3.21 ResultWithNoValueViolation

[SWS_CORE_13009] ViolationMessage ResultWithNoValueViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

Dlt-Message	ResultWithNoValueViolation		
Description	Violation message that is sent in case value of a Result is accessed while there is no value contained in this Result object.		
MessageId	0x80001ff6		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator.	uint8 [encoding UTF-8]	
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	

⌋

7.6.3.22 WrongDomainOfAnErrorCodeViolation

[SWS_CORE_13008] ViolationMessage WrongDomainOfAnErrorCodeViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

DIt-Message	WrongDomainOfAnErrorCodeViolation		
Description	Violation message that is sent in case an error code from a different error domain was passed to an error domain object for the error code's message retrieval.		
MessageId	0x80001ff7		
MessageType Info	DLT_LOG_FATAL		
DIt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator.	uint8 [encoding UTF-8]	
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	
errorCodeValue	Domain-specific error code value of the passed error code.	uint8 [encoding UTF-8]	
errorCodeDomainName	ErrorDomain associated with the passed error code.	uint8 [encoding UTF-8]	
errorDomainName	The function calls ErrorDomain name.	uint8 [encoding UTF-8]	

⌋

7.6.4 Production Errors

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

8 API specification

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

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

Table 8.1: Explanation of an API table

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

8.1 Header: ara/core/array.h

[SWS_CORE_01200] Definition of Header File ara/core/array.h

Upstream requirements: RS_AP_00130

⌈

Kind:	Header File
Syntax:	`ara/core/array.h`
Description:	Implements `[array.syn]` in [11]. Unless explicitly overridden in the API documentation, APIs always adhere in behavior to the ISO specification. To reduce duplication, the APIs here specify only any deviations from [11] or any AUTOSAR specific overrides. For APIs which would ordinarily throw `Exception`s, and those `Exception`s have been suppressed in favor of AUTOSAR violations, generally the semantics of the `Exception` from [11] still apply (unless locally overridden here).

⌋

8.1.1 Non-Member Functions

8.1.1.1 Other

8.1.1.1.1 get(Array<T, N>&)

[SWS_CORE_01282] Definition of API function ara::core::get ⌈

Kind:	function
Header file:	#include "ara/core/array.h"
Scope:	namespace ara::core
Syntax:	`template <std::size_t I, typename T, std::size_t N>` `constexpr T & get (Array< T, N > &a) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[array]`
Notes:	The implementation shall flag the condition I >= N as a compile error.

⌋

8.1.1.1.2 get(const Array<T, N>&)

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

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

▽

△

Syntax:	`template <std::size_t I, typename T, std::size_t N>` `constexpr T const & get (const Array< T, N > &a) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[array]`
Notes:	The implementation shall flag the condition I >= N as a compile error.

⌋

8.1.1.1.3 get(const Array<T, N>&&)

[SWS_CORE_01286] Definition of API function ara::core::get ⌈

Kind:	function
Header file:	#include "ara/core/array.h"
Scope:	namespace ara::core
Syntax:	`template <size_t I, class T, size_t N>` `constexpr const T && get (const Array< T, N > &&) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[array]`

⌋

8.1.1.1.4 get(Array<T, N>&&)

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

Kind:	function
Header file:	#include "ara/core/array.h"
Scope:	namespace ara::core
Syntax:	`template <std::size_t I, typename T, std::size_t N>` `constexpr T && get (Array< T, N > &&a) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[array]`
Notes:	The implementation shall flag the condition I >= N as a compile error.

⌋

8.1.1.1.5 operator!=

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

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

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

Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[array]`

⌋

8.1.1.1.6 operator<

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

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

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

Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[array]`

⌋

8.1.1.1.7 operator<=

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

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

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

Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[array]`

⌋

8.1.1.1.8 operator==

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

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

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

Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[array]`

⌋

8.1.1.1.9 operator>

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

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

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

Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[array]`

⌋

8.1.1.1.10 operator>=

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

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

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

Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[array]`

⌋

8.1.1.1.11 swap

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

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

Syntax:	`template <typename T, std::size_t N>`
	`void swap (Array< T, N > &lhs, Array< T, N > &rhs)`
	`noexcept(noexcept(lhs.swap(rhs)));`

Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[array]`

⌋

8.1.2 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	--
	std::size_t N	--
Description:	Implementation of `std::array` type from `[array.syn]` The implicitly-declared special member functions specified in `[array.cons]` are not explicitly re-declared here.	

⌋

8.1.2.1 Public Member Types

8.1.2.1.1 Type Alias: const_iterator

[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:	As per corresponding usage in `[array]`

⌋

8.1.2.1.2 Type Alias: const_pointer

[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:	As per corresponding usage in `[array]`

⌋

8.1.2.1.3 Type Alias: const_reference

[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:	As per corresponding usage in `[array]`

⌋

8.1.2.1.4 Type Alias: const_reverse_iterator

[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:	As per corresponding usage in `[array]`

⌋

8.1.2.1.5 Type Alias: difference_type

[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:	As per corresponding usage in `[array]`

⌋

8.1.2.1.6 Type Alias: iterator

[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:	As per corresponding usage in `[array]`

⌋

8.1.2.1.7 Type Alias: pointer

[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:	As per corresponding usage in `[array]`

⌋

8.1.2.1.8 Type Alias: reference

[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:	As per corresponding usage in `[array]`

⌋

8.1.2.1.9 Type Alias: reverse_iterator

[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:	As per corresponding usage in `[array]`

⌋

8.1.2.1.10 Type Alias: size_type

[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:	As per corresponding usage in `[array]`

⌋

8.1.2.1.11 Type Alias: value_type

[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:	As per corresponding usage in `[array]`

⌋

8.1.2.2 Public Member Functions

8.1.2.2.1 Member Functions

8.1.2.2.1.1 fill

[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`
Syntax:	`void fill (const T &u) noexcept(std::is_nothrow_copy_assignable< T >::value);`
Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[array]`

⌋

8.1.2.2.1.2 swap

[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`
Syntax:	`void swap (Array< T, N > &other) noexcept(ara::core::is_nothrow_swappable< T >::value);`
Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[array]` except for the following deviations: 1. `ara::core::is_nothrow_swappable` shall be used in place of the `std::` implementation

⌋

8.1.2.2.2 Capacity

8.1.2.2.2.1 empty

[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`
Syntax:	`[[nodiscard]] constexpr bool empty () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[array]`

⌋

8.1.2.2.2.2 max_size

[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`
Syntax:	`constexpr size_type max_size () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[array]`

⌋

8.1.2.2.2.3 size

[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`
Syntax:	`constexpr size_type size () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[array]`

⌋

8.1.2.2.3 Element access

8.1.2.2.3.1 at(size_type)

[SWS_CORE_01273] Definition of API function ara::core::Array::at ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`class ara::core::Array`	
Syntax:	`constexpr reference at (size_type n) noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdOut-OfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[array]` except for the following deviations: 1. if (n >= `size()`) `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range`	

⌋

8.1.2.2.3.2 at(size_type) const

[SWS_CORE_01274] Definition of API function ara::core::Array::at ⌈

Kind:	function	
Header file:	#include "ara/core/array.h"	
Scope:	`class ara::core::Array`	
Syntax:	`constexpr const_reference at (size_type n) const noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdOut-OfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[array]` except for the following deviations: 1. if (n >= `size()`) `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range`	

⌋

8.1.2.2.3.3 back()

[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`
Syntax:	`constexpr reference back () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[array]`
Notes:	Calling this function on an `empty()` array shall be a compile-time error

⌋

8.1.2.2.3.4 back() const

[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`
Syntax:	`constexpr const_reference back () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[array]`
Notes:	Calling this function on an `empty()` array shall be a compile-time error

⌋

8.1.2.2.3.5 data() const

[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`
Syntax:	`constexpr const_pointer data () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[array]`
Notes:	This function shall return `nullptr` in case the size of the array is 0

⌋

8.1.2.2.3.6 data()

[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`
Syntax:	`constexpr pointer data () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[array]`
Notes:	This function shall return `nullptr` in case the size of the array is 0

⌋

8.1.2.2.3.7 front() const

[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`
Syntax:	`constexpr const_reference front () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[array]`
Notes:	Calling this function on an `empty()` array shall be a compile-time error

⌋

8.1.2.2.3.8 front()

[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`
Syntax:	`constexpr reference front () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[array]`
Notes:	Calling this function on an `empty()` array shall be a compile-time error

⌋

8.1.2.2.3.9 operator[](size_type) const

[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`
Syntax:	`constexpr const_reference operator[] (size_type n) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[array]`

⌋

8.1.2.2.3.10 operator[](size_type)

[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`
Syntax:	`constexpr reference operator[] (size_type n) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[array]`

⌋

8.1.2.2.4 Iterators

8.1.2.2.4.1 begin() const

[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`
Syntax:	`constexpr const_iterator begin () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[array]`
Undefined Behavior:	De-referencing the returned iterator on an `empty()` array is undefined behavior

⌋

8.1.2.2.4.2 begin()

[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`
Syntax:	`constexpr iterator begin () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[array]`
Undefined Behavior:	De-referencing the returned iterator on an `empty()` array is undefined behavior

⌋

8.1.2.2.4.3 cbegin() const

[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`
Syntax:	`constexpr const_iterator cbegin () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[array]`

⌋

8.1.2.2.4.4 cend() const

[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`
Syntax:	`constexpr const_iterator cend () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[array]`

⌋

8.1.2.2.4.5 crbegin() const

[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`
Syntax:	`constexpr const_reverse_iterator crbegin () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[array]`

⌋

8.1.2.2.4.6 crend() const

[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`
Syntax:	`constexpr const_reverse_iterator crend () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[array]`

⌋

8.1.2.2.4.7 end()

[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`
Syntax:	`constexpr iterator end () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[array]`

⌋

8.1.2.2.4.8 end() const

[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`
Syntax:	`constexpr const_iterator end () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[array]`

⌋

8.1.2.2.4.9 rbegin() const

[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`
Syntax:	`constexpr const_reverse_iterator rbegin () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[array]`

⌋

8.1.2.2.4.10 rbegin()

[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`
Syntax:	`constexpr reverse_iterator rbegin () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[array]`

⌋

8.1.2.2.4.11 rend()

[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`
Syntax:	`constexpr reverse_iterator rend () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[array]`

⌋

8.1.2.2.4.12 rend() const

[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`
Syntax:	`constexpr const_reverse_iterator rend () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[array]`

⌋

8.1.3 Struct: tuple_element

[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 > > {...};`
Description:	As per corresponding usage in `[array]`
Notes:	The condition `I >= N` shall be a compile-time error

⌋

8.1.3.1 Public Member Types

8.1.3.1.1 Type Alias: type

[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:	As per corresponding usage in `[array]`
Notes:	The condition `I >= N` shall be a compile-time error

⌋

8.1.4 Struct: tuple_size

[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< size_t, N > {...};`
Description:	As per corresponding usage in `[array]`
Notes:	This specialization shall meet the `UnaryTypeTrait` requirements (see `[meta.rqmts]`) with a *base characteristic* of `std::integral_constant<std::size_t, N>`

⌋

8.2 Header: ara/core/bitset.h

[SWS_CORE_01700] Definition of Header File ara/core/bitset.h

Upstream requirements: RS_AP_00130

⌈

Kind:	Header File
Syntax:	`ara/core/bitset.h`
Description:	Implements `[bitset.syn]` in [11]. Unless explicitly overridden in the API documentation, APIs always adhere in behavior to the ISO specification. To reduce duplication, the APIs here specify only any deviations from [11] or any AUTOSAR specific overrides. For APIs which would ordinarily throw `Exception`s, and those `Exception`s have been suppressed in favor of AUTOSAR violations, generally the semantics of the `Exception` from [11] still apply (unless locally overridden here).

⌋

8.2.1 Non-Member Functions

8.2.1.1 Other

8.2.1.1.1 operator&

[SWS_CORE_01752] Definition of API function ara::core::operator& ⌈

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

Syntax:	`template <std::size_t N>`
	`Bitset< N > operator& (const Bitset< N > &, const Bitset< N > &)`
	`noexcept;`

Exception Safety:	exception safe
Description:	As per corresponding usage in `[bitset]`

⌋

8.2.1.1.2 operator<<

[SWS_CORE_01756] Definition of API function ara::core::operator<< ⌈

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

Syntax:	`template <class CharT = char, class Traits = std::char_traits<CharT>,`
	`std::size_t N>`
	`std::basic_ostream< CharT, Traits > & operator<< (std::basic_ostream<`
	`CharT, Traits > &os, const Bitset< N > &x);`

Exception Safety:	not exception safe
Description:	AAs per corresponding usage in `[bitset]`

⌋

8.2.1.1.3 operator>>

[SWS_CORE_01755] Definition of API function ara::core::operator>> ⌈

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

Syntax:	`template <class CharT, class Traits, std::size_t N>`
	`std::basic_istream< CharT, Traits > & operator>> (std::basic_istream<`
	`CharT, Traits > &is, Bitset< N > &x);`

Exception Safety:	not exception safe
Description:	As per corresponding usage in `[bitset]`

⌋

8.2.1.1.4 operator^

[SWS_CORE_01754] Definition of API function ara::core::operator^ ⌈

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

Syntax:	`template <std::size_t N>`
	`Bitset< N > operator^ (const Bitset< N > &, const Bitset< N > &)`
	`noexcept;`

Exception Safety:	exception safe
Description:	As per corresponding usage in `[bitset]`

⌋

8.2.1.1.5 operator|

[SWS_CORE_01753] Definition of API function ara::core::operator| ⌈

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

Syntax:	`template <std::size_t N>`
	`Bitset< N > operator| (const Bitset< N > &, const Bitset< N > &)`
	`noexcept;`

Exception Safety:	exception safe
Description:	As per corresponding usage in `[bitset]`

⌋

8.2.2 Class: Bitset

[SWS_CORE_01701] Definition of API class ara::core::Bitset ⌈

Kind:	class	
Header file:	#include "ara/core/bitset.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	namespace ara::core	
Symbol:	Bitset	
Syntax:	`template <std::size_t N>` `class Bitset {...};`	
Template param:	std::size_t N	--
Description:	Implementation of `std::bitset` type from `[bitset]`	

⌋

8.2.2.1 Public Member Functions

8.2.2.1.1 Special Member Functions

8.2.2.1.1.1 Move Constructor

[SWS_CORE_01716] Definition of API function ara::core::Bitset::Bitset

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset`
Syntax:	`Bitset (Bitset &&) noexcept=default;`
Exception Safety:	exception safe
Description:	**[AUTOSAR defined]** not present in ISO-14882. Move constructor

⌋

8.2.2.1.1.2 Default Constructor

[SWS_CORE_01713] Definition of API function ara::core::Bitset::Bitset ⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset`
Syntax:	`constexpr Bitset () noexcept=default;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[bitset]`

⌋

8.2.2.1.1.3 Copy Constructor

[SWS_CORE_01714] Definition of API function ara::core::Bitset::Bitset

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset`
Syntax:	`Bitset (const Bitset &) noexcept=default;`
Exception Safety:	exception safe
Description:	**[AUTOSAR defined]** not present in ISO-14882. Copy constructor

⌋

8.2.2.1.1.4 Move Assignment Operator

[SWS_CORE_01717] Definition of API function ara::core::Bitset::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset`
Syntax:	`Bitset & operator= (Bitset &&) noexcept=default;`
Exception Safety:	exception safe
Description:	**[AUTOSAR defined]** not present in ISO-14882. Move assignment operator

⌋

8.2.2.1.1.5 Copy Assignment Operator

[SWS_CORE_01715] Definition of API function ara::core::Bitset::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset`
Syntax:	`Bitset & operator= (const Bitset &) noexcept=default;`
Exception Safety:	exception safe
Description:	**[AUTOSAR defined]** not present in ISO-14882. Copy assignment operator

⌋

8.2.2.1.1.6 Destructor

[SWS_CORE_01718] Definition of API function ara::core::Bitset::~Bitset

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset`
Syntax:	`~Bitset () noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	**[AUTOSAR defined]** not present in ISO-14882. Destructor

⌋

8.2.2.1.2 Constructors

8.2.2.1.2.1 Bitset(const BasicString<Allocator>&, typename BasicString<Allocator>::size_type, typename BasicString<Allocator>::size_type, CharT, CharT)

[SWS_CORE_01749] Definition of API function ara::core::Bitset::Bitset ⌈

Kind:	function	
Header file:	#include "ara/core/bitset.h"	
Scope:	`class ara::core::Bitset`	
Syntax:	`template <class CharT, class Traits, class Allocator> explicit Bitset (const BasicString< Allocator > &str, typename BasicString< Allocator >::size_type pos=0, typename BasicString< Allocator >::size_type n=BasicString< Allocator >::npos, CharT zero=CharT('0'), CharT one=CharT('1')) noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdOutOfRangeViolation`	Description as per std::out_of_range.
	`IsoStdInvalidArgumentViolation`	Description as per std::invalid_argument
Description:	As per corresponding usage in `[bitset]` except for the following deviations: 1. The function is `noexcept` 2. If (`pos` > `size()`), `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range` 3. If any character is not `1` or `0`, `IsoStdInvalidArgumentViolation` is raised instead of `std::invalid_argument` 4. Function arguments have been adapted to conform to `ara::core::BasicString` constructor parameters	

⌋

8.2.2.1.2.2 Bitset(unsigned long long)

[SWS_CORE_01719] Definition of API function ara::core::Bitset::Bitset ⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset`
Syntax:	`constexpr Bitset (unsigned long long val) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[bitset]`

⌋

8.2.2.1.2.3 Bitset(const CharT *, typename BasicString<CharT>::size_type, CharT, CharT)

[SWS_CORE_01750] Definition of API function ara::core::Bitset::Bitset ⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset`
Syntax:	`template <class CharT>` `explicit Bitset (const CharT *str, typename BasicString< CharT >::size_type n=BasicString< CharT >::npos, CharT zero=CharT('0'), CharT one=CharT('1'));`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[bitset]` except for the following deviations: 1. Function arguments have been adapted to conform to `ara::core::BasicString` constructor parameters

⌋

8.2.2.1.3 Capacity

8.2.2.1.3.1 size

[SWS_CORE_01741] Definition of API function ara::core::Bitset::size ⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset`
Syntax:	`constexpr std::size_t size () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[bitset]`

⌋

8.2.2.1.4 Conversions

8.2.2.1.4.1 to_string

[SWS_CORE_01751] Definition of API function ara::core::Bitset::to_string ⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset`
Syntax:	`template <class Allocator = std::allocator<char>>` `BasicString< Allocator > to_string (char zero=char('0'), char one=char('1')) const;`
Exception Safety:	not exception safe

▽

△

Description:	As per corresponding usage in `[bitset]` except for the following deviations: 1. Template arguments have been adapted to `ara::core::BasicString` in place of `std::basic_string`

⌋

8.2.2.1.4.2 to_uint16t

[SWS_CORE_01737] Definition of API function ara::core::Bitset::to_uint16t

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/bitset.h"	
Scope:	`class ara::core::Bitset`	
Syntax:	`constexpr std::uint16_t to_uint16t () const noexcept;`	
Return value:	std::uint16_t	The bits in the bitset, whereby the first bit of the bitset corresponds to the least significant bit of the returned number and the last bit corresponds to the most significant bit
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`IsStdOverflowErrorViolation`	If the integral value `x` corresponding to the bits in the bitset cannot be represented as the return type
Description:	**[AUTOSAR defined]** not present in ISO-14882. Converts the contents of the bitset to a `std::uint16_t` from [SWS_APT_00025]	

⌋

8.2.2.1.4.3 to_uint32t

[SWS_CORE_01738] Definition of API function ara::core::Bitset::to_uint32t

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/bitset.h"	
Scope:	`class ara::core::Bitset`	
Syntax:	`constexpr std::uint32_t to_uint32t () const noexcept;`	
Return value:	std::uint32_t	The bits in the bitset, whereby the first bit of the bitset corresponds to the least significant bit of the returned number and the last bit corresponds to the most significant bit
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`IsStdOverflowErrorViolation`	If the integral value `x` corresponding to the bits in the bitset cannot be represented as the return type

▽

△

Description:	**[AUTOSAR defined]** not present in ISO-14882. Alias for `ara::core::Bitset::to_ulong` but explicitly using the `std::uint32_t` type from [SWS_APT_00028]

⌋

8.2.2.1.4.4 to_uint64t

[SWS_CORE_01739] Definition of API function ara::core::Bitset::to_uint64t

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/bitset.h"	
Scope:	`class ara::core::Bitset`	
Syntax:	`constexpr std::uint64_t to_uint64t () const noexcept;`	
Return value:	std::uint64_t	The bits in the bitset, whereby the first bit of the bitset corresponds to the least significant bit of the returned number and the last bit corresponds to the most significant bit
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`IsoStdOverflowErrorViolation`	If the integral value `x` corresponding to the bits in the bitset cannot be represented as the return type
Description:	**[AUTOSAR defined]** not present in ISO-14882. Alias for `ara::core::Bitset::to_ullong` but explicitly using the `std::uint64_t` type from [SWS_APT_00031]	

⌋

8.2.2.1.4.5 to_uint8t

[SWS_CORE_01736] Definition of API function ara::core::Bitset::to_uint8t

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/bitset.h"	
Scope:	`class ara::core::Bitset`	
Syntax:	`constexpr std::uint8_t to_uint8t () const noexcept;`	
Return value:	std::uint8_t	The bits in the bitset, whereby the first bit of the bitset corresponds to the least significant bit of the returned number and the last bit corresponds to the most significant bit
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`IsoStdOverflowErrorViolation`	If the integral value `x` corresponding to the bits in the bitset cannot be represented as the return type

▽

△

Description:	**[AUTOSAR defined]** not present in ISO-14882. Converts the contents of the bitset to a `std::uint8_t` from [SWS_APT_00022]

⌋

8.2.2.1.4.6 to_ullong

[SWS_CORE_01735] Definition of API function ara::core::Bitset::to_ullong ⌈

Kind:	function	
Header file:	#include "ara/core/bitset.h"	
Scope:	`class ara::core::Bitset`	
Syntax:	`constexpr unsigned long long to_ullong () const noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdOverflowErrorViolation`	Description as per std::overflow_error
Description:	As per corresponding usage in `[bitset]` except for the following deviations: 1. Function is constexpr 2. The function is `noexcept` 3. If the integral value `x` corresponding to the bits in the bitset cannot be represented as the return type, `IsoStdOverflowErrorViolation` is raised instead of `std::overflow_error`	

⌋

8.2.2.1.4.7 to_ulong

[SWS_CORE_01734] Definition of API function ara::core::Bitset::to_ulong ⌈

Kind:	function	
Header file:	#include "ara/core/bitset.h"	
Scope:	`class ara::core::Bitset`	
Syntax:	`constexpr unsigned long to_ulong () const noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdOverflowErrorViolation`	Description as per std::overflow_error
Description:	As per corresponding usage in `[bitset]` except for the following deviations: 1. Function is constexpr 2. The function is `noexcept` 3. If the integral value `x` corresponding to the bits in the bitset cannot be represented as the return type, `IsoStdOverflowErrorViolation` is raised instead of `std::overflow_error`	

⌋

8.2.2.1.5 Element access

8.2.2.1.5.1 all

[SWS_CORE_01744] Definition of API function ara::core::Bitset::all ⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset`
Syntax:	`bool all () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[bitset]`

⌋

8.2.2.1.5.2 any

[SWS_CORE_01745] Definition of API function ara::core::Bitset::any ⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset`
Syntax:	`bool any () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[bitset]`

⌋

8.2.2.1.5.3 count

[SWS_CORE_01740] Definition of API function ara::core::Bitset::count ⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset`
Syntax:	`std::size_t count () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[bitset]`

⌋

8.2.2.1.5.4 none

[SWS_CORE_01746] Definition of API function ara::core::Bitset::none ⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset`
Syntax:	`bool none () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[bitset]`

⌋

8.2.2.1.5.5 operator[](std::size_t)

[SWS_CORE_01733] Definition of API function ara::core::Bitset::operator[] ⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset`
Syntax:	`reference operator[] (std::size_t pos) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[bitset]` except for the following deviations: 1. The function is `noexcept`

⌋

8.2.2.1.5.6 operator[](std::size_t) const

[SWS_CORE_01732] Definition of API function ara::core::Bitset::operator[] ⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset`
Syntax:	`constexpr bool operator[] (std::size_t pos) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[bitset]` except for the following deviations: 1. The function is `noexcept`

⌋

8.2.2.1.5.7 test

[SWS_CORE_01743] Definition of API function ara::core::Bitset::test ⌈

Kind:	function	
Header file:	#include "ara/core/bitset.h"	
Scope:	`class ara::core::Bitset`	
Syntax:	`bool test (std::size_t pos) const noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdOut-OfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[bitset]` except for the following deviations: 1. The function is `noexcept` 2. If `pos` does not correspond to a valid bit position, `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range`	

⌋

8.2.2.1.6 Modifiers

8.2.2.1.6.1 flip(std::size_t)

[SWS_CORE_01731] Definition of API function ara::core::Bitset::flip ⌈

Kind:	function	
Header file:	#include "ara/core/bitset.h"	
Scope:	`class ara::core::Bitset`	
Syntax:	`Result< Bitset< N > & > flip (std::size_t pos) noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdOut-OfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[bitset]` except for the following deviations: 1. The function is `noexcept` 2. If `pos` does not correspond to a valid bit position, `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range`	

⌋

8.2.2.1.6.2 flip()

[SWS_CORE_01730] Definition of API function ara::core::Bitset::flip ⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset`
Syntax:	`Bitset< N > & flip () noexcept;`
Exception Safety:	exception safe

▽

△

Description:	As per corresponding usage in `[bitset]`

⌋

8.2.2.1.6.3 operator&=

[SWS_CORE_01720] Definition of API function ara::core::Bitset::operator&= ⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset`
Syntax:	`Bitset< N > & operator&= (const Bitset< N > &rhs) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[bitset]`

⌋

8.2.2.1.6.4 operator<<

[SWS_CORE_01747] Definition of API function ara::core::Bitset::operator<< ⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset`
Syntax:	`Bitset< N > operator<< (std::size_t pos) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[bitset]`

⌋

8.2.2.1.6.5 operator<<=

[SWS_CORE_01723] Definition of API function ara::core::Bitset::operator<<= ⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset`
Syntax:	`Bitset< N > & operator<<= (std::size_t pos) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[bitset]`

⌋

8.2.2.1.6.6 operator>>

[SWS_CORE_01748] Definition of API function ara::core::Bitset::operator>> ⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset`
Syntax:	`Bitset< N > operator>> (std::size_t pos) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[bitset]`

⌋

8.2.2.1.6.7 operator>>=

[SWS_CORE_01724] Definition of API function ara::core::Bitset::operator>>= ⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset`
Syntax:	`Bitset< N > & operator>>= (std::size_t pos) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[bitset]`

⌋

8.2.2.1.6.8 operatorˆ=

[SWS_CORE_01722] Definition of API function ara::core::Bitset::operatorˆ= ⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset`
Syntax:	`Bitset< N > & operatorˆ= (const Bitset< N > &rhs) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[bitset]`

⌋

8.2.2.1.6.9 operator|=

[SWS_CORE_01721] Definition of API function ara::core::Bitset::operator|= ⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset`
Syntax:	`Bitset< N > & operator|= (const Bitset< N > &rhs) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[bitset]`

⌋

8.2.2.1.6.10 operator~

[SWS_CORE_01729] Definition of API function ara::core::Bitset::operator~ ⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset`
Syntax:	`Bitset< N > operator~ () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[bitset]`

⌋

8.2.2.1.6.11 reset(std::size_t)

[SWS_CORE_01728] Definition of API function ara::core::Bitset::reset ⌈

Kind:	function	
Header file:	#include "ara/core/bitset.h"	
Scope:	`class ara::core::Bitset`	
Syntax:	`Result< Bitset< N > & > reset (std::size_t pos) noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdOut-OfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[bitset]` except for the following deviations: 1. The function is `noexcept` 2. If `pos` does not correspond to a valid bit position, `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range`	

⌋

8.2.2.1.6.12 reset()

[SWS_CORE_01727] Definition of API function ara::core::Bitset::reset ⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset`
Syntax:	`Bitset< N > & reset () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[bitset]`

⌋

8.2.2.1.6.13 set(std::size_t, bool)

[SWS_CORE_01726] Definition of API function ara::core::Bitset::set ⌈

Kind:	function	
Header file:	#include "ara/core/bitset.h"	
Scope:	`class ara::core::Bitset`	
Syntax:	`Result< Bitset< N > & > set (std::size_t pos, bool val=true) noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdOut-OfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[bitset]` except for the following deviations: 1. The function is `noexcept` 2. If `pos` does not correspond to a valid bit position, `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range`	

⌋

8.2.2.1.6.14 set()

[SWS_CORE_01725] Definition of API function ara::core::Bitset::set ⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset`
Syntax:	`Bitset< N > & set () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[bitset]`

⌋

8.2.2.1.7 Other

8.2.2.1.7.1 operator==

[SWS_CORE_01742] Definition of API function ara::core::Bitset::operator== ⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset`
Syntax:	`bool operator== (const Bitset< N > &rhs) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[bitset]`

⌋

8.2.3 Class: reference

[SWS_CORE_01759] Definition of API class ara::core::Bitset::reference ⌈

Kind:	class
Header file:	#include "ara/core/bitset.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	`class ara::core::Bitset`
Symbol:	reference
Syntax:	`class reference {...};`
Description:	Implements `std::bitset::reference` type from `[bitset]`

⌋

8.2.3.1 Public Member Functions

8.2.3.1.1 Special Member Functions

8.2.3.1.1.1 Copy Assignment Operator

[SWS_CORE_01705] Definition of API function ara::core::Bitset::reference::operator= ⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset::reference`
Syntax:	`reference & ara::core::Bitset< N >::reference::operator= (const reference &) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[bitset]`

⌋

8.2.3.1.1.2 Move Assignment Operator

[SWS_CORE_01707] Definition of API function ara::core::Bitset::reference::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset::reference`
Syntax:	`reference & ara::core::Bitset< N >::reference::operator= (reference &&)=delete;`
Description:	**[AUTOSAR defined]** not present in ISO-14882. Move assignment operator deletion

⌋

8.2.3.1.1.3 Move Constructor

[SWS_CORE_01706] Definition of API function ara::core::Bitset::reference::reference

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset::reference`
Syntax:	`ara::core::Bitset< N >::reference::reference (reference &&)=delete;`
Description:	**[AUTOSAR defined]** not present in ISO-14882. Move constructor deletion

⌋

8.2.3.1.1.4 Copy Constructor

[SWS_CORE_01704] Definition of API function ara::core::Bitset::reference::reference ⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset::reference`
Syntax:	`ara::core::Bitset< N >::reference::reference (const reference &) noexcept=default;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[bitset]`

⌋

8.2.3.1.1.5 Destructor

[SWS_CORE_01708] Definition of API function ara::core::Bitset::reference::~reference ⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset::reference`
Syntax:	`ara::core::Bitset< N >::reference::~reference () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[bitset]`

⌋

8.2.3.1.2 Member Functions

8.2.3.1.2.1 flip

[SWS_CORE_01712] Definition of API function ara::core::Bitset::reference::flip

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset::reference`
Syntax:	`reference & ara::core::Bitset< N >::reference::flip () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[bitset]`

⌋

8.2.3.1.2.2 operator bool

[SWS_CORE_01711] Definition of API function ara::core::Bitset::reference::operator bool ⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset::reference`
Syntax:	`ara::core::Bitset< N >::reference::operator bool () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[bitset]`

⌋

8.2.3.1.2.3 operator=

[SWS_CORE_01709] Definition of API function ara::core::Bitset::reference::operator= ⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset::reference`
Syntax:	`reference & ara::core::Bitset< N >::reference::operator= (bool x) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[bitset]`

⌋

8.2.3.1.2.4 operator~

[SWS_CORE_01710] Definition of API function ara::core::Bitset::reference::operator~ ⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset::reference`
Syntax:	`bool ara::core::Bitset< N >::reference::operator~ () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[bitset]`

⌋

8.2.3.2 Private Member Functions

8.2.3.2.1 Special Member Functions

8.2.3.2.1.1 Default Constructor

[SWS_CORE_01703] Definition of API function ara::core::Bitset::reference::reference ⌈

Kind:	function
Header file:	#include "ara/core/bitset.h"
Scope:	`class ara::core::Bitset::reference`
Syntax:	`ara::core::Bitset< N >::reference::reference () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[bitset]`
Visibility:	private

⌋

8.2.4 Struct: hash

[SWS_CORE_01757] Definition of API class ara::core::hash ⌈

Kind:	struct	
Header file:	#include "ara/core/bitset.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	namespace ara::core	
Symbol:	hash	
Syntax:	`template <class T>` `struct hash;`	
Template param:	T	--
Description:	As per corresponding usage in `[bitset]`	

⌋

8.3 Header: ara/core/map.h

[SWS_CORE_01401] Definition of Header File ara/core/map.h

Upstream requirements: RS_AP_00130

⌈

Kind:	Header File
Syntax:	`ara/core/map.h`
Description:	Implements `[associative.map.syn]` in [11]. Unless explicitly overridden in the API documentation, APIs always adhere in behavior to the ISO specification. To reduce duplication, the APIs here specify only any deviations from [11] or any AUTOSAR specific overrides. For APIs which would ordinarily throw `Exception`s, and those `Exception`s have been suppressed in favor of AUTOSAR violations, generally the semantics of the `Exception` from [11] still apply (unless locally overridden here).

⌋

8.3.1 Non-Member Functions

8.3.1.1 Other

8.3.1.1.1 swap

[SWS_CORE_01496] Definition of API function ara::core::swap

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	namespace ara::core
Syntax:	`template <class Key, class T, class Compare, class Allocator> void swap (Map< Key, T, Compare, Allocator > &lhs, Map< Key, T, Compare, Allocator > &rhs) noexcept(noexcept(lhs.swap(rhs)));`
Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2 Class: Map

[SWS_CORE_01400] 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 Key, typename T, typename Compare = std::less<Key>, typename Allocator = {<fq-allocator>}> class Map {...};`	
Template param:	typename Key	--
	typename T	--
	typename Compare = std::less<Key>	--
	typename Allocator = `{<fq-allocator>}`	--
Description:	Implementation of `std::pmr::map` container from `[map]`	
Descriptors:	`{<fq-allocator>}`	Implementation-defined. Fully qualified `Allocator` class. The Allocator shall satisfy `[allocator.requirements.completeness]` in [11]

⌋

8.3.2.1 Public Member Types

8.3.2.1.1 Type Alias: allocator_type

[SWS_CORE_01406] Definition of API type ara::core::Map::allocator_type ⌈

Kind:	type alias
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Symbol:	allocator_type
Syntax:	`using allocator_type = Allocator;`
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.1.2 Type Alias: const_iterator

[SWS_CORE_01414] Definition of API type ara::core::Map::const_iterator ⌈

Kind:	type alias	
Header file:	#include "ara/core/map.h"	
Scope:	`class ara::core::Map`	
Symbol:	const_iterator	
Syntax:	`using const_iterator = {<const-iterator-type>};`	
Description:	As per corresponding usage in `[map]`	
Implementation Defined Behavior:	`{<const-iterator-type>}` shall be an implementation-defined type	
Descriptors:	`{<const-iterator-type>}`	implementation-defined

⌋

8.3.2.1.3 Type Alias: const_pointer

[SWS_CORE_01408] Definition of API type ara::core::Map::const_pointer ⌈

Kind:	type alias
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Symbol:	const_pointer
Syntax:	`using const_pointer = typename std::allocator_ traits<Allocator>::const_pointer;`
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.1.4 Type Alias: const_reference

[SWS_CORE_01410] Definition of API type ara::core::Map::const_reference ⌈

Kind:	type alias
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Symbol:	const_reference
Syntax:	`using const_reference = const value_type&;`
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.1.5 Type Alias: const_reverse_iterator

[SWS_CORE_01416] Definition of API type ara::core::Map::const_reverse_iterator ⌈

Kind:	type alias
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Symbol:	const_reverse_iterator
Syntax:	`using const_reverse_iterator = std::reverse_iterator<const_iterator>;`
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.1.6 Type Alias: difference_type

[SWS_CORE_01412] Definition of API type ara::core::Map::difference_type ⌈

Kind:	type alias	
Header file:	#include "ara/core/map.h"	
Scope:	`class ara::core::Map`	
Symbol:	difference_type	
Syntax:	`using difference_type = {<difference-type>};`	
Description:	As per corresponding usage in `[map]`	
Implementation Defined Behavior:	`{<difference-type>}` shall be an implementation-defined type	
Descriptors:	`{<difference-type>}`	implementation-defined

⌋

8.3.2.1.7 Type Alias: insert_return_type

[SWS_CORE_01418] Definition of API type ara::core::Map::insert_return_type ⌈

Kind:	type alias	
Header file:	#include "ara/core/map.h"	
Scope:	`class ara::core::Map`	
Symbol:	insert_return_type	
Syntax:	`using insert_return_type = {<insert-return-type>}<iterator, node_type>;`	
Description:	As per corresponding usage in `[map]`	
Implementation Defined Behavior:	`{<insert-return-type>}` shall be an implementation-defined type	
Descriptors:	`{<insert-return-type>}`	implementation-defined

⌋

8.3.2.1.8 Type Alias: iterator

[SWS_CORE_01413] Definition of API type ara::core::Map::iterator ⌈

Kind:	type alias	
Header file:	#include "ara/core/map.h"	
Scope:	`class ara::core::Map`	
Symbol:	iterator	
Syntax:	`using iterator = {<iterator-type>};`	
Description:	As per corresponding usage in `[map]`	
Implementation Defined Behavior:	`{<iterator-type>}` shall be an implementation-defined type	
Descriptors:	`{<iterator-type>}`	implementation-defined

⌋

8.3.2.1.9 Type Alias: key_compare

[SWS_CORE_01405] Definition of API type ara::core::Map::key_compare ⌈

Kind:	type alias
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Symbol:	key_compare
Syntax:	`using key_compare = Compare;`
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.1.10 Type Alias: key_type

[SWS_CORE_01402] Definition of API type ara::core::Map::key_type ⌈

Kind:	type alias
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Symbol:	key_type
Syntax:	`using key_type = Key;`
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.1.11 Type Alias: mapped_type

[SWS_CORE_01403] Definition of API type ara::core::Map::mapped_type ⌈

Kind:	type alias
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Symbol:	mapped_type
Syntax:	`using mapped_type = T;`
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.1.12 Type Alias: node_type

[SWS_CORE_01417] Definition of API type ara::core::Map::node_type ⌈

Kind:	type alias	
Header file:	#include "ara/core/map.h"	
Scope:	`class ara::core::Map`	
Symbol:	node_type	
Syntax:	`using node_type = {<node-type>};`	
Description:	As per corresponding usage in `[map]`	
Implementation Defined Behavior:	`{<node-type>}` shall be an implementation-defined type	
Descriptors:	`{<node-type>}`	implementation-defined

⌋

8.3.2.1.13 Type Alias: pointer

[SWS_CORE_01407] Definition of API type ara::core::Map::pointer ⌈

Kind:	type alias
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Symbol:	pointer
Syntax:	`using pointer = typename std::allocator_traits<Allocator>::pointer;`
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.1.14 Type Alias: reference

[SWS_CORE_01409] Definition of API type ara::core::Map::reference ⌈

Kind:	type alias
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Symbol:	reference
Syntax:	`using reference = value_type&;`
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.1.15 Type Alias: reverse_iterator

[SWS_CORE_01415] Definition of API type ara::core::Map::reverse_iterator ⌈

Kind:	type alias
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Symbol:	reverse_iterator
Syntax:	`using reverse_iterator = std::reverse_iterator<iterator>;`
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.1.16 Type Alias: size_type

[SWS_CORE_01411] Definition of API type ara::core::Map::size_type ⌈

Kind:	type alias	
Header file:	#include "ara/core/map.h"	
Scope:	`class ara::core::Map`	
Symbol:	size_type	
Syntax:	`using size_type = {<size-type>};`	
Description:	As per corresponding usage in `[map]`	
Implementation Defined Behavior:	`{<size-type>}` shall be an implementation-defined type	
Descriptors:	`{<size-type>}`	implementation-defined

⌋

8.3.2.1.17 Type Alias: value_type

[SWS_CORE_01404] Definition of API type ara::core::Map::value_type ⌈

Kind:	type alias
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Symbol:	value_type
Syntax:	`using value_type = std::pair<const Key, T>;`
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2 Public Member Functions

8.3.2.2.1 Special Member Functions

8.3.2.2.1.1 Map(Map&&)

[SWS_CORE_01426] Definition of API function ara::core::Map::Map

Upstream requirements: RS_AP_00145

⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`Map (Map &&x)=default;`
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.1.2 Map()

[SWS_CORE_01428] Definition of API function ara::core::Map::Map ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`Map ();`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.1.3 Map(const Map&)

[SWS_CORE_01424] Definition of API function ara::core::Map::Map

Upstream requirements: RS_AP_00145

⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`Map (const Map &x)=default;`
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.1.4 operator=(const Map&)

[SWS_CORE_01425] Definition of API function ara::core::Map::operator=

Upstream requirements: RS_AP_00145

⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`Map & operator= (const Map &x)=default;`
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.1.5 operator=(Map&&)

[SWS_CORE_01427] Definition of API function ara::core::Map::operator=

Upstream requirements: RS_AP_00145

⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`Map & operator= (Map &&x) noexcept(std::allocator_traits< Allocator >::is_always_equal::value &&std::is_nothrow_move_assignable< Compare >::value);`
Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.1.6 Destructor

[SWS_CORE_01437] Definition of API function ara::core::Map::~Map

Upstream requirements: RS_AP_00145

⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`~Map ();`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.2 Constructors

8.3.2.2.2.1 Map(std::initializer_list<value_type>, const Compare&, const Allocator&)

[SWS_CORE_01434] Definition of API function ara::core::Map::Map ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`Map (std::initializer_list< value_type >, const Compare &=Compare(), const Allocator &=Allocator());`
Exception Safety:	not exception safe

▽

△

Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.2.2 Map(const Compare&, const Allocator&)

[SWS_CORE_01429] Definition of API function ara::core::Map::Map ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`explicit Map (const Compare &comp, const Allocator &=Allocator());`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.2.3 Map(InputIterator, InputIterator, const Compare&, const Allocator&)

[SWS_CORE_01430] Definition of API function ara::core::Map::Map ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`template <class InputIterator>` `Map (InputIterator first, InputIterator last, const Compare &comp=Compare(), const Allocator &=Allocator());`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.2.4 Map(const Allocator&)

[SWS_CORE_01431] Definition of API function ara::core::Map::Map ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`explicit Map (const Allocator &);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.2.5 Map(const Map&, const Allocator&)

[SWS_CORE_01432] Definition of API function ara::core::Map::Map ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`Map (const Map &, const Allocator &);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.2.6 Map(Map&&, const Allocator&)

[SWS_CORE_01433] Definition of API function ara::core::Map::Map ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`Map (Map &&, const Allocator &);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.2.7 Map(std::initializer_list<value_type>, const Allocator&)

[SWS_CORE_01436] Definition of API function ara::core::Map::Map ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`Map (std::initializer_list< value_type > il, const Allocator &a);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.2.8 Map(InputIterator, InputIterator, const Allocator&)

[SWS_CORE_01435] Definition of API function ara::core::Map::Map ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`template <class InputIterator>` `Map (InputIterator first, InputIterator last, const Allocator &a);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.3 Capacity

8.3.2.2.3.1 empty

[SWS_CORE_01511] Definition of API function ara::core::Map::empty ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`[[nodiscard]] bool empty () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.3.2 max_size

[SWS_CORE_01453] Definition of API function ara::core::Map::max_size ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`size_type max_size () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.3.3 size

[SWS_CORE_01452] Definition of API function ara::core::Map::size ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`size_type size () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.4 Element access

8.3.2.2.4.1 at(const key_type&)

[SWS_CORE_01456] Definition of API function ara::core::Map::at ⌈

Kind:	function	
Header file:	#include "ara/core/map.h"	
Scope:	`class ara::core::Map`	
Syntax:	`mapped_type & at (const key_type &x) noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdOut-OfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[map]` except for the following deviations: 1. if (`x` is not found in the container) `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range`	

⌋

8.3.2.2.4.2 at(const key_type&) const

[SWS_CORE_01457] Definition of API function ara::core::Map::at ⌈

Kind:	function	
Header file:	#include "ara/core/map.h"	
Scope:	`class ara::core::Map`	
Syntax:	`const mapped_type & at (const key_type &x) const noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdOut-OfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[map]` except for the following deviations: 1. if (`x` is not found in the container) `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range`	

⌋

8.3.2.2.4.3 operator[](const key_type&)

[SWS_CORE_01454] Definition of API function ara::core::Map::operator[] ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`mapped_type & operator[] (const key_type &x);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.4.4 operator[](key_type&&)

[SWS_CORE_01455] Definition of API function ara::core::Map::operator[] ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`mapped_type & operator[] (key_type &&x);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.5 Iterators

8.3.2.2.5.1 begin()

[SWS_CORE_01440] Definition of API function ara::core::Map::begin ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`iterator begin () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.5.2 begin() const

[SWS_CORE_01441] Definition of API function ara::core::Map::begin ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`const_iterator begin () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.5.3 cbegin

[SWS_CORE_01448] Definition of API function ara::core::Map::cbegin ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`const_iterator cbegin () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.5.4 cend

[SWS_CORE_01449] Definition of API function ara::core::Map::cend ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`const_iterator cend () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.5.5 crbegin

[SWS_CORE_01450] Definition of API function ara::core::Map::crbegin ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`const_reverse_iterator crbegin () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.5.6 crend

[SWS_CORE_01451] Definition of API function ara::core::Map::crend ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`const_reverse_iterator crend () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.5.7 end() const

[SWS_CORE_01443] Definition of API function ara::core::Map::end ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`const_iterator end () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.5.8 end()

[SWS_CORE_01442] Definition of API function ara::core::Map::end ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`iterator end () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.5.9 rbegin() const

[SWS_CORE_01445] Definition of API function ara::core::Map::rbegin ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`const_reverse_iterator rbegin () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.5.10 rbegin()

[SWS_CORE_01444] Definition of API function ara::core::Map::rbegin ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`reverse_iterator rbegin () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.5.11 rend()

[SWS_CORE_01446] Definition of API function ara::core::Map::rend ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`reverse_iterator rend () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.5.12 rend() const

[SWS_CORE_01447] Definition of API function ara::core::Map::rend ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`const_reverse_iterator rend () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.6 Lookup

8.3.2.2.6.1 contains(const key_type&) const

[SWS_CORE_01510] Definition of API function ara::core::Map::contains ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`bool contains (const key_type &x) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.6.2 contains(const K&) const

[SWS_CORE_01497] Definition of API function ara::core::Map::contains ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`template <class K>` `bool contains (const K &x) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.6.3 count(const K&) const

[SWS_CORE_01495] Definition of API function ara::core::Map::count ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`template <class K>` `size_type count (const K &x) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.6.4 count(const key_type&) const

[SWS_CORE_01494] Definition of API function ara::core::Map::count ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`size_type count (const key_type &x) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.6.5 equal_range(const K&) const

[SWS_CORE_01509] Definition of API function ara::core::Map::equal_range ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`template <class K>` `std::pair< const_iterator, const_iterator > equal_range (const K &x) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.6.6 equal_range(const K&)

[SWS_CORE_01508] Definition of API function ara::core::Map::equal_range ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`template <class K>` `std::pair< iterator, iterator > equal_range (const K &x);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.6.7 equal_range(const key_type&)

[SWS_CORE_01506] Definition of API function ara::core::Map::equal_range ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`std::pair< iterator, iterator > equal_range (const key_type &x);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.6.8 equal_range(const key_type&) const

[SWS_CORE_01507] Definition of API function ara::core::Map::equal_range ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`std::pair< const_iterator, const_iterator > equal_range (const key_type &x) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.6.9 find(const key_type&)

[SWS_CORE_01490] Definition of API function ara::core::Map::find ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`iterator find (const key_type &x);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.6.10 find(const K&) const

[SWS_CORE_01493] Definition of API function ara::core::Map::find ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`template <class K>` `const_iterator find (const K &x) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.6.11 find(const K&)

[SWS_CORE_01492] Definition of API function ara::core::Map::find ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`template <class K>` `iterator find (const K &x);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.6.12 find(const key_type&) const

[SWS_CORE_01491] Definition of API function ara::core::Map::find ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`const_iterator find (const key_type &x) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.6.13 lower_bound(const key_type&)

[SWS_CORE_01498] Definition of API function ara::core::Map::lower_bound ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`iterator lower_bound (const key_type &x);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.6.14 lower_bound(const K&)

[SWS_CORE_01500] Definition of API function ara::core::Map::lower_bound ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`template <class K>` `iterator lower_bound (const K &x);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.6.15 lower_bound(const K&) const

[SWS_CORE_01501] Definition of API function ara::core::Map::lower_bound ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`template <class K>` `const_iterator lower_bound (const K &x) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.6.16 lower_bound(const key_type&) const

[SWS_CORE_01499] Definition of API function ara::core::Map::lower_bound ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`const_iterator lower_bound (const key_type &x) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.6.17 upper_bound(const K&)

[SWS_CORE_01504] Definition of API function ara::core::Map::upper_bound ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`template <class K>` `iterator upper_bound (const K &x);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.6.18 upper_bound(const key_type&) const

[SWS_CORE_01503] Definition of API function ara::core::Map::upper_bound ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`const_iterator upper_bound (const key_type &x) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.6.19 upper_bound(const key_type&)

[SWS_CORE_01502] Definition of API function ara::core::Map::upper_bound ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`iterator upper_bound (const key_type &x);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.6.20 upper_bound(const K&) const

[SWS_CORE_01505] Definition of API function ara::core::Map::upper_bound ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`template <class K>` `const_iterator upper_bound (const K &x) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.7 Modifiers

8.3.2.2.7.1 clear

[SWS_CORE_01485] Definition of API function ara::core::Map::clear ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`void clear () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.7.2 emplace(Args ...&&)

[SWS_CORE_01458] Definition of API function ara::core::Map::emplace ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`template <class... Args>` `std::pair< iterator, bool > emplace (Args &&... args);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.7.3 emplace_hint(const_iterator, Args ...&&)

[SWS_CORE_01459] Definition of API function ara::core::Map::emplace_hint ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`template <class... Args>` `iterator emplace_hint (const_iterator position, Args &&... args);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.7.4 erase(const_iterator, const_iterator)

[SWS_CORE_01483] Definition of API function ara::core::Map::erase ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`iterator erase (const_iterator first, const_iterator last);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.7.5 erase(const_iterator)

[SWS_CORE_01481] Definition of API function ara::core::Map::erase ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`iterator erase (const_iterator position);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.7.6 erase(const key_type&)

[SWS_CORE_01482] Definition of API function ara::core::Map::erase ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`size_type erase (const key_type &x);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.7.7 erase(iterator)

[SWS_CORE_01480] Definition of API function ara::core::Map::erase ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`iterator erase (iterator position);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.7.8 extract(const_iterator)

[SWS_CORE_01470] Definition of API function ara::core::Map::extract ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`node_type extract (const_iterator position);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.7.9 extract(const key_type&)

[SWS_CORE_01471] Definition of API function ara::core::Map::extract ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`node_type extract (const key_type &x);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.7.10 insert(std::initializer_list<value_type>)

[SWS_CORE_01467] Definition of API function ara::core::Map::insert ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`void insert (std::initializer_list< value_type >);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.7.11 insert(const_iterator, P&&)

[SWS_CORE_01465] Definition of API function ara::core::Map::insert ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`template <class P>` `iterator insert (const_iterator position, P &&);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.7.12 insert(const_iterator, node_type&&)

[SWS_CORE_01469] Definition of API function ara::core::Map::insert ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`iterator insert (const_iterator hint, node_type &&nh);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.7.13 insert(const_iterator, value_type&&)

[SWS_CORE_01464] Definition of API function ara::core::Map::insert ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`iterator insert (const_iterator position, value_type &&x);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.7.14 insert(const_iterator, const value_type&)

[SWS_CORE_01463] Definition of API function ara::core::Map::insert ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`iterator insert (const_iterator position, const value_type &x);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.7.15 insert(P&&)

[SWS_CORE_01462] Definition of API function ara::core::Map::insert ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`template <class P>` `std::pair< iterator, bool > insert (P &&x);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.7.16 insert(InputIterator, InputIterator)

[SWS_CORE_01466] Definition of API function ara::core::Map::insert ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`template <class InputIterator>` `void insert (InputIterator first, InputIterator last);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.7.17 insert(value_type&&)

[SWS_CORE_01461] Definition of API function ara::core::Map::insert ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`std::pair< iterator, bool > insert (value_type &&x);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.7.18 insert(node_type&&)

[SWS_CORE_01468] Definition of API function ara::core::Map::insert ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`insert_return_type insert (node_type &&nh);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.7.19 insert(const value_type&)

[SWS_CORE_01460] Definition of API function ara::core::Map::insert ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`std::pair< iterator, bool > insert (const value_type &x);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.7.20 insert_or_assign(const_iterator, key_type&&, M&&)

[SWS_CORE_01479] Definition of API function ara::core::Map::insert_or_assign ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`template <class M>` `iterator insert_or_assign (const_iterator hint, key_type &&k, M &&obj);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.7.21 insert_or_assign(const key_type&, M&&)

[SWS_CORE_01476] Definition of API function ara::core::Map::insert_or_assign

⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`template <class M>` `std::pair< iterator, bool > insert_or_assign (const key_type &k, M &&obj);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.7.22 insert_or_assign(key_type&&, M&&)

[SWS_CORE_01477] Definition of API function ara::core::Map::insert_or_assign

⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`template <class M>` `std::pair< iterator, bool > insert_or_assign (key_type &&k, M &&obj);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.7.23 insert_or_assign(const_iterator, const key_type&, M&&)

[SWS_CORE_01478] Definition of API function ara::core::Map::insert_or_assign

⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`template <class M>` `iterator insert_or_assign (const_iterator hint, const key_type &k, M &&obj);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.7.24 merge(Map<Key, T, C2, Allocator>&)

[SWS_CORE_01486] Definition of API function ara::core::Map::merge ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`template <class C2>` `void merge (Map< Key, T, C2, Allocator > &source);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.7.25 merge(Map<Key, T, C2, Allocator>&&)

[SWS_CORE_01487] Definition of API function ara::core::Map::merge ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`template <class C2>` `void merge (Map< Key, T, C2, Allocator > &&source);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.7.26 swap

[SWS_CORE_01484] Definition of API function ara::core::Map::swap ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`void swap (Map &) noexcept(std::allocator_traits< Allocator >::is_always_equal::value &&ara::core::is_nothrow_swappable< Compare >::value);`
Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[map]` except for the following deviations: 1. `ara::core::is_nothrow_swappable` shall be used in place of the `std::` implementation

⌋

8.3.2.2.7.27 try_emplace(key_type&&, Args ...&&)

[SWS_CORE_01473] Definition of API function ara::core::Map::try_emplace ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`template <class... Args>` `std::pair< iterator, bool > try_emplace (key_type &&k, Args &&... args);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.7.28 try_emplace(const key_type&, Args ...&&)

[SWS_CORE_01472] Definition of API function ara::core::Map::try_emplace ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`template <class... Args>` `std::pair< iterator, bool > try_emplace (const key_type &k, Args &&... args);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.7.29 try_emplace(const_iterator, const key_type&, Args ...&&)

[SWS_CORE_01474] Definition of API function ara::core::Map::try_emplace ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`template <class... Args>` `iterator try_emplace (const_iterator hint, const key_type &k, Args &&... args);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.7.30 try_emplace(const_iterator, key_type&&, Args ...&&)

[SWS_CORE_01475] Definition of API function ara::core::Map::try_emplace ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`template <class... Args>` `iterator try_emplace (const_iterator hint, key_type &&k, Args &&... args);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.8 Observers

8.3.2.2.8.1 key_comp

[SWS_CORE_01488] Definition of API function ara::core::Map::key_comp ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`key_compare key_comp () const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.8.2 value_comp

[SWS_CORE_01489] Definition of API function ara::core::Map::value_comp ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`value_compare value_comp () const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.9 Operator=

8.3.2.2.9.1 operator=

[SWS_CORE_01438] Definition of API function ara::core::Map::operator= ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`Map & operator= (std::initializer_list< value_type >);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.2.2.10 Other

8.3.2.2.10.1 get_allocator

[SWS_CORE_01439] Definition of API function ara::core::Map::get_allocator ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map`
Syntax:	`allocator_type get_allocator () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.3 Class: value_compare

[SWS_CORE_01419] Definition of API class ara::core::Map::value_compare ⌈

Kind:	class
Header file:	#include "ara/core/map.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	`class ara::core::Map`
Symbol:	value_compare
Syntax:	`class value_compare {...};`
Description:	As per corresponding usage in `[map]`

⌋

8.3.3.1 Protected Member Variables

8.3.3.1.1 comp

[SWS_CORE_01421] Definition of API variable ara::core::Map::value_compare::comp ⌈

Kind:	variable
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map::value_compare`
Symbol:	comp
Type:	Compare
Syntax:	`Compare ara::core::Map< Key, T, Compare, Allocator >::value_compare::comp;`
Description:	As per corresponding usage in `[map]`
Visibility:	protected

⌋

8.3.3.2 Public Member Functions

8.3.3.2.1 Member Functions

8.3.3.2.1.1 operator()

[SWS_CORE_01423] Definition of API function ara::core::Map::value_compare::operator() ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map::value_compare`
Syntax:	`bool ara::core::Map< Key, T, Compare, Allocator >::value_compare::operator() (const value_type &x, const value_type &y) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`

⌋

8.3.3.3 Protected Member Functions

8.3.3.3.1 Constructors

8.3.3.3.1.1 value_compare

[SWS_CORE_01422] Definition of API function ara::core::Map::value_compare::value_compare ⌈

Kind:	function
Header file:	#include "ara/core/map.h"
Scope:	`class ara::core::Map::value_compare`
Syntax:	`ara::core::Map< Key, T, Compare, Allocator >::value_compare::value_compare (Compare c);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[map]`
Visibility:	protected

⌋

8.4 Header: ara/core/memory_resource.h

[SWS_CORE_06578] Definition of Header File ara/core/memory_resource.h

Upstream requirements: RS_AP_00130

⌈

Kind:	Header File
Syntax:	`ara/core/memory_resource.h`
Description:	Implements `[mem.res]` in [15]. Unless explicitly overridden in the API documentation, the APIs in this library always adhere in behavior to the ISO specification. To reduce duplication, the APIs here specify only any deviations from [15] or any AUTOSAR specific overrides. For APIs which would ordinarily throw `Exception`s, and those `Exception`s have been suppressed in favor of AUTOSAR violations, generally the semantics of the `Exception` from [15] still apply (unless locally overridden here).

⌋

8.4.1 Non-Member Functions

8.4.1.1 Other

8.4.1.1.1 GetDefaultResource

[SWS_CORE_06565] Definition of API function ara::core::GetDefaultResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	namespace ara::core
Syntax:	`MemoryResource * GetDefaultResource () noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.res]` except for the following deviations: 1. Description references to `std::pmr::new_delete_resource()` are replaced by `ara::core::NewDeleteResource()` 2. Description references to `std::pmr::set_default_resource()` are replaced by `ara::core::SetDefaultResource()` 3. Return type `std::pmr::memory_resource` is replaced with `ara::core::MemoryResource`

⌋

8.4.1.1.2 NewDeleteResource

[SWS_CORE_06562] Definition of API function ara::core::NewDeleteResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	namespace ara::core
Syntax:	`MemoryResource * NewDeleteResource () noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.res]` except for the following deviations: 1. Return type `std::pmr::memory_resource` is replaced with `ara::core::MemoryResource`

⌋

8.4.1.1.3 NullMemoryResource

[SWS_CORE_06563] Definition of API function ara::core::NullMemoryResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	namespace ara::core
Syntax:	`MemoryResource * NullMemoryResource () noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.res]` except for the following deviations: 1. Return type `std::pmr::memory_resource` is replaced with `ara::core::MemoryResource` 2. It's `allocate()` function returns `nullptr` instead of throwing `std::bad_alloc`

⌋

8.4.1.1.4 SetDefaultResource

[SWS_CORE_06564] Definition of API function ara::core::SetDefaultResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	namespace ara::core
Syntax:	`MemoryResource * SetDefaultResource (MemoryResource *r) noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.res]` except for the following deviations: 1. Description references to `std::pmr::new_delete_resource()` are replaced by `ara::core::NewDeleteResource()` 2. Description references to `std::pmr::get_default_resource()` are replaced by `ara::core::GetDefaultResource()` 3. Return type `std::pmr::memory_resource` is replaced with `ara::core::MemoryResource`

⌋

8.4.1.1.5 operator==

[SWS_CORE_06561] Definition of API function ara::core::operator==

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	namespace ara::core
Syntax:	`bool operator== (const MemoryResource &a, const MemoryResource &b) noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.res]`

⌋

8.4.1.1.6 operator==

[SWS_CORE_06560] Definition of API function ara::core::operator==

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	namespace ara::core
Syntax:	`template <class T1, class T2>` `bool operator== (const PolymorphicAllocator< T1 > &a, const PolymorphicAllocator< T2 > &b) noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.res]`

⌋

8.4.2 Class: MemoryResource

[SWS_CORE_06500] Definition of API class ara::core::MemoryResource

Upstream requirements: RS_AP_00130

⌈

Kind:	class
Header file:	#include "ara/core/memory_resource.h"
Forwarding header file:	#include "ara/core/core_fwd.h"

▽

△

Scope:	namespace ara::core
Symbol:	MemoryResource
Syntax:	`class MemoryResource {...};`
Description:	Implementation of `std::pmr::memory_resource` type from `[mem.res.class]`

⌋

8.4.2.1 Public Member Functions

8.4.2.1.1 Special Member Functions

8.4.2.1.1.1 Default Constructor

[SWS_CORE_06501] Definition of API function ara::core::MemoryResource::MemoryResource ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MemoryResource`
Syntax:	`MemoryResource () noexcept=default;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[mem.res]` except for the following deviations: 1. The function is `noexcept`

⌋

8.4.2.1.1.2 Move Constructor

[SWS_CORE_06574] Definition of API function ara::core::MemoryResource::MemoryResource

Upstream requirements: RS_AP_00145

⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	`class ara::core::MemoryResource`	
Syntax:	`MemoryResource (MemoryResource &&other) noexcept=default;`	
Parameters (in):	other	other MemoryResource to move from
Exception Safety:	exception safe	
Description:	**[AUTOSAR defined]** not present in ISO-14882. Move constructor	

⌋

8.4.2.1.1.3 Copy Constructor

[SWS_CORE_06502] Definition of API function ara::core::MemoryResource::MemoryResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MemoryResource`
Syntax:	`MemoryResource (const MemoryResource &other) noexcept=default;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[mem.res]` except for the following deviations: 1. The function is `noexcept`

⌋

8.4.2.1.1.4 Move Assignment Operator

[SWS_CORE_06575] Definition of API function ara::core::MemoryResource::operator=

Upstream requirements: RS_AP_00145

⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	`class ara::core::MemoryResource`	
Syntax:	`MemoryResource & operator= (MemoryResource &&other) noexcept=default;`	
Parameters (in):	other	other MemoryResource to move from
Exception Safety:	exception safe	
Description:	**[AUTOSAR defined]** not present in ISO-14882. Move assignment operator	

⌋

8.4.2.1.1.5 Copy Assignment Operator

[SWS_CORE_06507] Definition of API function ara::core::MemoryResource::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MemoryResource`
Syntax:	`MemoryResource & operator= (const MemoryResource &other) noexcept=default;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[mem.res]` except for the following deviations: 1. The function is `noexcept`

⌋

8.4.2.1.1.6 Destructor

[SWS_CORE_06506] Definition of API function ara::core::MemoryResource::~MemoryResource ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MemoryResource`
Syntax:	`virtual ~MemoryResource () noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding usage in `[mem.res]` except for the following deviations: 1. The function is `noexcept`

⌋

8.4.2.1.2 Member Functions

8.4.2.1.2.1 allocate

[SWS_CORE_06503] Definition of API function ara::core::MemoryResource::allocate ⌈

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

▽

△

Syntax:	`[[nodiscard]] void * allocate (std::size_t bytes, std::size_t alignment=alignof(std::max_align_t)) noexcept;`	
Return value:	void *	• If successful: As per corresponding usage in `[mem.res]` • If unsuccessful: the returned pointer shall be `nullptr`
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	As per corresponding usage in `[mem.res]` except for the following deviations: 1. The function is `noexcept` 2. The function returns `nullptr` instead of (potentially) throwing a `std::bad_alloc` exception	

⌋

8.4.2.1.2.2 deallocate

[SWS_CORE_06504] Definition of API function ara::core::MemoryResource::deallocate ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MemoryResource`
Syntax:	`void deallocate (void *p, std::size_t bytes, std::size_t alignment=max_align) noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding usage in `[mem.res]` except for the following deviations: 1. The function is `noexcept` 2. If unsuccessful: errors will be silently ignored

⌋

8.4.2.1.2.3 is_equal

[SWS_CORE_06505] Definition of API function ara::core::MemoryResource::is_equal ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MemoryResource`
Syntax:	`bool is_equal (const MemoryResource &other) const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.res]`

⌋

8.4.2.2 Private Member Functions

8.4.2.2.1 Member Functions

8.4.2.2.1.1 do_allocate

[SWS_CORE_06566] Definition of API function ara::core::MemoryResource::do_allocate ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	`class ara::core::MemoryResource`	
Syntax:	`virtual void * do_allocate (std::size_t bytes, std::size_t alignment) noexcept=0;`	
Return value:	void *	• If successful: As per corresponding usage in `[mem.res]` • If unsuccessful: the returned pointer shall be `nullptr`
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	As per corresponding usage in `[mem.res]` in [15] except for the following deviations: 1. The function is `noexcept` 2. The function returns `nullptr` instead of (potentially) throwing a `std::bad_alloc` exception	
Visibility:	private	

⌋

8.4.2.2.1.2 do_deallocate

[SWS_CORE_06567] Definition of API function ara::core::MemoryResource::do_deallocate ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MemoryResource`
Syntax:	`virtual void do_deallocate (void *p, std::size_t bytes, std::size_t alignment) noexcept=0;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding usage in `[mem.res]` except for the following deviations: 1. The function is `noexcept` 2. If unsuccessful: errors will be silently ignored
Visibility:	private

⌋

8.4.2.2.1.3 do_is_equal

[SWS_CORE_06568] Definition of API function ara::core::MemoryResource::do_is_equal ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MemoryResource`
Syntax:	`virtual bool do_is_equal (const MemoryResource &other) const noexcept=0;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.res]`
Visibility:	private

⌋

8.4.3 Class: MonotonicBufferResource

[SWS_CORE_06520] Definition of API class ara::core::MonotonicBufferResource

Upstream requirements: RS_AP_00130

⌈

Kind:	class
Header file:	#include "ara/core/memory_resource.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	namespace ara::core
Symbol:	MonotonicBufferResource
Base class:	MemoryResource
Syntax:	`class MonotonicBufferResource : public MemoryResource {...};`
Description:	Implementation of `std::pmr::monotonic_buffer_resource` type from `[mem.res.monotonic.buffer]`

⌋

8.4.3.1 Public Member Functions

8.4.3.1.1 Special Member Functions

8.4.3.1.1.1 Move Constructor

[SWS_CORE_06576] Definition of API function ara::core::MonotonicBufferResource::MonotonicBufferResource

Upstream requirements: RS_AP_00145

⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	`class ara::core::MonotonicBufferResource`	
Syntax:	`MonotonicBufferResource (MonotonicBufferResource &&other) noexcept=default;`	
Parameters (in):	other	other MonotonicBufferResource to move from
Exception Safety:	exception safe	
Description:	**[AUTOSAR defined]** not present in ISO-14882. Move constructor	

⌋

8.4.3.1.1.2 Default Constructor

[SWS_CORE_06524] Definition of API function ara::core::MonotonicBufferResource::MonotonicBufferResource ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MonotonicBufferResource`
Syntax:	`MonotonicBufferResource () noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.res.monotonic.buffer]` except for the following deviations: 1. The function is `noexcept`

⌋

8.4.3.1.1.3 Copy Constructor

[SWS_CORE_06527] Definition of API function ara::core::MonotonicBufferResource::MonotonicBufferResource ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MonotonicBufferResource`
Syntax:	`MonotonicBufferResource (const MonotonicBufferResource &)=delete;`
Description:	As per corresponding usage in `[mem.res.monotonic.buffer]`

⌋

8.4.3.1.1.4 Copy Assignment Operator

[SWS_CORE_06529] Definition of API function ara::core::MonotonicBufferResource::operator= ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MonotonicBufferResource`
Syntax:	`MonotonicBufferResource & operator= (const MonotonicBufferResource &)=delete;`
Description:	As per corresponding usage in `[mem.res.monotonic.buffer]`

⌋

8.4.3.1.1.5 Move Assignment Operator

[SWS_CORE_06577] Definition of API function ara::core::MonotonicBufferResource::operator=

Upstream requirements: RS_AP_00145

⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	`class ara::core::MonotonicBufferResource`	
Syntax:	`MonotonicBufferResource & operator= (MonotonicBufferResource &&other) noexcept=default;`	
Parameters (in):	other	other MonotonicBufferResource to move from
Exception Safety:	exception safe	
Description:	**[AUTOSAR defined]** not present in ISO-14882. Move assignment operator	

⌋

8.4.3.1.1.6 Destructor

[SWS_CORE_06528] Definition of API function ara::core::MonotonicBufferResource::~MonotonicBufferResource ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MonotonicBufferResource`
Syntax:	`~MonotonicBufferResource () noexcept override;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding usage in `[mem.res.monotonic.buffer]` except for the following deviations: 1. The function is `noexcept` 2. The function is not `virtual`

⌋

8.4.3.1.2 Constructors

8.4.3.1.2.1 MonotonicBufferResource(std::size_t)

[SWS_CORE_06525] Definition of API function ara::core::MonotonicBufferResource::MonotonicBufferResource ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MonotonicBufferResource`
Syntax:	`explicit MonotonicBufferResource (std::size_t initial_size) noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding usage in `[mem.res.monotonic.buffer]` except for the following deviations: 1. The function is `noexcept`

⌋

8.4.3.1.2.2 MonotonicBufferResource(void *, std::size_t, MemoryResource *)

[SWS_CORE_06523] Definition of API function ara::core::MonotonicBufferResource::MonotonicBufferResource ⌈

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

▽

△

Syntax:	`MonotonicBufferResource (void *buffer, std::size_t buffer_size, MemoryResource *upstream) noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding usage in `[mem.res.monotonic.buffer]` except for the following deviations: 1. The function is `noexcept`

⌋

8.4.3.1.2.3 MonotonicBufferResource(MemoryResource *)

[SWS_CORE_06521] Definition of API function ara::core::MonotonicBufferResource::MonotonicBufferResource ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MonotonicBufferResource`
Syntax:	`explicit MonotonicBufferResource (MemoryResource *upstream) noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding usage in `[mem.res.monotonic.buffer]` except for the following deviations: 1. The function is `noexcept`

⌋

8.4.3.1.2.4 MonotonicBufferResource(std::size_t, MemoryResource *)

[SWS_CORE_06522] Definition of API function ara::core::MonotonicBufferResource::MonotonicBufferResource ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MonotonicBufferResource`
Syntax:	`MonotonicBufferResource (std::size_t initial_size, MemoryResource *upstream) noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding usage in `[mem.res.monotonic.buffer]` except for the following deviations: 1. The function is `noexcept`

⌋

8.4.3.1.2.5 MonotonicBufferResource

[SWS_CORE_06526] Definition of API function ara::core::MonotonicBufferResource::MonotonicBufferResource ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MonotonicBufferResource`
Syntax:	`MonotonicBufferResource (void *buffer, std::size_t buffer_size) noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding usage in `[mem.res.monotonic.buffer]` except for the following deviations: 1. The function is `noexcept`

⌋

8.4.3.1.3 Member Functions

8.4.3.1.3.1 release

[SWS_CORE_06530] Definition of API function ara::core::MonotonicBufferResource::release ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MonotonicBufferResource`
Syntax:	`void release () noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding usage in `[mem.res.monotonic.buffer]` except for the following deviations: 1. The function is `noexcept`

⌋

8.4.3.1.3.2 upstream_resource

[SWS_CORE_06531] Definition of API function ara::core::MonotonicBufferResource::upstream_resource ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MonotonicBufferResource`
Syntax:	`MemoryResource * upstream_resource () const noexcept;`
Exception Safety:	exception safe

▽

△

Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.res.monotonic.buffer]` except for the following deviations: 1. The function is `noexcept`

⌋

8.4.3.2 Protected Member Functions

8.4.3.2.1 Member Functions

8.4.3.2.1.1 do_allocate

[SWS_CORE_06569] Definition of API function ara::core::MonotonicBufferResource::do_allocate ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	`class ara::core::MonotonicBufferResource`	
Syntax:	`void * do_allocate (std::size_t bytes, std::size_t alignment) noexcept override;`	
Return value:	void *	• If successful: As per corresponding usage in `[mem.res.monotonic.buffer]` • If unsuccessful: the returned pointer shall be `nullptr`
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	As per corresponding usage in `[mem.res.monotonic.buffer]` except for the following deviations: 1. The function is `noexcept` 2. The function returns `nullptr` instead of (potentially) throwing a `std::bad_alloc` exception	
Visibility:	protected	

⌋

8.4.3.2.1.2 do_deallocate

[SWS_CORE_06570] Definition of API function ara::core::MonotonicBufferResource::do_deallocate ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MonotonicBufferResource`
Syntax:	`void do_deallocate (void *p, std::size_t bytes, std::size_t alignment) noexcept override;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe

▽

△

Description:	As per corresponding usage in `[mem.res.monotonic.buffer]` except for the following deviations: 1. The function is `noexcept` 2. If unsuccessful: errors will be silently ignored
Visibility:	protected

⌋

8.4.3.2.1.3 do_is_equal

[SWS_CORE_06571] Definition of API function ara::core::MonotonicBufferResource::do_is_equal ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::MonotonicBufferResource`
Syntax:	`bool do_is_equal (const MemoryResource &other) const noexcept override;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.res.monotonic.buffer]` except for the following deviations: 1. The function is `noexcept`
Visibility:	protected

⌋

8.4.4 Class: PolymorphicAllocator

[SWS_CORE_06540] Definition of API class ara::core::PolymorphicAllocator

Upstream requirements: RS_AP_00130

⌈

Kind:	class
Header file:	#include "ara/core/memory_resource.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	namespace ara::core
Symbol:	PolymorphicAllocator
Syntax:	`template <class Tp = ara::core::Byte>` `class PolymorphicAllocator {...};`
Description:	Implementation of `std::pmr::polymorphic_allocator` from `[mem.poly.allocator.class]`

⌋

8.4.4.1 Public Member Types

8.4.4.1.1 Type Alias: value_type

[SWS_CORE_06572] Definition of API type ara::core::PolymorphicAllocator::value_type ⌈

Kind:	type alias
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`
Symbol:	value_type
Syntax:	`using value_type = Tp;`
Description:	As per corresponding usage in `[mem.poly.allocator.class]`

⌋

8.4.4.2 Public Member Functions

8.4.4.2.1 Special Member Functions

8.4.4.2.1.1 Copy Constructor

[SWS_CORE_06544] Definition of API function ara::core::PolymorphicAllocator::PolymorphicAllocator ⌈

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

Syntax:	`template <class U>`
	`PolymorphicAllocator (const PolymorphicAllocator< U > &other)`
	`noexcept;`

Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.poly.allocator.class]`

⌋

8.4.4.2.1.2 Default Constructor

[SWS_CORE_06541] Definition of API function ara::core::PolymorphicAllocator::PolymorphicAllocator ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`
Syntax:	`PolymorphicAllocator () noexcept;`

▽

△

Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.poly.allocator.class]`

⌋

8.4.4.2.1.3 Copy Constructor

[SWS_CORE_06543] Definition of API function ara::core::PolymorphicAllocator::PolymorphicAllocator ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`
Syntax:	`PolymorphicAllocator (const PolymorphicAllocator &other) noexcept=default;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[mem.poly.allocator.class]` except for the following deviations: 1. The function is `noexcept`

⌋

8.4.4.2.1.4 Move Constructor

[SWS_CORE_06546] Definition of API function ara::core::PolymorphicAllocator::PolymorphicAllocator

Upstream requirements: RS_AP_00145

⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	`class ara::core::PolymorphicAllocator`	
Syntax:	`PolymorphicAllocator (PolymorphicAllocator &&other) noexcept;`	
Parameters (in):	other	other PolymorphicAllocator to move from
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	**[AUTOSAR defined]** not present in ISO-14882. Move constructor	

⌋

8.4.4.2.1.5 Move Assignment Operator

[SWS_CORE_00066] Definition of API function ara::core::PolymorphicAllocator::operator=

Upstream requirements: RS_AP_00145

⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	`class ara::core::PolymorphicAllocator`	
Syntax:	`PolymorphicAllocator & operator= (PolymorphicAllocator &&other) noexcept=default;`	
Parameters (in):	other	other PolymorphicAllocator to move from
Exception Safety:	exception safe	
Description:	**[AUTOSAR defined]** not present in ISO-14882. Move assignment operator	

⌋

8.4.4.2.1.6 Copy Assignment Operator

[SWS_CORE_06545] Definition of API function ara::core::PolymorphicAllocator::operator= ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`
Syntax:	`PolymorphicAllocator & operator= (const PolymorphicAllocator &)=delete;`
Description:	As per corresponding usage in `[mem.poly.allocator.class]`

⌋

8.4.4.2.2 Constructors

8.4.4.2.2.1 PolymorphicAllocator

[SWS_CORE_06542] Definition of API function ara::core::PolymorphicAllocator::PolymorphicAllocator ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`
Syntax:	`PolymorphicAllocator (MemoryResource *r) noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe

▽

△

Description:	As per corresponding usage in `[mem.poly.allocator.class]` except for the following deviations: 1. The function is `noexcept`

⌋

8.4.4.2.3 Member Functions

8.4.4.2.3.1 allocate

[SWS_CORE_06547] Definition of API function ara::core::PolymorphicAllocator::allocate ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	`class ara::core::PolymorphicAllocator`	
Syntax:	`[[nodiscard]] Tp * allocate (std::size_t n) noexcept;`	
Return value:	Tp *	• If successful: As per corresponding usage in `[mem.poly.allocator.class]` • If unsuccessful: the returned pointer shall be `nullptr`
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	As per corresponding usage in `[mem.poly.allocator.class]` except for the following deviations: 1. The function is `noexcept` 2. If unsuccessful: `nullptr` is returned, instead of raising any `Exception`	

⌋

8.4.4.2.3.2 allocate_bytes

[SWS_CORE_06549] Definition of API function ara::core::PolymorphicAllocator::allocate_bytes ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`
Syntax:	`[[nodiscard]] void * allocate_bytes (std::size_t nbytes, std::size_t alignment=alignof(std::max_align_t)) noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding usage in `[mem.poly.allocator.class]` except for the following deviations: 1. The function is `noexcept` 2. If unsuccessful: the returned pointer shall be `nullptr`

⌋

8.4.4.2.3.3 allocate_object

[SWS_CORE_06551] Definition of API function ara::core::PolymorphicAllocator::allocate_object ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`
Syntax:	`template <class T>` `[[nodiscard]] T * allocate_object (std::size_t n=1) noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding usage in `[mem.poly.allocator.class]` except for the following deviations: 1. The function is `noexcept` 2. If unsuccessful: the returned pointer shall be `nullptr`

⌋

8.4.4.2.3.4 construct

[SWS_CORE_06555] Definition of API function ara::core::PolymorphicAllocator::construct ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`
Syntax:	`template <class T, class... Args>` `void construct (T *p, Args &&... args) noexcept(std::is_nothrow_constructible< T, Args... >::value);`
Exception Safety:	conditionally exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding usage in `[mem.poly.allocator.class]` except for the following deviations: 1. Function is conditionally `noexcept`

⌋

8.4.4.2.3.5 deallocate

[SWS_CORE_06548] Definition of API function ara::core::PolymorphicAllocator::deallocate ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`
Syntax:	`void deallocate (Tp *p, std::size_t n) noexcept;`

▽

△

Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding usage in `[mem.poly.allocator.class]` except for the following deviations: 1. The function is `noexcept` 2. If unsuccessful: errors will be silently ignored

⌋

8.4.4.2.3.6 deallocate_bytes

[SWS_CORE_06550] Definition of API function ara::core::PolymorphicAllocator::deallocate_bytes ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`
Syntax:	`void deallocate_bytes (void *p, std::size_t nbytes, std::size_t alignment=alignof(std::max_align_t)) noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding usage in `[mem.poly.allocator.class]` except for the following deviations: 1. The function is `noexcept` 2. If unsuccessful: errors will be silently ignored

⌋

8.4.4.2.3.7 deallocate_object

[SWS_CORE_06552] Definition of API function ara::core::PolymorphicAllocator::deallocate_object ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`
Syntax:	`template <class T>` `void deallocate_object (T *p, std::size_t n=1) noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding usage in `[mem.poly.allocator.class]` except for the following deviations: 1. Function is `nullptr` 2. If unsuccessful: errors will be silently ignored

⌋

8.4.4.2.3.8 delete_object

[SWS_CORE_06554] Definition of API function ara::core::PolymorphicAllocator::delete_object ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`
Syntax:	`template <class T>` `void delete_object (T *p) noexcept(std::is_nothrow_destructible< T >::value);`
Exception Safety:	conditionally exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding usage in `[mem.poly.allocator.class]` except for the following deviations: 1. Function is conditionally `noexcept` 2. If unsuccessful: errors will be silently ignored

⌋

8.4.4.2.3.9 destroy

[SWS_CORE_06556] Definition of API function ara::core::PolymorphicAllocator::destroy ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`
Syntax:	`template <class T>` `void destroy (T *p) noexcept(std::is_nothrow_destructible< T >::value);`
Exception Safety:	conditionally exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding usage in `[mem.poly.allocator.class]` except for the following deviations: 1. Function is conditionally `noexcept`

⌋

8.4.4.2.3.10 new_object

[SWS_CORE_06553] Definition of API function ara::core::PolymorphicAllocator::new_object ⌈

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

▽

△

Syntax:	`template <class T, class... CtorArgs>`
	`[[nodiscard]] T * new_object (CtorArgs &&... ctor_args)`
	`noexcept(std::is_nothrow_constructible< T, CtorArgs... >::value);`	

Return value:	T *	• If successful: As per corresponding usage in `[mem.poly.allocator.class]` • If unsuccessful: the returned pointer shall be `nullptr`
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	As per corresponding usage in `[mem.poly.allocator.class]` except for the following deviations: 1. Function is conditionally `noexcept` 2. If unsuccessful: `nullptr` is returned, instead of raising any `Exception`	

⌋

8.4.4.2.3.11 resource

[SWS_CORE_06557] Definition of API function ara::core::PolymorphicAllocator::resource ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`
Syntax:	`MemoryResource * resource () const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.poly.allocator.class]`

⌋

8.4.4.2.3.12 select_on_container_copy_construction

[SWS_CORE_06573] Definition of API function ara::core::PolymorphicAllocator::select_on_container_copy_construction ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::PolymorphicAllocator`
Syntax:	`PolymorphicAllocator select_on_container_copy_construction () const noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding usage in `[mem.poly.allocator.class]` except for the following deviations: 1. The function is `noexcept`

⌋

8.4.5 Struct: PoolOptions

[SWS_CORE_00026] Definition of API class ara::core::PoolOptions

Upstream requirements: RS_AP_00130

⌈

Kind:	struct
Header file:	#include "ara/core/memory_resource.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	namespace ara::core
Symbol:	PoolOptions
Syntax:	`struct PoolOptions {...};`
Description:	Implementation of `std::pmr::pool_options` from `[mem.poly.pool_options.class]`

⌋

8.4.5.1 Public Member Variables

8.4.5.1.1 largest_required_pool_block

[SWS_CORE_00028] Definition of API variable ara::core::PoolOptions::largest_required_pool_block ⌈

Kind:	variable
Header file:	#include "ara/core/memory_resource.h"
Scope:	`struct ara::core::PoolOptions`
Symbol:	largest_required_pool_block
Type:	std::size_t
Syntax:	`std::size_t largest_required_pool_block = 0;`
Description:	As per corresponding usage in `[mem.res.pool]`

⌋

8.4.5.1.2 max_blocks_per_chunk

[SWS_CORE_00027] Definition of API variable ara::core::PoolOptions::max_blocks_per_chunk ⌈

Kind:	variable
Header file:	#include "ara/core/memory_resource.h"
Scope:	`struct ara::core::PoolOptions`
Symbol:	max_blocks_per_chunk
Type:	std::size_t
Syntax:	`std::size_t max_blocks_per_chunk = 0;`

▽

△

Description:	As per corresponding usage in `[mem.res.pool]`

⌋

8.4.6 Class: SynchronizedPoolResource

[SWS_CORE_00029] Definition of API class ara::core::SynchronizedPoolResource

Upstream requirements: RS_AP_00130

⌈

Kind:	class
Header file:	#include "ara/core/memory_resource.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	namespace ara::core
Symbol:	SynchronizedPoolResource
Base class:	MemoryResource
Syntax:	`class SynchronizedPoolResource : public MemoryResource {...};`
Description:	Implementation of `std::pmr::synchronized_pool_resource` from `[mem.res.pool]`

⌋

8.4.6.1 Public Member Functions

8.4.6.1.1 Special Member Functions

8.4.6.1.1.1 Move Constructor

[SWS_CORE_00043] Definition of API function ara::core::SynchronizedPoolResource::SynchronizedPoolResource

Upstream requirements: RS_AP_00145

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::SynchronizedPoolResource`
Syntax:	`SynchronizedPoolResource (SynchronizedPoolResource &&other) noexcept=default;`
Exception Safety:	exception safe
Description:	**[AUTOSAR defined]** not present in ISO-14882. Move constructor

⌋

8.4.6.1.1.2 Default Constructor

[SWS_CORE_00031] Definition of API function ara::core::SynchronizedPoolResource::SynchronizedPoolResource ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::SynchronizedPoolResource`
Syntax:	`SynchronizedPoolResource ();`
Exception Safety:	not exception safe
Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.res.pool]`

⌋

8.4.6.1.1.3 Copy Constructor

[SWS_CORE_00034] Definition of API function ara::core::SynchronizedPoolResource::SynchronizedPoolResource ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::SynchronizedPoolResource`
Syntax:	`SynchronizedPoolResource (const SynchronizedPoolResource &)=delete;`
Description:	As per corresponding usage in `[mem.res.pool]`

⌋

8.4.6.1.1.4 Move Assignment Operator

[SWS_CORE_00044] Definition of API function ara::core::SynchronizedPoolResource::operator=

Upstream requirements: RS_AP_00145

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::SynchronizedPoolResource`
Syntax:	`SynchronizedPoolResource & operator= (SynchronizedPoolResource &&other) noexcept=default;`
Exception Safety:	exception safe
Description:	**[AUTOSAR defined]** not present in ISO-14882. Move assignment operator

⌋

8.4.6.1.1.5 Copy Assignment Operator

[SWS_CORE_00036] Definition of API function ara::core::SynchronizedPoolResource::operator= ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::SynchronizedPoolResource`
Syntax:	`SynchronizedPoolResource & operator= (const SynchronizedPoolResource &)=delete;`
Description:	As per corresponding usage in `[mem.res.pool]`

⌋

8.4.6.1.1.6 Destructor

[SWS_CORE_00035] Definition of API function ara::core::SynchronizedPoolResource::~SynchronizedPoolResource ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::SynchronizedPoolResource`
Syntax:	`~SynchronizedPoolResource () noexcept override;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.res.pool]` except for the following deviations: 1. The function is not `virtual`

⌋

8.4.6.1.2 Constructors

8.4.6.1.2.1 SynchronizedPoolResource

[SWS_CORE_00032] Definition of API function ara::core::SynchronizedPoolResource::SynchronizedPoolResource ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::SynchronizedPoolResource`
Syntax:	`explicit SynchronizedPoolResource (MemoryResource *upstream);`
Exception Safety:	not exception safe
Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.res.pool]`

⌋

8.4.6.1.2.2 SynchronizedPoolResource

[SWS_CORE_00030] Definition of API function ara::core::SynchronizedPoolResource::SynchronizedPoolResource ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::SynchronizedPoolResource`
Syntax:	`SynchronizedPoolResource (const PoolOptions &opts, MemoryResource *upstream) noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.res.pool]`

⌋

8.4.6.1.2.3 SynchronizedPoolResource

[SWS_CORE_00033] Definition of API function ara::core::SynchronizedPoolResource::SynchronizedPoolResource ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::SynchronizedPoolResource`
Syntax:	`explicit SynchronizedPoolResource (const PoolOptions &opts);`
Exception Safety:	not exception safe
Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.res.pool]`

⌋

8.4.6.1.3 Member Functions

8.4.6.1.3.1 options

[SWS_CORE_00039] Definition of API function ara::core::SynchronizedPoolResource::options ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::SynchronizedPoolResource`
Syntax:	`PoolOptions options () const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.res.pool]`

⌋

8.4.6.1.3.2 release

[SWS_CORE_00037] Definition of API function ara::core::SynchronizedPoolResource::release ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::SynchronizedPoolResource`
Syntax:	`void release () noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.res.pool]`

⌋

8.4.6.1.3.3 upstream_resource

[SWS_CORE_00038] Definition of API function ara::core::SynchronizedPoolResource::upstream_resource ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::SynchronizedPoolResource`
Syntax:	`MemoryResource * upstream_resource () const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.res.pool]`

⌋

8.4.6.2 Protected Member Functions

8.4.6.2.1 Member Functions

8.4.6.2.1.1 do_allocate

[SWS_CORE_00061] Definition of API function ara::core::SynchronizedPoolResource::do_allocate ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::SynchronizedPoolResource`
Syntax:	`void * do_allocate (std::size_t bytes, std::size_t alignment) noexcept override;`

▽

△

Return value:	void *	• If successful: As per corresponding usage in `[mem.res.pool]` • If unsuccessful: the returned pointer shall be `nullptr`
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	As per corresponding usage in `[mem.res.pool]` except for the following deviations: 1. The function is `noexcept` 2. The function returns `nullptr` instead of (potentially) throwing a `std::bad_alloc` exception	
Visibility:	protected	

⌋

8.4.6.2.1.2 do_deallocate

[SWS_CORE_00041] Definition of API function ara::core::SynchronizedPoolResource::do_deallocate ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::SynchronizedPoolResource`
Syntax:	`void do_deallocate (void *p, std::size_t bytes, std::size_t alignment) noexcept override;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.res.pool]` except for the following deviations: 1. The function is `noexcept` 2. If unsuccessful: errors will be silently ignored
Visibility:	protected

⌋

8.4.6.2.1.3 do_is_equal

[SWS_CORE_00042] Definition of API function ara::core::SynchronizedPoolResource::do_is_equal ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::SynchronizedPoolResource`
Syntax:	`bool do_is_equal (const MemoryResource &other) const noexcept override;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.res.pool]`
Visibility:	protected

⌋

8.4.7 Class: UnsynchronizedPoolResource

[SWS_CORE_00045] Definition of API class ara::core::UnsynchronizedPoolResource

Upstream requirements: RS_AP_00130

⌈

Kind:	class
Header file:	#include "ara/core/memory_resource.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	namespace ara::core
Symbol:	UnsynchronizedPoolResource
Base class:	MemoryResource
Syntax:	`class UnsynchronizedPoolResource : public MemoryResource {...};`
Description:	Implementation of `std::pmr::unsynchronized_pool_resource` from `[mem.res.pool]`

⌋

8.4.7.1 Public Member Functions

8.4.7.1.1 Special Member Functions

8.4.7.1.1.1 Move Constructor

[SWS_CORE_00048] Definition of API function ara::core::UnsynchronizedPoolResource::UnsynchronizedPoolResource

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::UnsynchronizedPoolResource`
Syntax:	`UnsynchronizedPoolResource (UnsynchronizedPoolResource &&other) noexcept=default;`
Exception Safety:	exception safe
Description:	**[AUTOSAR defined]** not present in ISO-14882. Move constructor

⌋

8.4.7.1.1.2 Default Constructor

[SWS_CORE_00063] Definition of API function ara::core::UnsynchronizedPool Resource::UnsynchronizedPoolResource ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::UnsynchronizedPoolResource`
Syntax:	`UnsynchronizedPoolResource ();`
Exception Safety:	not exception safe
Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.res.pool]`

⌋

8.4.7.1.1.3 Copy Constructor

[SWS_CORE_00046] Definition of API function ara::core::UnsynchronizedPool Resource::UnsynchronizedPoolResource ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::UnsynchronizedPoolResource`
Syntax:	`UnsynchronizedPoolResource (const UnsynchronizedPoolResource &)=delete;`
Description:	As per corresponding usage in `[mem.res.pool]`

⌋

8.4.7.1.1.4 Copy Assignment Operator

[SWS_CORE_00047] Definition of API function ara::core::UnsynchronizedPool Resource::operator= ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::UnsynchronizedPoolResource`
Syntax:	`UnsynchronizedPoolResource & operator= (const UnsynchronizedPool Resource &)=delete;`
Description:	As per corresponding usage in `[mem.res.pool]`

⌋

8.4.7.1.1.5 Move Assignment Operator

[SWS_CORE_00049] Definition of API function ara::core::UnsynchronizedPool Resource::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::UnsynchronizedPoolResource`
Syntax:	`UnsynchronizedPoolResource & operator= (UnsynchronizedPoolResource &&other) noexcept=default;`
Exception Safety:	exception safe
Description:	**[AUTOSAR defined]** not present in ISO-14882. Move assignment operator

⌋

8.4.7.1.1.6 Destructor

[SWS_CORE_00062] Definition of API function ara::core::UnsynchronizedPool Resource::~UnsynchronizedPoolResource ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::UnsynchronizedPoolResource`
Syntax:	`~UnsynchronizedPoolResource () noexcept override;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.res.pool]` except for the following deviations: 1. The function is not `virtual`

⌋

8.4.7.1.2 Constructors

8.4.7.1.2.1 UnsynchronizedPoolResource

[SWS_CORE_00064] Definition of API function ara::core::UnsynchronizedPool Resource::UnsynchronizedPoolResource ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::UnsynchronizedPoolResource`
Syntax:	`explicit UnsynchronizedPoolResource (MemoryResource *upstream);`
Exception Safety:	not exception safe

▽

△

Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.res.pool]`

⌋

8.4.7.1.2.2 UnsynchronizedPoolResource

[SWS_CORE_00067] Definition of API function ara::core::UnsynchronizedPool Resource::UnsynchronizedPoolResource ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::UnsynchronizedPoolResource`
Syntax:	`UnsynchronizedPoolResource (const PoolOptions &opts, MemoryResource *upstream) noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.res.pool]`

⌋

8.4.7.1.2.3 UnsynchronizedPoolResource

[SWS_CORE_00068] Definition of API function ara::core::UnsynchronizedPool Resource::UnsynchronizedPoolResource ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::UnsynchronizedPoolResource`
Syntax:	`explicit UnsynchronizedPoolResource (const PoolOptions &opts);`
Exception Safety:	not exception safe
Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.res.pool]`

⌋

8.4.7.1.3 Member Functions

8.4.7.1.3.1 options

[SWS_CORE_00057] Definition of API function ara::core::UnsynchronizedPool Resource::options ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::UnsynchronizedPoolResource`
Syntax:	`PoolOptions options () const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.res.pool]`

⌋

8.4.7.1.3.2 release

[SWS_CORE_00065] Definition of API function ara::core::UnsynchronizedPool Resource::release ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::UnsynchronizedPoolResource`
Syntax:	`void release () noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.res.pool]`

⌋

8.4.7.1.3.3 upstream_resource

[SWS_CORE_00056] Definition of API function ara::core::UnsynchronizedPool Resource::upstream_resource ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::UnsynchronizedPoolResource`
Syntax:	`MemoryResource * upstream_resource () const noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.res.pool]`

⌋

8.4.7.2 Protected Member Functions

8.4.7.2.1 Member Functions

8.4.7.2.1.1 do_allocate

[SWS_CORE_00058] Definition of API function ara::core::UnsynchronizedPool Resource::do_allocate ⌈

Kind:	function	
Header file:	#include "ara/core/memory_resource.h"	
Scope:	`class ara::core::UnsynchronizedPoolResource`	
Syntax:	`void * do_allocate (size_t bytes, size_t alignment) noexcept override;`	
Return value:	void *	• If successful: As per corresponding usage in `[mem.res.pool]` • If unsuccessful: the returned pointer shall be `nullptr`
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	As per corresponding usage in `[mem.res.pool]` except for the following deviations: 1. The function is `noexcept` 2. The function returns `nullptr` instead of (potentially) throwing a `std::bad_alloc` exception	
Visibility:	protected	

⌋

8.4.7.2.1.2 do_deallocate

[SWS_CORE_00059] Definition of API function ara::core::UnsynchronizedPool Resource::do_deallocate ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::UnsynchronizedPoolResource`
Syntax:	`void do_deallocate (void *p, size_t bytes, size_t alignment) noexcept override;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	As per corresponding usage in `[mem.res.pool]` except for the following deviations: 1. The function is `noexcept` 2. If unsuccessful: errors will be silently ignored
Visibility:	protected

⌋

8.4.7.2.1.3 do_is_equal

[SWS_CORE_00060] Definition of API function ara::core::UnsynchronizedPool Resource::do_is_equal ⌈

Kind:	function
Header file:	#include "ara/core/memory_resource.h"
Scope:	`class ara::core::UnsynchronizedPoolResource`
Syntax:	`bool do_is_equal (const MemoryResource &other) const noexcept override;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	As per corresponding usage in `[mem.res.pool]`
Visibility:	protected

⌋

8.5 Header: ara/core/optional.h

[SWS_CORE_01141] Definition of Header File ara/core/optional.h

Upstream requirements: RS_AP_00130

⌈

Kind:	Header File
Syntax:	`ara/core/optional.h`
Description:	Implements `[optional.syn]` in [11]. Unless explicitly overridden in the API documentation, APIs always adhere in behavior to the ISO specification. To reduce duplication, the APIs here specify only any deviations from [11] or any AUTOSAR specific overrides. For APIs which would ordinarily throw `Exception`s, and those `Exception`s have been suppressed in favor of AUTOSAR violations, generally the semantics of the `Exception` from [11] still apply (unless locally overridden here).

⌋

8.5.1 Global Variables

8.5.1.1 nullopt

[SWS_CORE_01101] Definition of API variable ara::core::nullopt

Status: DRAFT

⌈

Kind:	variable
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Symbol:	nullopt
Type:	nullopt_t
Syntax:	`constexpr nullopt_t nullopt {};`
Description:	As per corresponding usage in `[optional]`

⌋

8.5.2 Non-Member Functions

8.5.2.1 Other

8.5.2.1.1 make_optional(Args ...&&)

[SWS_CORE_01139] Definition of API function ara::core::make_optional

Status: DRAFT

⌈

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

Syntax:	`template <typename T, typename... Args>`
	`constexpr Optional< T > make_optional (Args &&... args)`
	`noexcept(std::is_nothrow_constructible< T, Args... >::value);`

Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[optional]` except for the following deviations: 1. Function is conditionally `noexcept`

⌋

8.5.2.1.2 make_optional(std::initializer_list<U>, Args ...&&)

[SWS_CORE_01140] Definition of API function ara::core::make_optional

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T, typename U, typename... Args>` `constexpr Optional< T > make_optional (std::initializer_list< U > il, Args &&... args) noexcept(std::is_nothrow_constructible< T, std::initializer_list< U > &, Args &&... >::value);`
Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[optional]` except for the following deviations: 1. Function is conditionally `noexcept`

⌋

8.5.2.1.3 make_optional(T&&)

[SWS_CORE_01138] Definition of API function ara::core::make_optional

Status: DRAFT

⌈

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

Syntax:	`template <typename T>`
	`constexpr Optional< std::decay_t< T > > make_optional (T &&)`
	`noexcept(std::is_nothrow_move_constructible< T >::value);`

Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[optional]` except for the following deviations: 1. Function is conditionally `noexcept`

⌋

8.5.2.1.4 operator!=(const Optional<T>&, nullopt_t)

[SWS_CORE_01188] Definition of API function ara::core::operator!= ⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T>` `constexpr bool operator!= (const Optional< T > &, nullopt_t) noexcept;`

▽

△

Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.2.1.5 operator!=(nullopt_t, const Optional<T>&)

[SWS_CORE_01189] Definition of API function ara::core::operator!= ⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T>` `constexpr bool operator!= (nullopt_t, const Optional< T > &) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.2.1.6 operator!=(const Optional<T>&, const Optional<U>&)

[SWS_CORE_01143] Definition of API function ara::core::operator!= ⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T, typename U>` `constexpr bool operator!= (const Optional< T > &, const Optional< U > &);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.2.1.7 operator!=(const Optional<T>&, const U&)

[SWS_CORE_01176] Definition of API function ara::core::operator!= ⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T, typename U>` `constexpr bool operator!= (const Optional< T > &, const U &);`
Exception Safety:	not exception safe

▽

△

Description:	As per corresponding usage in `[optional]`

⌋

8.5.2.1.8 operator!=(const U&, const Optional<T>&)

[SWS_CORE_01177] Definition of API function ara::core::operator!= ⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T, typename U>` `constexpr bool operator!= (const U &, const Optional< T > &);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.2.1.9 operator<(const Optional<T>&, nullopt_t)

[SWS_CORE_01190] Definition of API function ara::core::operator< ⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T>` `constexpr bool operator< (const Optional< T > &, nullopt_t) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.2.1.10 operator<(const U&, const Optional<T>&)

[SWS_CORE_01179] Definition of API function ara::core::operator< ⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T, typename U>` `constexpr bool operator< (const U &, const Optional< T > &);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.2.1.11 operator<(const Optional<T>&, const Optional<U>&)

[SWS_CORE_01144] Definition of API function ara::core::operator< ⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T, typename U>` `constexpr bool operator< (const Optional< T > &, const Optional< U > &);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.2.1.12 operator<(const Optional<T>&, const U&)

[SWS_CORE_01178] Definition of API function ara::core::operator< ⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T, typename U>` `constexpr bool operator< (const Optional< T > &, const U &);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.2.1.13 operator<(nullopt_t, const Optional<T>&)

[SWS_CORE_01191] Definition of API function ara::core::operator< ⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T>` `constexpr bool operator< (nullopt_t, const Optional< T > &) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.2.1.14 operator<=(const Optional<T>&, nullopt_t)

[SWS_CORE_01192] Definition of API function ara::core::operator<= ⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T>` `constexpr bool operator<= (const Optional< T > &, nullopt_t) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.2.1.15 operator<=(nullopt_t, const Optional<T>&)

[SWS_CORE_01193] Definition of API function ara::core::operator<= ⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T>` `constexpr bool operator<= (nullopt_t, const Optional< T > &) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.2.1.16 operator<=(const Optional<T>&, const U&)

[SWS_CORE_01180] Definition of API function ara::core::operator<= ⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T, typename U>` `constexpr bool operator<= (const Optional< T > &, const U &);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.2.1.17 operator<=(const U&, const Optional<T>&)

[SWS_CORE_01181] Definition of API function ara::core::operator<= ⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T, typename U>` `constexpr bool operator<= (const U &, const Optional< T > &);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.2.1.18 operator<=(const Optional<T>&, const Optional<U>&)

[SWS_CORE_01146] Definition of API function ara::core::operator<= ⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T, typename U>` `constexpr bool operator<= (const Optional< T > &, const Optional< U > &);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.2.1.19 operator==(nullopt_t, const Optional<T>&)

[SWS_CORE_01187] Definition of API function ara::core::operator== ⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T>` `constexpr bool operator== (nullopt_t, const Optional< T > &) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.2.1.20 operator==(const Optional<T>&, nullopt_t)

[SWS_CORE_01186] Definition of API function ara::core::operator== ⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T>` `constexpr bool operator== (const Optional< T > &, nullopt_t) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.2.1.21 operator==(const Optional<T>&, const Optional<U>&)

[SWS_CORE_01142] Definition of API function ara::core::operator== ⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T, typename U>` `constexpr bool operator== (const Optional< T > &, const Optional< U > &);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.2.1.22 operator==(const Optional<T>&, const U&)

[SWS_CORE_01148] Definition of API function ara::core::operator== ⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T, typename U>` `constexpr bool operator== (const Optional< T > &, const U &);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.2.1.23 operator==(const U&, const Optional<T>&)

[SWS_CORE_01149] Definition of API function ara::core::operator== ⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T, typename U>` `constexpr bool operator== (const U &, const Optional< T > &);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.2.1.24 operator>(const Optional<T>&, const Optional<U>&)

[SWS_CORE_01145] Definition of API function ara::core::operator> ⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T, typename U>` `constexpr bool operator> (const Optional< T > &, const Optional< U > &);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.2.1.25 operator>(const Optional<T>&, const U&)

[SWS_CORE_01182] Definition of API function ara::core::operator> ⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T, typename U>` `constexpr bool operator> (const Optional< T > &, const U &);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.2.1.26 operator>(const U&, const Optional<T>&)

[SWS_CORE_01183] Definition of API function ara::core::operator> ⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T, typename U>` `constexpr bool operator> (const U &, const Optional< T > &);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.2.1.27 operator>(const Optional<T>&, nullopt_t)

[SWS_CORE_01194] Definition of API function ara::core::operator> ⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T>` `constexpr bool operator> (const Optional< T > &, nullopt_t) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.2.1.28 operator>(nullopt_t, const Optional<T>&)

[SWS_CORE_01195] Definition of API function ara::core::operator> ⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T>` `constexpr bool operator> (nullopt_t, const Optional< T > &) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.2.1.29 operator>=(const Optional<T>&, const U&)

[SWS_CORE_01184] Definition of API function ara::core::operator>= ⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T, typename U>` `constexpr bool operator>= (const Optional< T > &, const U &);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.2.1.30 operator>=(nullopt_t, const Optional<T>&)

[SWS_CORE_01197] Definition of API function ara::core::operator>= ⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T>` `constexpr bool operator>= (nullopt_t, const Optional< T > &) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.2.1.31 operator>=(const Optional<T>&, const Optional<U>&)

[SWS_CORE_01147] Definition of API function ara::core::operator>= ⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T, typename U>` `constexpr bool operator>= (const Optional< T > &, const Optional< U > &);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.2.1.32 operator>=(const Optional<T>&, nullopt_t)

[SWS_CORE_01196] Definition of API function ara::core::operator>= ⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T>` `constexpr bool operator>= (const Optional< T > &, nullopt_t) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.2.1.33 operator>=(const U&, const Optional<T>&)

[SWS_CORE_01185] Definition of API function ara::core::operator>= ⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T, typename U>` `constexpr bool operator>= (const U &, const Optional< T > &);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.2.1.34 swap

[SWS_CORE_01096] Definition of API function ara::core::swap

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	namespace ara::core
Syntax:	`template <typename T>` `void swap (Optional< T > &lhs, Optional< T > &rhs) noexcept(std::is_nothrow_move_constructible< T >::value &&std::is_nothrow_move_assignable< T >::value);`
Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[optional]` except for the following deviations: 1. noexcept specifier conditions are modified

⌋

8.5.3 Class: Optional

[SWS_CORE_01033] Definition of API class ara::core::Optional

Status: DRAFT

⌈

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	--
Description:	Implementation of `std::optional<T>` type from `[optional.syn]`	

⌋

8.5.3.1 Public Member Types

8.5.3.1.1 Type Alias: value_type

[SWS_CORE_01102] Definition of API type ara::core::Optional::value_type

Status: DRAFT

⌈

Kind:	type alias
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional`
Symbol:	value_type
Syntax:	`using value_type = T;`
Description:	As per corresponding usage in `[optional]`

⌋

8.5.3.2 Public Member Functions

8.5.3.2.1 Special Member Functions

8.5.3.2.1.1 Copy Constructor

[SWS_CORE_01105] Definition of API function ara::core::Optional::Optional

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional`
Syntax:	`constexpr Optional (const Optional &other) noexcept(std::is_nothrow_copy_constructible< T >::value);`
Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[optional]` except for the following deviations: 1. Function is conditionally `noexcept`

⌋

8.5.3.2.1.2 Copy Constructor

[SWS_CORE_01110] Definition of API function ara::core::Optional::Optional

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	
Syntax:	`template <typename U>` `{<cond-explicit-2>} Optional (const Optional< U > &) noexcept(std::is_nothrow_constructible< T, const U & >::value);`	
Exception Safety:	conditionally exception safe	
Description:	As per corresponding usage in `[optional]` except for the following deviations: 1. Function is conditionally `noexcept`	
Descriptors:	`{<cond-explicit-2>}`	See condition in `std::optional`

⌋

8.5.3.2.1.3 Move Constructor

[SWS_CORE_01106] Definition of API function ara::core::Optional::Optional

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional`
Syntax:	`constexpr Optional (Optional &&other) noexcept(std::is_nothrow_move_constructible< T >::value);`
Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.3.2.1.4 Default Constructor

[SWS_CORE_01103] Definition of API function ara::core::Optional::Optional

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional`
Syntax:	`constexpr Optional () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.3.2.1.5 Move Constructor

[SWS_CORE_01111] Definition of API function ara::core::Optional::Optional

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional`
Syntax:	`template <typename U>` `{<cond-explicit-3>} Optional (Optional< U > &&) noexcept(std::is_nothrow_constructible< T, U >::value);`
Exception Safety:	conditionally exception safe

▽

△

Description:	As per corresponding usage in `[optional]` except for the following deviations: 1. Function is conditionally `noexcept`	
Descriptors:	`{<cond-explicit-3>` `}`	See condition in `std::optional`

⌋

8.5.3.2.1.6 Copy Assignment Operator

[SWS_CORE_01114] Definition of API function ara::core::Optional::operator=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional`
Syntax:	`Optional & operator= (const Optional &other) &noexcept(std::is_nothrow_copy_constructible< T >::value &&std::is_nothrow_copy_assignable< T >::value);`
Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[optional]` except for the following deviations: 1. Function is conditionally `noexcept` 2. Function is declared with the ref-qualifier &

⌋

8.5.3.2.1.7 Move Assignment Operator

[SWS_CORE_01115] Definition of API function ara::core::Optional::operator=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional`
Syntax:	`Optional & operator= (Optional &&other) &noexcept(std::is_nothrow_move_assignable< T >::value &&std::is_nothrow_move_constructible< T >::value);`
Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[optional]` except for the following deviations: 1. Function is declared with the ref-qualifier &

⌋

8.5.3.2.1.8 Move Assignment Operator

[SWS_CORE_01118] Definition of API function ara::core::Optional::operator=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional`
Syntax:	`template <typename U>` `Optional & operator= (Optional< U > &&) &noexcept(std::is_nothrow_constructible< T, U >::value &&std::is_nothrow_assignable< T &, U >::value);`
Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[optional]` except for the following deviations: 1. Function is conditionally `noexcept` 2. Function is declared with the ref-qualifier &

⌋

8.5.3.2.1.9 Destructor

[SWS_CORE_01112] Definition of API function ara::core::Optional::~Optional

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional`
Syntax:	`~Optional () noexcept(std::is_nothrow_destructible< T >::value);`
Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[optional]` except for the following deviations: 1. Function is conditionally `noexcept`

⌋

8.5.3.2.2 Constructors

8.5.3.2.2.1 Optional(nullopt_t)

[SWS_CORE_01104] Definition of API function ara::core::Optional::Optional

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional`
Syntax:	`constexpr Optional (nullopt_t) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.3.2.2.2 Optional(in_place_t, Args&&)

[SWS_CORE_01107] Definition of API function ara::core::Optional::Optional

Status: DRAFT

⌈

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

Syntax:	`template <typename... Args>`
	`explicit constexpr Optional (in_place_t, Args &&...)`
	`noexcept(std::is_nothrow_constructible< T, Args... >::value);`

Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[optional]` except for the following deviations: 1. Function is conditionally `noexcept`

⌋

8.5.3.2.2.3 Optional(in_place_t, std::initializer_list<U>, Args&&)

[SWS_CORE_01108] Definition of API function ara::core::Optional::Optional

Status: DRAFT

⌈

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

▽

△

Syntax:	`template <typename U, typename... Args>` `explicit constexpr Optional (in_place_t, std::initializer_list< U >, Args &&...) noexcept(std::is_nothrow_constructible< T, std::initializer_list< U > &, Args &&... >::value);`
Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[optional]` except for the following deviations: 1. Function is conditionally `noexcept`

⌋

8.5.3.2.2.4 Optional(U&&)

[SWS_CORE_01109] Definition of API function ara::core::Optional::Optional

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	
Syntax:	`template <typename U = T>` `{<cond-explicit-1>} constexpr Optional (U &&) noexcept(std::is_nothrow_constructible< T, U >::value);`	
Exception Safety:	conditionally exception safe	
Description:	As per corresponding usage in `[optional]` except for the following deviations: 1. Function is conditionally `noexcept`	
Descriptors:	`{<cond-explicit-1>}`	See condition in `std::optional`

⌋

8.5.3.2.3 Modifiers

8.5.3.2.3.1 emplace(Args&&)

[SWS_CORE_01119] Definition of API function ara::core::Optional::emplace

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional`
Syntax:	`template <typename... Args>` `T & emplace (Args &&...) noexcept(std::is_nothrow_constructible< T, Args &&... >::value);`
Exception Safety:	conditionally exception safe

▽

△

Description:	As per corresponding usage in `[optional]` except for the following deviations: 1. Function is conditionally `noexcept`

⌋

8.5.3.2.3.2 emplace(std::initializer_list<U>, Args&&)

[SWS_CORE_01120] Definition of API function ara::core::Optional::emplace

Status: DRAFT

⌈

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

Syntax:	`template <typename U, typename... Args>`
	`T & emplace (std::initializer_list< U >, Args &&...)`
	`noexcept(std::is_nothrow_constructible< T, std::initializer_list< U > &, Args &&... >::value);`

Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[optional]` except for the following deviations: 1. Function is conditionally `noexcept`

⌋

8.5.3.2.3.3 reset

[SWS_CORE_01136] Definition of API function ara::core::Optional::reset

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional`
Syntax:	`void reset () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.3.2.3.4 swap

[SWS_CORE_01121] Definition of API function ara::core::Optional::swap

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional`
Syntax:	`void swap (Optional &) noexcept(std::is_nothrow_move_constructible< T >::value &&std::is_nothrow_move_assignable< T >::value);`
Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[optional]` except for the following deviations: 1. Conditions for the conditional exception safety are modified

⌋

8.5.3.2.4 Observers

8.5.3.2.4.1 has_value

[SWS_CORE_01129] Definition of API function ara::core::Optional::has_value

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional`
Syntax:	`constexpr bool has_value () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.3.2.4.2 operator bool

[SWS_CORE_01128] Definition of API function ara::core::Optional::operator bool

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional`
Syntax:	`explicit constexpr operator bool () const noexcept;`

▽

△

Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.3.2.4.3 operator*() const &&

[SWS_CORE_01127] Definition of API function ara::core::Optional::operator*

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	
Syntax:	`constexpr const T && operator* () const &&noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdBadOption-alAccessViolation`	If `*this` does not contain a value
Description:	As per corresponding usage in `[optional]` except for the following deviations: 1. The function is `noexcept` 2. A `IsoStdBadOptionalAccessViolation` may conditionally be raised (see condition)	

⌋

8.5.3.2.4.4 operator*() &

[SWS_CORE_01125] Definition of API function ara::core::Optional::operator*

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	
Syntax:	`constexpr T & operator* () & noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdBadOption-alAccessViolation`	If `*this` does not contain a value
Description:	As per corresponding usage in `[optional]` except for the following deviations: 1. The function is `noexcept` 2. A `IsoStdBadOptionalAccessViolation` may conditionally be raised (see condition)	

⌋

8.5.3.2.4.5 operator*() &&

[SWS_CORE_01126] Definition of API function ara::core::Optional::operator*

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	
Syntax:	`constexpr T && operator* () &&noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdBadOption-alAccessViolation`	If `*this` does not contain a value
Description:	As per corresponding usage in `[optional]` except for the following deviations: 1. The function is `noexcept` 2. A `IsoStdBadOptionalAccessViolation` may conditionally be raised (see condition)	

⌋

8.5.3.2.4.6 operator*() const &

[SWS_CORE_01124] Definition of API function ara::core::Optional::operator*

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	
Syntax:	`constexpr const T & operator* () const & noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdBadOption-alAccessViolation`	If `*this` does not contain a value
Description:	As per corresponding usage in `[optional]` except for the following deviations: 1. The function is `noexcept` 2. A `IsoStdBadOptionalAccessViolation` may conditionally be raised (see condition)	

⌋

8.5.3.2.4.7 operator->

[SWS_CORE_01123] Definition of API function ara::core::Optional::operator->

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	
Syntax:	`T * operator-> () noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdBadOption-alAccessViolation`	If `*this` does not contain a value
Description:	As per corresponding usage in `[optional]` except for the following deviations: 1. The function is `noexcept` 2. Function is not constexpr 3. A `IsoStdBadOptionalAccessViolation` may be raised (see condition)	

⌋

8.5.3.2.4.8 operator->() const

[SWS_CORE_01122] Definition of API function ara::core::Optional::operator->

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	
Syntax:	`const T * operator-> () const noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdBadOption-alAccessViolation`	If `*this` does not contain a value
Description:	As per corresponding usage in `[optional]` except for the following deviations: 1. The function is `noexcept` 2. Function is not constexpr 3. A `IsoStdBadOptionalAccessViolation` may be raised (see condition)	

⌋

8.5.3.2.4.9 value() const &

[SWS_CORE_01130] Definition of API function ara::core::Optional::value

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	
Syntax:	`constexpr const T & value () const & noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdBadOption-alAccessViolation`	If `*this` does not contain a value
Description:	As per corresponding usage in `[optional]` except for the following deviations: 1. The function is `noexcept` 2. A `IsoStdBadOptionalAccessViolation` may conditionally be raised (see condition)	

⌋

8.5.3.2.4.10 value() &

[SWS_CORE_01131] Definition of API function ara::core::Optional::value

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	
Syntax:	`constexpr T & value () & noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdBadOption-alAccessViolation`	If `*this` does not contain a value
Description:	As per corresponding usage in `[optional]` except for the following deviations: 1. The function is `noexcept` 2. A `IsoStdBadOptionalAccessViolation` may conditionally be raised (see condition)	

⌋

8.5.3.2.4.11 value() &&

[SWS_CORE_01132] Definition of API function ara::core::Optional::value

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	
Syntax:	`constexpr T && value () &&noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdBadOption-alAccessViolation`	If `*this` does not contain a value
Description:	As per corresponding usage in `[optional]` except for the following deviations: 1. The function is `noexcept` 2. A `IsoStdBadOptionalAccessViolation` may conditionally be raised (see condition)	

⌋

8.5.3.2.4.12 value() const &&

[SWS_CORE_01133] Definition of API function ara::core::Optional::value

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional`	
Syntax:	`constexpr const T && value () const &&noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdBadOption-alAccessViolation`	If `*this` does not contain a value
Description:	As per corresponding usage in `[optional]` except for the following deviations: 1. The function is `noexcept` 2. A `IsoStdBadOptionalAccessViolation` may conditionally be raised (see condition)	

⌋

8.5.3.2.4.13 value_or(U&&) &&

[SWS_CORE_01135] Definition of API function ara::core::Optional::value_or

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional`
Syntax:	`template <typename U>` `constexpr T value_or (U &&) &&;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.3.2.4.14 value_or(U&&) const &

[SWS_CORE_01134] Definition of API function ara::core::Optional::value_or

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional`
Syntax:	`template <typename U>` `constexpr T value_or (U &&) const &;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.3.2.5 Operator=

8.5.3.2.5.1 operator=(U&&) &

[SWS_CORE_01116] Definition of API function ara::core::Optional::operator=

Status: DRAFT

⌈

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

▽

△

Syntax:	`template <typename U = T>` `Optional & operator= (U &&u) &noexcept(std::is_nothrow_constructible< T, U >::value &&std::is_nothrow_assignable< T &, U >::value);`
Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[optional]` except for the following deviations: 1. Function is conditionally `noexcept` 2. Function is declared with the ref-qualifier &

⌋

8.5.3.2.5.2 operator=(const Optional<U>&) &

[SWS_CORE_01117] Definition of API function ara::core::Optional::operator=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional`
Syntax:	`template <typename U>` `Optional & operator= (const Optional< U > &) &noexcept(std::is_nothrow_constructible< T, const U &>::value &&std::is_nothrow_assignable< T &, const U &>::value);`
Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[optional]` except for the following deviations: 1. Function is conditionally `noexcept` 2. Function is declared with the ref-qualifier &

⌋

8.5.3.2.5.3 operator=

[SWS_CORE_01113] Definition of API function ara::core::Optional::operator=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional`
Syntax:	`Optional & operator= (nullopt_t) & noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.4 Class: Optional<T &>

[SWS_CORE_01150] Definition of API class ara::core::Optional< T & >

Status: DRAFT

⌈

Kind:	class	
Header file:	#include "ara/core/optional.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	namespace ara::core	
Symbol:	Optional< T & >	
Syntax:	`template <typename T>` `class Optional< T & > final {...};`	
Template param:	typename T	--
Description:	Specialization of for type `T&`	

⌋

8.5.4.1 Public Member Types

8.5.4.1.1 Type Alias: value_type

[SWS_CORE_01151] Definition of API type ara::core::Optional< T & >::value_type

Status: DRAFT

⌈

Kind:	type alias
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional< T & >`
Symbol:	value_type
Syntax:	`using value_type = T&;`
Description:	As per corresponding usage in `[optional]`

⌋

8.5.4.2 Public Member Functions

8.5.4.2.1 Special Member Functions

8.5.4.2.1.1 Default Constructor

[SWS_CORE_01152] Definition of API function ara::core::Optional< T & >::Optional

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional< T & >`
Syntax:	`constexpr Optional () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.4.2.1.2 Copy Constructor

[SWS_CORE_01154] Definition of API function ara::core::Optional< T & >::Optional

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional< T & >`
Syntax:	`constexpr Optional (const Optional &other) noexcept=default;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.4.2.1.3 Move Constructor

[SWS_CORE_01155] Definition of API function ara::core::Optional< T & >::Optional

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional< T & >`
Syntax:	`constexpr Optional (Optional &&other) noexcept=default;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.4.2.1.4 Copy Assignment Operator

[SWS_CORE_01163] Definition of API function ara::core::Optional< T & >::operator=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional< T & >`
Syntax:	`template <typename U = T>` `Optional & operator= (const Optional< U > &other) & noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.4.2.1.5 Move Assignment Operator

[SWS_CORE_01160] Definition of API function ara::core::Optional< T & >::operator=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional< T & >`

▽

△

Syntax:	`constexpr Optional & operator= (Optional &&other) & noexcept=default;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.4.2.1.6 Copy Assignment Operator

[SWS_CORE_01159] Definition of API function ara::core::Optional< T & >::operator=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional< T & >`
Syntax:	`constexpr Optional & operator= (const Optional &other) & noexcept=default;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.4.2.1.7 Destructor

[SWS_CORE_01158] Definition of API function ara::core::Optional< T & >::~Optional

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional< T & >`
Syntax:	`~Optional () noexcept=default;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.4.2.2 Constructors

8.5.4.2.2.1 Optional(U&&)

[SWS_CORE_01156] Definition of API function ara::core::Optional< T & >::Optional

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional< T & >`
Syntax:	`template <typename U = T>` `explicit constexpr Optional (U &&u) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.4.2.2.2 Optional(Optional<U> const&)

[SWS_CORE_01157] Definition of API function ara::core::Optional< T & >::Optional

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional< T & >`
Syntax:	`template <typename U>` `explicit constexpr Optional (Optional< U > const &other) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.4.2.2.3 Optional

[SWS_CORE_01153] Definition of API function ara::core::Optional< T & >::Optional

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional< T & >`
Syntax:	`constexpr Optional (nullopt_t) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.4.2.3 Modifiers

8.5.4.2.3.1 emplace

[SWS_CORE_01164] Definition of API function ara::core::Optional< T & >::emplace

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional< T & >`
Syntax:	`template <typename U = T>` `constexpr Optional & emplace (U &&u) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.4.2.3.2 reset

[SWS_CORE_01165] Definition of API function ara::core::Optional< T & >::reset

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional< T & >`
Syntax:	`constexpr void reset () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.4.2.3.3 swap

[SWS_CORE_01166] Definition of API function ara::core::Optional< T & >::swap

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional< T & >`
Syntax:	`constexpr void swap (Optional &other) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.4.2.4 Observers

8.5.4.2.4.1 has_value

[SWS_CORE_01172] Definition of API function ara::core::Optional< T & >::has_value

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional< T & >`
Syntax:	`constexpr bool has_value () const noexcept;`

▽

△

Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.4.2.4.2 operator bool

[SWS_CORE_01171] Definition of API function ara::core::Optional< T & >::operator bool

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional< T & >`
Syntax:	`explicit constexpr operator bool () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.4.2.4.3 operator*() const

[SWS_CORE_01169] Definition of API function ara::core::Optional< T & >::operator*

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional< T & >`	
Syntax:	`constexpr T const & operator* () const noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdBadOptionalAccessViolation`	If `*this` does not contain a value
Description:	As per corresponding usage in `[optional]`	

⌋

8.5.4.2.4.4 operator*

[SWS_CORE_01170] Definition of API function ara::core::Optional< T & >::operator*

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional< T & >`	
Syntax:	`constexpr T & operator* () noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdBadOptionalAccessViolation`	If `*this` does not contain a value
Description:	As per corresponding usage in `[optional]`	

⌋

8.5.4.2.4.5 operator->

[SWS_CORE_01168] Definition of API function ara::core::Optional< T & >::operator->

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional< T & >`	
Syntax:	`T * operator-> () noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdBadOptionalAccessViolation`	If `*this` does not contain a value
Description:	As per corresponding usage in `[optional]`	

⌋

8.5.4.2.4.6 operator->() const

[SWS_CORE_01167] Definition of API function ara::core::Optional< T & >::operator->

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional< T & >`	
Syntax:	`T const * operator-> () const noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdBadOption-alAccessViolation`	If `*this` does not contain a value
Description:	As per corresponding usage in `[optional]`	

⌋

8.5.4.2.4.7 value() const

[SWS_CORE_01174] Definition of API function ara::core::Optional< T & >::value

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/optional.h"	
Scope:	`class ara::core::Optional< T & >`	
Syntax:	`constexpr T const & value () const;`	
Exception Safety:	not exception safe	
Violations:	`IsoStdBadOption-alAccessViolation`	If `*this` does not contain a value
Description:	As per corresponding usage in `[optional]`	

⌋

8.5.4.2.4.8 value

[SWS_CORE_01173] Definition of API function ara::core::Optional< T & >::value

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional< T & >`

▽

△

Syntax:	`constexpr T & value ();`	
Exception Safety:	not exception safe	
Violations:	`IsoStdBadOption-alAccessViolation`	If `*this` does not contain a value
Description:	As per corresponding usage in `[optional]`	

⌋

8.5.4.2.4.9 value_or

[SWS_CORE_01175] Definition of API function ara::core::Optional< T & >::value_or

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional< T & >`
Syntax:	`template <typename U>` `constexpr T value_or (U &&u) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.4.2.5 Operator=

8.5.4.2.5.1 operator=(nullopt_t) &

[SWS_CORE_01161] Definition of API function ara::core::Optional< T & >::operator=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional< T & >`
Syntax:	`constexpr Optional & operator= (nullopt_t) & noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.4.2.5.2 operator=(U&&) &

[SWS_CORE_01162] Definition of API function ara::core::Optional< T & >::operator=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`class ara::core::Optional< T & >`
Syntax:	`template <typename U = T>` `constexpr Optional & operator= (U &&u) & noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[optional]`

⌋

8.5.5 Struct: nullopt_t

[SWS_CORE_01100] Definition of API class ara::core::nullopt_t

Status: DRAFT

⌈

Kind:	struct
Header file:	#include "ara/core/optional.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	namespace ara::core
Symbol:	nullopt_t
Syntax:	`struct nullopt_t {...};`
Description:	As per corresponding usage in `[optional]` Shall implement `std::nullopt_t`

⌋

8.5.6 Struct: hash

[SWS_CORE_01198] Definition of API class std::hash< ara::core::Optional< T > > ⌈

Kind:	struct
Header file:	#include "ara/core/optional.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	namespace std
Symbol:	hash< ara::core::Optional< T > >

▽

△

Syntax:	`template <typename T>` `struct hash< ara::core::Optional< T > > {...};`	
Template param:	typename T	--
Description:	Specialization of `std::hash` (hashing function for `ara::core::Optional`)	

⌋

8.5.6.1 Public Member Functions

8.5.6.1.1 Member Functions

8.5.6.1.1.1 operator()

[SWS_CORE_01199] Definition of API function std::hash< ara::core::Optional< T > >::operator() ⌈

Kind:	function
Header file:	#include "ara/core/optional.h"
Scope:	`struct std::hash< ara::core::Optional< T > >`
Syntax:	`size_t operator() (ara::core::Optional< T > const &v) const noexcept;`
Exception Safety:	exception safe
Description:	Invoke the hashing function

⌋

8.6 Header: ara/core/span.h

[SWS_CORE_01902] Definition of Header File ara/core/span.h

Upstream requirements: RS_AP_00130

⌈

Kind:	Header File
Syntax:	`ara/core/span.h`
Description:	Implements `[span.syn]` in [15]. Unless explicitly overridden in the API documentation, the APIs in this library always adhere in behavior to the ISO specification. To reduce duplication, the APIs here specify only any deviations from [15] or any AUTOSAR specific overrides. For APIs which would ordinarily throw `Exception`s, and those `Exception`s have been suppressed in favor of AUTOSAR violations, generally the semantics of the `Exception` from [15] still apply (unless locally overridden here).

⌋

8.6.1 Global Variables

8.6.1.1 dynamic_extent

[SWS_CORE_01901] 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:	As per corresponding usage in `[views.span]`

⌋

8.6.2 Non-Member Functions

8.6.2.1 Other

8.6.2.1.1 MakeSpan(T *, typename Span<T>::size_type)

[SWS_CORE_01990] Definition of API function ara::core::MakeSpan

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	namespace ara::core	
Syntax:	`template <typename T>` `constexpr Span< T > MakeSpan (T *ptr, typename Span< T >::size_type count) noexcept;`	
Template param:	T	the type of elements
Parameters (in):	ptr	the pointer
	count	the number of elements to take from ptr
Return value:	Span< T >	the new Span
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	**[AUTOSAR defined]** not present in ISO-14882. Create a new Span from the given pointer and size	

⌋

8.6.2.1.2 MakeSpan(T *, T *)

[SWS_CORE_01991] Definition of API function ara::core::MakeSpan

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	namespace ara::core	
Syntax:	`template <typename T>` `constexpr Span< T > MakeSpan (T *firstElem, T *lastElem) noexcept;`	
Template param:	T	the type of elements
Parameters (in):	firstElem	pointer to the first element
	lastElem	pointer to past the last element
Return value:	Span< T >	the new Span
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	**[AUTOSAR defined]** not present in ISO-14882. Create a new Span from the left-closed, right-open range: `[firstElem, lastElem)`	

⌋

8.6.2.1.3 MakeSpan(T()&)

[SWS_CORE_01992] Definition of API function ara::core::MakeSpan

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	namespace ara::core	
Syntax:	`template <typename T, std::size_t N>` `constexpr Span< T, N > MakeSpan (T(&arr)[N]) noexcept;`	
Template param:	T	the type of elements
	N	the size of the raw array
Parameters (in):	arr	the raw array
Return value:	Span< T, N >	the new Span
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	**[AUTOSAR defined]** not present in ISO-14882. Create a new Span from the given raw array	

⌋

8.6.2.1.4 MakeSpan(Container&)

[SWS_CORE_01993] Definition of API function ara::core::MakeSpan

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	namespace ara::core	
Syntax:	`template <typename Container>` `constexpr Span< typename Container::value_type > MakeSpan (Container &cont) noexcept;`	
Template param:	Container	the type of container
Parameters (in):	cont	the container
Return value:	Span< typename Container::value_type >	the new Span
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	**[AUTOSAR defined]** not present in ISO-14882. Create a new Span from the given const container	

⌋

8.6.2.1.5 MakeSpan(const Container&)

[SWS_CORE_01994] Definition of API function ara::core::MakeSpan

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	namespace ara::core	
Syntax:	`template <typename Container>` `constexpr Span< typename Container::value_type const > MakeSpan (const Container &cont) noexcept;`	
Template param:	Container	the type of container
Parameters (in):	cont	the container
Return value:	Span< typename Container::value_type const >	the new Span
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	**[AUTOSAR defined]** not present in ISO-14882. Create a new Span from the given const container	

⌋

8.6.2.1.6 as_bytes

[SWS_CORE_01980] Definition of API function ara::core::as_bytes ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	namespace ara::core
Syntax:	`template <typename ElementType, std::size_t Extent> Span< const Byte, Extent==dynamic_extent ? dynamic_extent :sizeof(ElementType) *Extent > as_bytes (Span< ElementType, Extent > s) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`

⌋

8.6.2.1.7 as_writable_bytes

[SWS_CORE_01981] Definition of API function ara::core::as_writable_bytes ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	namespace ara::core
Syntax:	`template <typename ElementType, std::size_t Extent> Span< Byte, Extent==dynamic_extent ? dynamic_extent :sizeof(Element Type) *Extent > as_writable_bytes (Span< ElementType, Extent > s) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`

⌋

8.6.3 Class: Span

[SWS_CORE_01900] 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	--
	std::size_t Extent = dynamic_extent	--
Description:	Implementation of `std::span` container from `[views.span]`	

⌋

8.6.3.1 Public Member Types

8.6.3.1.1 Type Alias: const_iterator

[SWS_CORE_01918] 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 = {<const-iterator-type>};`	
Description:	As per corresponding usage in `[views.span]`	
Implementation Defined Behavior:	`{<const-iterator-type>}` shall be an implementation-defined type	
Descriptors:	`{<const-iterator-type>}`	implementation-defined

⌋

8.6.3.1.2 Type Alias: const_pointer

[SWS_CORE_01922] 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:	As per corresponding usage in `[views.span]`

⌋

8.6.3.1.3 Type Alias: const_reference

[SWS_CORE_01923] 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:	As per corresponding usage in `[views.span]`

⌋

8.6.3.1.4 Type Alias: const_reverse_iterator

[SWS_CORE_01920] 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:	As per corresponding usage in `[views.span]`

⌋

8.6.3.1.5 Type Alias: difference_type

[SWS_CORE_01914] 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:	As per corresponding usage in `[views.span]`

⌋

8.6.3.1.6 Type Alias: element_type

[SWS_CORE_01911] 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:	As per corresponding usage in `[views.span]`

⌋

8.6.3.1.7 Type Alias: iterator

[SWS_CORE_01917] 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 = {<iterator-type>};`	
Description:	As per corresponding usage in `[views.span]`	
Implementation Defined Behavior:	`{<iterator-type>}` shall be an implementation-defined type	
Descriptors:	`{<iterator-type>}`	implementation-defined

⌋

8.6.3.1.8 Type Alias: pointer

[SWS_CORE_01915] 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:	As per corresponding usage in `[views.span]`

⌋

8.6.3.1.9 Type Alias: reference

[SWS_CORE_01916] 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:	As per corresponding usage in `[views.span]`

⌋

8.6.3.1.10 Type Alias: reverse_iterator

[SWS_CORE_01919] 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:	As per corresponding usage in `[views.span]`

⌋

8.6.3.1.11 Type Alias: size_type

[SWS_CORE_01921] 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:	As per corresponding usage in `[views.span]`

⌋

8.6.3.1.12 Type Alias: value_type

[SWS_CORE_01912] 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:	As per corresponding usage in `[views.span]`

⌋

8.6.3.2 Public Member Variables

8.6.3.2.1 extent

[SWS_CORE_01931] 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:	As per corresponding usage in `[views.span]`

⌋

8.6.3.3 Public Member Functions

8.6.3.3.1 Special Member Functions

8.6.3.3.1.1 Span(const Span<U, N>&)

[SWS_CORE_01950] Definition of API function ara::core::Span::Span ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`template <typename U, std::size_t N>` `constexpr Span (const Span< U, N > &s) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`

⌋

8.6.3.3.1.2 Default Constructor

[SWS_CORE_01941] Definition of API function ara::core::Span::Span ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`constexpr Span () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`

⌋

8.6.3.3.1.3 Copy Constructor

[SWS_CORE_01949] Definition of API function ara::core::Span::Span ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`constexpr Span (const Span &other) noexcept=default;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`

⌋

8.6.3.3.1.4 Copy Assignment Operator

[SWS_CORE_01952] Definition of API function ara::core::Span::operator= ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`constexpr Span & operator= (const Span &other) noexcept=default;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`

⌋

8.6.3.3.1.5 Destructor

[SWS_CORE_01951] Definition of API function ara::core::Span::~Span ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`~Span () noexcept=default;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`

⌋

8.6.3.3.2 Constructors

8.6.3.3.2.1 Span(pointer, pointer)

[SWS_CORE_01943] Definition of API function ara::core::Span::Span ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`constexpr Span (pointer firstElem, pointer lastElem) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`

⌋

8.6.3.3.2.2 Span(element_type()&)

[SWS_CORE_01944] Definition of API function ara::core::Span::Span ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`template <std::size_t N>` `constexpr Span (element_type(&arr)[N]) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`

⌋

8.6.3.3.2.3 Span(ara::core::Array<U, N>&)

[SWS_CORE_01945] Definition of API function ara::core::Span::Span ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`template <typename U, std::size_t N>` `constexpr Span (ara::core::Array< U, N > &arr) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`

⌋

8.6.3.3.2.4 Span(const ara::core::Array<U, N>&)

[SWS_CORE_01946] Definition of API function ara::core::Span::Span ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`template <typename U, std::size_t N>` `constexpr Span (const ara::core::Array< U, N > &arr) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`

⌋

8.6.3.3.2.5 Span(Container&)

[SWS_CORE_01947] Definition of API function ara::core::Span::Span ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`template <typename Container>` `constexpr Span (Container &cont) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`

⌋

8.6.3.3.2.6 Span(const Container&)

[SWS_CORE_01948] Definition of API function ara::core::Span::Span ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`template <typename Container>` `constexpr Span (const Container &cont) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`

⌋

8.6.3.3.2.7 Span(pointer, size_type)

[SWS_CORE_01942] Definition of API function ara::core::Span::Span ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`constexpr Span (pointer ptr, size_type count) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`

⌋

8.6.3.3.3 Element access

8.6.3.3.3.1 back() const

[SWS_CORE_01960] Definition of API function ara::core::Span::back ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`constexpr reference back () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`

⌋

8.6.3.3.3.2 data() const

[SWS_CORE_01971] Definition of API function ara::core::Span::data ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`constexpr pointer data () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`

⌋

8.6.3.3.3.3 front() const

[SWS_CORE_01959] Definition of API function ara::core::Span::front ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`constexpr reference front () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`

⌋

8.6.3.3.3.4 operator[](size_type) const

[SWS_CORE_01970] Definition of API function ara::core::Span::operator[] ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`constexpr reference operator[] (size_type idx) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`

⌋

8.6.3.3.4 Iterators

8.6.3.3.4.1 begin

[SWS_CORE_01972] Definition of API function ara::core::Span::begin ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`constexpr iterator begin () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`

⌋

8.6.3.3.4.2 cbegin

[SWS_CORE_01974] Definition of API function ara::core::Span::cbegin ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`constexpr const_iterator cbegin () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`

⌋

8.6.3.3.4.3 cend

[SWS_CORE_01975] Definition of API function ara::core::Span::cend ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`constexpr const_iterator cend () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`

⌋

8.6.3.3.4.4 crbegin

[SWS_CORE_01978] Definition of API function ara::core::Span::crbegin ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`constexpr const_reverse_iterator crbegin () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`

⌋

8.6.3.3.4.5 crend

[SWS_CORE_01979] Definition of API function ara::core::Span::crend ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`constexpr const_reverse_iterator crend () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`

⌋

8.6.3.3.4.6 end

[SWS_CORE_01973] Definition of API function ara::core::Span::end ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`constexpr iterator end () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`

⌋

8.6.3.3.4.7 rbegin

[SWS_CORE_01976] Definition of API function ara::core::Span::rbegin ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`constexpr reverse_iterator rbegin () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`

⌋

8.6.3.3.4.8 rend

[SWS_CORE_01977] Definition of API function ara::core::Span::rend ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`constexpr reverse_iterator rend () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`

⌋

8.6.3.3.5 Observers

8.6.3.3.5.1 empty

[SWS_CORE_01969] Definition of API function ara::core::Span::empty ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`constexpr bool empty () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`

⌋

8.6.3.3.5.2 size

[SWS_CORE_01967] Definition of API function ara::core::Span::size ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`constexpr size_type size () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`

⌋

8.6.3.3.5.3 size_bytes

[SWS_CORE_01968] Definition of API function ara::core::Span::size_bytes ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`constexpr size_type size_bytes () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`

⌋

8.6.3.3.6 Subviews

8.6.3.3.6.1 first(size_type) const

[SWS_CORE_01962] Definition of API function ara::core::Span::first ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`constexpr Span< element_type, dynamic_extent > first (size_type count) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`
Undefined Behavior:	The behavior of this function is undefined if: `(count > size())`

⌋

8.6.3.3.6.2 first() const

[SWS_CORE_01961] Definition of API function ara::core::Span::first ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`template <std::size_t Count>` `constexpr Span< element_type, Count > first () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`
Undefined Behavior:	The behavior of this function is undefined if: `(Count > size())`

⌋

8.6.3.3.6.3 last(size_type) const

[SWS_CORE_01964] Definition of API function ara::core::Span::last ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`constexpr Span< element_type, dynamic_extent > last (size_type count) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`
Undefined Behavior:	The behavior of this function is undefined if: `(count > size())`

⌋

8.6.3.3.6.4 last() const

[SWS_CORE_01963] Definition of API function ara::core::Span::last ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`template <std::size_t Count>` `constexpr Span< element_type, Count > last () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`
Undefined Behavior:	The behavior of this function is undefined if: `(Count > size())`

⌋

8.6.3.3.6.5 subspan() const

[SWS_CORE_01965] Definition of API function ara::core::Span::subspan ⌈

Kind:	function	
Header file:	#include "ara/core/span.h"	
Scope:	`class ara::core::Span`	
Syntax:	`template <std::size_t Offset, std::size_t Count = dynamic_extent>` `constexpr auto subspan () const noexcept -> Span< element_type, {<extent-or-dynamicextent>} >;`	
Exception Safety:	exception safe	
Description:	As per corresponding usage in `[views.span]`	
Descriptors:	`{<extent-or-dynamicextent>}`	See condition in `std::span`

⌋

8.6.3.3.6.6 subspan(size_type, size_type) const

[SWS_CORE_01966] Definition of API function ara::core::Span::subspan ⌈

Kind:	function
Header file:	#include "ara/core/span.h"
Scope:	`class ara::core::Span`
Syntax:	`constexpr Span< element_type, dynamic_extent > subspan (size_type offset, size_type count=dynamic_extent) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[views.span]`
Undefined Behavior:	The behavior of this function is undefined unless: `(offset <= size() && (count == dynamic_extent || count <= size() - offset)) == true`

⌋

8.7 Header: ara/core/string_view.h

[SWS_CORE_02002] Definition of Header File ara/core/string_view.h

Upstream requirements: RS_AP_00130

⌈

Kind:	Header File
Syntax:	`ara/core/string_view.h`
Description:	Implements `[string.view]` in [11]. Unless explicitly overridden in the API documentation, APIs always adhere in behavior to the ISO specification. To reduce duplication, the APIs here specify only any deviations from [11] or any AUTOSAR specific overrides. For APIs which would ordinarily throw `Exception`s, and those `Exception`s have been suppressed in favor of AUTOSAR violations, generally the semantics of the `Exception` from [11] still apply (unless locally overridden here).

⌋

8.7.1 Non-Member Functions

8.7.1.1 Other

8.7.1.1.1 operator!=

[SWS_CORE_02182] Definition of API function ara::core::operator!=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	namespace ara::core
Syntax:	`constexpr bool operator!= (StringView lhs, StringView rhs) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.1.1.2 operator""_SV

[SWS_CORE_02188] Definition of API function ara::core::literals::string_view_literals::operator""_SV

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	namespace ara::core::literals::string_view_literals
Syntax:	`constexpr StringView operator""_SV (const char *str, std::size_t len) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.1.1.3 operator<

[SWS_CORE_02183] Definition of API function ara::core::operator<

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	namespace ara::core
Syntax:	`constexpr bool operator< (StringView lhs, StringView rhs) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.1.1.4 operator<<

[SWS_CORE_02187] Definition of API function ara::core::operator<<

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	namespace ara::core
Syntax:	`std::ostream & operator<< (std::ostream &os, StringView sv);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.1.1.5 operator<=

[SWS_CORE_02185] Definition of API function ara::core::operator<=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	namespace ara::core
Syntax:	`constexpr bool operator<= (StringView lhs, StringView rhs) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.1.1.6 operator==

[SWS_CORE_02181] Definition of API function ara::core::operator==

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	namespace ara::core
Syntax:	`constexpr bool operator== (StringView lhs, StringView rhs) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.1.1.7 operator>

[SWS_CORE_02184] Definition of API function ara::core::operator>

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	namespace ara::core
Syntax:	`constexpr bool operator> (StringView lhs, StringView rhs) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.1.1.8 operator>=

[SWS_CORE_02186] Definition of API function ara::core::operator>=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	namespace ara::core
Syntax:	`constexpr bool operator>= (StringView lhs, StringView rhs) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2 Class: StringView

[SWS_CORE_02001] Definition of API class ara::core::StringView

Status: DRAFT

⌈

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 {...};`
Description:	Implementation of `std::basic_string_view` type from `[string.view]`

⌋

8.7.2.1 Public Member Types

8.7.2.1.1 Type Alias: const_iterator

[SWS_CORE_02105] Definition of API type ara::core::StringView::const_iterator

Status: DRAFT

⌈

Kind:	type alias	
Header file:	#include "ara/core/string_view.h"	
Scope:	`class ara::core::StringView`	
Symbol:	const_iterator	
Syntax:	`using const_iterator = {<const-iterator-type>};`	
Description:	As per corresponding usage in `[string.view]`	
Implementation Defined Behavior:	`{<const-iterator-type>}` shall be an implementation-defined type	
Descriptors:	`{<const-iterator-type>}`	implementation-defined

⌋

8.7.2.1.2 Type Alias: const_pointer

[SWS_CORE_02102] Definition of API type ara::core::StringView::const_pointer

Status: DRAFT

⌈

Kind:	type alias
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Symbol:	const_pointer
Syntax:	`using const_pointer = const value_type*;`
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. Pointer type shall be specialized to `value_type`

⌋

8.7.2.1.3 Type Alias: const_reference

[SWS_CORE_02104] Definition of API type ara::core::StringView::const_reference

Status: DRAFT

⌈

Kind:	type alias
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Symbol:	const_reference
Syntax:	`using const_reference = const value_type&;`
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. Reference type shall be specialized to `value_type`

⌋

8.7.2.1.4 Type Alias: const_reverse_iterator

[SWS_CORE_02107] Definition of API type ara::core::StringView::const_reverse_iterator

Status: DRAFT

⌈

Kind:	type alias
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`

▽

△

Symbol:	const_reverse_iterator
Syntax:	`using const_reverse_iterator = std::reverse_iterator<const_iterator>;`
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.1.5 Type Alias: difference_type

[SWS_CORE_02110] Definition of API type ara::core::StringView::difference_type

Status: DRAFT

⌈

Kind:	type alias
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Symbol:	difference_type
Syntax:	`using difference_type = std::ptrdiff_t;`
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.1.6 Type Alias: iterator

[SWS_CORE_02106] Definition of API type ara::core::StringView::iterator

Status: DRAFT

⌈

Kind:	type alias
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Symbol:	iterator
Syntax:	`using iterator = const_iterator;`
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.1.7 Type Alias: pointer

[SWS_CORE_02101] Definition of API type ara::core::StringView::pointer

Status: DRAFT

⌈

Kind:	type alias
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Symbol:	pointer
Syntax:	`using pointer = value_type*;`
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. Pointer type shall be specialized to `value_type`

⌋

8.7.2.1.8 Type Alias: reference

[SWS_CORE_02103] Definition of API type ara::core::StringView::reference

Status: DRAFT

⌈

Kind:	type alias
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Symbol:	reference
Syntax:	`using reference = value_type&;`
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. Reference type shall be specialized to `value_type`

⌋

8.7.2.1.9 Type Alias: reverse_iterator

[SWS_CORE_02108] Definition of API type ara::core::StringView::reverse_iterator

Status: DRAFT

⌈

Kind:	type alias
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Symbol:	reverse_iterator

▽

△

Syntax:	`using reverse_iterator = const_reverse_iterator;`
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.1.10 Type Alias: size_type

[SWS_CORE_02109] Definition of API type ara::core::StringView::size_type

Status: DRAFT

⌈

Kind:	type alias
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Symbol:	size_type
Syntax:	`using size_type = std::size_t;`
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.1.11 Type Alias: value_type

[SWS_CORE_02100] Definition of API type ara::core::StringView::value_type

Status: DRAFT

⌈

Kind:	type alias
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Symbol:	value_type
Syntax:	`using value_type = char;`
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. Type shall be specialized to `char`

⌋

8.7.2.2 Public Member Variables

8.7.2.2.1 npos

[SWS_CORE_02111] Definition of API variable ara::core::StringView::npos

Status: DRAFT

⌈

Kind:	variable
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Symbol:	npos
Type:	size_type
Syntax:	`static constexpr size_type npos = size_type(-1);`
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.3 Public Member Functions

8.7.2.3.1 Special Member Functions

8.7.2.3.1.1 Default Constructor

[SWS_CORE_02112] Definition of API function ara::core::StringView::StringView

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr StringView () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.3.1.2 Copy Constructor

[SWS_CORE_02115] Definition of API function ara::core::StringView::StringView

Status: DRAFT
Upstream requirements: RS_AP_00145

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr StringView (const StringView &other) noexcept=default;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.3.1.3 Move Constructor

[SWS_CORE_02117] Definition of API function ara::core::StringView::StringView

Status: DRAFT
Upstream requirements: RS_AP_00145

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr StringView (StringView &&other) noexcept=default;`
Exception Safety:	exception safe
Description:	**[AUTOSAR defined]** not present in ISO-14882. Move constructor

⌋

8.7.2.3.1.4 Copy Assignment Operator

[SWS_CORE_02116] Definition of API function ara::core::StringView::operator=

Status: DRAFT
Upstream requirements: RS_AP_00145

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr StringView & operator= (const StringView &other) noexcept=default;`

▽

△

Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.3.1.5 Move Assignment Operator

[SWS_CORE_02118] Definition of API function ara::core::StringView::operator=

Status: DRAFT
Upstream requirements: RS_AP_00145

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr StringView & operator= (StringView &&other) noexcept=default;`
Exception Safety:	exception safe
Description:	**[AUTOSAR defined]** not present in ISO-14882. Move assignment operator

⌋

8.7.2.3.1.6 Destructor

[SWS_CORE_02119] Definition of API function ara::core::StringView::~StringView

Status: DRAFT
Upstream requirements: RS_AP_00145

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`~StringView () noexcept=default;`
Exception Safety:	exception safe
Description:	**[AUTOSAR defined]** not present in ISO-14882. Destructor

⌋

8.7.2.3.2 Constructors

8.7.2.3.2.1 StringView(const char *, size_type)

[SWS_CORE_02114] Definition of API function ara::core::StringView::StringView

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr StringView (const char *str, size_type len) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. The function is `noexcept`

⌋

8.7.2.3.2.2 StringView(const char *)

[SWS_CORE_02113] Definition of API function ara::core::StringView::StringView

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr StringView (const char *str) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. The function is `noexcept`

⌋

8.7.2.3.3 Capacity

8.7.2.3.3.1 empty

[SWS_CORE_02131] Definition of API function ara::core::StringView::empty

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`[[nodiscard]] constexpr bool empty () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.3.3.2 length

[SWS_CORE_02129] Definition of API function ara::core::StringView::length

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type length () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.3.3.3 max_size

[SWS_CORE_02130] Definition of API function ara::core::StringView::max_size

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type max_size () const noexcept;`
Exception Safety:	exception safe

▽

△

Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.3.3.4 size

[SWS_CORE_02128] Definition of API function ara::core::StringView::size

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type size () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.3.4 Element access

8.7.2.3.4.1 at

[SWS_CORE_02133] Definition of API function ara::core::StringView::at

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/string_view.h"	
Scope:	`class ara::core::StringView`	
Syntax:	`constexpr const_reference at (size_type pos) const noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdOut-OfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. The function is `noexcept` 2. if (pos >= `size()`) `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range`	

⌋

8.7.2.3.4.2 back

[SWS_CORE_02135] Definition of API function ara::core::StringView::back

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr const_reference back () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. The function is `noexcept`
Undefined Behavior:	The behavior of this function is undefined if: (`empty()` == true)

⌋

8.7.2.3.4.3 data

[SWS_CORE_02136] Definition of API function ara::core::StringView::data

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr const_pointer data () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.3.4.4 front

[SWS_CORE_02134] Definition of API function ara::core::StringView::front

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr const_reference front () const noexcept;`
Exception Safety:	exception safe

▽

△

Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. The function is `noexcept`
Undefined Behavior:	The behavior of this function is undefined if: (`empty()` == true)

⌋

8.7.2.3.4.5 operator[]

[SWS_CORE_02132] Definition of API function ara::core::StringView::operator[]

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr const_reference operator[] (size_type pos) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. The function is `noexcept`
Undefined Behavior:	The behavior of this function is undefined if: (pos >= `size()`)

⌋

8.7.2.3.4.6 remove_prefix

[SWS_CORE_02137] Definition of API function ara::core::StringView::remove_prefix

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr void remove_prefix (size_type n) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. The function is `noexcept`
Undefined Behavior:	The behavior of this function is undefined if: (n > `size()`)

⌋

8.7.2.3.4.7 remove_suffix

[SWS_CORE_02138] Definition of API function ara::core::StringView::remove_suffix

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr void remove_suffix (size_type n) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. The function is `noexcept`
Undefined Behavior:	The behavior of this function is undefined if: (n > `size()`)

⌋

8.7.2.3.4.8 swap

[SWS_CORE_02139] Definition of API function ara::core::StringView::swap

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr void swap (StringView &other) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. The function is `noexcept`
Undefined Behavior:	The behavior of this function is undefined if: (n > `size()`)

⌋

8.7.2.3.5 Iterators

8.7.2.3.5.1 begin() const

[SWS_CORE_02120] Definition of API function ara::core::StringView::begin

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr const_iterator begin () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.3.5.2 cbegin() const

[SWS_CORE_02122] Definition of API function ara::core::StringView::cbegin

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr const_iterator cbegin () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.3.5.3 cend() const

[SWS_CORE_02123] Definition of API function ara::core::StringView::cend

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr const_iterator cend () const noexcept;`
Exception Safety:	exception safe

▽

△

Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.3.5.4 crbegin() const

[SWS_CORE_02126] Definition of API function ara::core::StringView::crbegin

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr const_reverse_iterator crbegin () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.3.5.5 crend() const

[SWS_CORE_02127] Definition of API function ara::core::StringView::crend

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr const_reverse_iterator crend () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.3.5.6 end() const

[SWS_CORE_02121] Definition of API function ara::core::StringView::end

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr const_iterator end () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.3.5.7 rbegin() const

[SWS_CORE_02124] Definition of API function ara::core::StringView::rbegin

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr const_reverse_iterator rbegin () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.3.5.8 rend() const

[SWS_CORE_02125] Definition of API function ara::core::StringView::rend

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr const_reverse_iterator rend () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.3.6 Operations

8.7.2.3.6.1 compare(size_type, size_type, StringView) const

[SWS_CORE_02143] Definition of API function ara::core::StringView::compare

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/string_view.h"	
Scope:	`class ara::core::StringView`	
Syntax:	`constexpr int compare (size_type pos1, size_type n1, StringView s) const noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdOut-OfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. The function is `noexcept` 2. if (pos1 > `size()`) `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range`	

⌋

8.7.2.3.6.2 compare(StringView) const

[SWS_CORE_02142] Definition of API function ara::core::StringView::compare

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr int compare (StringView s) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.3.6.3 compare(const char *) const

[SWS_CORE_02145] Definition of API function ara::core::StringView::compare

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr int compare (const char *s) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. The function is `noexcept`

⌋

8.7.2.3.6.4 compare(size_type, size_type, const char *) const

[SWS_CORE_02146] Definition of API function ara::core::StringView::compare

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/string_view.h"	
Scope:	`class ara::core::StringView`	
Syntax:	`constexpr int compare (size_type pos1, size_type n1, const char *s) const noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdOut-OfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. The function is `noexcept` 2. if (pos1 > `size()`) `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range`	

⌋

8.7.2.3.6.5 compare(size_type, size_type, StringView, size_type, size_type) const

[SWS_CORE_02144] Definition of API function ara::core::StringView::compare

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/string_view.h"	
Scope:	`class ara::core::StringView`	
Syntax:	`constexpr int compare (size_type pos1, size_type n1, StringView s, size_type pos2, size_type n2) const noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdOut-OfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. The function is `noexcept` 2. if (pos1 > `size()` || pos2 > s.`size()`) `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range`	

⌋

8.7.2.3.6.6 compare(size_type, size_type, const char *, size_type) const

[SWS_CORE_02147] Definition of API function ara::core::StringView::compare

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/string_view.h"	
Scope:	`class ara::core::StringView`	
Syntax:	`constexpr int compare (size_type pos1, size_type n1, const char *s, size_type n2) const noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdOut-OfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. The function is `noexcept` 2. if (pos1 > `size()`) `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range`	

⌋

8.7.2.3.6.7 contains(const char *) const

[SWS_CORE_02156] Definition of API function ara::core::StringView::contains

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr bool contains (const char *str) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]` in [16] except for the following deviations: 1. The function is `noexcept`

⌋

8.7.2.3.6.8 contains(StringView) const

[SWS_CORE_02154] Definition of API function ara::core::StringView::contains

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr bool contains (StringView sv) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]` in [16]

⌋

8.7.2.3.6.9 contains(char) const

[SWS_CORE_02155] Definition of API function ara::core::StringView::contains

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr bool contains (char c) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]` in [16]

⌋

8.7.2.3.6.10 copy

[SWS_CORE_02140] Definition of API function ara::core::StringView::copy

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/string_view.h"	
Scope:	`class ara::core::StringView`	
Syntax:	`size_type copy (char *s, size_type n, size_type pos=0) const noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdOut-OfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. The function is `noexcept` 2. if (pos > `size()`) `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range`	

⌋

8.7.2.3.6.11 ends_with(char) const

[SWS_CORE_02152] Definition of API function ara::core::StringView::ends_with

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr bool ends_with (char c) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]` in [15]

⌋

8.7.2.3.6.12 ends_with(StringView) const

[SWS_CORE_02151] Definition of API function ara::core::StringView::ends_with

Status: DRAFT

⌈

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

▽

△

Syntax:	`constexpr bool ends_with (StringView sv) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]` in [15]

⌋

8.7.2.3.6.13 ends_with(const char *) const

[SWS_CORE_02153] Definition of API function ara::core::StringView::ends_with

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr bool ends_with (const char *str) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]` in [15] 1. The function is `noexcept`

⌋

8.7.2.3.6.14 find(char, size_type) const

[SWS_CORE_02158] Definition of API function ara::core::StringView::find

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find (char c, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.3.6.15 find(StringView, size_type) const

[SWS_CORE_02157] Definition of API function ara::core::StringView::find

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find (StringView s, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.3.6.16 find(const char *, size_type, size_type) const

[SWS_CORE_02159] Definition of API function ara::core::StringView::find

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find (const char *s, size_type pos, size_type n) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. The function is `noexcept`

⌋

8.7.2.3.6.17 find(const char *, size_type) const

[SWS_CORE_02160] Definition of API function ara::core::StringView::find

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find (const char *s, size_type pos=0) const noexcept;`

▽

△

Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. The function is `noexcept`

⌋

8.7.2.3.6.18 find_first_not_of(char, size_type) const

[SWS_CORE_02174] Definition of API function ara::core::StringView::find_first_not_of

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find_first_not_of (char c, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.3.6.19 find_first_not_of(const char *, size_type) const

[SWS_CORE_02176] Definition of API function ara::core::StringView::find_first_not_of

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find_first_not_of (const char *s, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. The function is `noexcept`

⌋

8.7.2.3.6.20 find_first_not_of(const char *, size_type, size_type) const

[SWS_CORE_02175] Definition of API function ara::core::StringView::find_first_not_of

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find_first_not_of (const char *s, size_type pos, size_type n) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. The function is `noexcept`

⌋

8.7.2.3.6.21 find_first_not_of(StringView, size_type) const

[SWS_CORE_02173] Definition of API function ara::core::StringView::find_first_not_of

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find_first_not_of (StringView s, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.3.6.22 find_first_of(const char *, size_type) const

[SWS_CORE_02168] Definition of API function ara::core::StringView::find_first_of

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find_first_of (const char *s, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. The function is `noexcept`

⌋

8.7.2.3.6.23 find_first_of(const char *, size_type, size_type) const

[SWS_CORE_02167] Definition of API function ara::core::StringView::find_first_of

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find_first_of (const char *s, size_type pos, size_type n) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. The function is `noexcept`

⌋

8.7.2.3.6.24 find_first_of(StringView, size_type) const

[SWS_CORE_02165] Definition of API function ara::core::StringView::find_first_of

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find_first_of (StringView s, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.3.6.25 find_first_of(char, size_type) const

[SWS_CORE_02166] Definition of API function ara::core::StringView::find_first_of

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find_first_of (char c, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.3.6.26 find_last_not_of(const char *, size_type) const

[SWS_CORE_02180] Definition of API function ara::core::StringView::find_last_not_of

Status: DRAFT

⌈

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

▽

△

Syntax:	`constexpr size_type find_last_not_of (const char *s, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. The function is `noexcept`

⌋

8.7.2.3.6.27 find_last_not_of(const char *, size_type, size_type) const

[SWS_CORE_02179] Definition of API function ara::core::StringView::find_last_not_of

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find_last_not_of (const char *s, size_type pos, size_type n) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. The function is `noexcept`

⌋

8.7.2.3.6.28 find_last_not_of(StringView, size_type) const

[SWS_CORE_02177] Definition of API function ara::core::StringView::find_last_not_of

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find_last_not_of (StringView s, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.3.6.29 find_last_not_of(char, size_type) const

[SWS_CORE_02178] Definition of API function ara::core::StringView::find_last_not_of

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find_last_not_of (char c, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.3.6.30 find_last_of(const char *, size_type, size_type) const

[SWS_CORE_02171] Definition of API function ara::core::StringView::find_last_of

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find_last_of (const char *s, size_type pos, size_type n) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. The function is `noexcept`

⌋

8.7.2.3.6.31 find_last_of(char, size_type) const

[SWS_CORE_02170] Definition of API function ara::core::StringView::find_last_of

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find_last_of (char c, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.3.6.32 find_last_of(StringView, size_type) const

[SWS_CORE_02169] Definition of API function ara::core::StringView::find_last_of

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type find_last_of (StringView s, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.3.6.33 find_last_of(const char *, size_type) const

[SWS_CORE_02172] Definition of API function ara::core::StringView::find_last_of

Status: DRAFT

⌈

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

▽

△

Syntax:	`constexpr size_type find_last_of (const char *s, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. The function is `noexcept`

⌋

8.7.2.3.6.34 rfind(char, size_type) const

[SWS_CORE_02162] Definition of API function ara::core::StringView::rfind

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type rfind (char c, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.3.6.35 rfind(StringView, size_type) const

[SWS_CORE_02161] Definition of API function ara::core::StringView::rfind

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type rfind (StringView s, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]`

⌋

8.7.2.3.6.36 rfind(const char *, size_type) const

[SWS_CORE_02164] Definition of API function ara::core::StringView::rfind

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type rfind (const char *s, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. The function is `noexcept`

⌋

8.7.2.3.6.37 rfind(const char *, size_type, size_type) const

[SWS_CORE_02163] Definition of API function ara::core::StringView::rfind

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr size_type rfind (const char *s, size_type pos, size_type n) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. The function is `noexcept`

⌋

8.7.2.3.6.38 starts_with(char) const

[SWS_CORE_02149] Definition of API function ara::core::StringView::starts_with

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr bool starts_with (char c) const noexcept;`

▽

△

Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]` in [15]

⌋

8.7.2.3.6.39 starts_with(const char *) const

[SWS_CORE_02150] Definition of API function ara::core::StringView::starts_with

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr bool starts_with (const char *str) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]` in [15] 1. The function is `noexcept`

⌋

8.7.2.3.6.40 starts_with(StringView) const

[SWS_CORE_02148] Definition of API function ara::core::StringView::starts_with

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`class ara::core::StringView`
Syntax:	`constexpr bool starts_with (StringView sv) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[string.view]` in [15]

⌋

8.7.2.3.6.41 substr

[SWS_CORE_02141] Definition of API function ara::core::StringView::substr

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/string_view.h"	
Scope:	`class ara::core::StringView`	
Syntax:	`constexpr StringView substr (size_type pos=0, size_type n=npos) const noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdOut-OfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[string.view]` except for the following deviations: 1. if (pos > `size()`) `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range`	

⌋

8.7.3 Struct: hash

[SWS_CORE_02189] Definition of API class std::hash< ara::core::StringView >

Status: DRAFT

⌈

Kind:	struct
Header file:	#include "ara/core/string_view.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	namespace std
Symbol:	hash< ara::core::StringView >
Syntax:	`template <>` `struct hash< ara::core::StringView > final {...};`
Description:	Specialization of `std::hash` (hashing function for `ara::core::StringView`)

⌋

8.7.3.1 Public Member Functions

8.7.3.1.1 Member Functions

8.7.3.1.1.1 operator()

[SWS_CORE_02190] Definition of API function std::hash< ara::core::StringView >::operator()

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string_view.h"
Scope:	`struct std::hash< ara::core::StringView >`
Syntax:	`size_t operator() (ara::core::StringView const &v) const noexcept;`
Exception Safety:	exception safe
Description:	Invoke the hashing function

⌋

8.8 Header: ara/core/string.h

[SWS_CORE_03002] Definition of Header File ara/core/string.h

Upstream requirements: RS_AP_00130

⌈

Kind:	Header File
Syntax:	`ara/core/string.h`
Description:	Implements `[string.syn]` in [11]. Unless explicitly overridden in the API documentation, APIs always adhere in behavior to the ISO specification. To reduce duplication, the APIs here specify only any deviations from [11] or any AUTOSAR specific overrides. For APIs which would ordinarily throw `Exception`s, and those `Exception`s have been suppressed in favor of AUTOSAR violations, generally the semantics of the `Exception` from [11] still apply (unless locally overridden here).

⌋

8.8.1 Non-Member Types

8.8.1.1 Type Alias: String

[SWS_CORE_03001] Definition of API type ara::core::String

Status: DRAFT

⌈

Kind:	type alias
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Symbol:	String
Syntax:	`using String = BasicString<>;`
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2 Non-Member Functions

8.8.2.1 Other

8.8.2.1.1 getline(std::basic_istream<charT, traits>&&, BasicString<Allocator>&, charT)

[SWS_CORE_03402] Definition of API function ara::core::getline ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <class charT = char, class traits = std::char_traits<char>, class Allocator = ara::core::PolymorphicAllocator<char>> std::basic_istream< charT, traits > & getline (std::basic_istream< charT, traits > &&is, BasicString< Allocator > &str, charT delim);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. Template args are specialized to `char`

⌋

8.8.2.1.2 getline(std::basic_istream<charT, traits>&, BasicString<Allocator>&, charT)

[SWS_CORE_03401] Definition of API function ara::core::getline ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <class charT = char, class traits = std::char_traits<char>, class Allocator = ara::core::PolymorphicAllocator<char>> std::basic_istream< charT, traits > & getline (std::basic_istream< charT, traits > &is, BasicString< Allocator > &str, charT delim);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. Template args are specialized to `char`

⌋

8.8.2.1.3 getline(std::basic_istream<charT, traits>&&, BasicString<Allocator>&)

[SWS_CORE_03404] Definition of API function ara::core::getline ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <class charT = char, class traits = std::char_traits<char>, class Allocator = ara::core::PolymorphicAllocator<char>> std::basic_istream< charT, traits > & getline (std::basic_istream< charT, traits > &&is, BasicString< Allocator > &str);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. Template args are specialized to `char`

⌋

8.8.2.1.4 getline(std::basic_istream<charT, traits>&, BasicString<Allocator>&)

[SWS_CORE_03403] Definition of API function ara::core::getline ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <class charT = char, class traits = std::char_traits<char>, class Allocator = ara::core::PolymorphicAllocator<char>> std::basic_istream< charT, traits > & getline (std::basic_istream< charT, traits > &is, BasicString< Allocator > &str);`
Exception Safety:	not exception safe

▽

△

Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. Template args are specialized to `char`

⌋

8.8.2.1.5 operator!=(const charT *, const BasicString<Allocator>&)

[SWS_CORE_03368] Definition of API function ara::core::operator!=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <typename Allocator, typename charT = typename std::allocator_traits<Allocator>::value_type> bool operator!= (const charT *lhs, const BasicString< Allocator > &rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2.1.6 operator!=(const BasicString<Allocator>&, const charT *)

[SWS_CORE_03369] Definition of API function ara::core::operator!=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <typename Allocator, typename charT = typename std::allocator_traits<Allocator>::value_type> bool operator!= (const BasicString< Allocator > &lhs, const charT *rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2.1.7 operator!=(const BasicString<Allocator>&, const BasicString<Allocator>&)

[SWS_CORE_03367] Definition of API function ara::core::operator!=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <typename Allocator, typename charT = typename std::allocator_traits<Allocator>::value_type> bool operator!= (const BasicString< Allocator > &lhs, const BasicString< Allocator > &rhs) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2.1.8 operator+(charT, const BasicString<Allocator>&)

[SWS_CORE_03358] Definition of API function ara::core::operator+

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <typename Allocator, typename charT = typename std::allocator_traits<Allocator>::value_type> BasicString< Allocator > operator+ (charT lhs, const BasicString< Allocator > &rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2.1.9 operator+(charT, BasicString<Allocator>&&)

[SWS_CORE_03359] Definition of API function ara::core::operator+

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <typename Allocator, typename charT = typename std::allocator_traits<Allocator>::value_type> BasicString< Allocator > operator+ (charT lhs, BasicString< Allocator > &&rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2.1.10 operator+(const charT *, BasicString<Allocator>&&)

[SWS_CORE_03357] Definition of API function ara::core::operator+

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <typename Allocator, typename charT = typename std::allocator_traits<Allocator>::value_type> BasicString< Allocator > operator+ (const charT *lhs, BasicString< Allocator > &&rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2.1.11 operator+(BasicString<Allocator>&&, const charT *)

[SWS_CORE_03361] Definition of API function ara::core::operator+

Status: DRAFT

⌈

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

▽

△

Syntax:	`template <typename Allocator, typename charT = typename std::allocator_traits<Allocator>::value_type> BasicString< Allocator > operator+ (BasicString< Allocator > &&lhs, const charT *rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2.1.12 operator+(const BasicString<Allocator>&, charT)

[SWS_CORE_03362] Definition of API function ara::core::operator+

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <typename Allocator, typename charT = typename std::allocator_traits<Allocator>::value_type> BasicString< Allocator > operator+ (const BasicString< Allocator > &lhs, charT rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2.1.13 operator+(BasicString<Allocator>&&, charT)

[SWS_CORE_03363] Definition of API function ara::core::operator+

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <typename Allocator, typename charT = typename std::allocator_traits<Allocator>::value_type> BasicString< Allocator > operator+ (BasicString< Allocator > &&lhs, charT rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2.1.14 operator+(const charT *, const BasicString<Allocator>&)

[SWS_CORE_03356] Definition of API function ara::core::operator+

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <typename Allocator, typename charT = typename std::allocator_traits<Allocator>::value_type> BasicString< Allocator > operator+ (const charT *lhs, const BasicString< Allocator > &rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2.1.15 operator+(BasicString<Allocator>&&, BasicString<Allocator>&&)

[SWS_CORE_03355] Definition of API function ara::core::operator+

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <typename Allocator> BasicString< Allocator > operator+ (BasicString< Allocator > &&lhs, BasicString< Allocator > &&rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2.1.16 operator+(const BasicString<Allocator>&, BasicString<Allocator>&&)

[SWS_CORE_03354] Definition of API function ara::core::operator+

Status: DRAFT

⌈

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

▽

△

Syntax:	`template <typename Allocator>` `BasicString< Allocator > operator+ (const BasicString< Allocator > &lhs, BasicString< Allocator > &&rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2.1.17 operator+(BasicString<Allocator>&&, const BasicString<Allocator>&)

[SWS_CORE_03353] Definition of API function ara::core::operator+

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <typename Allocator>` `BasicString< Allocator > operator+ (BasicString< Allocator > &&lhs, const BasicString< Allocator > &rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2.1.18 operator+(const BasicString<Allocator>&, const BasicString<Allocator>&)

[SWS_CORE_03352] Definition of API function ara::core::operator+

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <typename Allocator>` `BasicString< Allocator > operator+ (const BasicString< Allocator > &lhs, const BasicString< Allocator > &rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2.1.19 operator+(const BasicString<Allocator>&, const charT *)

[SWS_CORE_03360] Definition of API function ara::core::operator+

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <typename Allocator, typename charT = typename std::allocator_traits<Allocator>::value_type> BasicString< Allocator > operator+ (const BasicString< Allocator > &lhs, const charT *rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2.1.20 operator<(const charT *, const BasicString<Allocator>&)

[SWS_CORE_03372] Definition of API function ara::core::operator<

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <typename Allocator, typename charT = typename std::allocator_traits<Allocator>::value_type> bool operator< (const charT *lhs, const BasicString< Allocator > &rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2.1.21 operator<(const BasicString<Allocator>&, const BasicString<Allocator>&)

[SWS_CORE_03370] Definition of API function ara::core::operator<

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <typename Allocator, typename charT = typename std::allocator_traits<Allocator>::value_type> bool operator< (const BasicString< Allocator > &lhs, const BasicString< Allocator > &rhs) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2.1.22 operator<(const BasicString<Allocator>&, const charT *)

[SWS_CORE_03371] Definition of API function ara::core::operator<

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <typename Allocator, typename charT = typename std::allocator_traits<Allocator>::value_type> bool operator< (const BasicString< Allocator > &lhs, const charT *rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2.1.23 operator<<(std::basic_ostream<charT, traits>&, const BasicString<Allocator>&)

[SWS_CORE_03400] Definition of API function ara::core::operator<< ⌈

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

▽

△

Syntax:	`template <class charT = char, class traits = std::char_traits<char>, class Allocator = ara::core::PolymorphicAllocator<char>> std::basic_ostream< charT, traits > & operator<< (std::basic_ostream< charT, traits > &os, const BasicString< Allocator > &str);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. Template args are specialized to `char`

⌋

8.8.2.1.24 operator<=(const BasicString<Allocator>&, const charT *)

[SWS_CORE_03377] Definition of API function ara::core::operator<=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <typename Allocator, typename charT = typename std::allocator_traits<Allocator>::value_type> bool operator<= (const BasicString< Allocator > &lhs, const charT *rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2.1.25 operator<=(const BasicString<Allocator>&, const BasicString<Allocator>&)

[SWS_CORE_03376] Definition of API function ara::core::operator<=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <typename Allocator, typename charT = typename std::allocator_traits<Allocator>::value_type> bool operator<= (const BasicString< Allocator > &lhs, const BasicString< Allocator > &rhs) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2.1.26 operator<=(const charT *, const BasicString<Allocator>&)

[SWS_CORE_03378] Definition of API function ara::core::operator<=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <typename Allocator, typename charT = typename std::allocator_traits<Allocator>::value_type> bool operator<= (const charT *lhs, const BasicString< Allocator > &rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2.1.27 operator==(const BasicString<Allocator>&, const BasicString<Allocator>&)

[SWS_CORE_03364] Definition of API function ara::core::operator==

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <typename Allocator, typename charT = typename std::allocator_traits<Allocator>::value_type> bool operator== (const BasicString< Allocator > &lhs, const BasicString< Allocator > &rhs) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2.1.28 operator==(const BasicString<Allocator>&, const charT *)

[SWS_CORE_03366] Definition of API function ara::core::operator==

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <typename Allocator, typename charT = typename std::allocator_traits<Allocator>::value_type> bool operator== (const BasicString< Allocator > &lhs, const charT *rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2.1.29 operator==(const charT *, const BasicString<Allocator>&)

[SWS_CORE_03365] Definition of API function ara::core::operator==

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <typename Allocator, typename charT = typename std::allocator_traits<Allocator>::value_type> bool operator== (const charT *lhs, const BasicString< Allocator > &rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2.1.30 operator>(const BasicString<Allocator>&, const BasicString<Allocator>&)

[SWS_CORE_03373] Definition of API function ara::core::operator>

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <typename Allocator, typename charT = typename std::allocator_traits<Allocator>::value_type> bool operator> (const BasicString< Allocator > &lhs, const BasicString< Allocator > &rhs) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2.1.31 operator>(const BasicString<Allocator>&, const charT *)

[SWS_CORE_03374] Definition of API function ara::core::operator>

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <typename Allocator, typename charT = typename std::allocator_traits<Allocator>::value_type> bool operator> (const BasicString< Allocator > &lhs, const charT *rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2.1.32 operator>(const charT *, const BasicString<Allocator>&)

[SWS_CORE_03375] Definition of API function ara::core::operator>

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <typename Allocator, typename charT = typename std::allocator_traits<Allocator>::value_type> bool operator> (const charT *lhs, const BasicString< Allocator > &rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2.1.33 operator>=(const BasicString<Allocator>&, const BasicString<Allocator>&)

[SWS_CORE_03379] Definition of API function ara::core::operator>=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <typename Allocator, typename charT = typename std::allocator_traits<Allocator>::value_type> bool operator>= (const BasicString< Allocator > &lhs, const BasicString< Allocator > &rhs) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2.1.34 operator>=(const charT *, const BasicString<Allocator>&)

[SWS_CORE_03381] Definition of API function ara::core::operator>=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <typename Allocator, typename charT = typename std::allocator_traits<Allocator>::value_type> bool operator>= (const charT *lhs, const BasicString< Allocator > &rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2.1.35 operator>=(const BasicString<Allocator>&, const charT *)

[SWS_CORE_03380] Definition of API function ara::core::operator>=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <typename Allocator, typename charT = typename std::allocator_traits<Allocator>::value_type> bool operator>= (const BasicString< Allocator > &lhs, const charT *rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2.1.36 operator>>(std::basic_istream<charT, traits>&, BasicString<Allocator>&)

[SWS_CORE_03399] Definition of API function ara::core::operator>> ⌈

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

▽

△

Syntax:	`template <class charT = char, class traits = std::char_traits<char>, class Allocator = ara::core::PolymorphicAllocator<char>> std::basic_istream< charT, traits > & operator>> (std::basic_istream< charT, traits > &is, BasicString< Allocator > &str);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. Template args are specialized to `char`

⌋

8.8.2.1.37 stod(const ara::core::String&, size_t *)

[SWS_CORE_03397] Definition of API function ara::core::stod ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`double stod (const ara::core::String &str, size_t *idx=nullptr);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. Arg#1=`std::string&` replaced with `ara::core::String&`

⌋

8.8.2.1.38 stof(const ara::core::String&, size_t *)

[SWS_CORE_03396] Definition of API function ara::core::stof ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`float stof (const ara::core::String &str, size_t *idx=nullptr);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. Arg#1=`std::string&` replaced with `ara::core::String&`

⌋

8.8.2.1.39 stoi(const ara::core::String&, size_t *, int)

[SWS_CORE_03391] Definition of API function ara::core::stoi ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`int stoi (const ara::core::String &str, size_t *idx=nullptr, int base=10);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. Arg#1=`std::string&` replaced with `ara::core::String&`

⌋

8.8.2.1.40 stol(const ara::core::String&, size_t *, int)

[SWS_CORE_03392] Definition of API function ara::core::stol ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`long stol (const ara::core::String &str, size_t *idx=nullptr, int base=10);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. Arg#1=`std::string&` replaced with `ara::core::String&`

⌋

8.8.2.1.41 stold(const ara::core::String&, size_t *)

[SWS_CORE_03398] Definition of API function ara::core::stold ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`long double stold (const ara::core::String &str, size_t *idx=nullptr);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. Arg#1=`std::string&` replaced with `ara::core::String&`

⌋

8.8.2.1.42 stoll(const ara::core::String&, size_t *, int)

[SWS_CORE_03394] Definition of API function ara::core::stoll ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`long long stoll (const ara::core::String &str, size_t *idx=nullptr, int base=10);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. Arg#1=`std::string&` replaced with `ara::core::String&`

⌋

8.8.2.1.43 stoul(const ara::core::String&, size_t *, int)

[SWS_CORE_03393] Definition of API function ara::core::stoul ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`unsigned long stoul (const ara::core::String &str, size_t *idx=nullptr, int base=10);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. Arg#1=`std::string&` replaced with `ara::core::String&`

⌋

8.8.2.1.44 stoull(const ara::core::String&, size_t *, int)

[SWS_CORE_03395] Definition of API function ara::core::stoull ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`unsigned long long stoull (const ara::core::String &str, size_t *idx=nullptr, int base=10);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. Arg#1=`std::string&` replaced with `ara::core::String&`

⌋

8.8.2.1.45 swap(BasicString<Allocator>&, BasicString<Allocator>&)

[SWS_CORE_03296] Definition of API function ara::core::swap

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`template <typename Allocator>` `void swap (BasicString< Allocator > &lhs, BasicString< Allocator > &rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.2.1.46 to_string(unsigned long long)

[SWS_CORE_03387] Definition of API function ara::core::to_string ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`String to_string (unsigned long long val);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. Return type `std::string` replaced with `ara::core::String`

⌋

8.8.2.1.47 to_string(float)

[SWS_CORE_03388] Definition of API function ara::core::to_string ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`String to_string (float val);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. Return type `std::string` replaced with `ara::core::String`

⌋

8.8.2.1.48 to_string(double)

[SWS_CORE_03389] Definition of API function ara::core::to_string ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`String to_string (double val);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. Return type `std::string` replaced with `ara::core::String`

⌋

8.8.2.1.49 to_string(long long)

[SWS_CORE_03386] Definition of API function ara::core::to_string ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`String to_string (long long val);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. Return type `std::string` replaced with `ara::core::String`

⌋

8.8.2.1.50 to_string(unsigned long)

[SWS_CORE_03385] Definition of API function ara::core::to_string ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`String to_string (unsigned long val);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. Return type `std::string` replaced with `ara::core::String`

⌋

8.8.2.1.51 to_string(long)

[SWS_CORE_03384] Definition of API function ara::core::to_string ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`String to_string (long val);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. Return type `std::string` replaced with `ara::core::String`

⌋

8.8.2.1.52 to_string(unsigned)

[SWS_CORE_03383] Definition of API function ara::core::to_string ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`String to_string (unsigned val);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. Return type `std::string` replaced with `ara::core::String`

⌋

8.8.2.1.53 to_string(int)

[SWS_CORE_03382] Definition of API function ara::core::to_string ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`String to_string (int val);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. Return type `std::string` replaced with `ara::core::String`

⌋

8.8.2.1.54 to_string(long double)

[SWS_CORE_03390] Definition of API function ara::core::to_string ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	namespace ara::core
Syntax:	`String to_string (long double val);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. Return type `std::string` replaced with `ara::core::String`

⌋

8.8.3 Class: BasicString

[SWS_CORE_03000] 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 = {<fq-allocator>}>` `class BasicString {...};`	
Template param:	typename Allocator = `{<fq-allocator>}`	--
Description:	Implementation of `std::pmr::basic_string` type from `[basic.string]`	
Descriptors:	`{<fq-allocator>}`	Implementation-defined. Fully qualified `Allocator` class. The Allocator shall satisfy `[allocator.requirements.completeness]` in [11]

⌋

8.8.3.1 Public Member Types

8.8.3.1.1 Type Alias: allocator_type

[SWS_CORE_03005] Definition of API type ara::core::BasicString::allocator_type ⌈

Kind:	type alias
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Symbol:	allocator_type
Syntax:	`using allocator_type = Allocator;`

▽

△

Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.1.2 Type Alias: const_iterator

[SWS_CORE_00072] Definition of API type ara::core::BasicString::const_iterator

⌈

Kind:	type alias	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Symbol:	const_iterator	
Syntax:	`using const_iterator = {<const-iterator-type>};`	
Description:	As per corresponding usage in `[basic.string]`	
Implementation Defined Behavior:	`{<const-iterator-type>}` shall be an implementation-defined type	
Descriptors:	`{<const-iterator-type>}`	implementation-defined

⌋

8.8.3.1.3 Type Alias: const_pointer

[SWS_CORE_03015] Definition of API type ara::core::BasicString::const_pointer

⌈

Kind:	type alias
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Symbol:	const_pointer
Syntax:	`using const_pointer = typename std::allocator_ traits<Allocator>::const_pointer;`
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.1.4 Type Alias: const_reference

[SWS_CORE_03017] Definition of API type ara::core::BasicString::const_reference ⌈

Kind:	type alias
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Symbol:	const_reference
Syntax:	`using const_reference = const value_type&;`
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.1.5 Type Alias: const_reverse_iterator

[SWS_CORE_03019] Definition of API type ara::core::BasicString::const_reverse_iterator ⌈

Kind:	type alias
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Symbol:	const_reverse_iterator
Syntax:	`using const_reverse_iterator = std::reverse_iterator<const_iterator>;`
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.1.6 Type Alias: difference_type

[SWS_CORE_03013] Definition of API type ara::core::BasicString::difference_type ⌈

Kind:	type alias
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Symbol:	difference_type
Syntax:	`using difference_type = typename std::allocator_traits<Allocator>::difference_type;`
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.1.7 Type Alias: iterator

[SWS_CORE_00071] Definition of API type ara::core::BasicString::iterator ⌈

Kind:	type alias	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Symbol:	iterator	
Syntax:	`using iterator = {<iterator-type>};`	
Description:	As per corresponding usage in `[basic.string]`	
Implementation Defined Behavior:	`{<iterator-type>}` shall be an implementation-defined type	
Descriptors:	`{<iterator-type>}`	implementation-defined

⌋

8.8.3.1.8 Type Alias: pointer

[SWS_CORE_03014] Definition of API type ara::core::BasicString::pointer ⌈

Kind:	type alias
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Symbol:	pointer
Syntax:	`using pointer = typename std::allocator_traits<Allocator>::pointer;`
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.1.9 Type Alias: reference

[SWS_CORE_03016] Definition of API type ara::core::BasicString::reference ⌈

Kind:	type alias
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Symbol:	reference
Syntax:	`using reference = value_type&;`
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.1.10 Type Alias: reverse_iterator

[SWS_CORE_03018] Definition of API type ara::core::BasicString::reverse_iterator ⌈

Kind:	type alias
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Symbol:	reverse_iterator
Syntax:	`using reverse_iterator = std::reverse_iterator<iterator>;`
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.1.11 Type Alias: size_type

[SWS_CORE_03012] 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 = typename std::allocator_traits<Allocator>::size_type;`
Description:	As per corresponding member in `[basic.string]` except for the following deviations: 1. Type shall be specialized to `std::size_t`

⌋

8.8.3.1.12 Type Alias: traits_type

[SWS_CORE_03003] Definition of API type ara::core::BasicString::traits_type ⌈

Kind:	type alias
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Symbol:	traits_type
Syntax:	`using traits_type = typename std::char_traits<char>;`
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. Type shall be specialized to `std::char_traits<char>`

⌋

8.8.3.1.13 Type Alias: value_type

[SWS_CORE_03004] Definition of API type ara::core::BasicString::value_type ⌈

Kind:	type alias
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Symbol:	value_type
Syntax:	`using value_type = typename std::allocator_traits<Allocator>::value_type;`
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. Type shall be specialized to `std::allocator_traits<Allocator>::value_type`

⌋

8.8.3.2 Public Member Variables

8.8.3.2.1 npos

[SWS_CORE_00073] Definition of API variable ara::core::BasicString::npos ⌈

Kind:	variable
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Symbol:	npos
Type:	size_type const
Syntax:	`static constexpr size_type const npos = size_type(-1);`
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3 Public Member Functions

8.8.3.3.1 Special Member Functions

8.8.3.3.1.1 BasicString(BasicString&&)

[SWS_CORE_03022] Definition of API function ara::core::BasicString::BasicString

Upstream requirements: RS_AP_00145

⌈

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

▽

△

Syntax:	`BasicString (BasicString &&str) noexcept=default;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.1.2 BasicString()

[SWS_CORE_03024] Definition of API function ara::core::BasicString::BasicString ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString () noexcept(noexcept(Allocator()));`
Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.1.3 BasicString(const BasicString&)

[SWS_CORE_03020] Definition of API function ara::core::BasicString::BasicString

Upstream requirements: RS_AP_00145

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString (const BasicString &str);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.1.4 operator=(const BasicString&)

[SWS_CORE_03021] Definition of API function ara::core::BasicString::operator=

Upstream requirements: RS_AP_00145

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & operator= (const BasicString &str);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.1.5 operator=(BasicString&&)

[SWS_CORE_03023] Definition of API function ara::core::BasicString::operator=

Upstream requirements: RS_AP_00145

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & operator= (BasicString &&str) noexcept(std::allocator_traits< Allocator >::propagate_on_container_move_assignment::value||std::allocator_traits< Allocator >::is_always_equal::value);`
Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.1.6 Destructor

[SWS_CORE_03038] Definition of API function ara::core::BasicString::~BasicString

Upstream requirements: RS_AP_00145

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`~BasicString ();`

▽

△

Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.2 Constructors

8.8.3.3.2.1 BasicString(const BasicString&, const Allocator&)

[SWS_CORE_03033] Definition of API function ara::core::BasicString::Basic String ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString (const BasicString &, const Allocator &);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.2.2 BasicString(std::initializer_list<value_type>, const Allocator&)

[SWS_CORE_03032] Definition of API function ara::core::BasicString::Basic String ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString (std::initializer_list< value_type >, const Allocator &=Allocator());`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.2.3 BasicString(const Allocator&)

[SWS_CORE_03025] Definition of API function ara::core::BasicString::BasicString ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`explicit BasicString (const Allocator &a) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.2.4 BasicString(const BasicString&, size_type, const Allocator&)

[SWS_CORE_03026] Definition of API function ara::core::BasicString::BasicString ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString (const BasicString &str, size_type pos, const Allocator &a=Allocator());`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.2.5 BasicString(const BasicString&, size_type, size_type, const Allocator&)

[SWS_CORE_03027] Definition of API function ara::core::BasicString::BasicString ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString (const BasicString &str, size_type pos, size_type n, const Allocator &a=Allocator());`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.2.6 BasicString(const T&, size_type, size_type, const Allocator&)

[SWS_CORE_03303] Definition of API function ara::core::BasicString::BasicString

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`template <class T>` `BasicString (const T &t, size_type pos, size_type n, const Allocator &a=Allocator());`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.2.7 BasicString(StringView, const Allocator&)

[SWS_CORE_03302] Definition of API function ara::core::BasicString::BasicString

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`explicit BasicString (StringView sv, const Allocator &a=Allocator());`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.2.8 BasicString(const value_type *, size_type, const Allocator&)

[SWS_CORE_03028] Definition of API function ara::core::BasicString::BasicString ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString (const value_type *s, size_type n, const Allocator &a=Allocator());`
Exception Safety:	not exception safe

▽

△

Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.2.9 BasicString(const value_type *, const Allocator&)

[SWS_CORE_03029] Definition of API function ara::core::BasicString::BasicString ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString (const value_type *s, const Allocator &a=Allocator());`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.2.10 BasicString(size_type, value_type, const Allocator&)

[SWS_CORE_03030] Definition of API function ara::core::BasicString::BasicString ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString (size_type n, value_type c, const Allocator &a=Allocator());`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.2.11 BasicString(InputIterator, InputIterator, const Allocator&)

[SWS_CORE_03031] Definition of API function ara::core::BasicString::BasicString ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`template <class InputIterator>` `BasicString (InputIterator begin, InputIterator end, const Allocator &a=Allocator());`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.2.12 BasicString(BasicString&&, const Allocator&)

[SWS_CORE_03034] Definition of API function ara::core::BasicString::BasicString ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString (BasicString &&, const Allocator &);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.3 Capacity

8.8.3.3.3.1 capacity

[SWS_CORE_03055] Definition of API function ara::core::BasicString::capacity ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`size_type capacity () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.3.2 clear

[SWS_CORE_03057] Definition of API function ara::core::BasicString::clear ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`void clear () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.3.3 empty

[SWS_CORE_03051] Definition of API function ara::core::BasicString::empty ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`bool empty () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.3.4 length

[SWS_CORE_03060] Definition of API function ara::core::BasicString::length ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`size_type length () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.3.5 max_size

[SWS_CORE_03053] Definition of API function ara::core::BasicString::max_size ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`size_type max_size () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.3.6 reserve

[SWS_CORE_03054] Definition of API function ara::core::BasicString::reserve ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`void reserve (size_type res_arg=0);`	
Exception Safety:	not exception safe	
Violations:	`IsoStdLengthErrorViolation`	Description as per std::length_error
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. if (res_arg > `max_size()`) `IsoStdLengthErrorViolation` is raised instead of `std::length_error`	

⌋

8.8.3.3.3.7 resize(size_type, value_type)

[SWS_CORE_03059] Definition of API function ara::core::BasicString::resize ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`void resize (size_type sz, value_type c);`	
Exception Safety:	not exception safe	
Violations:	`IsoStdLengthErrorViolation`	Description as per std::length_error
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type` 2. if (sz > `max_size()`) `IsoStdLengthErrorViolation` is raised instead of `std::length_error`	

⌋

8.8.3.3.3.8 resize(size_type)

[SWS_CORE_03058] Definition of API function ara::core::BasicString::resize ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`void resize (size_type sz);`	
Exception Safety:	not exception safe	
Violations:	`IsoStdLengthErrorViolation`	Description as per std::length_error
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. if (sz > `max_size()`) `IsoStdLengthErrorViolation` is raised instead of `std::length_error`	

⌋

8.8.3.3.3.9 shrink_to_fit

[SWS_CORE_03056] Definition of API function ara::core::BasicString::shrink_to_fit ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`void shrink_to_fit ();`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.3.10 size

[SWS_CORE_03052] Definition of API function ara::core::BasicString::size ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`size_type size () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.4 Element access

8.8.3.3.4.1 at(size_type)

[SWS_CORE_03064] Definition of API function ara::core::BasicString::at ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`reference at (size_type pos);`	
Exception Safety:	not exception safe	
Violations:	`IsoStdOut-OfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. if (pos >= `size()`) `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range`	

⌋

8.8.3.3.4.2 at(size_type) const

[SWS_CORE_03063] Definition of API function ara::core::BasicString::at ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`const_reference at (size_type pos) const;`	
Exception Safety:	not exception safe	
Violations:	`IsoStdOut-OfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. if (pos >= `size()`) `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range`	

⌋

8.8.3.3.4.3 back() const

[SWS_CORE_03068] Definition of API function ara::core::BasicString::back ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`const_reference back () const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.4.4 back()

[SWS_CORE_03067] Definition of API function ara::core::BasicString::back ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`reference back ();`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.4.5 c_str

[SWS_CORE_03069] Definition of API function ara::core::BasicString::c_str ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`const value_type * c_str () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.4.6 data

[SWS_CORE_03070] Definition of API function ara::core::BasicString::data ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`const value_type * data () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.4.7 data

[SWS_CORE_03071] Definition of API function ara::core::BasicString::data ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`value_type * data () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.4.8 front() const

[SWS_CORE_03066] Definition of API function ara::core::BasicString::front ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`const_reference front () const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.4.9 front()

[SWS_CORE_03065] Definition of API function ara::core::BasicString::front ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`reference front ();`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.4.10 operator StringView

[SWS_CORE_03301] Definition of API function ara::core::BasicString::operator StringView

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`operator StringView () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.4.11 operator[](size_type) const

[SWS_CORE_03062] Definition of API function ara::core::BasicString::operator[]

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`const_reference operator[] (size_type n) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.4.12 operator[](size_type)

[SWS_CORE_03061] Definition of API function ara::core::BasicString::operator[]

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`reference operator[] (size_type n);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.5 Iterators

8.8.3.3.5.1 begin() const

[SWS_CORE_03040] Definition of API function ara::core::BasicString::begin ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`const_iterator begin () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.5.2 begin()

[SWS_CORE_03039] Definition of API function ara::core::BasicString::begin ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`iterator begin () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.5.3 cbegin

[SWS_CORE_03047] Definition of API function ara::core::BasicString::cbegin ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`const_iterator cbegin () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.5.4 cend

[SWS_CORE_03048] Definition of API function ara::core::BasicString::cend ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`const_iterator cend () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.5.5 crbegin

[SWS_CORE_03049] Definition of API function ara::core::BasicString::crbegin ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`const_reverse_iterator crbegin () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.5.6 crend

[SWS_CORE_03050] Definition of API function ara::core::BasicString::crend ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`const_reverse_iterator crend () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.5.7 end()

[SWS_CORE_03041] Definition of API function ara::core::BasicString::end ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`iterator end () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.5.8 end() const

[SWS_CORE_03042] Definition of API function ara::core::BasicString::end ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`const_iterator end () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.5.9 rbegin()

[SWS_CORE_03043] Definition of API function ara::core::BasicString::rbegin ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`reverse_iterator rbegin () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.5.10 rbegin() const

[SWS_CORE_03044] Definition of API function ara::core::BasicString::rbegin ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`const_reverse_iterator rbegin () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.5.11 rend() const

[SWS_CORE_03046] Definition of API function ara::core::BasicString::rend ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`const_reverse_iterator rend () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.5.12 rend()

[SWS_CORE_03045] Definition of API function ara::core::BasicString::rend ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`reverse_iterator rend () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.6 Modifiers

8.8.3.3.6.1 append(const BasicString&, size_type, size_type)

[SWS_CORE_03077] Definition of API function ara::core::BasicString::append ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`BasicString & append (const BasicString &str, size_type pos, size_type n=npos);`	
Exception Safety:	not exception safe	
Violations:	`IsoStdOut-OfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. if (pos > str.`size()`) `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range`	

⌋

8.8.3.3.6.2 append(StringView)

[SWS_CORE_03308] Definition of API function ara::core::BasicString::append

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & append (StringView sv);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.6.3 append(const BasicString&)

[SWS_CORE_03076] Definition of API function ara::core::BasicString::append ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & append (const BasicString &str);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.6.4 append(const T&, size_type, size_type)

[SWS_CORE_03309] Definition of API function ara::core::BasicString::append

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`template <class T>` `BasicString & append (const T &t, size_type pos, size_type n=npos);`	
Exception Safety:	not exception safe	
Violations:	`IsoStdOut-OfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. if (pos > sv.`size()`) `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range`	

⌋

8.8.3.3.6.5 append(const value_type *, size_type)

[SWS_CORE_03078] Definition of API function ara::core::BasicString::append ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`BasicString & append (const value_type *s, size_type n);`	
Exception Safety:	not exception safe	
Violations:	`IsoStdLengthEr-rorViolation`	Description as per std::length_error
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type` 2. if (`size()` > n + `max_size()`) `IsoStdLengthErrorViolation` is raised instead of `std::length_error`	

⌋

8.8.3.3.6.6 append(const value_type *)

[SWS_CORE_03079] Definition of API function ara::core::BasicString::append ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & append (const value_type *s);`
Exception Safety:	not exception safe

▽

△

Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.6.7 append(size_type, value_type)

[SWS_CORE_03080] Definition of API function ara::core::BasicString::append ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & append (size_type n, value_type c);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.6.8 append(InputIterator, InputIterator)

[SWS_CORE_03081] Definition of API function ara::core::BasicString::append ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`template <class InputIterator>` `BasicString & append (InputIterator first, InputIterator last);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.6.9 append(std::initializer_list<value_type>)

[SWS_CORE_03082] Definition of API function ara::core::BasicString::append ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & append (std::initializer_list< value_type >);`
Exception Safety:	not exception safe

▽

△

Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.6.10 copy

[SWS_CORE_03410] Definition of API function ara::core::BasicString::copy ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`size_type copy (value_type *s, size_type n, size_type pos=0) const;`	
Exception Safety:	not exception safe	
Violations:	`IsoStdOut-OfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type` 2. if (pos > `size()`) `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range`	

⌋

8.8.3.3.6.11 erase(const_iterator, const_iterator)

[SWS_CORE_03104] Definition of API function ara::core::BasicString::erase ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`iterator erase (const_iterator first, const_iterator last);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.6.12 erase(const_iterator)

[SWS_CORE_03103] Definition of API function ara::core::BasicString::erase ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`iterator erase (const_iterator p);`

▽

△

Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.6.13 erase(size_type, size_type)

[SWS_CORE_03102] Definition of API function ara::core::BasicString::erase ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`BasicString & erase (size_type pos=0, size_type n=npos);`	
Exception Safety:	not exception safe	
Violations:	`IsoStdOut-OfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type` 2. if (pos > `size()`) `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range`	

⌋

8.8.3.3.6.14 insert(const_iterator, std::initializer_list<value_type>)

[SWS_CORE_03101] Definition of API function ara::core::BasicString::insert ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`iterator insert (const_iterator p, std::initializer_list< value_type >);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.6.15 insert(size_type, const BasicString&, size_type, size_type)

[SWS_CORE_03094] Definition of API function ara::core::BasicString::insert ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`BasicString & insert (size_type pos1, const BasicString &str, size_type pos2, size_type n=npos);`	
Exception Safety:	not exception safe	
Violations:	`IsoStdOut-OfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. if (pos1 > `size()` || pos2 > str.`size()`) `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range`	

⌋

8.8.3.3.6.16 insert(size_type, const BasicString&)

[SWS_CORE_03093] Definition of API function ara::core::BasicString::insert ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & insert (size_type pos, const BasicString &str);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.6.17 insert(const_iterator, InputIterator, InputIterator)

[SWS_CORE_03100] Definition of API function ara::core::BasicString::insert ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`template <class InputIterator>` `iterator insert (const_iterator p, InputIterator first, InputIterator last);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.6.18 insert(const_iterator, size_type, value_type)

[SWS_CORE_03099] Definition of API function ara::core::BasicString::insert ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`iterator insert (const_iterator p, size_type n, value_type c);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.6.19 insert(const_iterator, value_type)

[SWS_CORE_03098] Definition of API function ara::core::BasicString::insert ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`iterator insert (const_iterator p, value_type c);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.6.20 insert(size_type, size_type, value_type)

[SWS_CORE_03097] Definition of API function ara::core::BasicString::insert ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & insert (size_type pos, size_type n, value_type c);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.6.21 insert(size_type, const T&, size_type, size_type)

[SWS_CORE_03311] Definition of API function ara::core::BasicString::insert

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`template <class T>` `BasicString & insert (size_type pos1, const T &t, size_type pos2, size_type n=npos);`	
Exception Safety:	not exception safe	
Violations:	`IsoStdOut-OfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. if (pos1 > `size()` || pos2 > sv.`size()`) `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range`	

⌋

8.8.3.3.6.22 insert(size_type, StringView)

[SWS_CORE_03310] Definition of API function ara::core::BasicString::insert

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & insert (size_type pos, StringView sv);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.6.23 insert(size_type, const value_type *, size_type)

[SWS_CORE_03095] Definition of API function ara::core::BasicString::insert ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & insert (size_type pos, const value_type *s, size_type n);`
Exception Safety:	not exception safe

▽

△

Violations:	`IsoStdLengthErrorViolation`	Description as per std::length_error
	`IsoStdOutOfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type` 2. if (pos1 > `size()`) `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range` 3. if (`size()` + n > `max_size()`) `IsoStdLengthErrorViolation` is raised instead of `std::length_error`	

⌋

8.8.3.3.6.24 insert(size_type, const value_type *)

[SWS_CORE_03096] Definition of API function ara::core::BasicString::insert ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & insert (size_type pos, const value_type *s);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.6.25 operator+=(const value_type *)

[SWS_CORE_03073] Definition of API function ara::core::BasicString::operator+= ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & operator+= (const value_type *s);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.6.26 operator+=(value_type)

[SWS_CORE_03074] Definition of API function ara::core::BasicString::operator+= ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & operator+= (value_type c);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.6.27 operator+=(const BasicString&)

[SWS_CORE_03072] Definition of API function ara::core::BasicString::operator+= ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & operator+= (const BasicString &str);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.6.28 operator+=(std::initializer_list<value_type>)

[SWS_CORE_03075] Definition of API function ara::core::BasicString::operator+= ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & operator+= (std::initializer_list< value_type >);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.6.29 operator+=(StringView)

[SWS_CORE_03307] Definition of API function ara::core::BasicString::operator+=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & operator+= (StringView sv);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.6.30 pop_back

[SWS_CORE_03105] Definition of API function ara::core::BasicString::pop_back

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`void pop_back ();`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.6.31 push_back(value_type)

[SWS_CORE_03083] Definition of API function ara::core::BasicString::push_back ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`void push_back (value_type c);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.6.32 replace(size_type, size_type, size_type, value_type)

[SWS_CORE_03110] Definition of API function ara::core::BasicString::replace ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & replace (size_type pos, size_type n1, size_type n2, value_type c);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.6.33 replace(size_type, size_type, const value_type *, size_type)

[SWS_CORE_03108] Definition of API function ara::core::BasicString::replace ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`BasicString & replace (size_type pos, size_type n1, const value_type *s, size_type n2);`	
Exception Safety:	not exception safe	
Violations:	`IsoStdLengthErrorViolation`	Description as per std::length_error
	`IsoStdOutOfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type` 2. if (pos1 > `size()`) `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range` 3. `IsoStdLengthErrorViolation` is raised instead of `std::length_error` (see condition in [11])	

⌋

8.8.3.3.6.34 replace(const_iterator, const_iterator, const BasicString&)

[SWS_CORE_03111] Definition of API function ara::core::BasicString::replace ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & replace (const_iterator i1, const_iterator i2, const BasicString &str);`
Exception Safety:	not exception safe

▽

△

Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.6.35 replace(size_type, size_type, StringView)

[SWS_CORE_03312] Definition of API function ara::core::BasicString::replace

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & replace (size_type pos1, size_type n1, StringView sv);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.6.36 replace(size_type, size_type, const BasicString&, size_type, size_type)

[SWS_CORE_03107] Definition of API function ara::core::BasicString::replace ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`BasicString & replace (size_type pos1, size_type n1, const BasicString &str, size_type pos2, size_type n2=npos);`	
Exception Safety:	not exception safe	
Violations:	`IsoStdOut-OfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type` 2. if (pos > `max_size()` || pos2 > str.`size()`) `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range`	

⌋

8.8.3.3.6.37 replace(size_type, size_type, const BasicString&)

[SWS_CORE_03106] Definition of API function ara::core::BasicString::replace ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & replace (size_type pos1, size_type n1, const BasicString &str);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.6.38 replace(const_iterator, const_iterator, const value_type *)

[SWS_CORE_03406] Definition of API function ara::core::BasicString::replace ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & replace (const_iterator i1, const_iterator i2, const value_type *s);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.6.39 replace(const_iterator, const_iterator, StringView)

[SWS_CORE_03314] Definition of API function ara::core::BasicString::replace

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & replace (const_iterator i1, const_iterator i2, StringView sv);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.6.40 replace(size_type, size_type, const T&, size_type, size_type)

[SWS_CORE_03313] Definition of API function ara::core::BasicString::replace

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`template <class T>` `BasicString & replace (size_type pos1, size_type n1, const T &t, size_type pos2, size_type n2=npos);`	
Exception Safety:	not exception safe	
Violations:	`IsoStdOut-OfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type` 2. if (pos1 > `size()` || pos2 > sv.`size()`) `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range`	

⌋

8.8.3.3.6.41 replace(const_iterator, const_iterator, const value_type *, size_type)

[SWS_CORE_03112] Definition of API function ara::core::BasicString::replace ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & replace (const_iterator i1, const_iterator i2, const value_type *s, size_type n);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.6.42 replace(const_iterator, const_iterator, size_type, value_type)

[SWS_CORE_03414] Definition of API function ara::core::BasicString::replace ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & replace (const_iterator i1, const_iterator i2, size_type n, value_type c);`

▽

△

Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.6.43 replace(const_iterator, const_iterator, InputIterator, InputIterator)

[SWS_CORE_03411] Definition of API function ara::core::BasicString::replace ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`template <class InputIterator>` `BasicString & replace (const_iterator i1, const_iterator i2, Input Iterator j1, InputIterator j2);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.6.44 replace(const_iterator, const_iterator, std::initializer_list<value_type>)

[SWS_CORE_03413] Definition of API function ara::core::BasicString::replace ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & replace (const_iterator, const_iterator, std::initializer_list< value_type >);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.6.45 replace(size_type, size_type, const value_type *)

[SWS_CORE_03109] Definition of API function ara::core::BasicString::replace ⌈

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

▽

△

Syntax:	`BasicString & replace (size_type pos, size_type n1, const value_type *s);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.6.46 swap

[SWS_CORE_03412] Definition of API function ara::core::BasicString::swap ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`void swap (BasicString &str) noexcept(std::allocator_traits< Allocator >::propagate_on_container_swap::value||std::allocator_traits< Allocator >::is_always_equal::value);`
Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.7 Operations

8.8.3.3.7.1 compare(const BasicString&) const

[SWS_CORE_03346] Definition of API function ara::core::BasicString::compare

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`int compare (const BasicString &str) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.7.2 compare(size_type, size_type, const value_type *, size_type) const

[SWS_CORE_03351] Definition of API function ara::core::BasicString::compare

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`int compare (size_type pos1, size_type n1, const value_type *s, size_type n2) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.7.3 compare(size_type, size_type, const BasicString&) const

[SWS_CORE_03347] Definition of API function ara::core::BasicString::compare

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`int compare (size_type pos1, size_type n1, const BasicString &str) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.7.4 compare(size_type, size_type, StringView) const

[SWS_CORE_03322] Definition of API function ara::core::BasicString::compare

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`int compare (size_type pos1, size_type n1, StringView sv) const;`
Exception Safety:	not exception safe

▽

△

Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.7.5 compare(size_type, size_type, const BasicString&, size_type, size_type) const

[SWS_CORE_03348] Definition of API function ara::core::BasicString::compare

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`int compare (size_type pos1, size_type n1, const BasicString &str, size_type pos2, size_type n2=npos) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.7.6 compare(const value_type *) const

[SWS_CORE_03349] Definition of API function ara::core::BasicString::compare

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`int compare (const value_type *s) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.7.7 compare(size_type, size_type, const value_type *) const

[SWS_CORE_03350] Definition of API function ara::core::BasicString::compare

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`int compare (size_type pos1, size_type n1, const value_type *s) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.7.8 compare(size_type, size_type, const T&, size_type, size_type) const

[SWS_CORE_03323] Definition of API function ara::core::BasicString::compare

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`template <typename T>` `int compare (size_type pos1, size_type n1, const T &t, size_type pos2, size_type n2=npos) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.7.9 compare(StringView) const

[SWS_CORE_03321] Definition of API function ara::core::BasicString::compare

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`int compare (StringView sv) const noexcept;`
Exception Safety:	exception safe

▽

△

Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.7.10 substr(size_type, size_type) const

[SWS_CORE_03345] Definition of API function ara::core::BasicString::substr ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`BasicString substr (size_type pos=0, size_type n=npos) const;`	
Exception Safety:	not exception safe	
Violations:	`IsoStdOut-OfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. if (pos > `size()`) `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range`	

⌋

8.8.3.3.8 Operator=

8.8.3.3.8.1 operator=(std::initializer_list<value_type>)

[SWS_CORE_03037] Definition of API function ara::core::BasicString::operator= ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & operator= (std::initializer_list< value_type >);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.8.2 operator=(value_type)

[SWS_CORE_03036] Definition of API function ara::core::BasicString::operator=

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & operator= (value_type c);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.8.3 operator=(StringView)

[SWS_CORE_03304] Definition of API function ara::core::BasicString::operator=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & operator= (StringView sv);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.8.4 operator=(const value_type *)

[SWS_CORE_03035] Definition of API function ara::core::BasicString::operator=

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & operator= (const value_type *s);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.9 Other

8.8.3.3.9.1 assign(const BasicString&)

[SWS_CORE_03084] Definition of API function ara::core::BasicString::assign ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & assign (const BasicString &str);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.9.2 assign(BasicString&&)

[SWS_CORE_03085] Definition of API function ara::core::BasicString::assign ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & assign (BasicString &&str) noexcept(std::allocator_traits< Allocator >::propagate_on_container_move_assignment::value||std::allocator_traits< Allocator >::is_always_equal::value);`
Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.9.3 assign(const BasicString&, size_type, size_type)

[SWS_CORE_03086] Definition of API function ara::core::BasicString::assign ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`BasicString & assign (const BasicString &str, size_type pos, size_type n=npos);`	
Exception Safety:	not exception safe	
Violations:	`IsoStdOut-OfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. if (pos > str.`size()`) `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range`	

⌋

8.8.3.3.9.4 assign(StringView)

[SWS_CORE_03087] Definition of API function ara::core::BasicString::assign

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & assign (StringView sv);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.9.5 assign(const T&, size_type, size_type)

[SWS_CORE_03306] Definition of API function ara::core::BasicString::assign

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`template <class T>` `BasicString & assign (const T &t, size_type pos, size_type n=npos);`	
Exception Safety:	not exception safe	
Violations:	`IsoStdOut-OfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. if (pos > sv.`size()`) `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range`	

⌋

8.8.3.3.9.6 assign(const value_type *, size_type)

[SWS_CORE_03088] Definition of API function ara::core::BasicString::assign ⌈

Kind:	function	
Header file:	#include "ara/core/string.h"	
Scope:	`class ara::core::BasicString`	
Syntax:	`BasicString & assign (const value_type *s, size_type n);`	
Exception Safety:	not exception safe	
Violations:	`IsoStdLengthEr-rorViolation`	Description as per std::length_error

▽

△

Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type` 2. if (n > `max_size()`) `IsoStdLengthErrorViolation` is raised instead of `std::length_error`

⌋

8.8.3.3.9.7 assign(const value_type *)

[SWS_CORE_03089] Definition of API function ara::core::BasicString::assign ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & assign (const value_type *s);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.9.8 assign(InputIterator, InputIterator)

[SWS_CORE_03091] Definition of API function ara::core::BasicString::assign ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`template <class InputIterator>` `BasicString & assign (InputIterator first, InputIterator last);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.9.9 assign(std::initializer_list<value_type>)

[SWS_CORE_03092] Definition of API function ara::core::BasicString::assign ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & assign (std::initializer_list< value_type >);`
Exception Safety:	not exception safe

▽

△

Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.9.10 assign(size_type, value_type)

[SWS_CORE_03090] Definition of API function ara::core::BasicString::assign ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`BasicString & assign (size_type n, value_type c);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.9.11 get_allocator() const

[SWS_CORE_03415] Definition of API function ara::core::BasicString::get_allocator ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`allocator_type get_allocator () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.10 Search

8.8.3.3.10.1 find(StringView, size_type) const

[SWS_CORE_03315] Definition of API function ara::core::BasicString::find

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`size_type find (StringView sv, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.10.2 find(const BasicString&, size_type) const

[SWS_CORE_03407] Definition of API function ara::core::BasicString::find ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`size_type find (const BasicString &str, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.10.3 find(const value_type *, size_type) const

[SWS_CORE_03408] Definition of API function ara::core::BasicString::find ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`size_type find (const value_type *s, size_type pos=0) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.10.4 find(value_type, size_type) const

[SWS_CORE_03324] Definition of API function ara::core::BasicString::find ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`size_type find (value_type c, size_type pos=0) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.10.5 find(const value_type *, size_type, size_type) const

[SWS_CORE_03409] Definition of API function ara::core::BasicString::find ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`size_type find (const value_type *s, size_type pos, size_type n) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.10.6 find_first_not_of(value_type, size_type) const

[SWS_CORE_03340] 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`
Syntax:	`size_type find_first_not_of (value_type c, size_type pos=0) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.10.7 find_first_not_of(const value_type *, size_type, size_type) const

[SWS_CORE_03338] 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`
Syntax:	`size_type find_first_not_of (const value_type *s, size_type pos, size_type n) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.10.8 find_first_not_of(const BasicString&, size_type) const

[SWS_CORE_03337] 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`
Syntax:	`size_type find_first_not_of (const BasicString &str, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.10.9 find_first_not_of(StringView, size_type) const

[SWS_CORE_03319] Definition of API function ara::core::BasicString::find_first_not_of

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`size_type find_first_not_of (StringView sv, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.10.10 find_first_not_of(const value_type *, size_type) const

[SWS_CORE_03339] 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`
Syntax:	`size_type find_first_not_of (const value_type *s, size_type pos=0) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.10.11 find_first_of(const value_type *, size_type) const

[SWS_CORE_03331] Definition of API function ara::core::BasicString::find_first_of ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`size_type find_first_of (const value_type *s, size_type pos=0) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.10.12 find_first_of(StringView, size_type) const

[SWS_CORE_03317] Definition of API function ara::core::BasicString::find_first_of

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`size_type find_first_of (StringView sv, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.10.13 find_first_of(value_type, size_type) const

[SWS_CORE_03332] Definition of API function ara::core::BasicString::find_first_of ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`size_type find_first_of (value_type c, size_type pos=0) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.10.14 find_first_of(const BasicString&, size_type) const

[SWS_CORE_03329] Definition of API function ara::core::BasicString::find_first_of ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`size_type find_first_of (const BasicString &str, size_type pos=0) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.10.15 find_first_of(const value_type *, size_type, size_type) const

[SWS_CORE_03330] Definition of API function ara::core::BasicString::find_first_of ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`size_type find_first_of (const value_type *s, size_type pos, size_type n) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.10.16 find_last_not_of(const value_type *, size_type, size_type) const

[SWS_CORE_03342] 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`
Syntax:	`size_type find_last_not_of (const value_type *s, size_type pos, size_type n) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.10.17 find_last_not_of(value_type, size_type) const

[SWS_CORE_03344] 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`
Syntax:	`size_type find_last_not_of (value_type c, size_type pos=npos) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.10.18 find_last_not_of(const BasicString&, size_type) const

[SWS_CORE_03341] 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`
Syntax:	`size_type find_last_not_of (const BasicString &str, size_type pos=npos) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.10.19 find_last_not_of(StringView, size_type) const

[SWS_CORE_03320] Definition of API function ara::core::BasicString::find_last_not_of

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`size_type find_last_not_of (StringView sv, size_type pos=npos) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.10.20 find_last_not_of(const value_type *, size_type) const

[SWS_CORE_03343] 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`
Syntax:	`size_type find_last_not_of (const value_type *s, size_type pos=npos) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.10.21 find_last_of(const BasicString&, size_type) const

[SWS_CORE_03333] Definition of API function ara::core::BasicString::find_last_of ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`size_type find_last_of (const BasicString &str, size_type pos=npos) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.10.22 find_last_of(const value_type *, size_type, size_type) const

[SWS_CORE_03334] Definition of API function ara::core::BasicString::find_last_of ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`size_type find_last_of (const value_type *s, size_type pos, size_type n) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.10.23 find_last_of(const value_type *, size_type) const

[SWS_CORE_03335] Definition of API function ara::core::BasicString::find_last_of ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`size_type find_last_of (const value_type *s, size_type pos=npos) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.10.24 find_last_of(value_type, size_type) const

[SWS_CORE_03336] Definition of API function ara::core::BasicString::find_last_of ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`size_type find_last_of (value_type c, size_type pos=npos) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.10.25 find_last_of(StringView, size_type) const

[SWS_CORE_03318] Definition of API function ara::core::BasicString::find_last_of

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`size_type find_last_of (StringView sv, size_type pos=npos) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.10.26 rfind(const value_type *, size_type, size_type) const

[SWS_CORE_03326] Definition of API function ara::core::BasicString::rfind ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`size_type rfind (const value_type *s, size_type pos, size_type n) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.10.27 rfind(const value_type *, size_type) const

[SWS_CORE_03327] Definition of API function ara::core::BasicString::rfind ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`size_type rfind (const value_type *s, size_type pos=npos) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.10.28 rfind(const BasicString&, size_type) const

[SWS_CORE_03325] Definition of API function ara::core::BasicString::rfind ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`size_type rfind (const BasicString &str, size_type pos=npos) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]`

⌋

8.8.3.3.10.29 rfind(value_type, size_type) const

[SWS_CORE_03328] Definition of API function ara::core::BasicString::rfind ⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`size_type rfind (value_type c, size_type pos=npos) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.3.3.10.30 rfind(StringView, size_type) const

[SWS_CORE_03316] Definition of API function ara::core::BasicString::rfind

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/string.h"
Scope:	`class ara::core::BasicString`
Syntax:	`size_type rfind (StringView sv, size_type pos=npos) const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[basic.string]` except for the following deviations: 1. `charT` replaced with `value_type`

⌋

8.8.4 Struct: hash

[SWS_CORE_03405] Definition of API class std::hash< ara::core::String >

Status: DRAFT

⌈

Kind:	struct
Header file:	#include "ara/core/string.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	namespace std
Symbol:	hash< ara::core::String >
Syntax:	`template <>` `struct hash< ara::core::String > final {...};`
Description:	Specialization of `std::hash` (hashing function for `ara::core::String`)

⌋

8.9 Header: ara/core/variant.h

[SWS_CORE_01672] Definition of Header File ara/core/variant.h

Upstream requirements: RS_AP_00130

⌈

Kind:	Header File
Syntax:	`ara/core/variant.h`
Description:	Implements `[variant.syn]` in [11]. Unless explicitly overridden in the API documentation, APIs always adhere in behavior to the ISO specification. To reduce duplication, the APIs here specify only any deviations from [11] or any AUTOSAR specific overrides. For APIs which would ordinarily throw `Exception`s, and those `Exception`s have been suppressed in favor of AUTOSAR violations, generally the semantics of the `Exception` from [11] still apply (unless locally overridden here).

⌋

8.9.1 Non-Member Types

8.9.1.1 Type Alias: variant_alternative_t

[SWS_CORE_01612] Definition of API type ara::core::variant_alternative_t

Status: DRAFT

⌈

Kind:	type alias
Header file:	#include "ara/core/variant.h"
Scope:	namespace ara::core
Symbol:	variant_alternative_t
Syntax:	`using variant_alternative_t = typename variant_alternative<I, T>::type;`
Description:	As per corresponding usage in `[variant]`

⌋

8.9.2 Global Variables

8.9.2.1 variant_size_v

[SWS_CORE_01606] Definition of API variable ara::core::variant_size_v

Status: DRAFT

⌈

Kind:	variable
Header file:	#include "ara/core/variant.h"
Scope:	namespace ara::core
Symbol:	variant_size_v
Type:	std::size_t
Syntax:	`template <typename T>` `constexpr std::size_t variant_size_v = variant_size<T>::value;`
Description:	As per corresponding usage in `[variant]`

⌋

8.9.3 Non-Member Functions

8.9.3.1 Other

8.9.3.1.1 get(Variant<Types...>&)

[SWS_CORE_01613] Definition of API function ara::core::get

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	namespace ara::core	
Syntax:	`template <std::size_t I, typename... Types> constexpr variant_alternative_t< I, Variant< Types... > > & get ( Variant< Types... > &variant) noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdBadVariantAccessViolation`	Description as per std::bad_variant_access.
Description:	As per corresponding usage in `[variant]` except for the following deviations: 1. The function is `noexcept` 2. if (variant.`index()` != I) `IsoStdBadVariantAccessViolation` is raised instead of `std::bad_variant_access`	

⌋

8.9.3.1.2 get(Variant<Types...>&&)

[SWS_CORE_01614] Definition of API function ara::core::get

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	namespace ara::core	
Syntax:	`template <std::size_t I, typename... Types> constexpr variant_alternative_t< I, Variant< Types... > > && get ( Variant< Types... > &&variant) noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdBadVariantAccessViolation`	Description as per std::bad_variant_access.
Description:	As per corresponding usage in `[variant]` except for the following deviations: 1. The function is `noexcept` 2. if (variant.`index()` != I) `IsoStdBadVariantAccessViolation` is raised instead of `std::bad_variant_access`	

⌋

8.9.3.1.3 get(const Variant<Types...>&)

[SWS_CORE_01615] Definition of API function ara::core::get

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	namespace ara::core	
Syntax:	`template <std::size_t I, typename... Types> constexpr const variant_alternative_t< I, Variant< Types... > > & get (const Variant< Types... > &variant) noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdBadVariantAccessViolation`	Description as per std::bad_variant_access.
Description:	As per corresponding usage in `[variant]` except for the following deviations: 1. The function is `noexcept` 2. if (variant.`index()` != I) `IsoStdBadVariantAccessViolation` is raised instead of `std::bad_variant_access`	

⌋

8.9.3.1.4 get(const Variant<Types...>&&)

[SWS_CORE_01616] Definition of API function ara::core::get

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	namespace ara::core	
Syntax:	`template <std::size_t I, typename... Types> constexpr const variant_alternative_t< I, Variant< Types... > > && get (const Variant< Types... > &&variant) noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdBadVariantAccessViolation`	Description as per std::bad_variant_access.
Description:	As per corresponding usage in `[variant]` except for the following deviations: 1. The function is `noexcept` 2. if (variant.`index()` != I) `IsoStdBadVariantAccessViolation` is raised instead of `std::bad_variant_access`	

⌋

8.9.3.1.5 get(Variant<Types...>&)

[SWS_CORE_01617] Definition of API function ara::core::get

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	namespace ara::core	
Syntax:	`template <typename T, typename... Types>` `constexpr T & get (Variant< Types... > &variant) noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdBadVariantAccessViolation`	Description as per std::bad_variant_access.
Description:	As per corresponding usage in `[variant]` except for the following deviations: 1. The function is `noexcept` 2. if (variant does not hold alternative T) `IsoStdBadVariantAccessViolation` is raised instead of `std::bad_variant_access`	

⌋

8.9.3.1.6 get(Variant<Types...>&&)

[SWS_CORE_01618] Definition of API function ara::core::get

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	namespace ara::core	
Syntax:	`template <typename T, typename... Types>` `constexpr T && get (Variant< Types... > &&variant) noexcept;`	
Exception Safety:	exception safe	
Violations:	`IsoStdBadVariantAccessViolation`	Description as per std::bad_variant_access.
Description:	As per corresponding usage in `[variant]` except for the following deviations: 1. The function is `noexcept` 2. if (variant does not hold alternative T) `IsoStdBadVariantAccessViolation` is raised instead of `std::bad_variant_access`	

⌋

8.9.3.1.7 get(const Variant<Types...>&)

[SWS_CORE_01619] Definition of API function ara::core::get

Status: DRAFT

⌈

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

Syntax:	`template <typename T, typename... Types>`
	`constexpr const T & get (const Variant< Types... > &variant)`
	`noexcept;`	

Exception Safety:	exception safe	
Violations:	`IsoStdBadVariantAccessViolation`	Description as per std::bad_variant_access.
Description:	As per corresponding usage in `[variant]` except for the following deviations: 1. The function is `noexcept` 2. if (variant does not hold alternative T) `IsoStdBadVariantAccessViolation` is raised instead of `std::bad_variant_access`	

⌋

8.9.3.1.8 get(const Variant<Types...>&&)

[SWS_CORE_01620] Definition of API function ara::core::get

Status: DRAFT

⌈

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

Syntax:	`template <typename T, typename... Types>`
	`constexpr const T && get (const Variant< Types... > &&variant)`
	`noexcept;`	

Exception Safety:	exception safe	
Violations:	`IsoStdBadVariantAccessViolation`	Description as per std::bad_variant_access.
Description:	As per corresponding usage in `[variant]` except for the following deviations: 1. The function is `noexcept` 2. if (variant does not hold alternative T) `IsoStdBadVariantAccessViolation` is raised instead of `std::bad_variant_access`	

⌋

8.9.3.1.9 get_if(Variant<Types...> *)

[SWS_CORE_01623] Definition of API function ara::core::get_if

Status: DRAFT

⌈

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

Syntax:	`template <typename T, typename... Types>`
	`std::add_pointer_t< T > get_if (Variant< Types... > *variant)`
	`noexcept;`

Exception Safety:	exception safe
Description:	As per corresponding usage in `[variant]`

⌋

8.9.3.1.10 get_if(const Variant<Types...> *)

[SWS_CORE_01624] Definition of API function ara::core::get_if

Status: DRAFT

⌈

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

Syntax:	`template <typename T, typename... Types>`
	`std::add_pointer_t< const T > get_if (const Variant< Types... >`
	`*variant) noexcept;`

Exception Safety:	exception safe
Description:	As per corresponding usage in `[variant]`

⌋

8.9.3.1.11 get_if(Variant<Types...> *)

[SWS_CORE_01621] Definition of API function ara::core::get_if

Status: DRAFT

⌈

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

▽

△

Syntax:	`template <std::size_t I, typename... Types>`
	`std::add_pointer_t< variant_alternative_t< I, Variant< Types... > > >`
	`get_if (Variant< Types... > *variant) noexcept;`

Exception Safety:	exception safe
Description:	As per corresponding usage in `[variant]`

⌋

8.9.3.1.12 get_if(const Variant<Types...> *)

[SWS_CORE_01622] Definition of API function ara::core::get_if

Status: DRAFT

⌈

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

Syntax:	`template <std::size_t I, typename... Types>`
	`std::add_pointer_t< const variant_alternative_t< I, Variant< Types...`
	`> > > get_if (const Variant< Types... > *variant) noexcept;`

Exception Safety:	exception safe
Description:	As per corresponding usage in `[variant]`

⌋

8.9.3.1.13 holds_alternative

[SWS_CORE_01626] Definition of API function ara::core::holds_alternative

Status: DRAFT

⌈

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

Syntax:	`template <typename T, typename... Types>`
	`constexpr bool holds_alternative (const Variant< Types... > &variant)`
	`noexcept;`

Exception Safety:	exception safe
Description:	As per corresponding usage in `[variant]`

⌋

8.9.3.1.14 operator!=(Monostate, Monostate)

[SWS_CORE_01642] Definition of API function ara::core::operator!=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/variant.h"
Scope:	namespace ara::core
Syntax:	`constexpr bool operator!= (Monostate lhs, Monostate rhs) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[variant]`

⌋

8.9.3.1.15 operator!=(const Variant<Types...>&, const Variant<Types...>&)

[SWS_CORE_01628] Definition of API function ara::core::operator!=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/variant.h"
Scope:	namespace ara::core
Syntax:	`template <typename... Types>` `constexpr bool operator!= (const Variant< Types... > &lhs, const Variant< Types... > &rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[variant]`

⌋

8.9.3.1.16 operator<(const Variant<Types...>&, const Variant<Types...>&)

[SWS_CORE_01629] Definition of API function ara::core::operator<

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/variant.h"
Scope:	namespace ara::core
Syntax:	`template <typename... Types>` `constexpr bool operator< (const Variant< Types... > &lhs, const Variant< Types... > &rhs);`
Exception Safety:	not exception safe

▽

△

Description:	As per corresponding usage in `[variant]`

⌋

8.9.3.1.17 operator<(Monostate, Monostate)

[SWS_CORE_01643] Definition of API function ara::core::operator<

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/variant.h"
Scope:	namespace ara::core
Syntax:	`constexpr bool operator< (Monostate lhs, Monostate rhs) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[variant]`

⌋

8.9.3.1.18 operator<=(Monostate, Monostate)

[SWS_CORE_01645] Definition of API function ara::core::operator<=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/variant.h"
Scope:	namespace ara::core
Syntax:	`constexpr bool operator<= (Monostate lhs, Monostate rhs) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[variant]`

⌋

8.9.3.1.19 operator<=(const Variant<Types...>&, const Variant<Types...>&)

[SWS_CORE_01631] Definition of API function ara::core::operator<=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/variant.h"
Scope:	namespace ara::core
Syntax:	`template <typename... Types>` `constexpr bool operator<= (const Variant< Types... > &lhs, const Variant< Types... > &rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[variant]`

⌋

8.9.3.1.20 operator==(Monostate, Monostate)

[SWS_CORE_01641] Definition of API function ara::core::operator==

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/variant.h"
Scope:	namespace ara::core
Syntax:	`constexpr bool operator== (Monostate lhs, Monostate rhs) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[variant]`

⌋

8.9.3.1.21 operator==(const Variant<Types...>&, const Variant<Types...>&)

[SWS_CORE_01627] Definition of API function ara::core::operator==

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/variant.h"
Scope:	namespace ara::core
Syntax:	`template <typename... Types>` `constexpr bool operator== (const Variant< Types... > &lhs, const Variant< Types... > &rhs);`
Exception Safety:	not exception safe

▽

△

Description:	As per corresponding usage in `[variant]`

⌋

8.9.3.1.22 operator>(Monostate, Monostate)

[SWS_CORE_01644] Definition of API function ara::core::operator>

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/variant.h"
Scope:	namespace ara::core
Syntax:	`constexpr bool operator> (Monostate lhs, Monostate rhs) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[variant]`

⌋

8.9.3.1.23 operator>(const Variant<Types...>&, const Variant<Types...>&)

[SWS_CORE_01630] Definition of API function ara::core::operator>

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/variant.h"
Scope:	namespace ara::core
Syntax:	`template <typename... Types>` `constexpr bool operator> (const Variant< Types... > &lhs, const Variant< Types... > &rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[variant]`

⌋

8.9.3.1.24 operator>=(Monostate, Monostate)

[SWS_CORE_01646] Definition of API function ara::core::operator>=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/variant.h"
Scope:	namespace ara::core
Syntax:	`constexpr bool operator>= (Monostate lhs, Monostate rhs) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[variant]`

⌋

8.9.3.1.25 operator>=(const Variant<Types...>&, const Variant<Types...>&)

[SWS_CORE_01632] Definition of API function ara::core::operator>=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/variant.h"
Scope:	namespace ara::core
Syntax:	`template <typename... Types>` `constexpr bool operator>= (const Variant< Types... > &lhs, const Variant< Types... > &rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[variant]`

⌋

8.9.3.1.26 swap

[SWS_CORE_01696] Definition of API function ara::core::swap

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/variant.h"
Scope:	namespace ara::core
Syntax:	`template <typename... Types>` `void swap (Variant< Types... > &lhs, Variant< Types... > &rhs) noexcept(noexcept(lhs.swap(rhs)));`
Exception Safety:	conditionally exception safe

▽

△

Description:	As per corresponding usage in `[variant]`

⌋

8.9.3.1.27 visit(Visitor&&, Variants ...&&)

[SWS_CORE_01634] Definition of API function ara::core::visit

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	namespace ara::core	
Syntax:	`template <typename Visitor, typename... Variants>` `constexpr {<invoke-result-decltype-2>} visit (Visitor &&visitor, Variants &&... variants);`	
Exception Safety:	not exception safe	
Description:	As per corresponding usage in `[variant]`	
Descriptors:	`{<invoke-result-decltype-2>}`	As per corresponding return type condition in `[variant.visit]`

⌋

8.9.3.1.28 visit(Visitor&&, Variants ...&&)

[SWS_CORE_01633] Definition of API function ara::core::visit

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	namespace ara::core	
Syntax:	`template <typename ReturnType, typename Visitor, typename... Variants>` `constexpr {<invoke-result-decltype-1>} visit (Visitor &&visitor, Variants &&... variants);`	
Exception Safety:	not exception safe	
Description:	As per corresponding usage in `[variant]`	
Descriptors:	`{<invoke-result-decltype-1>}`	As per corresponding return type condition in `[variant.visit]`

⌋

8.9.4 Struct: Monostate

[SWS_CORE_01640] Definition of API class ara::core::Monostate

Status: DRAFT

⌈

Kind:	struct
Header file:	#include "ara/core/variant.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	namespace ara::core
Symbol:	Monostate
Syntax:	`struct Monostate final {...};`
Description:	As per corresponding usage in `[variant]`

⌋

8.9.5 Class: Variant

[SWS_CORE_01601] Definition of API class ara::core::Variant

Status: DRAFT
Upstream requirements: RS_AP_00130

⌈

Kind:	class	
Header file:	#include "ara/core/variant.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	namespace ara::core	
Symbol:	Variant	
Syntax:	`template <typename... Types>` `class Variant {...};`	
Template param:	typename... Types	--
Description:	Implementation of `std::variant` type from `[basic.string]`	

⌋

8.9.5.1 Public Member Functions

8.9.5.1.1 Special Member Functions

8.9.5.1.1.1 Move Constructor

[SWS_CORE_01651] Definition of API function ara::core::Variant::Variant

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`class ara::core::Variant`	
Syntax:	`Variant (Variant &&other) noexcept({<cond-noexcept-3>});`	
Exception Safety:	conditionally exception safe	
Description:	As per corresponding usage in `[variant]` Move constructor	
Descriptors:	`{<cond-noexcept-3>}`	As per corresponding exception condition in `[variant]`

⌋

8.9.5.1.1.2 Default Constructor

[SWS_CORE_01649] Definition of API function ara::core::Variant::Variant

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`class ara::core::Variant`	
Syntax:	`constexpr Variant () noexcept({<cond-noexcept-1>});`	
Exception Safety:	conditionally exception safe	
Description:	As per corresponding usage in `[variant]`	
Descriptors:	`{<cond-noexcept-1>}`	As per corresponding exception condition in `[variant]`

⌋

8.9.5.1.1.3 Copy Constructor

[SWS_CORE_01650] Definition of API function ara::core::Variant::Variant

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`class ara::core::Variant`	
Syntax:	`Variant (const Variant &other) noexcept({<cond-noexcept-2>});`	
Exception Safety:	conditionally exception safe	
Description:	As per corresponding usage in `[variant]` except for the following deviations: 1. Function is conditionally `noexcept`	
Descriptors:	`{<cond-noexcept-2>}`	As per corresponding exception condition in `[variant]`. Condition is the logical AND of `std::is_nothrow_copy_constructible<Ti>::value` for all i.

⌋

8.9.5.1.1.4 Move Assignment Operator

[SWS_CORE_01659] Definition of API function ara::core::Variant::operator=

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`class ara::core::Variant`	
Syntax:	`constexpr Variant & operator= (Variant &&other) noexcept({<cond-noexcept-9>});`	
Exception Safety:	conditionally exception safe	
Description:	As per corresponding usage in `[variant]`	
Descriptors:	`{<cond-noexcept-9>}`	As per corresponding exception condition in `[variant]`

⌋

8.9.5.1.1.5 Copy Assignment Operator

[SWS_CORE_01658] Definition of API function ara::core::Variant::operator=

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`class ara::core::Variant`	
Syntax:	`constexpr Variant & operator= (const Variant &other) noexcept( {<cond-noexcept-8>});`	
Exception Safety:	conditionally exception safe	
Description:	As per corresponding usage in `[variant]` except for the following deviations: 1. Function is conditionally `noexcept`	
Descriptors:	`{<cond-noexcept-8>}`	As per corresponding exception condition in `[variant]`. Condition is the logical AND of `is_nothrow_copy_constructible<Ti>::value && is_nothrow_copy_assignable<Ti>::value` for all i

⌋

8.9.5.1.1.6 Destructor

[SWS_CORE_01657] Definition of API function ara::core::Variant::~Variant

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`class ara::core::Variant`	
Syntax:	`~Variant () noexcept({<cond-noexcept-7>});`	
Exception Safety:	conditionally exception safe	
Description:	As per corresponding usage in `[variant]` except for the following deviations: 1. Function is conditionally `noexcept`	
Descriptors:	`{<cond-noexcept-7>}`	As per corresponding exception condition in `[variant]`. Condition is the logical AND of `std::is_nothrow_destructible<Ti>::value` for all i

⌋

8.9.5.1.2 Constructors

8.9.5.1.2.1 Variant(in_place_type_t<T>, Args ...&&)

[SWS_CORE_01653] Definition of API function ara::core::Variant::Variant

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/variant.h"
Scope:	`class ara::core::Variant`
Syntax:	`template <typename T, typename... Args>` `explicit constexpr Variant (in_place_type_t< T > tag, Args &&... args) noexcept(std::is_nothrow_constructible< T, Args... >::value);`
Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[variant]` except for the following deviations: 1. Function is conditionally `noexcept`

⌋

8.9.5.1.2.2 Variant(in_place_type_t<T>, std::initializer_list<U>, Args ...&&)

[SWS_CORE_01654] Definition of API function ara::core::Variant::Variant

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/variant.h"
Scope:	`class ara::core::Variant`
Syntax:	`template <typename T, typename U, typename... Args>` `explicit constexpr Variant (in_place_type_t< T > tag, std::initializer_list< U > il, Args &&... args) noexcept(std::is_nothrow_constructible< T, std::initializer_list< U > &, Args &&... >::value);`
Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[variant]` except for the following deviations: 1. Function is conditionally `noexcept`

⌋

8.9.5.1.2.3 Variant(in_place_index_t<I>, Args ...&&)

[SWS_CORE_01655] Definition of API function ara::core::Variant::Variant

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`class ara::core::Variant`	
Syntax:	`template <std::size_t I, typename... Args>` `explicit constexpr Variant (in_place_index_t< I > tag, Args &&... args) noexcept({<cond-noexcept-5>});`	
Exception Safety:	conditionally exception safe	
Description:	As per corresponding usage in `[variant]` except for the following deviations: 1. Function is conditionally `noexcept`	
Descriptors:	`{<cond-noexcept-5>}`	As per corresponding exception condition in `[variant]`. Condition is equivalent to `std::is_nothrow_constructible <T_I, Args...>::value`.

⌋

8.9.5.1.2.4 Variant(in_place_index_t<I>, std::initializer_list<U>, Args ...&&)

[SWS_CORE_01656] Definition of API function ara::core::Variant::Variant

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`class ara::core::Variant`	
Syntax:	`template <std::size_t I, typename U, typename... Args>` `explicit constexpr Variant (in_place_index_t< I > tag, std::initializer_list< U > il, Args &&... args) noexcept( {<cond-noexcept-6>});`	
Exception Safety:	conditionally exception safe	
Description:	As per corresponding usage in `[variant]` except for the following deviations: 1. Function is conditionally `noexcept`	
Descriptors:	`{<cond-noexcept-6>}`	As per corresponding exception condition in `[variant]`. Condition is equivalent to `std::is_nothrow_constructible<T_I, std::initializer_list<U>&, Args&&...>::value`

⌋

8.9.5.1.2.5 Variant(T&&)

[SWS_CORE_01652] Definition of API function ara::core::Variant::Variant

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`class ara::core::Variant`	
Syntax:	`template <typename T>` `constexpr Variant (T &&value) noexcept({<cond-noexcept-4>});`	
Exception Safety:	conditionally exception safe	
Description:	As per corresponding usage in `[variant]`	
Descriptors:	`{<cond-noexcept-4>` `}`	As per corresponding exception condition in `[variant]`

⌋

8.9.5.1.3 Member Functions

8.9.5.1.3.1 emplace(Args ...&&)

[SWS_CORE_01661] Definition of API function ara::core::Variant::emplace

Status: DRAFT

⌈

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

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

Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[variant]` except for the following deviations: 1. Function is conditionally `noexcept`

⌋

8.9.5.1.3.2 emplace(std::initializer_list<U>, Args ...&&)

[SWS_CORE_01664] Definition of API function ara::core::Variant::emplace

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`class ara::core::Variant`	
Syntax:	`template <std::size_t I, typename U, typename... Args> variant_alternative_t< I, Variant< Types... > > & emplace (std::initializer_list< U > il, Args &&... args) noexcept( {<cond-noexcept-12>});`	
Exception Safety:	conditionally exception safe	
Description:	As per corresponding usage in `[variant]` except for the following deviations: 1. Function is conditionally `noexcept`	
Descriptors:	`{ <cond-noexcept-12> }`	As per corresponding exception condition in `[variant]`. Condition is equivalent to `std::is_nothrow_constructible<T_I, std::initializer_list<U>&, Args&&...>::value`

⌋

8.9.5.1.4 Modifiers

8.9.5.1.4.1 emplace(std::initializer_list<U>, Args ...&&)

[SWS_CORE_01662] Definition of API function ara::core::Variant::emplace

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/variant.h"
Scope:	`class ara::core::Variant`
Syntax:	`template <typename T, typename U, typename... Args> T & emplace (std::initializer_list< U > il, Args &&... args) noexcept(std::is_nothrow_constructible< T, std::initializer_list< U > &, Args &&... >::value);`
Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[variant]` except for the following deviations: 1. Function is conditionally `noexcept`

⌋

8.9.5.1.4.2 emplace(Args ...&&)

[SWS_CORE_01663] Definition of API function ara::core::Variant::emplace

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`class ara::core::Variant`	
Syntax:	`template <std::size_t I, typename... Args> variant_alternative_t< I, Variant< Types... > > & emplace (Args &&... args) noexcept({<cond-noexcept-11>});`	
Exception Safety:	conditionally exception safe	
Description:	As per corresponding usage in `[variant]` except for the following deviations: 1. Function is conditionally `noexcept`	
Descriptors:	`{ <cond-noexcept-11> }`	As per corresponding exception condition in `[variant]`. Condition is equivalent to `std::is_nothrow_constructible <T_I, Args...>::value`

⌋

8.9.5.1.4.3 swap

[SWS_CORE_01667] Definition of API function ara::core::Variant::swap

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "ara/core/variant.h"	
Scope:	`class ara::core::Variant`	
Syntax:	`void swap (Variant &other) noexcept({<cond-noexcept-13>});`	
Exception Safety:	conditionally exception safe	
Description:	As per corresponding usage in `[variant]` except for the following deviations: 1. Conditions for the conditional exception safety are modified	
Descriptors:	`{ <cond-noexcept-13> }`	As per corresponding exception condition in `[variant]`. Condition is the logical AND of `std::is_nothrow_move_ constructible<Ti>::value && std::is_nothrow_move_ assignable<Ti>::value` for all i

⌋

8.9.5.1.5 Observers

8.9.5.1.5.1 index

[SWS_CORE_01665] Definition of API function ara::core::Variant::index

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/variant.h"
Scope:	`class ara::core::Variant`
Syntax:	`constexpr std::size_t index () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[variant]`

⌋

8.9.5.1.5.2 valueless_by_exception

[SWS_CORE_01666] Definition of API function ara::core::Variant::valueless_by_exception

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/variant.h"
Scope:	`class ara::core::Variant`
Syntax:	`constexpr bool valueless_by_exception () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[variant]`

⌋

8.9.5.1.6 Operator=

8.9.5.1.6.1 operator=

[SWS_CORE_01660] Definition of API function ara::core::Variant::operator=

Status: DRAFT

⌈

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

▽

△

Syntax:	`template <typename T>` `Variant & operator= (T &&value) noexcept({<cond-noexcept-10>});`	
Exception Safety:	conditionally exception safe	
Description:	As per corresponding usage in `[variant]`	

Descriptors:	`{`
	`<cond-noexcept-10>`
	`}`
		As per corresponding exception condition in `[variant]`

]

8.9.6 Struct: variant_alternative

[SWS_CORE_01607] Definition of API class ara::core::variant_alternative

Status: DRAFT

[

Kind:	struct	
Header file:	#include "ara/core/variant.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	namespace ara::core	
Symbol:	variant_alternative	
Syntax:	`template <std::size_t I, typename T>` `struct variant_alternative;`	
Template param:	std::size_t I	--
	typename T	--
Description:	As per corresponding usage in `[variant]`	

]

8.9.7 Struct: variant_alternative<I, Variant<Types...>>

[SWS_CORE_01611] Definition of API class ara::core::variant_alternative< I, Variant< Types... > >

Status: DRAFT

[

Kind:	struct	
Header file:	#include "ara/core/variant.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	namespace ara::core	
Symbol:	variant_alternative< I, Variant< Types... > >	
Syntax:	`template <std::size_t I, typename... Types>` `struct variant_alternative< I, Variant< Types... > > {...};`	
Template param:	std::size_t I	--

▽

△

	typename... Types	--
Description:	As per corresponding usage in `[variant]`	

⌋

8.9.8 Struct: variant_alternative<I, const T>

[SWS_CORE_01608] Definition of API class ara::core::variant_alternative< I, const T >

Status: DRAFT

⌈

Kind:	struct	
Header file:	#include "ara/core/variant.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	namespace ara::core	
Symbol:	variant_alternative< I, const T >	
Syntax:	`template <std::size_t I, typename T>` `struct variant_alternative< I, const T > {...};`	
Template param:	std::size_t I	--
	typename T	--
Description:	As per corresponding usage in `[variant]`	

⌋

8.9.9 Struct: variant_alternative<I, const volatile T>

[SWS_CORE_01610] Definition of API class ara::core::variant_alternative< I, const volatile T >

Status: DRAFT

⌈

Kind:	struct	
Header file:	#include "ara/core/variant.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	namespace ara::core	
Symbol:	variant_alternative< I, const volatile T >	
Syntax:	`template <std::size_t I, typename T>` `struct variant_alternative< I, const volatile T > {...};`	
Template param:	std::size_t I	--
	typename T	--
Description:	As per corresponding usage in `[variant]`	

⌋

8.9.10 Struct: variant_alternative<I, volatile T>

[SWS_CORE_01609] Definition of API class ara::core::variant_alternative< I, volatile T >

Status: DRAFT

⌈

Kind:	struct	
Header file:	#include "ara/core/variant.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	namespace ara::core	
Symbol:	variant_alternative< I, volatile T >	
Syntax:	`template <std::size_t I, typename T>` `struct variant_alternative< I, volatile T > {...};`	
Template param:	std::size_t I	--
	typename T	--
Description:	As per corresponding usage in `[variant]`	

⌋

8.9.11 Struct: variant_size

[SWS_CORE_01600] Definition of API class ara::core::variant_size

Status: DRAFT

⌈

Kind:	struct	
Header file:	#include "ara/core/variant.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	namespace ara::core	
Symbol:	variant_size	
Syntax:	`template <typename T>` `struct variant_size;`	
Template param:	typename T	--
Description:	As per corresponding usage in `[variant]`	

⌋

8.9.12 Struct: variant_size<Variant<Types...>>

[SWS_CORE_01605] Definition of API class ara::core::variant_size< Variant< Types... > >

Status: DRAFT

⌈

Kind:	struct	
Header file:	#include "ara/core/variant.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	namespace ara::core	
Symbol:	variant_size< Variant< Types... > >	
Syntax:	`template <typename... Types>` `struct variant_size< Variant< Types... > > {...};`	
Template param:	typename... Types	--
Description:	As per corresponding usage in `[variant]`	

⌋

8.9.13 Struct: variant_size<const T>

[SWS_CORE_01602] Definition of API class ara::core::variant_size< const T >

Status: DRAFT

⌈

Kind:	struct	
Header file:	#include "ara/core/variant.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	namespace ara::core	
Symbol:	variant_size< const T >	
Syntax:	`template <typename T>` `struct variant_size< const T > {...};`	
Template param:	typename T	--
Description:	As per corresponding usage in `[variant]`	

⌋

8.9.14 Struct: variant_size<const volatile T>

[SWS_CORE_01604] Definition of API class ara::core::variant_size< const volatile T >

Status: DRAFT

⌈

Kind:	struct	
Header file:	#include "ara/core/variant.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	namespace ara::core	
Symbol:	variant_size< const volatile T >	
Syntax:	`template <typename T>` `struct variant_size< const volatile T > {...};`	
Template param:	typename T	--
Description:	As per corresponding usage in `[variant]`	

⌋

8.9.15 Struct: variant_size<volatile T>

[SWS_CORE_01603] Definition of API class ara::core::variant_size< volatile T >

Status: DRAFT

⌈

Kind:	struct	
Header file:	#include "ara/core/variant.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	namespace ara::core	
Symbol:	variant_size< volatile T >	
Syntax:	`template <typename T>` `struct variant_size< volatile T > {...};`	
Template param:	typename T	--
Description:	As per corresponding usage in `[variant]`	

⌋

8.9.16 Struct: Monostate>

[SWS_CORE_01670] Definition of API class std::hash< ara::core::Monostate > ⌈

Kind:	struct
Header file:	#include "ara/core/variant.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	namespace std

▽

△

Symbol:	hash< ara::core::Monostate >
Syntax:	`template <>` `struct hash< ara::core::Monostate > {...};`
Description:	Specialization of `std::hash` (hashing function for `ara::core::Monostate`)

⌋

8.9.16.1 Public Member Functions

8.9.16.1.1 Member Functions

8.9.16.1.1.1 operator()

[SWS_CORE_01671] Definition of API function std::hash< ara::core::Monostate >::operator() ⌈

Kind:	function
Header file:	#include "ara/core/variant.h"
Scope:	`struct std::hash< ara::core::Monostate >`
Syntax:	`size_t operator() (ara::core::Monostate const &v) const noexcept;`
Exception Safety:	exception safe
Description:	Invoke the hashing function

⌋

8.9.17 Struct: Variant<Types...>>

[SWS_CORE_01668] Definition of API class std::hash< ara::core::Variant< Types... > > ⌈

Kind:	struct	
Header file:	#include "ara/core/variant.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	namespace std	
Symbol:	hash< ara::core::Variant< Types... > >	
Syntax:	`template <typename... Types>` `struct hash< ara::core::Variant< Types... > > {...};`	
Template param:	typename... Types	--
Description:	Specialization of `std::hash` (hashing function for `ara::core::Variant<Types...>`)	

⌋

8.9.17.1 Public Member Functions

8.9.17.1.1 Member Functions

8.9.17.1.1.1 operator()

[SWS_CORE_01669] Definition of API function std::hash< ara::core::Variant< Types... > >::operator() ⌈

Kind:	function
Header file:	#include "ara/core/variant.h"
Scope:	`struct std::hash< ara::core::Variant< Types... > >`
Syntax:	`size_t operator() (ara::core::Variant< Types... > const &v) const noexcept;`
Exception Safety:	exception safe
Description:	Invoke the hashing function

⌋

8.10 Header: ara/core/vector.h

[SWS_CORE_01300] Definition of Header File ara/core/vector.h

Upstream requirements: RS_AP_00130

⌈

Kind:	Header File
Syntax:	`ara/core/vector.h`
Description:	Implements `[vector.syn]` in [11]. Unless explicitly overridden in the API documentation, APIs always adhere in behavior to the ISO specification. To reduce duplication, the APIs here specify only any deviations from [11] or any AUTOSAR specific overrides. For APIs which would ordinarily throw `Exception`s, and those `Exception`s have been suppressed in favor of AUTOSAR violations, generally the semantics of the `Exception` from [11] still apply (unless locally overridden here).

⌋

8.10.1 Non-Member Functions

8.10.1.1 Other

8.10.1.1.1 operator!=

[SWS_CORE_01391] Definition of API function ara::core::operator!=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	namespace ara::core
Syntax:	`template <typename T, typename Allocator>` `bool operator!= (const Vector< T, Allocator > &lhs, const Vector< T, Allocator > &rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.1.1.2 operator<

[SWS_CORE_01392] Definition of API function ara::core::operator<

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	namespace ara::core
Syntax:	`template <typename T, typename Allocator>` `bool operator< (const Vector< T, Allocator > &lhs, const Vector< T, Allocator > &rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.1.1.3 operator<=

[SWS_CORE_01393] Definition of API function ara::core::operator<=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	namespace ara::core
Syntax:	`template <typename T, typename Allocator>` `bool operator<= (const Vector< T, Allocator > &lhs, const Vector< T, Allocator > &rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.1.1.4 operator==

[SWS_CORE_01390] Definition of API function ara::core::operator==

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	namespace ara::core
Syntax:	`template <typename T, typename Allocator>` `bool operator== (const Vector< T, Allocator > &lhs, const Vector< T, Allocator > &rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.1.1.5 operator>

[SWS_CORE_01394] Definition of API function ara::core::operator>

Status: DRAFT

⌈

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

▽

△

Syntax:	`template <typename T, typename Allocator>` `bool operator> (const Vector< T, Allocator > &lhs, const Vector< T, Allocator > &rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.1.1.6 operator>=

[SWS_CORE_01395] Definition of API function ara::core::operator>=

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	namespace ara::core
Syntax:	`template <typename T, typename Allocator>` `bool operator>= (const Vector< T, Allocator > &lhs, const Vector< T, Allocator > &rhs);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.1.1.7 swap

[SWS_CORE_01396] Definition of API function ara::core::swap

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	namespace ara::core
Syntax:	`template <typename T, typename Allocator>` `void swap (Vector< T, Allocator > &lhs, Vector< T, Allocator > &rhs) noexcept(noexcept(lhs.swap(rhs)));`
Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2 Class: Vector

[SWS_CORE_01301] Definition of API class ara::core::Vector

Status: DRAFT

⌈

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 = {<fq-allocator>}> class Vector {...};`	
Template param:	typename T	--
	typename Allocator = {<fq-allocator>}	--
Description:	Implementation of `std::pmr::vector` container from `[vector]`	
Descriptors:	{<fq-allocator>}	Implementation-defined. Fully qualified `Allocator` class. The Allocator shall satisfy `[allocator.requirements.completeness]` in [11]

⌋

8.10.2.1 Public Member Types

8.10.2.1.1 Type Alias: allocator_type

[SWS_CORE_01303] Definition of API type ara::core::Vector::allocator_type ⌈

Kind:	type alias
Header file:	#include "ara/core/vector.h"
Scope:	class ara::core::Vector
Symbol:	allocator_type
Syntax:	`using allocator_type = Allocator;`
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.1.2 Type Alias: const_iterator

[SWS_CORE_01311] Definition of API type ara::core::Vector::const_iterator ⌈

Kind:	type alias
Header file:	#include "ara/core/vector.h"
Scope:	class ara::core::Vector
Symbol:	const_iterator

▽

△

Syntax:	`using const_iterator = {<const-iterator-type>};`	
Description:	As per corresponding usage in `[vector]`	
Implementation Defined Behavior:	`{<const-iterator-type>}` shall be an implementation-defined type	
Descriptors:	`{<const-iterator-type>}`	implementation-defined

⌋

8.10.2.1.3 Type Alias: const_pointer

[SWS_CORE_01305] Definition of API type ara::core::Vector::const_pointer ⌈

Kind:	type alias
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Symbol:	const_pointer
Syntax:	`using const_pointer = typename std::allocator_traits<Allocator>::const_pointer;`
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.1.4 Type Alias: const_reference

[SWS_CORE_01307] Definition of API type ara::core::Vector::const_reference ⌈

Kind:	type alias
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Symbol:	const_reference
Syntax:	`using const_reference = const value_type&;`
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.1.5 Type Alias: const_reverse_iterator

[SWS_CORE_01313] Definition of API type ara::core::Vector::const_reverse_iterator ⌈

Kind:	type alias
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Symbol:	const_reverse_iterator
Syntax:	`using const_reverse_iterator = std::reverse_iterator<const_iterator>;`
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.1.6 Type Alias: difference_type

[SWS_CORE_01309] Definition of API type ara::core::Vector::difference_type ⌈

Kind:	type alias	
Header file:	#include "ara/core/vector.h"	
Scope:	`class ara::core::Vector`	
Symbol:	difference_type	
Syntax:	`using difference_type = {<difference-type>};`	
Description:	As per corresponding usage in `[vector]`	
Implementation Defined Behavior:	`{<difference-type>}` shall be an implementation-defined type	
Descriptors:	`{<difference-type>}`	implementation-defined

⌋

8.10.2.1.7 Type Alias: iterator

[SWS_CORE_01310] Definition of API type ara::core::Vector::iterator ⌈

Kind:	type alias	
Header file:	#include "ara/core/vector.h"	
Scope:	`class ara::core::Vector`	
Symbol:	iterator	
Syntax:	`using iterator = {<iterator-type>};`	
Description:	As per corresponding usage in `[vector]`	
Implementation Defined Behavior:	`{<iterator-type>}` shall be an implementation-defined type	
Descriptors:	`{<iterator-type>}`	implementation-defined

⌋

8.10.2.1.8 Type Alias: pointer

[SWS_CORE_01304] Definition of API type ara::core::Vector::pointer ⌈

Kind:	type alias
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Symbol:	pointer
Syntax:	`using pointer = typename std::allocator_traits<Allocator>::pointer;`
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.1.9 Type Alias: reference

[SWS_CORE_01306] Definition of API type ara::core::Vector::reference ⌈

Kind:	type alias
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Symbol:	reference
Syntax:	`using reference = value_type&;`
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.1.10 Type Alias: reverse_iterator

[SWS_CORE_01312] Definition of API type ara::core::Vector::reverse_iterator ⌈

Kind:	type alias
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Symbol:	reverse_iterator
Syntax:	`using reverse_iterator = std::reverse_iterator<iterator>;`
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.1.11 Type Alias: size_type

[SWS_CORE_01308] Definition of API type ara::core::Vector::size_type ⌈

Kind:	type alias	
Header file:	#include "ara/core/vector.h"	
Scope:	`class ara::core::Vector`	
Symbol:	size_type	
Syntax:	`using size_type = {<size-type>};`	
Description:	As per corresponding usage in `[vector]`	
Implementation Defined Behavior:	`{<size-type>}` shall be an implementation-defined type	
Descriptors:	`{<size-type>}`	implementation-defined

⌋

8.10.2.1.12 Type Alias: value_type

[SWS_CORE_01302] Definition of API type ara::core::Vector::value_type ⌈

Kind:	type alias
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Symbol:	value_type
Syntax:	`using value_type = T;`
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2 Public Member Functions

8.10.2.2.1 Special Member Functions

8.10.2.2.1.1 Move Constructor

[SWS_CORE_01316] Definition of API function ara::core::Vector::Vector

Upstream requirements: RS_AP_00145

⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`Vector (Vector &&other) noexcept(std::allocator_traits< Allocator >::propagate_on_container_move_assignment::value||std::allocator_traits< Allocator >::is_always_equal::value)=default;`
Exception Safety:	conditionally exception safe

▽

△

Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.1.2 Default Constructor

[SWS_CORE_01318] Definition of API function ara::core::Vector::Vector ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`Vector () noexcept(noexcept(Allocator()));`
Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.1.3 Copy Constructor

[SWS_CORE_01314] Definition of API function ara::core::Vector::Vector

Upstream requirements: RS_AP_00145

⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`Vector (const Vector &other) noexcept(std::is_nothrow_copy_constructible< T >::value)=default;`
Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[vector]` except for the following deviations: 1. The function is `noexcept`

⌋

8.10.2.2.1.4 Copy Assignment Operator

[SWS_CORE_01315] Definition of API function ara::core::Vector::operator=

Upstream requirements: RS_AP_00145

⌈

Kind:	function	
Header file:	#include "ara/core/vector.h"	
Scope:	`class ara::core::Vector`	
Syntax:	`Vector & operator= (const Vector &other) noexcept(std::is_nothrow_copy_assignable< T >::value)=default;`	
Exception Safety:	conditionally exception safe	
Description:	As per corresponding usage in `[vector]` except for the following deviations: 1. Function is conditionally `noexcept`	
Implementation Defined Behavior:	`{<may-throw-on-allocate>}` shall be an implementation-defined type	
Descriptors:	`{<may-throw-on-allocate>}`	implementation-defined

⌋

8.10.2.2.1.5 Move Assignment Operator

[SWS_CORE_01317] Definition of API function ara::core::Vector::operator=

Upstream requirements: RS_AP_00145

⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`Vector & operator= (Vector &&x) noexcept(std::allocator_traits< Allocator >::propagate_on_container_move_assignment::value||std::allocator_traits< Allocator >::is_always_equal::value);`
Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.1.6 Destructor

[SWS_CORE_01326] Definition of API function ara::core::Vector::~Vector ⌈

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

▽

△

Syntax:	`~Vector () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[vector]` except for the following deviations: 1. The function is `noexcept`

⌋

8.10.2.2.2 Constructors

8.10.2.2.2.1 Vector(Vector&&, const Allocator&)

[SWS_CORE_01324] Definition of API function ara::core::Vector::Vector ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`Vector (Vector &&, const Allocator &);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.2.2 Vector(const Allocator&)

[SWS_CORE_01319] Definition of API function ara::core::Vector::Vector ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`explicit Vector (const Allocator &) noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.2.3 Vector(size_type, const Allocator&)

[SWS_CORE_01320] Definition of API function ara::core::Vector::Vector ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`explicit Vector (size_type n, const Allocator &=Allocator());`

▽

△

Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.2.4 Vector(size_type, const T&, const Allocator&)

[SWS_CORE_01321] Definition of API function ara::core::Vector::Vector ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`Vector (size_type n, const T &value, const Allocator &=Allocator());`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.2.5 Vector(InputIterator, InputIterator, const Allocator&)

[SWS_CORE_01322] Definition of API function ara::core::Vector::Vector ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`template <class InputIterator>` `Vector (InputIterator first, InputIterator last, const Allocator &=Allocator());`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.2.6 Vector(const Vector&, const Allocator&)

[SWS_CORE_01323] Definition of API function ara::core::Vector::Vector ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`Vector (const Vector &, const Allocator &);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.2.7 Vector(std::initializer_list<T>, const Allocator&)

[SWS_CORE_01325] Definition of API function ara::core::Vector::Vector ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`Vector (std::initializer_list< T >, const Allocator &=Allocator());`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.3 Capacity

8.10.2.2.3.1 capacity

[SWS_CORE_01348] Definition of API function ara::core::Vector::capacity ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`size_type capacity () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.3.2 empty

[SWS_CORE_01344] Definition of API function ara::core::Vector::empty ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`bool empty () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.3.3 max_size

[SWS_CORE_01346] Definition of API function ara::core::Vector::max_size ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`size_type max_size () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.3.4 reserve

[SWS_CORE_01347] Definition of API function ara::core::Vector::reserve ⌈

Kind:	function	
Header file:	#include "ara/core/vector.h"	
Scope:	`class ara::core::Vector`	
Syntax:	`void reserve (size_type n);`	
Exception Safety:	not exception safe	
Violations:	`IsoStdLengthErrorViolation`	Description as per std::length_error
Description:	As per corresponding usage in `[vector]` except for the following deviations: 1. if (n < `max_size()`) `IsoStdLengthErrorViolation` is raised instead of `std::length_error`	

⌋

8.10.2.2.3.5 shrink_to_fit

[SWS_CORE_01349] Definition of API function ara::core::Vector::shrink_to_fit ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`void shrink_to_fit ();`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.3.6 size

[SWS_CORE_01345] Definition of API function ara::core::Vector::size ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`size_type size () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.4 Element access

8.10.2.2.4.1 at(size_type)

[SWS_CORE_01355] Definition of API function ara::core::Vector::at ⌈

Kind:	function	
Header file:	#include "ara/core/vector.h"	
Scope:	`class ara::core::Vector`	
Syntax:	`reference at (size_type n);`	
Exception Safety:	not exception safe	
Violations:	`IsoStdOut-OfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[vector]` 1. if (n >= `size()`) `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range`	

⌋

8.10.2.2.4.2 at(size_type) const

[SWS_CORE_01354] Definition of API function ara::core::Vector::at ⌈

Kind:	function	
Header file:	#include "ara/core/vector.h"	
Scope:	`class ara::core::Vector`	
Syntax:	`const_reference at (size_type n) const;`	
Exception Safety:	not exception safe	
Violations:	`IsoStdOut-OfRangeViolation`	Description as per std::out_of_range.
Description:	As per corresponding usage in `[vector]` except for the following deviations: 1. if (n >= `size()`) `IsoStdOutOfRangeViolation` is raised instead of `std::out_of_range`	

⌋

8.10.2.2.4.3 back()

[SWS_CORE_01358] Definition of API function ara::core::Vector::back ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`reference back ();`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.4.4 back() const

[SWS_CORE_01359] Definition of API function ara::core::Vector::back ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`const_reference back () const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.4.5 data() const

[SWS_CORE_01361] Definition of API function ara::core::Vector::data ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`const T * data () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.4.6 data()

[SWS_CORE_01360] Definition of API function ara::core::Vector::data ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`T * data () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.4.7 front()

[SWS_CORE_01356] Definition of API function ara::core::Vector::front ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`reference front ();`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.4.8 front() const

[SWS_CORE_01357] Definition of API function ara::core::Vector::front ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`const_reference front () const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.4.9 operator[](size_type)

[SWS_CORE_01352] Definition of API function ara::core::Vector::operator[] ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`reference operator[] (size_type n);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.4.10 operator[](size_type) const

[SWS_CORE_01353] Definition of API function ara::core::Vector::operator[] ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`const_reference operator[] (size_type n) const;`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.5 Iterators

8.10.2.2.5.1 begin() const

[SWS_CORE_01333] Definition of API function ara::core::Vector::begin ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`const_iterator begin () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.5.2 begin()

[SWS_CORE_01332] Definition of API function ara::core::Vector::begin ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`iterator begin () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.5.3 cbegin

[SWS_CORE_01340] Definition of API function ara::core::Vector::cbegin ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`const_iterator cbegin () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.5.4 cend

[SWS_CORE_01341] Definition of API function ara::core::Vector::cend ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`const_iterator cend () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.5.5 crbegin

[SWS_CORE_01342] Definition of API function ara::core::Vector::crbegin ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`const_reverse_iterator crbegin () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.5.6 crend

[SWS_CORE_01343] Definition of API function ara::core::Vector::crend ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`const_reverse_iterator crend () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.5.7 end()

[SWS_CORE_01334] Definition of API function ara::core::Vector::end ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`iterator end () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.5.8 end() const

[SWS_CORE_01335] Definition of API function ara::core::Vector::end ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`const_iterator end () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.5.9 rbegin() const

[SWS_CORE_01337] Definition of API function ara::core::Vector::rbegin ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`const_reverse_iterator rbegin () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.5.10 rbegin()

[SWS_CORE_01336] Definition of API function ara::core::Vector::rbegin ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`reverse_iterator rbegin () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.5.11 rend() const

[SWS_CORE_01339] Definition of API function ara::core::Vector::rend ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`const_reverse_iterator rend () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.5.12 rend()

[SWS_CORE_01338] Definition of API function ara::core::Vector::rend ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`reverse_iterator rend () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.6 Modifiers

8.10.2.2.6.1 clear

[SWS_CORE_01375] Definition of API function ara::core::Vector::clear ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`void clear () noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.6.2 emplace

[SWS_CORE_01366] Definition of API function ara::core::Vector::emplace ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`template <class... Args>` `iterator emplace (const_iterator position, Args &&... args);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.6.3 emplace_back

[SWS_CORE_01362] Definition of API function ara::core::Vector::emplace_back ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`template <class... Args>` `reference emplace_back (Args &&... args);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.6.4 erase(const_iterator)

[SWS_CORE_01372] Definition of API function ara::core::Vector::erase ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`iterator erase (const_iterator position);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.6.5 erase(const_iterator, const_iterator)

[SWS_CORE_01373] Definition of API function ara::core::Vector::erase ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`iterator erase (const_iterator first, const_iterator last);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.6.6 insert(const_iterator, InputIterator, InputIterator)

[SWS_CORE_01370] Definition of API function ara::core::Vector::insert ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`template <class InputIterator>` `iterator insert (const_iterator position, InputIterator first, Input Iterator last);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.6.7 insert(const_iterator, std::initializer_list<T>)

[SWS_CORE_01371] Definition of API function ara::core::Vector::insert ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`iterator insert (const_iterator position, std::initializer_list< T > il);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.6.8 insert(const_iterator, T&&)

[SWS_CORE_01368] Definition of API function ara::core::Vector::insert ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`iterator insert (const_iterator position, T &&x);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.6.9 insert(const_iterator, const T&)

[SWS_CORE_01367] Definition of API function ara::core::Vector::insert ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`iterator insert (const_iterator position, const T &x);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.6.10 insert(const_iterator, size_type, const T&)

[SWS_CORE_01369] Definition of API function ara::core::Vector::insert ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`iterator insert (const_iterator position, size_type n, const T &x);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.6.11 pop_back

[SWS_CORE_01365] Definition of API function ara::core::Vector::pop_back ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`void pop_back ();`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.6.12 push_back(const T&)

[SWS_CORE_01363] Definition of API function ara::core::Vector::push_back ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`void push_back (const T &x) noexcept(std::is_nothrow_copy_assignable< T >::value);`
Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[vector]` except for the following deviations: 1. Function is conditionally `noexcept`: (see `[vector.modifiers]`)

⌋

8.10.2.2.6.13 push_back(T&&)

[SWS_CORE_01364] Definition of API function ara::core::Vector::push_back ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`void push_back (T &&x) noexcept(std::is_nothrow_move_assignable< T >::value);`
Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[vector]` except for the following deviations: 1. Function is conditionally `noexcept`: (see `[vector.modifiers]`)

⌋

8.10.2.2.6.14 resize(size_type, const T&)

[SWS_CORE_01351] Definition of API function ara::core::Vector::resize ⌈

Kind:	function	
Header file:	#include "ara/core/vector.h"	
Scope:	`class ara::core::Vector`	
Syntax:	`void resize (size_type sz, const T &c);`	
Exception Safety:	not exception safe	
Violations:	`IsoStdLengthErrorViolation`	Description as per std::length_error
Description:	As per corresponding usage in `[vector]` except for the following deviations: 1. if (sz > `max_size()`) `IsoStdLengthErrorViolation` is raised instead of `std::length_error`	

⌋

8.10.2.2.6.15 resize(size_type)

[SWS_CORE_01350] Definition of API function ara::core::Vector::resize ⌈

Kind:	function	
Header file:	#include "ara/core/vector.h"	
Scope:	`class ara::core::Vector`	
Syntax:	`void resize (size_type sz);`	
Exception Safety:	not exception safe	
Violations:	`IsoStdLengthErrorViolation`	Description as per std::length_error
Description:	As per corresponding usage in `[vector]` except for the following deviations: 1. if (sz > `max_size()`) `IsoStdLengthErrorViolation` is raised instead of `std::length_error`	

⌋

8.10.2.2.6.16 swap

[SWS_CORE_01374] Definition of API function ara::core::Vector::swap ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`void swap (Vector &) noexcept(std::allocator_traits< Allocator >::propagate_on_container_swap::value||std::allocator_traits< Allocator >::is_always_equal::value);`
Exception Safety:	conditionally exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.7 Operator=

8.10.2.2.7.1 operator=(std::initializer_list<T>)

[SWS_CORE_01327] Definition of API function ara::core::Vector::operator= ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`Vector & operator= (std::initializer_list< T >);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.8 Other

8.10.2.2.8.1 assign(InputIterator, InputIterator)

[SWS_CORE_01328] Definition of API function ara::core::Vector::assign ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`template <class InputIterator>` `void assign (InputIterator first, InputIterator last);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.8.2 assign(std::initializer_list<T>)

[SWS_CORE_01330] Definition of API function ara::core::Vector::assign ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`void assign (std::initializer_list< T >);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.8.3 assign(size_type, const T&)

[SWS_CORE_01329] Definition of API function ara::core::Vector::assign ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`void assign (size_type n, const T &u);`
Exception Safety:	not exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.2.2.8.4 get_allocator

[SWS_CORE_01331] Definition of API function ara::core::Vector::get_allocator ⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`class ara::core::Vector`
Syntax:	`allocator_type get_allocator () const noexcept;`
Exception Safety:	exception safe
Description:	As per corresponding usage in `[vector]`

⌋

8.10.3 Struct: hash

[SWS_CORE_01376] Definition of API class std::hash< ara::core::Vector< T, Allocator > >

Status: DRAFT

⌈

Kind:	struct
Header file:	#include "ara/core/vector.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	namespace std
Symbol:	hash< ara::core::Vector< T, Allocator > >
Syntax:	`template <typename T, typename Allocator>` `struct hash< ara::core::Vector< T, Allocator > > final {...};`
Description:	Specialization of `std::hash` (hashing function for `ara::core::Vector`)

⌋

8.10.3.1 Public Member Functions

8.10.3.1.1 Member Functions

8.10.3.1.1.1 operator()

[SWS_CORE_01377] Definition of API function std::hash< ara::core::Vector< T, Allocator > >::operator()

Status: DRAFT

⌈

Kind:	function
Header file:	#include "ara/core/vector.h"
Scope:	`struct std::hash< ara::core::Vector< T, Allocator > >`
Syntax:	`size_t operator() (ara::core::Vector< T, Allocator > const &v) const noexcept;`
Exception Safety:	exception safe
Description:	Invoke the hashing function

⌋

8.11 Header: ara/core/abort.h

8.11.1 Non-Member Types

8.11.1.1 Type Alias: AbortHandler

[SWS_CORE_00050] Definition of API type ara::core::AbortHandler

Upstream requirements: RS_AP_00159

⌈

Kind:	type alias
Header file:	#include "ara/core/abort.h"
Scope:	namespace ara::core
Symbol:	AbortHandler
Syntax:	`using AbortHandler = decltype(&AbortHandlerPrototype);`
Description:	The type of a handler for `ara::core::SetAbortHandler`

⌋

8.11.2 Non-Member Functions

8.11.2.1 Other

8.11.2.1.1 Abort

[SWS_CORE_00052] Definition of API function ara::core::Abort

Upstream requirements: RS_AP_00127, RS_AP_00159, RS_AP_00136

⌈

Kind:	function	
Header file:	#include "ara/core/abort.h"	
Scope:	namespace ara::core	
Syntax:	`template <typename... Args>` `[[noreturn]] void Abort (const Args &... args) noexcept;`	
Template param:	Args...	the types of arguments given to this function
Parameters (in):	args	custom texts to be added in the log message being output
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Abort the current operation This function will never return to its caller. The stack is not unwound: destructors of variables with automatic storage duration are not called. Calling this function is ill-formed if any of the arguments is not convertible to `ara::core::StringView`.	

⌋

8.11.2.1.2 AbortHandlerPrototype

[SWS_CORE_00053] Definition of API function ara::core::AbortHandlerPrototype

Upstream requirements: RS_AP_00159

⌈

Kind:	function
Header file:	#include "ara/core/abort.h"
Scope:	namespace ara::core
Syntax:	`void AbortHandlerPrototype () noexcept;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	A function declaration with the correct prototype for `ara::core::SetAbortHandler` This declaration exists only for providing a function type that includes `noexcept` and that acts as base type for a type alias, which is defined in [SWS_CORE_00050]. This compensates for the fact that the C++ standard (up to and including C++14) prohibits that `noexcept` appears in an alias-declaration. There is no implementation of this function.

⌋

8.11.2.1.3 AddAbortHandler

[SWS_CORE_00054] Definition of API function ara::core::AddAbortHandler

Upstream requirements: RS_AP_00159

⌈

Kind:	function	
Header file:	#include "ara/core/abort.h"	
Scope:	namespace ara::core	
Syntax:	`bool AddAbortHandler (AbortHandler handler) noexcept;`	
Parameters (in):	handler	a custom Abort handler
Return value:	bool	true if the given handler was successfully installed; false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Add a custom Abort handler function. Implementations support at least 8 AbortHandlers.	
Implementation Defined Behavior:	`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	

⌋

8.11.2.1.4 SetAbortHandler

[SWS_CORE_00051] Definition of API function ara::core::SetAbortHandler

Upstream requirements: RS_AP_00159

⌈

Kind:	function	
Header file:	#include "ara/core/abort.h"	
Scope:	namespace ara::core	
Syntax:	`AbortHandler SetAbortHandler (AbortHandler handler) noexcept;`	
Parameters (in):	handler	a custom Abort handler (or `nullptr`)
Return value:	AbortHandler	the most recently installed Abort handler (or `nullptr` if none was installed)
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Add a custom Abort handler function and return the most recently added one. By setting `nullptr`, the implementation may restore the default handler instead; this will remove all previously installed handlers.	
Implementation Defined Behavior:	This function can be called from multiple threads simultaneously; these calls are performed in an implementation-defined sequence	

⌋

8.12 Header: ara/core/exception.h

8.12.1 Class: Exception

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.12.1.1 Public Member Functions

8.12.1.1.1 Special Member Functions

8.12.1.1.1.1 Move Constructor

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.12.1.1.1.2 Move Assignment Operator

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.12.1.1.1.3 Destructor

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

Upstream requirements: RS_AP_00130, RS_AP_00145

⌈

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

⌋

8.12.1.1.2 Constructors

8.12.1.1.2.1 Exception

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

Upstream requirements: RS_AP_00130

⌈

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

▽

△

Syntax:	`explicit Exception (ErrorCode err) noexcept;`	
Parameters (in):	err	the ErrorCode
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Construct a new Exception object with a specific ErrorCode	

⌋

8.12.1.1.3 Member Functions

8.12.1.1.3.1 Error

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.12.1.1.3.2 what

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

Upstream requirements: RS_AP_00130

⌈

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

▽

△

Description:	Return the explanatory string. This function overrides the virtual function `std::exception::what`. All guarantees about the lifetime of the returned pointer that are given for std::exception::what are preserved.

⌋

8.12.1.2 Protected Member Functions

8.12.1.2.1 Special Member Functions

8.12.1.2.1.1 Copy Constructor

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.12.1.2.1.2 Copy Assignment Operator

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

Upstream requirements: RS_AP_00130

⌈

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

▽

△

Visibility:	protected
Notes:	This function has `protected` access specifier in order to prevent some opportunities for accidental object slicing

⌋

8.13 Header: ara/core/error_domain.h

8.13.1 Class: ErrorDomain

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.13.1.1 Public Member Types

8.13.1.1.1 Type Alias: CodeType

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.13.1.1.2 Type Alias: IdType

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.13.1.1.3 Type Alias: SupportDataType

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.13.1.2 Public Member Functions

8.13.1.2.1 Special Member Functions

8.13.1.2.1.1 Copy Constructor

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.13.1.2.1.2 Move Constructor

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.13.1.2.1.3 Move Assignment Operator

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.13.1.2.1.4 Copy Assignment Operator

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.13.1.2.2 Member Functions

8.13.1.2.2.1 Id

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

Upstream requirements: RS_AP_00130

⌈

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

▽

△

Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	Return the unique domain identifier

⌋

8.13.1.2.2.2 Message

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.13.1.2.2.3 Name

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/error_domain.h"	
Scope:	`class ara::core::ErrorDomain`	
Syntax:	`virtual const char * Name () const noexcept=0;`	
Return value:	const char *	the name as a null-terminated string, never `nullptr`
Exception Safety:	exception safe	
Thread Safety:	thread-safe	

▽

△

Description:	Return the name of this `ErrorDomain` The returned pointer remains owned by class `ara::core::ErrorDomain` and shall not be freed by clients.

⌋

8.13.1.2.2.4 ThrowAsException

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/error_domain.h"	
Scope:	`class ara::core::ErrorDomain`	
Syntax:	`virtual void ThrowAsException (const ErrorCode &errorCode) const noexcept(false)=0;`	
Parameters (in):	errorCode	the `errorCode`
Return value:	None	
Exceptions:	Any exception of the type as defined in [SWS_CORE_10953] containing the given `ErrorCode`	
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Throw the given `errorCode` as an `ara::core::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`.	
Notes:	This function shall not participate in overload resolution when C++ exceptions are disabled in the compiler toolchain. As per [SWS_CORE_10304].	

⌋

8.13.1.2.2.5 operator!=

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/error_domain.h"	
Scope:	`class ara::core::ErrorDomain`	
Syntax:	`constexpr bool operator!= (const ErrorDomain &other) const noexcept;`	
Parameters (in):	other	the other instance

▽

△

Return value:	bool	• `TRUE`: if `other` is equal to `*this` • `FALSE`: if `other` is NOT equal to `*this`
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Compare for non-equality with another `ara::core::ErrorDomain` instance	

⌋

8.13.1.2.2.6 operator==

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/error_domain.h"	
Scope:	`class ara::core::ErrorDomain`	
Syntax:	`constexpr bool operator== (const ErrorDomain &other) const noexcept;`	
Parameters (in):	other	the other instance
Return value:	bool	• `TRUE`: if `other` is equal to `*this` • `FALSE`: if `other` is NOT equal to `*this`
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Compare for equality with another `ara::core::ErrorDomain` instance. Two `ara::core::ErrorDomain` instances compare equal when their identifiers (returned by `Id()`) are equal	

⌋

8.13.1.3 Protected Member Functions

8.13.1.3.1 Special Member Functions

8.13.1.3.1.1 Destructor

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/error_domain.h"
Scope:	`class ara::core::ErrorDomain`
Syntax:	`~ErrorDomain () noexcept=default;`
Exception Safety:	exception safe
Description:	Destructor This dtor is non-virtual (and trivial) so that this class can be a literal type. While this class has virtual functions, no polymorphic destruction is needed.
Visibility:	protected

⌋

8.13.1.3.2 Constructors

8.13.1.3.2.1 ErrorDomain

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/error_domain.h"	
Scope:	`class ara::core::ErrorDomain`	
Syntax:	`explicit constexpr ErrorDomain (IdType id) noexcept;`	
Parameters (in):	id	the unique identifier
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Construct a new instance with the given `id`. Identifiers are expected to be system-wide unique.	
Visibility:	protected	

⌋

8.14 Header: ara/core/error_code.h

8.14.1 Non-Member Functions

8.14.1.1 Other

8.14.1.1.1 operator!=

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/error_code.h"	
Scope:	namespace ara::core	
Syntax:	`constexpr bool operator!= (const ErrorCode &lhs, const ErrorCode &rhs) noexcept;`	
Parameters (in):	lhs	the left hand side of the comparison
	rhs	the right hand side of the comparison
Return value:	bool	• `TRUE`: if the two instances compare NOT equal • `FALSE`: if the two instances compare equal
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Negative comparison of two `ErrorCode`s. Comparison semantics as per [SWS_CORE_00571].	

⌋

8.14.1.1.2 operator==

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/error_code.h"	
Scope:	namespace ara::core	
Syntax:	`constexpr bool operator== (const ErrorCode &lhs, const ErrorCode &rhs) noexcept;`	
Parameters (in):	lhs	the left hand side of the comparison
	rhs	the right hand side of the comparison
Return value:	bool	• `TRUE`: if the two instances compare equal • `FALSE`: if the two instances compare NOT equal
Exception Safety:	exception safe	
Thread Safety:	thread-safe	

▽

△

Description:	Comparison of two `ErrorCode`s. Two `ErrorCode` instances compare equal if the results of their `Value()` and `Domain()` functions are equal. The result of `SupportData()` is not considered for equality.

⌋

8.14.2 Class: ErrorCode

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

Upstream requirements: RS_AP_00130, RS_AP_00140

⌈

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

⌋

8.14.2.1 Public Member Functions

8.14.2.1.1 Constructors

8.14.2.1.1.1 ErrorCode

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/error_code.h"	
Scope:	`class ara::core::ErrorCode`	
Syntax:	`template <typename EnumT>` `constexpr ErrorCode (EnumT e, ErrorDomain::SupportDataType data=ErrorDomain::SupportDataType()) noexcept;`	
Template param:	EnumT	an enum type that contains error code values
Parameters (in):	e	a domain-specific error code value
	data	optional vendor-specific supplementary error context data
Exception Safety:	exception safe	
Thread Safety:	thread-safe	

▽

△

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

⌋

8.14.2.1.1.2 ErrorCode

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.14.2.1.2 Member Functions

8.14.2.1.2.1 Domain

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.14.2.1.2.2 Message

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.14.2.1.2.3 SupportData

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.14.2.1.2.4 ThrowAsException

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.14.2.1.2.5 Value

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.15 Header: ara/core/core_error_domain.h

8.15.1 Non-Member Types

8.15.1.1 Enumeration: CoreErrc

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

Upstream requirements: RS_AP_00130

⌈

Kind:	enumeration	
Header file:	#include "ara/core/core_error_domain.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	namespace ara::core	
Symbol:	CoreErrc	
Underlying type:	ErrorDomain::CodeType	
Syntax:	`enum class CoreErrc : ErrorDomain::CodeType {...};`	
Values:	kInvalidArgument	= 22
		The argument is invalid, corrupt or missing **status**=obsolete
	kInvalidMetaModel Shortname	= 137
		Given string is not a valid model element `shortName`
	kInvalidMetaModelPath	= 138
		Missing or invalid path to model element
	kAlreadyInitialized	= 141
		ARA has already been initialized and has completed
	kNotInitialized	= 142
		ARA is not currently initialized (i.e. has already been successfully de-initialized or has never been initialized)
Description:	Defines all errors of the `ara::core` Functional Cluster	

⌋

8.15.2 Non-Member Functions

8.15.2.1 Other

8.15.2.1.1 GetCoreErrorDomain

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

Upstream requirements: RS_AP_00130

⌈

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

▽

△

Syntax:	`constexpr const ErrorDomain & GetCoreErrorDomain () noexcept;`	
Return value:	const ErrorDomain &	the `ara::core::CoreErrorDomain`
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a reference to the global `ara::core::CoreErrorDomain`	

⌋

8.15.2.1.2 MakeErrorCode

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/core_error_domain.h"	
Scope:	namespace ara::core	
Syntax:	`constexpr ErrorCode MakeErrorCode (CoreErrc code, ErrorDomain::SupportDataType data) noexcept;`	
Parameters (in):	code	the `ara::core::CoreErrorDomain`-specific error code value
	data	optional vendor-specific error data
Return value:	ErrorCode	a new ErrorCode instance
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Create a new ErrorCode within `ara::core::CoreErrorDomain`. This function is used internally by constructors of ErrorCode. It is usually not used directly by users.	

⌋

8.15.3 Class: CoreErrorDomain

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

Upstream requirements: RS_AP_00130

⌈

Kind:	class
Header file:	#include "ara/core/core_error_domain.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	namespace ara::core
Symbol:	CoreErrorDomain
Base class:	ErrorDomain
Syntax:	`class CoreErrorDomain final :  public ErrorDomain {...};`
Unique ID:	As per `ara::core::CoreErrorDomain` in [SWS_CORE_90023]

▽

△

Description:	An error domain for errors originating from the `ara::core` Functional Cluster.

⌋

8.15.3.1 Public Member Types

8.15.3.1.1 Type Alias: Errc

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.15.3.1.2 Type Alias: Exception

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

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/core_error_domain.h"
Scope:	`class ara::core::CoreErrorDomain`
Symbol:	Exception
Syntax:	`using Exception = CoreException;`
Description:	Alias for the exception base class

⌋

8.15.3.2 Public Member Functions

8.15.3.2.1 Special Member Functions

8.15.3.2.1.1 Default Constructor

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.15.3.2.2 Member Functions

8.15.3.2.2.1 Message

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/core_error_domain.h"	
Scope:	`class ara::core::CoreErrorDomain`	
Syntax:	`const char * Message (ErrorDomain::CodeType errorCode) const noexcept override;`	
Parameters (in):	errorCode	the error code value
Return value:	const char *	the text message, never `nullptr`
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Translate an error code value into a text message	

⌋

8.15.3.2.2.2 Name

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/core_error_domain.h"	
Scope:	`class ara::core::CoreErrorDomain`	
Syntax:	`const char * Name () const noexcept override;`	
Return value:	const char *	As per `ara::core::CoreErrorDomain` in [SWS_CORE_90023]
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return the `ApApplicationErrorDomain`. `shortName` of this error domain	

⌋

8.15.3.2.2.3 ThrowAsException

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/core_error_domain.h"	
Scope:	`class ara::core::CoreErrorDomain`	
Syntax:	`void ThrowAsException (const ErrorCode &errorCode) const noexcept(false) override;`	
Parameters (in):	errorCode	the ErrorCode instance
Return value:	None	
Exceptions:	`CoreException`	an exception containing the given `ErrorCode`
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Throw the exception type corresponding to the given ErrorCode.	
Notes:	This function shall not participate in overload resolution when C++ exceptions are disabled in the compiler toolchain. As per [SWS_CORE_10304]	

⌋

8.15.4 Class: CoreException

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

Upstream requirements: RS_AP_00130

⌈

Kind:	class
Header file:	#include "ara/core/core_error_domain.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	namespace ara::core
Symbol:	CoreException
Base class:	Exception
Syntax:	`class CoreException : public Exception {...};`
Description:	Exception type thrown for `ara::core` errors.

⌋

8.15.4.1 Public Member Functions

8.15.4.1.1 Constructors

8.15.4.1.1.1 CoreException

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.16 Header: ara/core/future.h

8.16.1 Non-Member Types

8.16.1.1 Enumeration: FutureStatus

[SWS_CORE_00361] Definition of API enum ara::core::FutureStatus

Upstream requirements: RS_AP_00130

⌈

Kind:	enumeration	
Header file:	#include "ara/core/future.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	namespace ara::core	
Symbol:	FutureStatus	
Underlying type:	std::uint8_t	
Syntax:	`enum class FutureStatus : std::uint8_t {...};`	
Values:	kReady	the `shared state` is ready
	kTimeout	the `shared state` did not become ready before the specified timeout has passed
Description:	Specifies the status of a `ara::core::Future` as returned by: • `Future<T,E>::wait_for()` • `Future<void,E>::wait_for()` • `Future<T,E>::wait_until()` • `Future<void,E>::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.	
Implementation Defined Behavior:	The numerical values of the enum literals are implementation-defined	

⌋

8.16.2 Class: Future

[SWS_CORE_00321] Definition of API class ara::core::Future

Upstream requirements: RS_AP_00130

⌈

Kind:	class	
Header file:	#include "ara/core/future.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	namespace ara::core	
Symbol:	Future	
Syntax:	`template <typename T, typename E = ErrorCode>` `class Future final {...};`	
Template param:	typename T	the type of values
	typename E = ErrorCode	the type of errors

▽

△

Description:	Implementation of `std::future` from `[future.syn]` in [11]. The implementation shall flag the condition `E != ara::core::ErrorCode` as a compile error.

⌋

8.16.2.1 Public Member Functions

8.16.2.1.1 Special Member Functions

8.16.2.1.1.1 Copy Constructor

[SWS_CORE_00334] Definition of API function ara::core::Future::Future

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.16.2.1.1.2 Default Constructor

[SWS_CORE_00322] Definition of API function ara::core::Future::Future

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.16.2.1.1.3 Move Constructor

[SWS_CORE_00323] Definition of API function ara::core::Future::Future

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future`	
Syntax:	`Future (Future &&other) noexcept;`	
Parameters (in):	other	the other instance
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Move constructor from another instance This function shall behave the same as the corresponding `std::future` function.	

⌋

8.16.2.1.1.4 Copy Assignment Operator

[SWS_CORE_00335] Definition of API function ara::core::Future::operator=

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.16.2.1.1.5 Move Assignment Operator

[SWS_CORE_00325] Definition of API function ara::core::Future::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future`	
Syntax:	`Future & operator= (Future &&other) noexcept;`	
Parameters (in):	other	the other instance
Return value:	Future &	`*this`

▽

△

Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	Move assign from another instance. This function shall behave the same as the corresponding `std::future` function.

⌋

8.16.2.1.1.6 Destructor

[SWS_CORE_00333] Definition of API function ara::core::Future::~Future

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	`class ara::core::Future`
Syntax:	`~Future () noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	Destructor for Future objects. Abandons any `shared state`

⌋

8.16.2.1.2 Member Functions

8.16.2.1.2.1 GetResult

[SWS_CORE_00336] Definition of API function ara::core::Future::GetResult

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	`class ara::core::Future`
Syntax:	`Result< T, E > GetResult () noexcept(std::is_nothrow_move_constructible< T >::value &&std::is_nothrow_move_constructible< E >::value);`

▽

△

Return value:	Result< T, E >	a `ara::core::Result` as `std::move`(result from the `shared state`) with either a value or an error that has been put into the corresponding `Promise`. This can be because either: • – `Promise::SetError(E&&)` – `Promise::SetError(const E&)` – `Promise::SetResult(const Result<T, E>&)` – `Promise::SetResult(Result<T, E>&&)` • the `Promise` was broken, meaning the `shared state` was abandoned by the corresponding `Promise`. Then the error is `kBrokenPromise` as defined in [SWS_CORE_00400].
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Errors:	FutureErrc::kBroken Promise	rollback_semantics
		if the `Promise` was broken, meaning the `shared state` was abandoned by the corresponding `Promise`.
Violations:	`FutureInvalidVio-lation`	If the `Future` is invalid (`valid()` ==FALSE).
Description:	Get the result. Similar to `Future::get()`, this call blocks until the value or an error is available. However, this call will never throw an exception. As per `std::future`, after this function is called the `Future` is no longer valid (`valid()` ==FALSE).	

⌋

8.16.2.1.2.2 get

[SWS_CORE_00326] Definition of API function ara::core::Future::get

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future`	
Syntax:	`T get () noexcept(false);`	
Return value:	T	value of type T
Exceptions:	Any 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(E&&)` – `Promise::SetError(const E&)` – `Promise::SetResult(const Result<T, E>&)` – `Promise::SetResult(Result<T, E>&&)` • the `Promise` was broken, meaning the `shared state` was abandoned by the corresponding `Promise`. Then the error is `kBrokenPromise` as defined in [SWS_CORE_00400].	
	`FutureException`	if the `Future` is invalid. The contained ErrorCode is `kNoState`
Exception Safety:	not exception safe	

▽

△

Thread Safety:	not thread-safe
Description:	Get the value. This function shall behave the same as the corresponding `std::future` function. As per `std::future`, after this function is called the `Future` is no longer valid (`valid()` `==FALSE`).
Notes:	This function shall not participate in overload resolution when C++ exceptions are disabled in the compiler toolchain

⌋

8.16.2.1.2.3 is_ready

[SWS_CORE_00332] Definition of API function ara::core::Future::is_ready

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future`	
Syntax:	`bool is_ready () const noexcept;`	
Return value:	bool	• `TRUE`: the `Future` contains a value or an error • `FALSE`: the `Future` does **NOT** contain a value or an error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`FutureInvalidViolation`	If the `Future` is invalid (`valid()` `==FALSE`).
Description:	Return whether the asynchronous operation has finished. If this function returns `TRUE`, `get()`, `GetResult()` and the wait() calls are guaranteed not to block.	

⌋

8.16.2.1.2.4 then(F&&, ExecutorT&&)

[SWS_CORE_00337] Definition of API function ara::core::Future::then

Upstream requirements: RS_AP_00130

⌈

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

▽

△

Syntax:	`template <typename F, typename ExecutorT>` `auto then (F &&func, ExecutorT &&executor) noexcept -> Future< <see below> >;`	
Template param:	F	the type of the `func` argument
	ExecutorT	the type of the executor argument
Parameters (in):	func	a callable to register
	executor	the execution context in which to execute `func`
Return value:	Future< *<see below>* >	a new `Future` instance for the result of the continuation
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Errors:	This function does not specify any standardized errors.	
Violations:	`FutureInvalidViolation`	If the `Future` is invalid (`valid` ==FALSE)
	`FutureContinuationHasThrownViolation`	If the continuation throws an exception
Description:	Create a new Future with the content returned by func when passed the Result of this `Future`. func is called in the context of the provided execution context executor. valid() == false on the original `Future` object immediately after it returns. The Callable input argument "func" takes a Result<T,E> object as parameter. This will be the Result obtained via GetResult from the `Future` instance itself, on which .then() is being called. The Result is passed to func as an rvalue expression. The return type of then depends on the return type of func (aka continuation). Let U be the return type of the continuation (i.e. a type equivalent to std::invoke_result_t<std::decay_t<F>,Result<T,E>>). • If U is Future<T2,E2> for some types T2, E2, then the return type of then() is Future<T2,E2>. This is known as implicit Future unwrapping. • If U is Result<T2,E2> for some types T2, E2, then the return type of then() is Future<T2,E2>. This is known as implicit Result unwrapping. • Otherwise it is Future<U,E>. Continuations that are registered with Future::then() are registered in the `Future` F that Future::then() returns. No such continuation shall be executed after the destructor of the `Future` F returns. A continuation that is already being executed shall run to completion. Note: that means that the destructor of `Future` might block until the currently running continuation has been completed. The continuation shall not throw, except for the purpose of implementing a `Violation`. Note: Exceptions can be used within the continuation, however if they do not realize a `Violation`, they must not escape the continuation. Note: Users who need to propagate information from closures' exceptions should translate them to an error and return an `ara::core::Result` or `ara::core::Future` from the continuation with the error stored in it.	

⌋

8.16.2.1.2.5 then(F&&)

[SWS_CORE_00331] Definition of API function ara::core::Future::then

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future`	
Syntax:	`template <typename F>` `auto then (F &&func) noexcept -> Future< <see below> >;`	
Parameters (in):	func	a callable to register
Return value:	Future< *<see below>* >	a new `Future` instance for the result of the continuation
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Errors:	This function does not specify any standardized errors.	
Violations:	`FutureInvalidViolation`	If the `Future` is invalid (`valid` ==FALSE)
	`FutureContinuationHasThrownViolation`	If the continuation throws an exception
Description:	Create a new `Future` with the content returned by `func` when passed the Result of this `Future`. The callable `func` may be called in the context of this call or in the context of • `Promise::set_result(const T&)` • `Promise::set_result(T&&)` • `Promise::SetError(E&&)` • `Promise::SetError(const E&)` 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::invoke_result_t<std::decay_t<F>,Result<T,E>>). Return type deduction: • If `U` is Future<T2,E2> for some types T2, E2, then the return type of `then(...)` is Future<T2,E2>. This is known as implicit `Future` unwrapping. • If `U` is Result<T2,E2> for some types T2, E2, then the return type of `then(...)` is Future<T2,E2>. This is known as implicit Result unwrapping. • Otherwise it is Future<U,E> Continuations that are registered with `then(...)` are registered in the `Future` F that `then(...)` returns. No such continuation shall be executed after the destructor of the `Future` F returns. A continuation that is already being executed shall run to completion. Note: that means that the destructor of `Future` might block until the currently running continuation has been completed. The continuation shall not throw, except for the purpose of implementing a `Violation`.	

▽

△

Notes:	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. Exceptions can be used within the continuation, however if they do not realize a `Violation`, they must not escape the continuation.

⌋

8.16.2.1.2.6 valid

[SWS_CORE_00327] Definition of API function ara::core::Future::valid

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future`	
Syntax:	`bool valid () const noexcept;`	
Return value:	bool	• `TRUE`: the `Future` is usable • `FALSE`: the `Future` is **NOT** usable
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Checks if the `Future` is valid, i.e. if it has a `shared state`. This function shall behave the same as the corresponding `std::future` function.	

⌋

8.16.2.1.2.7 wait

[SWS_CORE_00328] Definition of API function ara::core::Future::wait

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future`	
Syntax:	`void wait () const noexcept;`	
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Violations:	`FutureInvalidViolation`	If the `Future` is invalid (`valid` ==`FALSE`)

▽

△

Description:	Wait for a value or an error to be available. This function shall behave the same as the corresponding `std::future` function

⌋

8.16.2.1.2.8 wait_for

[SWS_CORE_00329] Definition of API function ara::core::Future::wait_for

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future`	
Syntax:	`template <typename Rep, typename Period>` `FutureStatus wait_for (const std::chrono::duration< Rep, Period > &timeoutDuration) const noexcept;`	
Parameters (in):	timeoutDuration	maximal duration to wait for
Return value:	FutureStatus	Status that indicates whether the `timeoutDuration` expired or if a value is available
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Violations:	`FutureInvalidViolation`	If the `Future` is invalid (`valid ==FALSE`)
Description:	Wait for the `timeoutDuration` or until a value or an error is available. This function shall behave the same as the corresponding `std::future` function.	

⌋

8.16.2.1.2.9 wait_until

[SWS_CORE_00330] Definition of API function ara::core::Future::wait_until

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future`	
Syntax:	`template <typename Clock, typename Duration>` `FutureStatus wait_until (const std::chrono::time_point< Clock, Duration > &deadline) const noexcept;`	
Parameters (in):	deadline	latest point in time to wait
Return value:	FutureStatus	status that indicates whether the `deadline` was reached or if a value is available

▽

△

Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Violations:	`FutureInvalidViolation`	If the `Future` is invalid (`valid` ==FALSE)
Description:	Wait until the `deadline` or until a value or an error is available. This function shall behave the same as the corresponding `std::future` function.	

⌋

8.16.3 Class: Future<void, E>

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

Upstream requirements: RS_AP_00130

⌈

Kind:	class	
Header file:	#include "ara/core/future.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	namespace ara::core	
Symbol:	Future< void, E >	
Syntax:	`template <typename E>` `class Future< void, E > final {...};`	
Template param:	typename E	the type of error
Description:	Specialization of class `Future` for "void" values. The implementation shall flag the condition: (`E != ara::core::ErrorCode`) as a compile error.	

⌋

8.16.3.1 Public Member Functions

8.16.3.1.1 Special Member Functions

8.16.3.1.1.1 Copy Constructor

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

Upstream requirements: RS_AP_00130

⌈

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

▽

△

Description:	Copy constructor shall be disabled

⌋

8.16.3.1.1.2 Default Constructor

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.16.3.1.1.3 Move Constructor

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future< void, E >`	
Syntax:	`Future (Future &&other) noexcept;`	
Parameters (in):	other	the other instance
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Move constructor from another instance This function shall behave the same as the corresponding `std::future` function.	

⌋

8.16.3.1.1.4 Copy Assignment Operator

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.16.3.1.1.5 Move Assignment Operator

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future< void, E >`	
Syntax:	`Future & operator= (Future &&other) noexcept;`	
Parameters (in):	other	the other instance
Return value:	Future &	`*this`
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Move assign from another instance. This function shall behave the same as the corresponding `std::future` function.	

⌋

8.16.3.1.1.6 Destructor

[SWS_CORE_06233] Definition of API function ara::core::Future< void, E >::~Future

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.16.3.1.2 Member Functions

8.16.3.1.2.1 GetResult

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future< void, E >`	
Syntax:	`Result< void, E > GetResult () noexcept(std::is_nothrow_move_constructible< E >::value);`	
Return value:	Result< void, E >	a Result as std::move(result from the `shared state`) with either a value or an error that has been put into the corresponding Promise. This can be because either: • explicit setting of the Error via Promise::SetError / Promise::Set Result or • the Promise was broken, meaning the `shared state` was abandoned by the corresponding Promise. Then the error is `kBrokenPromise` as defined in [SWS_CORE_00400]
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Errors:	FutureErrc::kBroken Promise	rollback_semantics
		if the Promise was broken, meaning the `shared state` was abandoned by the corresponding Promise.
Violations:	`FutureInvalidViolation`	If the Future is invalid (`valid` ==FALSE)

▽

△

Description:	Get the result. Similar to `Future::get()`, this call blocks until the value or an error is available. However, this call will never throw an exception. Part of the behavior taken from `Future::get()` is that after a call to this function the Future is not valid anymore (`valid()` == false).

⌋

8.16.3.1.2.2 get

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future< void, E >`	
Syntax:	`void get () noexcept(false);`	
Return value:	None	
Exceptions:	Any 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(E&&)` `Promise::SetError(const E&)` / `Promise::SetResult(const Result<T, E>&)` `Promise::SetResult(Result<T, E>&&)` • the Promise was broken, meaning the `shared state` was abandoned by the corresponding Promise. Then the error is `kBrokenPromise` as defined in [SWS_CORE_00400].	
	`FutureException`	in case the Future is invalid. The contained ErrorCode is `kNoState`
Exception Safety:	not exception safe	
Thread Safety:	not thread-safe	
Description:	Returns nothing. This function shall behave the same as the corresponding `std::future` function. Part of the behavior taken from the `std::future` is that after a call to this function the Future is not valid anymore (`valid()` == false).	
Notes:	This function shall not participate in overload resolution when C++ exceptions are disabled in the compiler toolchain	

⌋

8.16.3.1.2.3 is_ready

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future< void, E >`	
Syntax:	`bool is_ready () const noexcept;`	
Return value:	bool	true if the Future contains a value or an error, false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`FutureInvalidViolation`	If the Future is invalid (`valid` returns false)
Description:	Return whether the asynchronous operation has finished. If this function returns true, get(), GetResult() and the wait calls are guaranteed not to block.	

⌋

8.16.3.1.2.4 then(F&&, ExecutorT&&)

[SWS_CORE_06237] Definition of API function ara::core::Future< void, E >::then

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future< void, E >`	
Syntax:	`template <typename F, typename ExecutorT>` `auto then (F &&func, ExecutorT &&executor) noexcept -> Future< <see below> >;`	
Template param:	F	the type of the func argument
	ExecutorT	the type of the executor argument
Parameters (in):	func	a callable to register
	executor	the execution context in which to execute the Callable func
Return value:	Future< *<see below>* >	a new Future instance for the result of the continuation
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Errors:	This function does not specify any standardized errors.	
Violations:	`FutureInvalidViolation`	If the Future is invalid (`valid` returns false)
	`FutureContinuationHasThrownViolation`	If the continuation throws an exception.

▽

△

Description:	Create a new Future with the content returned by func when passed the Result of this Future. func is called in the context of the provided execution context executor. valid() == false on the original future object immediately after it returns. The Callable input argument "func" takes a Result<void,E> object as parameter. This will be the Result obtained via GetResult from the Future instance itself, on which .then() is being called. The Result is passed to func as an rvalue expression. The return type of then depends on the return type of func (aka continuation). Let U be the return type of the continuation (i.e. a type equivalent to std::invoke_result_t<std::decay_t<F>,Result<void,E>>). • If U is Future<T2,E2> for some types T2, E2, then the return type of then() is Future<T2,E2>. This is known as implicit Future unwrapping. • If U is Result<T2,E2> for some types T2, E2, then the return type of then() is Future<T2,E2>. This is known as implicit Result unwrapping. • Otherwise it is Future<U,E>. Continuations that are registered with Future::then() are registered in the Future F that Future::then() returns. No such continuation shall be executed after the destructor of the Future F returns. A continuation that is already being executed shall run to completion. Note: that means that the destructor of Future might block until the currently running continuation has been completed. The continuation shall not throw, except for the purpose of implementing a `Violation`. Note: Exceptions can be used within the continuation, however if they do not realize a `Violation`, they must not escape the continuation. Note: Users who need to propagate information from closures' exceptions should translate them to an error and return an `ara::core::Result` or `ara::core::Future` from the continuation with the error stored in it.

⌋

8.16.3.1.2.5 then(F&&)

[SWS_CORE_06231] Definition of API function ara::core::Future< void, E >::then

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future< void, E >`	
Syntax:	`template <typename F>` `auto then (F &&func) noexcept -> Future< <see below> >;`	
Parameters (in):	func	a callable to register
Return value:	Future< *<see below>* >	a new Future instance for the result of the continuation
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Errors:	This function does not specify any standardized errors.	
Violations:	`FutureInvalidViolation`	If the Future is invalid (`valid` returns false)
	`FutureContinuationHasThrownViolation`	If the continuation throws an exception

▽

△

Description:	Create a new Future with the content returned by func when passed the Result of this Future. func may be called in the context of this call or in the context of Promise::set_value() or Promise::SetError() or somewhere else. valid() == false on the original future object immediately after it returns. The Callable input argument "func" takes a Result<void,E> object as parameter. This will be the Result obtained via GetResult from the Future instance itself, on which .then() is being called. The Result is passed to func as an rvalue expression. The return type of then depends on the return type of func (aka continuation). Let U be the return type of the continuation (i.e. a type equivalent to std::invoke_result_t<std::decay_t<F>,Result<void,E>>). • If U is Future<T2,E2> for some types T2, E2, then the return type of then() is Future<T2,E2>. This is known as implicit Future unwrapping. • If U is Result<T2,E2> for some types T2, E2, then the return type of then() is Future<T2,E2>. This is known as implicit Result unwrapping. • Otherwise it is Future<U,E>. Continuations that are registered with Future::then() are registered in the Future F that Future::then() returns. No such continuation shall be executed after the destructor of the Future F returns. A continuation that is already being executed shall run to completion. Note: that means that the destructor of Future might block until the currently running continuation has been completed. The continuation shall not throw, except for the purpose of implementing a `Violation`. Note: Exceptions can be used within the continuation, however if they do not realize a `Violation`, they must not escape the continuation. Note: Users who need to propagate information from closures' exceptions should translate them to an error and return an `ara::core::Result` or `ara::core::Future` from the continuation with the error stored in it.

⌋

8.16.3.1.2.6 valid

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future< void, E >`	
Syntax:	`bool valid () const noexcept;`	
Return value:	bool	• `TRUE`: the `Future` is usable • `FALSE`: the `Future` is **NOT** usable
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Checks if the `Future` is valid, i.e. if it has a `shared state`. This function shall behave the same as the corresponding `std::future` function.	

⌋

8.16.3.1.2.7 wait

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future< void, E >`	
Syntax:	`void wait () const noexcept;`	
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Violations:	`FutureInvalidViolation`	If the Future is invalid (`valid` returns false)
Description:	Wait for a value or an error to be available. This function shall behave the same as the corresponding std::future function	

⌋

8.16.3.1.2.8 wait_for

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future< void, E >`	
Syntax:	`template <typename Rep, typename Period>` `FutureStatus wait_for (const std::chrono::duration< Rep, Period > &timeoutDuration) const noexcept;`	
Parameters (in):	timeoutDuration	maximal duration to wait for
Return value:	FutureStatus	status that indicates whether the timeout hit or if a value is available
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Violations:	`FutureInvalidViolation`	If the Future is invalid (`valid` returns false)
Description:	Wait for the given period, or until a value or an error is available. This function shall behave the same as the corresponding std::future function	

⌋

8.16.3.1.2.9 wait_until

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Future< void, E >`	
Syntax:	`template <typename Clock, typename Duration>` `FutureStatus wait_until (const std::chrono::time_point< Clock, Duration > &deadline) const noexcept;`	
Parameters (in):	deadline	latest point in time to wait
Return value:	FutureStatus	status that indicates whether the time was reached or if a value is available
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Violations:	`FutureInvalidViolation`	If the Future is invalid (`valid` returns false)
Description:	Wait until the given time, or until a value or an error is available. This function shall behave the same as the corresponding `std::future` function.	

⌋

8.16.4 Class: Promise

[SWS_CORE_00340] Definition of API class ara::core::Promise

Upstream requirements: RS_AP_00130

⌈

Kind:	class	
Header file:	#include "ara/core/future.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	namespace ara::core	
Symbol:	Promise	
Syntax:	`template <typename T, typename E = ErrorCode>` `class Promise final {...};`	
Template param:	typename T	the type of value
	typename E = ErrorCode	the type of error
Description:	ara::core specific variant of `std::promise` class The implementation shall flag the condition `E!=ara::core::ErrorCode` as a compile error.	

⌋

8.16.4.1 Public Member Functions

8.16.4.1.1 Special Member Functions

8.16.4.1.1.1 Copy Constructor

[SWS_CORE_00350] Definition of API function ara::core::Promise::Promise

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.16.4.1.1.2 Default Constructor

[SWS_CORE_00341] Definition of API function ara::core::Promise::Promise

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	`class ara::core::Promise`
Syntax:	`Promise () noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	Default constructor This function shall behave the same as the corresponding `std::promise` function.

⌋

8.16.4.1.1.3 Move Constructor

[SWS_CORE_00342] Definition of API function ara::core::Promise::Promise

Upstream requirements: RS_AP_00130

⌈

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

▽

△

Syntax:	`Promise (Promise &&other) noexcept;`	
Parameters (in):	other	the other instance
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Move constructor This function shall behave the same as the corresponding `std::promise` function.	

⌋

8.16.4.1.1.4 Copy Assignment Operator

[SWS_CORE_00351] Definition of API function ara::core::Promise::operator=

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.16.4.1.1.5 Move Assignment Operator

[SWS_CORE_00343] Definition of API function ara::core::Promise::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise`	
Syntax:	`Promise & operator= (Promise &&other) noexcept;`	
Parameters (in):	other	the other instance
Return value:	Promise &	`*this`
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Move assignment Abandons any `shared state` and then as if Promise(std::move(other)).swap(*this).	

⌋

8.16.4.1.1.6 Destructor

[SWS_CORE_00349] Definition of API function ara::core::Promise::~Promise

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	`class ara::core::Promise`
Syntax:	`~Promise () noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	Destructor for Promise objects Abandons any `shared state`.

⌋

8.16.4.1.2 Member Functions

8.16.4.1.2.1 SetError(E&&)

[SWS_CORE_00353] Definition of API function ara::core::Promise::SetError

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise`	
Syntax:	`void SetError (E &&error) noexcept(std::is_nothrow_move_constructible< E >::value);`	
Parameters (in):	error	the error to store
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Violations:	`IsoStdFutureErrcPromiseAlreadySatisfiedViolation`	If the `shared state` already has a stored value or error.
	`PromiseNoSharedStateToSetViolation`	If `*this` has no `shared state`.
Description:	Move an error into the `shared state` and make the `shared state` ready.	

⌋

8.16.4.1.2.2 SetError(const E&)

[SWS_CORE_00354] Definition of API function ara::core::Promise::SetError

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise`	
Syntax:	`void SetError (const E &error) noexcept(std::is_nothrow_copy_constructible< E >::value);`	
Parameters (in):	error	the error to store
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Violations:	`IsStdFutureErrcPromiseAlreadySatisfiedViolation`	If the `shared state` already has a stored value or error.
	`PromiseNoSharedStateToSetViolation`	If `*this` has no `shared state`.
Description:	Copy an error into the `shared state` and make the `shared state` ready.	

⌋

8.16.4.1.2.3 SetResult(Result<T, E>&&)

[SWS_CORE_00356] Definition of API function ara::core::Promise::SetResult

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise`	
Syntax:	`void SetResult (Result< T, E > &&result) noexcept(std::is_nothrow_move_constructible< T >::value &&std::is_nothrow_move_constructible< E >::value);`	
Parameters (in):	result	the result to store
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Violations:	`IsStdFutureErrcPromiseAlreadySatisfiedViolation`	If the `shared state` already has a stored value or error.
	`PromiseNoSharedStateToSetViolation`	If `*this` has no `shared state`.

▽

△

Description:	Move a Result into the `shared state` and make the `shared state` ready.

⌋

8.16.4.1.2.4 SetResult(const Result<T, E>&)

[SWS_CORE_00355] Definition of API function ara::core::Promise::SetResult

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise`	
Syntax:	`void SetResult (const Result< T, E > &result) noexcept(std::is_nothrow_copy_constructible< T >::value &&std::is_nothrow_copy_constructible< E >::value);`	
Parameters (in):	result	the result to store
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Violations:	`IsoStdFutureErrcPromiseAlreadySatisfiedViolation`	If the `shared state` already has a stored value or error.
	`PromiseNoSharedStateToSetViolation`	If `*this` has no `shared state`.
Description:	Copy a Result into the `shared state` and make the `shared state` ready.	

⌋

8.16.4.1.2.5 get_future

[SWS_CORE_00344] Definition of API function ara::core::Promise::get_future

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise`	
Syntax:	`Future< T, E > get_future () noexcept;`	
Return value:	Future< T, E >	a Future with the same `shared state` as `*this`
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Errors:	This function does not specify any standardized errors.	

▽

△

Violations:	`IsoStdFutureErrc-FutureAlreadyRetrievedViolation`	If the function is called more than once on the same `shared state`.
	`IsoStdFutureErrc-NoStateViolation`	If `*this` has no `shared state`.
Description:	Return an associated Future with the same `shared state` as `*this` The returned Future is set as soon as this Promise receives the result, value, or an error. This fuction must only be called once as it is not allowed to have multiple Futures per Promise.	

⌋

8.16.4.1.2.6 set_value(const T&)

[SWS_CORE_00345] Definition of API function ara::core::Promise::set_value

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise`	
Syntax:	`void set_value (const T &value) noexcept(std::is_nothrow_copy_constructible< T >::value);`	
Parameters (in):	value	the value to store
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Violations:	`IsoStdFutureErrcPromiseAlreadySatisfiedViolation`	If the `shared state` already has a stored value or error.
	`PromiseNoSharedStateToSetViolation`	If `*this` has no `shared state`.
Description:	Copy a value into the `shared state` and make the `shared state` ready.	

⌋

8.16.4.1.2.7 set_value(T&&)

[SWS_CORE_00346] Definition of API function ara::core::Promise::set_value

Upstream requirements: RS_AP_00130

⌈

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

▽

△

Syntax:	`void set_value (T &&value) noexcept(std::is_nothrow_move_constructible< T >::value);`	
Parameters (in):	value	the value to store
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Violations:	`IsoStdFutureErrcPromiseAlreadySatisfiedViolation`	If the `shared state` already has a stored value or error.
	`PromiseNoSharedStateToSetViolation`	If `*this` has no `shared state`.
Description:	Move a value into the `shared state` and make the `shared state` ready.	

⌋

8.16.4.1.2.8 swap

[SWS_CORE_00352] Definition of API function ara::core::Promise::swap

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise`	
Syntax:	`void swap (Promise &other) noexcept;`	
Parameters (inout):	other	the other instance
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Swap the contents of this instance with another one's. This function shall behave the same as the corresponding `std::promise` function.	

⌋

8.16.5 Class: Promise<void, E>

[SWS_CORE_06340] Definition of API class ara::core::Promise< void, E >

Upstream requirements: RS_AP_00130

⌈

Kind:	class
Header file:	#include "ara/core/future.h"
Forwarding header file:	#include "ara/core/core_fwd.h"

▽

△

Scope:	namespace ara::core	
Symbol:	Promise< void, E >	
Syntax:	`template <typename E>` `class Promise< void, E > final {...};`	
Template param:	typename E	the type of error
Description:	Specialization of class Promise for `void` values The implementation shall flag the condition `E != *ara::core::ErrorCode` as a compile error.	

⌋

8.16.5.1 Public Member Functions

8.16.5.1.1 Special Member Functions

8.16.5.1.1.1 Move Constructor

[SWS_CORE_06342] Definition of API function ara::core::Promise< void, E >::Promise

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise< void, E >`	
Syntax:	`Promise (Promise &&other) noexcept;`	
Parameters (in):	other	the other instance
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Move constructor This function shall behave the same as the corresponding `std::promise` function.	

⌋

8.16.5.1.1.2 Default Constructor

[SWS_CORE_06341] Definition of API function ara::core::Promise< void, E >::Promise

Upstream requirements: RS_AP_00130

⌈

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

▽

△

Syntax:	`Promise () noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	Default constructor This function shall behave the same as the corresponding `std::promise` function.

⌋

8.16.5.1.1.3 Copy Constructor

[SWS_CORE_06350] Definition of API function ara::core::Promise< void, E >::Promise

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.16.5.1.1.4 Copy Assignment Operator

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.16.5.1.1.5 Move Assignment Operator

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise< void, E >`	
Syntax:	`Promise & operator= (Promise &&other) noexcept;`	
Parameters (in):	other	the other instance
Return value:	Promise &	`*this`
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Move assignment Abandons any `shared state` and then as if Promise(std::move(other)).swap(*this).	

⌋

8.16.5.1.1.6 Destructor

[SWS_CORE_06349] Definition of API function ara::core::Promise< void, E >::~Promise

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/future.h"
Scope:	`class ara::core::Promise< void, E >`
Syntax:	`~Promise () noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	Destructor for Promise objects Abandons any `shared state`.

⌋

8.16.5.1.2 Member Functions

8.16.5.1.2.1 SetError(E&&)

[SWS_CORE_06353] Definition of API function ara::core::Promise< void, E >::SetError

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise< void, E >`	
Syntax:	`void SetError (E &&error) noexcept(std::is_nothrow_move_constructible< E >::value);`	
Parameters (in):	error	the error to store
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Violations:	`IsoStdFutureErrcPromiseAlreadySatisfiedViolation`	If the `shared state` already has a stored value or error.
	`PromiseNoSharedStateToSetViolation`	If `*this` has no `shared state`.
Description:	Move an error into the `shared state` and make the `shared state` ready.	

⌋

8.16.5.1.2.2 SetError(const E&)

[SWS_CORE_06354] Definition of API function ara::core::Promise< void, E >::SetError

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise< void, E >`	
Syntax:	`void SetError (const E &error) noexcept(std::is_nothrow_copy_constructible< E >::value);`	
Parameters (in):	error	the error to store
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	

▽

△

Violations:	`IsoStdFutureErrcPromiseAlreadySatisfiedViolation`	If the `shared state` already has a stored value or error.
	`PromiseNoSharedStateToSetViolation`	If `*this` has no `shared state`.
Description:	Copy an error into the `shared state` and make the `shared state` ready.	

⌋

8.16.5.1.2.3 SetResult(Result<void, E>&&)

[SWS_CORE_06356] Definition of API function ara::core::Promise< void, E >::SetResult

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise< void, E >`	
Syntax:	`void SetResult (Result< void, E > &&result) noexcept(std::is_nothrow_move_constructible< E >::value);`	
Parameters (in):	result	the result to store
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Violations:	`IsoStdFutureErrcPromiseAlreadySatisfiedViolation`	If the `shared state` already has a stored value or error.
	`PromiseNoSharedStateToSetViolation`	If `*this` has no `shared state`.
Description:	Move a Result into the `shared state` and make the `shared state` ready.	

⌋

8.16.5.1.2.4 SetResult(const Result<void, E>&)

[SWS_CORE_06355] Definition of API function ara::core::Promise< void, E >::SetResult

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise< void, E >`	
Syntax:	`void SetResult (const Result< void, E > &result) noexcept(std::is_nothrow_copy_constructible< E >::value);`	
Parameters (in):	result	the result to store
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Violations:	`IsoStdFutureErrcPromiseAlreadySatisfiedViolation`	If the `shared state` already has a stored value or error.
	`PromiseNoSharedStateToSetViolation`	If `*this` has no `shared state`.
Description:	Copy a Result into the `shared state` and make the `shared state` ready.	

⌋

8.16.5.1.2.5 get_future

[SWS_CORE_06344] Definition of API function ara::core::Promise< void, E >::get_future

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise< void, E >`	
Syntax:	`Future< void, E > get_future () noexcept;`	
Return value:	Future< void, E >	a Future with the same `shared state` as `*this`
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Errors:	This function does not specify any standardized errors.	
Violations:	`IsoStdFutureErrcFutureAlreadyRetrievedViolation`	If the function is called more than once on the same `shared state`.
	`IsoStdFutureErrcNoStateViolation`	If `*this` has no `shared state`.

▽

△

Description:	Return an associated Future with the same `shared state` as `*this` The returned Future is set as soon as this Promise receives the result, value, or an error. This fuction must only be called once as it is not allowed to have multiple Futures per Promise.

⌋

8.16.5.1.2.6 set_value()

[SWS_CORE_06345] Definition of API function ara::core::Promise< void, E >::set_value

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise< void, E >`	
Syntax:	`void set_value () noexcept;`	
Return value:	None	
Exception Safety:	exception safe	
Violations:	`IsoStdFutureErrcPromiseAlreadySatisfiedViolation`	If the `shared state` already has a stored value or error.
	`PromiseNoSharedStateToSetViolation`	If `*this` has no `shared state`.
Description:	Set the `shared state` value and make the `shared state` ready.	

⌋

8.16.5.1.2.7 swap

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future.h"	
Scope:	`class ara::core::Promise< void, E >`	
Syntax:	`void swap (Promise &other) noexcept;`	
Parameters (inout):	other	the other instance
Return value:	None	
Exception Safety:	exception safe	

▽

△

Thread Safety:	not thread-safe
Description:	Swap the contents of this instance with another one's. This function shall behave the same as the corresponding `std::promise` function.

⌋

8.17 Header: ara/core/future_error_domain.h

8.17.1 Non-Member Types

8.17.1.1 Enumeration: FutureErrc

[SWS_CORE_00400] Definition of API enum ara::core::FutureErrc

Upstream requirements: RS_AP_00130

⌈

Kind:	enumeration	
Header file:	#include "ara/core/future_error_domain.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	namespace ara::core	
Symbol:	FutureErrc	
Underlying type:	ara::core::ErrorDomain::CodeType	
Syntax:	`enum class FutureErrc : ara::core::ErrorDomain::CodeType {...};`	
Values:	kBrokenPromise	= 101
		The asynchronous task abandoned its `shared state`
	kNoState	= 104
		Attempt to access or without an associated `shared state`
Description:	Specifies the errors that can occur upon calling `Future::get()` or `Future::GetResult()`.	

⌋

8.17.2 Non-Member Functions

8.17.2.1 Other

8.17.2.1.1 GetFutureErrorDomain

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.17.2.1.2 MakeErrorCode

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.17.3 Class: FutureErrorDomain

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

Upstream requirements: RS_AP_00130

⌈

Kind:	class
Header file:	#include "ara/core/future_error_domain.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	namespace ara::core
Symbol:	FutureErrorDomain
Base class:	ErrorDomain
Syntax:	`class FutureErrorDomain final : public ErrorDomain {...};`
Unique ID:	As per `ara::core::FutureErrorDomain` in [SWS_CORE_90023]
Description:	Error domain for errors originating from `ara::core::Future` and `ara::core::Promise`

⌋

8.17.3.1 Public Member Types

8.17.3.1.1 Type Alias: Errc

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.17.3.1.2 Type Alias: Exception

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

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/future_error_domain.h"
Scope:	`class ara::core::FutureErrorDomain`
Symbol:	Exception
Syntax:	`using Exception = FutureException;`
Description:	Alias for the exception base class

⌋

8.17.3.2 Public Member Functions

8.17.3.2.1 Special Member Functions

8.17.3.2.1.1 Default Constructor

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.17.3.2.2 Member Functions

8.17.3.2.2.1 Message

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future_error_domain.h"	
Scope:	`class ara::core::FutureErrorDomain`	
Syntax:	`const char * Message (ErrorDomain::CodeType errorCode) const noexcept override;`	
Parameters (in):	errorCode	the error code value
Return value:	const char *	the text message, never `nullptr`
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Translate an error code value into a text message	

⌋

8.17.3.2.2.2 Name

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future_error_domain.h"	
Scope:	`class ara::core::FutureErrorDomain`	
Syntax:	`const char * Name () const noexcept override;`	
Return value:	const char *	As per `ara::core::FutureErrorDomain` in [SWS_CORE_90023]
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return the name of this error domain	

⌋

8.17.3.2.2.3 ThrowAsException

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/future_error_domain.h"	
Scope:	`class ara::core::FutureErrorDomain`	
Syntax:	`void ThrowAsException (const ErrorCode &errorCode) const noexcept(false) override;`	
Parameters (in):	errorCode	the ErrorCode instance
Return value:	None	
Exceptions:	`FutureException`	an exception containing the given `ErrorCode`
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Throw the exception type corresponding to the given `ErrorCode`	
Notes:	This function shall not participate in overload resolution when C++ exceptions are disabled in the compiler toolchain.	

⌋

8.17.4 Class: FutureException

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

Upstream requirements: RS_AP_00130

⌈

Kind:	class
Header file:	#include "ara/core/future_error_domain.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	namespace ara::core
Symbol:	FutureException
Base class:	Exception
Syntax:	`class FutureException : public Exception {...};`
Description:	Exception type thrown by `ara::core::Future` and `ara::core::Promise`

⌋

8.17.4.1 Public Member Functions

8.17.4.1.1 Constructors

8.17.4.1.1.1 FutureException

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.18 Header: ara/core/steady_clock.h

8.18.1 Class: SteadyClock

[SWS_CORE_06401] Definition of API class ara::core::SteadyClock

Upstream requirements: RS_AP_00130

⌈

Kind:	class
Header file:	#include "ara/core/steady_clock.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	namespace ara::core
Symbol:	SteadyClock
Syntax:	`class SteadyClock final {...};`
Description:	This clock represents a monotonic clock. The time points of this clock cannot decrease as physical time moves forward and the time between ticks of this clock is constant.

⌋

8.18.1.1 Public Member Types

8.18.1.1.1 Type Alias: duration

[SWS_CORE_06411] Definition of API type ara::core::SteadyClock::duration

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/steady_clock.h"
Scope:	`class ara::core::SteadyClock`
Symbol:	duration
Syntax:	`using duration = std::chrono::duration<rep, period>;`
Description:	std::chrono::duration<rep, period>

⌋

8.18.1.1.2 Type Alias: period

[SWS_CORE_06413] Definition of API type ara::core::SteadyClock::period

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/steady_clock.h"
Scope:	`class ara::core::SteadyClock`
Symbol:	period
Syntax:	`using period = std::nano;`
Description:	A std::ratio type representing the tick period of the clock, in seconds .

⌋

8.18.1.1.3 Type Alias: rep

[SWS_CORE_06412] Definition of API type ara::core::SteadyClock::rep

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/steady_clock.h"
Scope:	`class ara::core::SteadyClock`
Symbol:	rep
Syntax:	`using rep = std::int64_t;`

▽

△

Description:	An arithmetic type representing the number of ticks in the clock's duration .

⌋

8.18.1.1.4 Type Alias: time_point

[SWS_CORE_06414] Definition of API type ara::core::SteadyClock::time_point

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "ara/core/steady_clock.h"
Scope:	`class ara::core::SteadyClock`
Symbol:	time_point
Syntax:	`using time_point = std::chrono::time_point<SteadyClock, duration>;`
Description:	std::chrono::time_point<ara::core::SteadyClock>

⌋

8.18.1.2 Public Member Variables

8.18.1.2.1 is_steady

[SWS_CORE_06431] Definition of API variable ara::core::SteadyClock::is_steady

Upstream requirements: RS_AP_00130

⌈

Kind:	variable
Header file:	#include "ara/core/steady_clock.h"
Scope:	`class ara::core::SteadyClock`
Symbol:	is_steady
Type:	bool
Syntax:	`static constexpr bool is_steady = true;`
Description:	steady clock flag, always true

⌋

8.18.1.3 Public Member Functions

8.18.1.3.1 Member Functions

8.18.1.3.1.1 now

[SWS_CORE_06432] Definition of API function ara::core::SteadyClock::now

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/steady_clock.h"	
Scope:	`class ara::core::SteadyClock`	
Syntax:	`static time_point now () noexcept;`	
Return value:	time_point	a time_point
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a time_point representing the current value of the clock.	

⌋

8.19 Header: ara/core/utility.h

8.19.1 Non-Member Types

8.19.1.1 Type Alias: Byte

[SWS_CORE_04200] Definition of API type ara::core::Byte

Upstream requirements: RS_AP_00130

⌈

Kind:	type alias	
Header file:	#include "ara/core/utility.h"	
Scope:	namespace ara::core	
Symbol:	Byte	
Syntax:	`using Byte = {<byte-type>};`	
Description:	A non-integral binary type. The requirements on this type are stated in **"See also:"**	
Implementation Defined Behavior:	`{<byte-type>}` shall be an implementation-defined type	
Descriptors:	`{<byte-type>}`	implementation-defined
See also:	[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]	

⌋

8.19.2 Global Variables

8.19.2.1 in_place

[SWS_CORE_04013] Definition of API variable ara::core::in_place

Upstream requirements: RS_AP_00130

⌈

Kind:	variable
Header file:	#include "ara/core/utility.h"
Scope:	namespace ara::core
Symbol:	in_place
Type:	in_place_t
Syntax:	`constexpr in_place_t in_place;`
Description:	The singleton instance of `ara::core::in_place_t`

⌋

8.19.2.2 in_place_index

[SWS_CORE_04033] Definition of API variable ara::core::in_place_index

Upstream requirements: RS_AP_00130

⌈

Kind:	variable	
Header file:	#include "ara/core/utility.h"	
Scope:	namespace ara::core	
Symbol:	in_place_index	
Type:	in_place_index_t< Index >	
Syntax:	`template <std::size_t Index>` `constexpr in_place_index_t<Index> in_place_index {};`	
Template param:	std::size_t Index	the index to address
Description:	The singleton instances (one for each `I` of `ara::core::in_place_index_t`	

⌋

8.19.2.3 in_place_type

[SWS_CORE_04023] Definition of API variable ara::core::in_place_type

Upstream requirements: RS_AP_00130

⌈

Kind:	variable	
Header file:	#include "ara/core/utility.h"	
Scope:	namespace ara::core	
Symbol:	in_place_type	
Type:	in_place_type_t< T >	
Syntax:	`template <typename T>` `constexpr in_place_type_t<T> in_place_type;`	
Template param:	typename T	the type to address
Description:	The singleton instances (one for each `T`) of `ara::core::in_place_type_t`	

⌋

8.19.3 Non-Member Functions

8.19.3.1 Other

8.19.3.1.1 data(T()&)

[SWS_CORE_04112] Definition of API function ara::core::data

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/utility.h"	
Scope:	namespace ara::core	
Syntax:	`template <typename T, std::size_t N>` `constexpr T * data (T(&array)[N]) noexcept;`	
Template param:	T	the type of array elements
	N	the number of elements in the array
Parameters (in):	array	reference to a raw array
Return value:	T *	a pointer to the first element of the array
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a pointer to the block of memory that contains the elements of a raw array.	

⌋

8.19.3.1.2 data(std::initializer_list<E>)

[SWS_CORE_04113] Definition of API function ara::core::data

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/utility.h"	
Scope:	namespace ara::core	
Syntax:	`template <typename E>` `constexpr const E * data (std::initializer_list< E > il) noexcept;`	
Template param:	E	the type of elements in the `std::initializer_list`
Parameters (in):	il	the `std::initializer_list`
Return value:	const E *	a pointer to the first element of the `std::initializer_list`
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return a pointer to the block of memory that contains the elements of a `std::initializer_list`	

⌋

8.19.3.1.3 data(Container&)

[SWS_CORE_04110] Definition of API function ara::core::data

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/utility.h"	
Scope:	namespace ara::core	
Syntax:	`template <typename Container>` `constexpr auto data (Container &c) noexcept(noexcept(c.data())) -> decltype(c.data());`	
Template param:	Container	a type with a `data()` method
Parameters (in):	c	an instance of Container
Return value:	decltype(c.data())	a pointer to the first element of the container
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return a pointer to the block of memory that contains the elements of a container.	

⌋

8.19.3.1.4 data(const Container&)

[SWS_CORE_04111] Definition of API function ara::core::data

Upstream requirements: RS_AP_00130

⌈

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

Syntax:	`template <typename Container>`
	`constexpr auto data (const Container &c) noexcept(noexcept(c.data()))`
	`-> decltype(c.data());`	

Template param:	Container	a type with a `data()` method
Parameters (in):	c	an instance of Container
Return value:	decltype(c.data())	a pointer to the first element of the container
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return a const_pointer to the block of memory that contains the elements of a container.	

⌋

8.19.3.1.5 empty(std::initializer_list<E>)

[SWS_CORE_04132] Definition of API function ara::core::empty

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/utility.h"	
Scope:	namespace ara::core	
Syntax:	`template <typename E>` `constexpr bool empty (std::initializer_list< E > il) noexcept;`	
Template param:	E	the type of elements in the `std::initializer_list`
Parameters (in):	il	the `std::initializer_list`
Return value:	bool	true if the `std::initializer_list` is empty, false otherwise
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return whether the given `std::initializer_list` is empty.	

⌋

8.19.3.1.6 empty(const T()&)

[SWS_CORE_04131] Definition of API function ara::core::empty

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/utility.h"	
Scope:	namespace ara::core	
Syntax:	`template <typename T, std::size_t N>` `constexpr bool empty (const T(&array)[N]) noexcept;`	
Template param:	T	the type of array elements
	N	the number of elements in the array
Parameters (in):	array	the raw array
Return value:	bool	false
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return whether the given raw array is empty. As raw arrays cannot have zero elements in C++, this function always returns false.	

⌋

8.19.3.1.7 empty(const Container&)

[SWS_CORE_04130] Definition of API function ara::core::empty

Upstream requirements: RS_AP_00130

⌈

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

Syntax:	`template <typename Container>`
	`constexpr auto empty (const Container &c)`
	`noexcept(noexcept(c.empty())) -> decltype(c.empty());`	

Template param:	Container	a type with a `empty()` method
Parameters (in):	c	an instance of Container
Return value:	decltype(c.empty())	true if the container is empty, false otherwise
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return whether the given container is empty	

⌋

8.19.3.1.8 size(const T()&)

[SWS_CORE_04121] Definition of API function ara::core::size

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/utility.h"	
Scope:	namespace ara::core	
Syntax:	`template <typename T, std::size_t N>` `constexpr std::size_t size (const T(&array)[N]) noexcept;`	
Template param:	T	the type of array elements
	N	the number of elements in the array
Parameters (in):	array	reference to a raw array
Return value:	std::size_t	the size of the array, i.e. N
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Return the size of a raw array.	

⌋

8.19.3.1.9 size(const Container&)

[SWS_CORE_04120] Definition of API function ara::core::size

Upstream requirements: RS_AP_00130

⌈

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

Syntax:	`template <typename Container>`
	`constexpr auto size (const Container &c) noexcept(noexcept(c.size()))`
	`-> decltype(c.size());`	

Template param:	Container	a type with a `size()` method
Parameters (in):	c	an instance of Container
Return value:	decltype(c.size())	the size of the container
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return the size of a container	

⌋

8.19.4 Struct: in_place_index_t

[SWS_CORE_04031] Definition of API class ara::core::in_place_index_t

Upstream requirements: RS_AP_00130

⌈

Kind:	struct	
Header file:	#include "ara/core/utility.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	namespace ara::core	
Symbol:	in_place_index_t	
Syntax:	`template <std::size_t Index>` `struct in_place_index_t {...};`	
Template param:	std::size_t Index	the index to address
Description:	Denote an index-distinguishing operation to be performed in-place. An instance of this type can be passed to certain constructors of `ara::core::Variant` to denote the intention that construction of the contained type shall be done in-place, i.e. without any copying taking place.	

⌋

8.19.4.1 Public Member Functions

8.19.4.1.1 Special Member Functions

8.19.4.1.1.1 Default Constructor

[SWS_CORE_04032] Definition of API function ara::core::in_place_index_t::in_place_index_t

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/utility.h"
Scope:	`struct ara::core::in_place_index_t`
Syntax:	`explicit in_place_index_t () noexcept=default;`
Exception Safety:	exception safe
Description:	Default constructor

⌋

8.19.5 Struct: in_place_t

[SWS_CORE_04011] Definition of API class ara::core::in_place_t

Upstream requirements: RS_AP_00130

⌈

Kind:	struct
Header file:	#include "ara/core/utility.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	namespace ara::core
Symbol:	in_place_t
Syntax:	`struct in_place_t {...};`
Description:	Denote an operation to be performed in-place. An instance of this type can be passed to certain constructors of `ara::core::Optional` to denote the intention that construction of the contained type shall be done in-place, i.e. without any copying taking place.

⌋

8.19.5.1 Public Member Functions

8.19.5.1.1 Special Member Functions

8.19.5.1.1.1 Default Constructor

[SWS_CORE_04012] Definition of API function ara::core::in_place_t::in_place_t

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/utility.h"
Scope:	`struct ara::core::in_place_t`
Syntax:	`explicit in_place_t () noexcept=default;`
Exception Safety:	exception safe
Description:	Default constructor

⌋

8.19.6 Struct: in_place_type_t

[SWS_CORE_04021] Definition of API class ara::core::in_place_type_t

Upstream requirements: RS_AP_00130

⌈

Kind:	struct	
Header file:	#include "ara/core/utility.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	namespace ara::core	
Symbol:	in_place_type_t	
Syntax:	`template <typename T>` `struct in_place_type_t {...};`	
Template param:	typename T	the type to address
Description:	Denote a type-distinguishing operation to be performed in-place. An instance of this type can be passed to certain constructors of `ara::core::Variant` to denote the intention that construction of the contained type shall be done in-place, i.e. without any copying taking place.	

⌋

8.19.6.1 Public Member Functions

8.19.6.1.1 Special Member Functions

8.19.6.1.1.1 Default Constructor

[SWS_CORE_04022] Definition of API function ara::core::in_place_type_t::in_place_type_t

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/utility.h"
Scope:	`struct ara::core::in_place_type_t`
Syntax:	`explicit in_place_type_t () noexcept=default;`
Exception Safety:	exception safe
Description:	Default constructor

⌋

8.19.7 Struct: is_nothrow_swappable

[SWS_CORE_04014] Definition of API class ara::core::is_nothrow_swappable

Status: DRAFT

⌈

Kind:	struct	
Header file:	#include "ara/core/utility.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	namespace ara::core	
Symbol:	is_nothrow_swappable	
Syntax:	`template <class T>` `struct is_nothrow_swappable;`	
Template param:	T	--
Description:	As per `std::is_nothrow_swappable`, see `[meta.unary.prop]` in [11]	

⌋

8.20 Header: ara/core/executor.h

8.20.1 Class: Executor

[SWS_CORE_00008] Definition of API class ara::core::Executor

Upstream requirements: RS_AP_00130

⌈

Kind:	class
Header file:	#include "ara/core/executor.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	namespace ara::core
Symbol:	Executor
Syntax:	`class Executor {...};`
Description:	Provides an interface for a executing context, which asynchronously invokes `Callable` ([4] `[func.wrap.func]`) objects in a guaranteed thread-safe context. The `execute(...)` method in [SWS_CORE_00024] returns a `ara::core::Future` object to the caller for further result/error extraction or management of the asynchronous-context; the `execute_oneway(...)` method in [SWS_CORE_00025] imply fire-and-forget semantics and thus no asynchronous-context (`ara::core::Future`) is returned.

⌋

8.20.1.1 Public Member Functions

8.20.1.1.1 Special Member Functions

8.20.1.1.1.1 Copy Constructor

[SWS_CORE_00009] Definition of API function ara::core::Executor::Executor

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/executor.h"	
Scope:	`class ara::core::Executor`	
Syntax:	`Executor (const Executor &other);`	
Parameters (in):	other	Other `ara::core::Executor` to copy
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Copy constructor	

⌋

8.20.1.1.1.2 Move Constructor

[SWS_CORE_00015] Definition of API function ara::core::Executor::Executor

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/executor.h"	
Scope:	`class ara::core::Executor`	
Syntax:	`Executor (Executor &&other) noexcept;`	
Parameters (in):	other	Other `ara::core::Executor` to move
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Move constructor	

⌋

8.20.1.1.1.3 Copy Assignment Operator

[SWS_CORE_00017] Definition of API function ara::core::Executor::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/executor.h"	
Scope:	`class ara::core::Executor`	
Syntax:	`Executor & operator= (const Executor &other);`	
Parameters (in):	other	Other `ara::core::Executor` to copy
Return value:	Executor &	The copied object
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Copy assignment operator	

⌋

8.20.1.1.1.4 Move Assignment Operator

[SWS_CORE_00018] Definition of API function ara::core::Executor::operator=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/executor.h"	
Scope:	`class ara::core::Executor`	
Syntax:	`Executor & operator= (Executor &&other) noexcept;`	
Parameters (in):	other	Other `ara::core::Executor` to move
Return value:	Executor &	The moved object
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Move assignment operator	

⌋

8.20.1.1.1.5 Destructor

[SWS_CORE_00019] Definition of API function ara::core::Executor::~Executor

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/executor.h"
Scope:	`class ara::core::Executor`
Syntax:	`~Executor () noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	Destructor

⌋

8.20.1.1.2 Member Functions

8.20.1.1.2.1 execute

[SWS_CORE_00024] Definition of API function ara::core::Executor::execute

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/executor.h"	
Scope:	`class ara::core::Executor`	
Syntax:	`template <class F, class... P> ara::core::Future< std::invoke_result_t< std::decay_t< F >, std::decay_t< P >... > > execute (F &&f, P &&... p) noexcept;`	
Template param:	F	a `Callable` type
	P	argument list to the `Callable`
Parameters (in):	f	a `Callable` to be executed
	p	parameters to be supplied with the `Callable`
Return value:	ara::core::Future< std::invoke_result_t< std::decay_t< F >, std::decay_t< P >... > >	• If successful: an `ara::core::Future` containing a `ara::core::Result::value_type` containing the result of the callable `F` • If unsuccessful: an `ara::core::Future` containing an `ErrorCode`
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Errors:	This function does not specify any standardized errors.	
Description:	Asynchronously invoke `F` with the provided arguments `P...`.	

⌋

8.20.1.1.2.2 oneway_execute

[SWS_CORE_00025] Definition of API function ara::core::Executor::oneway_execute

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/executor.h"	
Scope:	`class ara::core::Executor`	
Syntax:	`template <class F, class... P>` `void oneway_execute (F &&f, P &&... p) noexcept;`	
Template param:	F	a `Callable` type
	P	argument list to the `Callable`
Parameters (in):	f	a `Callable` to be executed
	p	parameters to be supplied with the `Callable`
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Asynchronously invoke `F` with the provided arguments `P...`.	

⌋

8.21 Header: ara/core/initialization.h

8.21.1 Non-Member Functions

8.21.1.1 Other

8.21.1.1.1 Deinitialize

[SWS_CORE_10002] Definition of API function ara::core::Deinitialize

Upstream requirements: RS_AP_00155

⌈

Kind:	function	
Header file:	#include "ara/core/initialization.h"	
Scope:	namespace ara::core	
Syntax:	`Result< void > Deinitialize () noexcept;`	
Return value:	Result< void >	• If successful: an `ara::core::Result` containing a `ara::core::Result::value_type` containing a `void` • If unsuccessful: an `ara::core::Result` containing an `ara::core::Result::error_type` i.e. a corresponding `ara::core::CoreErrc`
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	

▽

△

Errors:	CoreErrc::kNotInitialized	rollback_semantics
		ARA is not currently initialized (i.e. has already been successfully de-initialized or has never been initialized)
Description:	Shutdown of the ARA Framework After this call, no interaction with the ARA is allowed with the exception of types intended to be used independently of initialization as defined in [SWS_CORE_15002]. As a prerequisite to calling this API it is expected that the use of ARA interfaces has completed (with the given exceptions). It is strongly recommended to make this call in a place where it is guaranteed that the static initialization has completed and destruction of statically initialized data has not yet started.	

⌋

8.21.1.1.2 Initialize(int&, char **)

[SWS_CORE_10003] Definition of API function ara::core::Initialize

Upstream requirements: RS_AP_00155

⌈

Kind:	function	
Header file:	#include "ara/core/initialization.h"	
Scope:	namespace ara::core	
Syntax:	`Result< void > Initialize (int &argc, char **argv) noexcept;`	
Parameters (inout):	argc	number of elements in `argv`
	argv	the process's command-line arguments, as received in `main()`
Return value:	Result< void >	• If successful: an `ara::core::Result` containing a `ara::core::Result::value_type` containing a `void` • If unsuccessful: an `ara::core::Result` containing an `ara::core::Result::error_type` i.e. a corresponding `ara::core::CoreErrc`
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Errors:	CoreErrc::kAlready Initialized	rollback_semantics
		ARA has already been initialized and has completed
Description:	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.	
Notes:	Both `argc` and `argv` are mutable in the function. This is done to allow injection and interpretation of vendor specific ARA configuration information. The injected arguments are removed in accordance with [SWS_CORE_15006]. Error scenarios, such as corrupted configuration files, are expected to be handled in an implementation-defined way - either as a `Non-Standardized Violation` ([SWS_CORE_00003]) or as a vendor-specific error code ([SWS_CORE_00092]).	

⌋

8.22 Header: ara/core/instance_specifier.h

8.22.1 Non-Member Functions

8.22.1.1 Other

8.22.1.1.1 operator!=

[SWS_CORE_08082] Definition of API function ara::core::operator!=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	namespace ara::core	
Syntax:	`bool operator!= (ara::core::StringView lhs, const InstanceSpecifier &rhs) noexcept;`	
Parameters (in):	lhs	stringified form of a `ara::core::InstanceSpecifier`
	rhs	an `ara::core::InstanceSpecifier`
Return value:	bool	true in case rhs string representation not equals lhs
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Negative equality comparison of two `ara::core::InstanceSpecifier`s	

⌋

8.22.1.1.2 operator==

[SWS_CORE_08081] Definition of API function ara::core::operator==

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	namespace ara::core	
Syntax:	`bool operator== (ara::core::StringView lhs, const InstanceSpecifier &rhs) noexcept;`	
Parameters (in):	lhs	stringified form of a `ara::core::InstanceSpecifier`
	rhs	an `ara::core::InstanceSpecifier`
Return value:	bool	true in case rhs string representation equals lhs
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Equality comparison of two `ara::core::InstanceSpecifier`s	

⌋

8.22.2 Class: InstanceSpecifier

[SWS_CORE_08001] Definition of API class ara::core::InstanceSpecifier

Upstream requirements: RS_AP_00140

⌈

Kind:	class
Header file:	#include "ara/core/instance_specifier.h"
Forwarding header file:	#include "ara/core/core_fwd.h"
Scope:	namespace ara::core
Symbol:	InstanceSpecifier
Syntax:	`class InstanceSpecifier final {...};`
Description:	Representation of an AUTOSAR Instance Specifier, which is basically an AUTOSAR `shortName` path wrapper.

⌋

8.22.2.1 Public Member Functions

8.22.2.1.1 Special Member Functions

8.22.2.1.1.1 Move Constructor

[SWS_CORE_08023] Definition of API function ara::core::InstanceSpecifier::InstanceSpecifier

Upstream requirements: RS_AP_00145

⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	`class ara::core::InstanceSpecifier`	
Syntax:	`InstanceSpecifier (InstanceSpecifier &&other) noexcept;`	
Parameters (in):	other	the other instance
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Move constructor	

⌋

8.22.2.1.1.2 Copy Constructor

[SWS_CORE_08022] Definition of API function ara::core::InstanceSpecifier::InstanceSpecifier

Upstream requirements: RS_AP_00145

⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	`class ara::core::InstanceSpecifier`	
Syntax:	`InstanceSpecifier (const InstanceSpecifier &other) noexcept;`	
Parameters (in):	other	the other instance
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Copy constructor	

⌋

8.22.2.1.1.3 operator=(const InstanceSpecifier&)

[SWS_CORE_08024] Definition of API function ara::core::InstanceSpecifier::operator=

Upstream requirements: RS_AP_00145

⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	`class ara::core::InstanceSpecifier`	
Syntax:	`InstanceSpecifier & operator= (const InstanceSpecifier &other) noexcept;`	
Parameters (in):	other	the other instance
Return value:	InstanceSpecifier &	*this
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Copy assignment operator	

⌋

8.22.2.1.1.4 operator=(InstanceSpecifier&&)

[SWS_CORE_08025] Definition of API function ara::core::InstanceSpecifier::operator=

Upstream requirements: RS_AP_00145

⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	`class ara::core::InstanceSpecifier`	
Syntax:	`InstanceSpecifier & operator= (InstanceSpecifier &&other) noexcept;`	
Parameters (in):	other	the other instance
Return value:	InstanceSpecifier &	*this
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Move assignment operator	

⌋

8.22.2.1.1.5 Destructor

[SWS_CORE_08029] Definition of API function ara::core::InstanceSpecifier::~InstanceSpecifier

Upstream requirements: RS_AP_00134, RS_AP_00145

⌈

Kind:	function
Header file:	#include "ara/core/instance_specifier.h"
Scope:	`class ara::core::InstanceSpecifier`
Syntax:	`~InstanceSpecifier () noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	Destructor

⌋

8.22.2.1.2 Constructors

8.22.2.1.2.1 InstanceSpecifier

[SWS_CORE_08021] Definition of API function ara::core::InstanceSpecifier::InstanceSpecifier

Upstream requirements: RS_AP_00145

⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	`class ara::core::InstanceSpecifier`	
Syntax:	`explicit InstanceSpecifier (ara::core::StringView qualifiedShortName) noexcept(false);`	
Parameters (in):	qualifiedShortName	String representation of a `ara::core::InstanceSpecifier`, according to the syntax rules in [SWS_CORE_10200] and [SWS_CORE_10203].
Exceptions:	`ara::core::CoreException`	As per `ara::core::CoreErrc::kInvalidMetaModelPath` :- the `qualifiedShortName` is not a valid `shortName` path to a model element.
	`ara::core::CoreException`	As per `ara::core::CoreErrc::kInvalidMetaModelShortname` :- the `qualifiedShortName` contains invalid characters or missing `shortName` path elements.
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Construct an `ara::core::InstanceSpecifier` from a `ara::core::StringView`	

⌋

8.22.2.1.3 Member Functions

8.22.2.1.3.1 ToString

[SWS_CORE_08041] Definition of API function ara::core::InstanceSpecifier::ToString

Upstream requirements: RS_AP_00120

⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	`class ara::core::InstanceSpecifier`	
Syntax:	`ara::core::StringView ToString () const noexcept;`	
Return value:	ara::core::StringView	Stringified form of `ara::core::InstanceSpecifier`. Lifetime of the underlying string is only guaranteed for the lifetime of the underlying string of the `ara::core::StringView` passed to the constructor.
Exception Safety:	exception safe	
Thread Safety:	thread-safe	

▽

△

Description:	method to return the stringified form of `ara::core::InstanceSpecifier`

⌋

8.22.2.1.3.2 operator!=(ara::core::StringView) const

[SWS_CORE_08045] Definition of API function ara::core::InstanceSpecifier::operator!=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	`class ara::core::InstanceSpecifier`	
Syntax:	`bool operator!= (ara::core::StringView other) const noexcept;`	
Parameters (in):	other	string representation to compare this one with
Return value:	bool	false in case this `ara::core::InstanceSpecifier` is denoting exactly the same model element as other, true otherwise.
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Negative comparison operator to compare with other `ara::core::InstanceSpecifier` string representation.	

⌋

8.22.2.1.3.3 operator!=(const InstanceSpecifier&) const

[SWS_CORE_08044] Definition of API function ara::core::InstanceSpecifier::operator!=

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	`class ara::core::InstanceSpecifier`	
Syntax:	`bool operator!= (const InstanceSpecifier &other) const noexcept;`	
Parameters (in):	other	`ara::core::InstanceSpecifier` instance to compare this one with.
Return value:	bool	false in case both `ara::core::InstanceSpecifier`s are denoting exactly the same model element, true otherwise.
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Negative comparison operator to compare with other `ara::core::InstanceSpecifier` instance.	

⌋

8.22.2.1.3.4 operator<

[SWS_CORE_08046] Definition of API function ara::core::InstanceSpecifier::operator<

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	`class ara::core::InstanceSpecifier`	
Syntax:	`bool operator< (const InstanceSpecifier &other) const noexcept;`	
Parameters (in):	other	`ara::core::InstanceSpecifier` instance to compare this one with.
Return value:	bool	true in case this `ara::core::InstanceSpecifier` is lexically lower than other, false otherwise.
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	lower than operator to compare with other `ara::core::InstanceSpecifier` for ordering purposes, for instance, when collecting identifiers in maps).	

⌋

8.22.2.1.3.5 operator==(const InstanceSpecifier&) const

[SWS_CORE_08042] Definition of API function ara::core::InstanceSpecifier::operator==

Upstream requirements: RS_AP_00120

⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	`class ara::core::InstanceSpecifier`	
Syntax:	`bool operator== (const InstanceSpecifier &other) const noexcept;`	
Parameters (in):	other	`ara::core::InstanceSpecifier` instance to compare this one with.
Return value:	bool	true in case both `ara::core::InstanceSpecifier`s are denoting exactly the same model element, false otherwise.
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	eq operator to compare with other `ara::core::InstanceSpecifier` instance.	

⌋

8.22.2.1.3.6 operator==(ara::core::StringView) const

[SWS_CORE_08043] Definition of API function ara::core::InstanceSpecifier::operator==

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	`class ara::core::InstanceSpecifier`	
Syntax:	`bool operator== (ara::core::StringView other) const noexcept;`	
Parameters (in):	other	string representation to compare this one with.
Return value:	bool	true in case this `ara::core::InstanceSpecifier` is denoting exactly the same model element as other, false otherwise.
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	eq operator to compare with other `ara::core::InstanceSpecifier` instance.	

⌋

8.22.2.1.4 Named Constructors

8.22.2.1.4.1 Create

[SWS_CORE_08032] Definition of API function ara::core::InstanceSpecifier::Create

Upstream requirements: RS_AP_00137, RS_AP_00136

⌈

Kind:	function	
Header file:	#include "ara/core/instance_specifier.h"	
Scope:	`class ara::core::InstanceSpecifier`	
Syntax:	`static ara::core::Result< InstanceSpecifier > Create ( ara::core::StringView qualifiedShortName) noexcept;`	
Parameters (in):	qualifiedShortName	string representation of a valid
Return value:	ara::core::Result< InstanceSpecifier >	a Result, containing either a syntactically valid `ara::core::InstanceSpecifier`, or an ErrorCode
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	CoreErrc::kInvalidMetaModelShortname	rollback_semantics
		if any of the path elements of `qualifiedShortName` is missing or contains invalid characters
	CoreErrc::kInvalidMetaModelPath	rollback_semantics
		if the `qualifiedShortName` is not a valid path to a model element

▽

△

Description:	Create a new instance of this class `ara::core::InstanceSpecifier`, according to the syntax rules given by [SWS_CORE_10200] and [SWS_CORE_10203].

⌋

8.23 Header: ara/core/result.h

8.23.1 Non-Member Functions

8.23.1.1 Other

8.23.1.1.1 operator!=(const Result<T, E>&, const E&)

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	namespace ara::core	
Syntax:	`template <typename T, typename E>` `bool operator!= (const Result< T, E > &lhs, const E &rhs);`	
Parameters (in):	lhs	the Result instance
	rhs	the error to compare with
Return value:	bool	true if the Result's error compares unequal to the rhs error, false otherwise
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Compare a Result instance for inequality to an error. A Result that contains no error is unequal to every error. A Result is equal to an error only if the Result contains an error of the same type, and the errors compare equal.	

⌋

8.23.1.1.2 operator!=(const Result<T, E>&, const Result<T, E>&)

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

Upstream requirements: RS_AP_00130

⌈

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

▽

△

Syntax:	`template <typename T, typename E>` `bool operator!= (const Result< T, E > &lhs, const Result< T, E > &rhs);`	
Parameters (in):	lhs	the left hand side of the comparison
	rhs	the right hand side of the comparison
Return value:	bool	true if the two instances compare unequal, false otherwise
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Compare two Result instances for inequality. A Result that contains a value is unequal to every Result containing an error. A Result containing a value is equal to another Result only if both contain the same type, and either that type is `void` or the value of that type compares equal. A Result containing an error is equal to another Result only if both contain the same error type, and the contained errors compare equal.	
See also:	[SWS_CORE_00780]	

⌋

8.23.1.1.3 operator!=(const T&, const Result<T, E>&)

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	namespace ara::core	
Syntax:	`template <typename T, typename E>` `bool operator!= (const T &lhs, const Result< T, E > &rhs);`	
Parameters (in):	lhs	the value to compare with
	rhs	the Result instance
Return value:	bool	true if the Result's value compares unequal to the lhs value, false otherwise
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Compare a Result instance for inequality to a value. A Result that contains no value is unequal to every value. A Result is equal to a value only if the Result contains a value of the same type, and the values compare equal.	

⌋

8.23.1.1.4 operator!=(const E&, const Result<T, E>&)

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	namespace ara::core	
Syntax:	`template <typename T, typename E>` `bool operator!= (const E &lhs, const Result< T, E > &rhs);`	
Parameters (in):	lhs	the error to compare with
	rhs	the Result instance
Return value:	bool	true if the Result's error compares unequal to the lhs error, false otherwise
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Compare a Result instance for inequality to an error. A Result that contains no error is unequal to every error. A Result is equal to an error only if the Result contains an error of the same type, and the errors compare equal.	

⌋

8.23.1.1.5 operator!=(const Result<T, E>&, const T&)

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	namespace ara::core	
Syntax:	`template <typename T, typename E>` `bool operator!= (const Result< T, E > &lhs, const T &rhs);`	
Parameters (in):	lhs	the Result instance
	rhs	the value to compare with
Return value:	bool	true if the Result's value compares unequal to the rhs value, false otherwise
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Compare a Result instance for inequality to a value. A Result that contains no value is unequal to every value. A Result is equal to a value only if the Result contains a value of the same type, and the values compare equal.	

⌋

8.23.1.1.6 operator==(const Result<T, E>&, const T&)

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	namespace ara::core	
Syntax:	`template <typename T, typename E>` `bool operator== (const Result< T, E > &lhs, const T &rhs);`	
Parameters (in):	lhs	the Result instance
	rhs	the value to compare with
Return value:	bool	true if the Result's value compares equal to the rhs value, false otherwise
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Compare a Result instance for equality to a value. A Result that contains no value is unequal to every value. A Result is equal to a value only if the Result contains a value of the same type, and the values compare equal.	

⌋

8.23.1.1.7 operator==(const Result<T, E>&, const E&)

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	namespace ara::core	
Syntax:	`template <typename T, typename E>` `bool operator== (const Result< T, E > &lhs, const E &rhs);`	
Parameters (in):	lhs	the Result instance
	rhs	the error to compare with
Return value:	bool	true if the Result's error compares equal to the rhs error, false otherwise
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Compare a Result instance for equality to an error. A Result that contains no error is unequal to every error. A Result is equal to an error only if the Result contains an error of the same type, and the errors compare equal.	

⌋

8.23.1.1.8 operator==(const E&, const Result<T, E>&)

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	namespace ara::core	
Syntax:	`template <typename T, typename E>` `bool operator== (const E &lhs, const Result< T, E > &rhs);`	
Parameters (in):	lhs	the error to compare with
	rhs	the Result instance
Return value:	bool	true if the Result's error compares equal to the lhs error, false otherwise
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Compare a Result instance for equality to an error. A Result that contains no error is unequal to every error. A Result is equal to an error only if the Result contains an error of the same type, and the errors compare equal.	

⌋

8.23.1.1.9 operator==(const Result<T, E>&, const Result<T, E>&)

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	namespace ara::core	
Syntax:	`template <typename T, typename E>` `bool operator== (const Result< T, E > &lhs, const Result< T, E > &rhs);`	
Parameters (in):	lhs	the left hand side of the comparison
	rhs	the right hand side of the comparison
Return value:	bool	true if the two instances compare equal, false otherwise
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Compare two Result instances for equality. A Result that contains a value is unequal to every Result containing an error. A Result containing a value is equal to another Result only if both contain the same type, and either that type is `void` or the value of that type compares equal. A Result containing an error is equal to another Result only if both contain the same error type, and the contained errors compare equal.	

⌋

8.23.1.1.10 operator==(const T&, const Result<T, E>&)

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	namespace ara::core	
Syntax:	`template <typename T, typename E>` `bool operator== (const T &lhs, const Result< T, E > &rhs);`	
Parameters (in):	lhs	the value to compare with
	rhs	the Result instance
Return value:	bool	true if the Result's value compares equal to the lhs value, false otherwise
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Compare a Result instance for equality to a value. A Result that contains no value is unequal to every value. A Result is equal to a value only if the Result contains a value of the same type, and the values compare equal.	

⌋

8.23.1.1.11 swap

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

Upstream requirements: RS_AP_00130

⌈

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

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

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

⌋

8.23.2 Class: Result

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

Upstream requirements: RS_AP_00130

⌈

Kind:	class	
Header file:	#include "ara/core/result.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	namespace ara::core	
Symbol:	Result	
Syntax:	`template <typename T, typename E = ErrorCode>` `class Result final {...};`	
Template param:	typename T	the type of value
	typename E = ErrorCode	the type of error
Description:	This class is a type that contains either a value or an error. The implementation shall flag the condition `E != ara::core::ErrorCode` as a compile error.	

⌋

8.23.2.1 Public Member Types

8.23.2.1.1 Type Alias: error_type

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.1.2 Type Alias: value_type

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.2 Public Member Functions

8.23.2.2.1 Special Member Functions

8.23.2.2.1.1 Copy Constructor

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.2.1.2 Move Constructor

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`Result (Result &&other) noexcept(std::is_nothrow_move_constructible< T >::value &&std::is_nothrow_move_constructible< E >::value);`	
Parameters (in):	other	the other instance
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	Move-construct a new Result from another instance	

⌋

8.23.2.2.1.3 Move Assignment Operator

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.2.1.4 Copy Assignment Operator

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.2.1.5 Destructor

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.2.2 Constructors

8.23.2.2.2.1 Result(E&&)

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.2.2.2 Result(T&&)

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.2.2.3 Result(const T&)

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.2.2.4 Result(const E&)

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.2.3 Member Functions

8.23.2.2.3.1 Bind

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`template <typename F>` `auto Bind (F &&f) const -> <see below>;`	
Template param:	F	the type of the `Callable` `f()`
Parameters (in):	f	the `Callable`
Return value:	*<see below>*	a new `ara::core::Result` instance of the possibly transformed type
Exception Safety:	not exception safe	
Thread Safety:	conditional,	depends on thread-safety of `f()`
Description:	Apply the given `Callable` to the value of this instance, and return a new `ara::core::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&);` • <ARTechTerm{XXX f(const T&);}> • <ARNewline{}> • <ARNewline{}> • <ARTechTerm{Callable}> • Result<XXX> or a `XXX` directly, where `XXX` can be any type that is suitable for use by `ara::core::Result`. The return type of this function is `decltype(f(Value()))` for a template argument F that returns a `ara::core::Result` type, and it is `Result<decltype(f(Value())), E>` for a template argument F that does not return a `ara::core::Result` type. If this instance does not contain a value, a new `Result<XXX, E>` is still created and returned, with the original error contents of this instance being copied into the new instance.}>}	

⌋

8.23.2.2.3.2 CheckError

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

Upstream requirements: RS_AP_00130

⌈

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

▽

△

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

⌋

8.23.2.2.3.3 EmplaceError

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`template <typename... Args>` `void EmplaceError (Args &&... args) noexcept(std::is_nothrow_constructible< E, Args... >::value);`	
Template param:	Args...	the types of arguments given to this function
Parameters (in):	args	the arguments used for constructing the error
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	Put a new error into this instance, constructed in-place from the given arguments.	

⌋

8.23.2.2.3.4 EmplaceValue

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

Upstream requirements: RS_AP_00130

⌈

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

▽

△

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

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

⌋

8.23.2.2.3.5 Err() &&

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.2.3.6 Err() const &

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

Upstream requirements: RS_AP_00130

⌈

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

▽

△

Description:	Return the contained error as an `ara::core::Optional`.

⌋

8.23.2.2.3.7 Error() &&

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.2.3.8 Error() &

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.2.3.9 Error() const &

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.2.3.10 ErrorOr(G&&) &&

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`template <typename G>` `E ErrorOr (G &&defaultError) &&noexcept(std::is_nothrow_move_constructible< E >::value &&std::is_nothrow_constructible< E, G && >::value);`	
Template param:	G	the type of defaultError
Parameters (in):	defaultError	the error to use if `*this` does not contain an error
Return value:	E	the error
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return the contained error or the given default error. If `*this` contains an error, it is `std::move`'d into the return value. Otherwise, the specified default error is returned, `static_cast`'d to `E`	

⌋

8.23.2.2.3.11 ErrorOr(G&&) const &

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`template <typename G>` `E ErrorOr (G &&defaultError) const &noexcept(std::is_nothrow_copy_constructible< E >::value &&std::is_nothrow_constructible< E, G &&>::value);`	
Template param:	G	the type of defaultError
Parameters (in):	defaultError	the error to use if `*this` does not contain an error
Return value:	E	the error
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return the contained error or the given default error. If `*this` contains an error, it is returned. Otherwise, the specified default error is returned, `static_cast`'d to `E`	

⌋

8.23.2.2.3.12 HasValue

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.2.3.13 Ok() const &

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.2.3.14 Ok() &&

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.2.3.15 Resolve

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`template <typename F>` `T Resolve (F &&f) const;`	
Template param:	F	the type of the Callable `f()` }
Parameters (in):	f	the Callable
Return value:	T	the value
Exception Safety:	not exception safe	
Thread Safety:	conditional,	depends on thread-safety of `f()`
Description:	Return the contained value or return the result of a function call. If `*this` contains a value, it is returned. Otherwise, the specified callable is invoked and its return value which is to be compatible to type `T` is returned from this function. The Callable is expected to be compatible to this interface: `T f(const E&);`	

⌋

8.23.2.2.3.16 Swap

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.2.3.17 Value() &&

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.2.3.18 Value() &

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.2.3.19 Value() const &

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.2.3.20 ValueOr(U&&) const &

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`template <typename U>` `T ValueOr (U &&defaultValue) const &noexcept(std::is_nothrow_copy_constructible< T >::value &&std::is_nothrow_constructible< T, U &&>::value);`	
Template param:	U	the type of `defaultValue`
Parameters (in):	defaultValue	the value to use if `*this` does not contain a value
Return value:	T	the value
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return the contained value or the given default value. If `*this` contains a value, it is returned. Otherwise, the specified default value is returned, `static_cast`'d to `T`	

⌋

8.23.2.2.3.21 ValueOr(U&&) &&

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`template <typename U>` `T ValueOr (U &&defaultValue) &&noexcept(std::is_nothrow_move_constructible< T >::value &&std::is_nothrow_constructible< T, U && >::value);`	
Template param:	U	the type of `defaultValue`
Parameters (in):	defaultValue	the value to use if `*this` does not contain a value
Return value:	T	the value
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return the contained value or the given default value. If `*this` contains a value, it is returned. Otherwise, the specified default value is returned, `static_cast`'d to `T`	

⌋

8.23.2.2.3.22 ValueOrThrow() &

[SWS_CORE_00778] Definition of API function ara::core::Result::ValueOrThrow

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.2.3.23 ValueOrThrow() const &

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.2.3.24 ValueOrThrow() &&

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.2.3.25 operator bool

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.2.3.26 operator*() &

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`T & operator* () & noexcept;`	
Return value:	T &	a reference to the contained value
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`ResultWithNoValue-Violation`	If `*this` does not contain a value.
Description:	Access the contained value.	
Notes:	This function shall not participate in overload resolution when C++ exceptions are disabled in the compiler toolchain	

⌋

8.23.2.2.3.27 operator*() &&

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.2.3.28 operator*() const &

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.2.3.29 operator->() const

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.2.3.30 operator->()

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.2.4 Named Constructors

8.23.2.2.4.1 FromError(Args ...&&)

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`template <typename... Args>` `static Result FromError (Args &&... args) noexcept(std::is_nothrow_constructible< E, Args... >::value);`	
Template param:	Args...	the types of arguments given to this function
Parameters (in):	args	the arguments used for constructing the error
Return value:	Result	a `ara::core::Result` that contains an error
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Build a new `ara::core::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 == 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 `ara::core::Result`	

⌋

8.23.2.2.4.2 FromError(const E&)

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.2.4.3 FromError(E&&)

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.2.4.4 FromValue(Args ...&&)

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result`	
Syntax:	`template <typename... Args>` `static Result FromValue (Args &&... args) noexcept(std::is_nothrow_constructible< T, Args... >::value);`	
Template param:	Args...	the types of arguments given to this function
Parameters (in):	args	the arguments used for constructing the value
Return value:	Result	a `ara::core::Result` that contains a value
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	Build a new `ara::core::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 == 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 `ara::core::Result`	

⌋

8.23.2.2.4.5 FromValue(T&&)

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.2.2.4.6 FromValue(const T&)

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.3 Class: Result<T &, E>

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

Upstream requirements: RS_AP_00130

⌈

Kind:	class	
Header file:	#include "ara/core/result.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	namespace ara::core	
Symbol:	Result< T &, E >	
Syntax:	`template <typename T, typename E>` `class Result< T &, E > final {...};`	
Template param:	typename T	the type of value
	typename E	the type of error
Description:	Specialization of class Result for reference to value types. The implementation shall flag the condition `E!=ara::core::ErrorCode` as a compile error.	

⌋

8.23.3.1 Public Member Types

8.23.3.1.1 Type Alias: error_type

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.3.1.2 Type Alias: value_type

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.3.2 Public Member Functions

8.23.3.2.1 Special Member Functions

8.23.3.2.1.1 Move Constructor

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`Result (Result &&other) noexcept(std::is_nothrow_move_constructible< E >::value);`	
Parameters (in):	other	the other instance
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	Move-construct a new Result from another instance	

⌋

8.23.3.2.1.2 Copy Constructor

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.3.2.1.3 Copy Assignment Operator

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.3.2.1.4 Move Assignment Operator

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.3.2.1.5 Destructor

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.3.2.2 Constructors

8.23.3.2.2.1 Result(const E&)

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.3.2.2.2 Result(E&&)

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.3.2.2.3 Result

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`template <typename U = T>` `Result (const U &&t) noexcept;`	
Template param:	U	the type of t
Parameters (in):	t	the value to bind to
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Construct a new Result containing a reference to value from the specified reference to value. If U is not an lvalue, the program is ill-formed.	

⌋

8.23.3.2.3 Member Functions

8.23.3.2.3.1 Bind

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`template <typename F>` `auto Bind (F &&f) const -> <see below>;`	
Template param:	F	the type of the Callable f
Parameters (in):	f	the Callable
Return value:	*<see below>*	a new Result instance of the possibly transformed type
Exception Safety:	not exception safe	
Thread Safety:	conditional,	depends on thread-safety of f

▽

△

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

⌋

8.23.3.2.3.2 CheckError

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`template <typename G>` `bool CheckError (G &&error) const;`	
Template param:	G	the type of the error argument `error`
Parameters (in):	error	the error to check
Return value:	bool	true if `*this` contains an error that is equivalent to the given error, false otherwise
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Return whether this instance contains the given error. This call compares the argument `error`, `static_cast`'d to `E`, with the return value from `Error()`	

⌋

8.23.3.2.3.3 EmplaceError

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`template <typename... Args>` `void EmplaceError (Args &&... args) noexcept(std::is_nothrow_constructible< E, Args... >::value);`	
Template param:	Args...	the types of arguments given to this function
Parameters (in):	args	the arguments used for constructing the error
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	Put a new error into this instance, constructed in-place from the given arguments.	

⌋

8.23.3.2.3.4 Err() const &

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.3.2.3.5 Err() &&

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.3.2.3.6 Error() &

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.3.2.3.7 Error() &&

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.3.2.3.8 Error() const &

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.3.2.3.9 ErrorOr(G&&) &&

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`template <typename G>` `E ErrorOr (G &&defaultError) &&noexcept(std::is_nothrow_move_constructible< E >::value &&std::is_nothrow_constructible< E, G && >::value);`	
Template param:	G	the type of defaultError
Parameters (in):	defaultError	the error to use if `*this` does not contain an error
Return value:	E	the error
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return the contained error or the given default error. If `*this` contains an error, it is `std::move`'d into the return value. Otherwise, the specified default error is returned, `static_cast`'d to `E`	

⌋

8.23.3.2.3.10 ErrorOr(G&&) const &

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`template <typename G>` `E ErrorOr (G &&defaultError) const &noexcept(std::is_nothrow_copy_constructible< E >::value &&std::is_nothrow_constructible< E, G &&>::value);`	
Template param:	G	the type of defaultError
Parameters (in):	defaultError	the error to use if `*this` does not contain an error
Return value:	E	the error
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return the contained error or the given default error. If `*this` contains an error, it is returned. Otherwise, the specified default error is returned, `static_cast`'d to `E`	

⌋

8.23.3.2.3.11 HasValue

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.3.2.3.12 Ok() const

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.3.2.3.13 Ok()

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.3.2.3.14 Resolve(F&&) const

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`template <typename F>` `T & Resolve (F &&f) const;`	
Template param:	F	the type of the Callable f
Parameters (in):	f	the Callable
Return value:	T &	the referred-to value
Exception Safety:	not exception safe	
Thread Safety:	conditional,	depends on thread safety of f
Description:	Return the referred-to value or return the result of a function call. If *this contains a value, it is returned. Otherwise, the specified callable is invoked and its return value which is to be compatible to type T& is returned from this function. The Callable is expected to be compatible to this interface: `T& f(const E&);`	

⌋

8.23.3.2.3.15 Resolve(F&&) const

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`template <typename F>` `const T & Resolve (F &&f) const;`	
Template param:	F	the type of the Callable f
Parameters (in):	f	the Callable
Return value:	const T &	the referred-to value
Exception Safety:	not exception safe	
Thread Safety:	conditional,	depends on thread safety of f
Description:	Return the referred-to value or return the result of a function call. If *this contains a value, it is returned. Otherwise, the specified callable is invoked and its return value which is to be compatible to type T& is returned from this function. The Callable is expected to be compatible to this interface: `const T& f(const E&);`	

⌋

8.23.3.2.3.16 Swap

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.3.2.3.17 Value() &

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.3.2.3.18 Value() const &

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.3.2.3.19 ValueOr(U&&) &&

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`template <typename U>` `T ValueOr (U &&defaultValue) &&noexcept(std::is_nothrow_move_constructible< T >::value &&std::is_nothrow_constructible< T, U && >::value);`	
Template param:	U	the type of defaultValue
Parameters (in):	defaultValue	the value to use if *this does not contain a value
Return value:	T	the value
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return a copy of the referred-to value or the given default value. If *this contains a value, it is returned. Otherwise, the specified default value is returned, static_cast'd to T.	

⌋

8.23.3.2.3.20 ValueOr(U&&) const &

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`template <typename U>` `T ValueOr (U &&defaultValue) const &noexcept(std::is_nothrow_copy_constructible< T >::value &&std::is_nothrow_constructible< T, U &&>::value);`	
Template param:	U	the type of defaultValue
Parameters (in):	defaultValue	the value to use if *this does not contain a value
Return value:	T	the value
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return a copy of the referred-to value or the given default value. If *this contains a value, it is returned. Otherwise, the specified default value is returned, static_cast'd to T.	

⌋

8.23.3.2.3.21 ValueOrThrow() const

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.3.2.3.22 ValueOrThrow()

[SWS_CORE_00949] Definition of API function ara::core::Result< T &, E >::ValueOrThrow

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.3.2.3.23 operator bool

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.3.2.3.24 operator*() &

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.3.2.3.25 operator*() const &

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.3.2.3.26 operator->() const

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.3.2.3.27 operator->()

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.3.2.4 Named Constructors

8.23.3.2.4.1 FromError

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`template <typename... Args>` `static Result FromError (Args &&... args) noexcept(std::is_nothrow_constructible< E, Args... >::value);`	
Template param:	Args...	the types of arguments given to this function
Parameters (in):	args	the parameter pack used for constructing the error
Return value:	Result	a Result that contains an error
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-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	

⌋

8.23.3.2.4.2 FromError(const E&)

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.3.2.4.3 FromError(E&&)

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.3.2.4.4 FromValue

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< T &, E >`	
Syntax:	`template <typename U = T>` `static Result FromValue (U &&u) noexcept;`	
Template param:	U	the type of u
Parameters (in):	u	the value to bind to
Return value:	Result	a Result that contains a reference to value
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Description:	Build a new Result from the reference to a specified value (given as lvalue). If U is not an lvalue, the program is ill-formed.	

⌋

8.23.4 Class: Result<void, E>

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

Upstream requirements: RS_AP_00130

⌈

Kind:	class	
Header file:	#include "ara/core/result.h"	
Forwarding header file:	#include "ara/core/core_fwd.h"	
Scope:	namespace ara::core	
Symbol:	Result< void, E >	
Syntax:	`template <typename E>` `class Result< void, E > final {...};`	
Template param:	typename E	the type of error
Description:	Specialization of class Result for "void" values The implementation shall flag the condition `E != ara::core::ErrorCode` as a compile error	

⌋

8.23.4.1 Public Member Types

8.23.4.1.1 Type Alias: error_type

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.4.1.2 Type Alias: value_type

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.4.2 Public Member Functions

8.23.4.2.1 Special Member Functions

8.23.4.2.1.1 Copy Constructor

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.4.2.1.2 Move Constructor

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.4.2.1.3 Default Constructor

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.4.2.1.4 Move Assignment Operator

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.4.2.1.5 Copy Assignment Operator

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.4.2.1.6 Destructor

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.4.2.2 Constructors

8.23.4.2.2.1 Result(E&&)

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.4.2.2.2 Result(const E&)

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.4.2.3 Member Functions

8.23.4.2.3.1 Bind

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	
Syntax:	`template <typename F>` `auto Bind (F &&f) const -> <see below>;`	
Template param:	F	the type of the Callable f
Parameters (in):	f	the Callable
Return value:	*<see below>*	a new Result instance of the possibly transformed type
Exception Safety:	not exception safe	
Thread Safety:	conditional,	depends on thread safety of f
Description:	Call the given Callable, and return a new Result with the result of the call. The Callable is expected to be compatible to one of these two interfaces: • `Result<XXX, E> f();` • `XXX f();` meaning that the Callable either returns a Result<XXX, E> or a XXX directly, where XXX can be any type that is suitable for use by class Result. The return type of this function is `decltype(f())` for a template argument F that returns a Result type, and it is `Result<decltype(f()), E>` for a template argument F that does not return a Result type. If this instance does not contain a value, a new Result<XXX, E> is still created and returned, with the original error contents of this instance being copied into the new instance	

⌋

8.23.4.2.3.2 CheckError

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	
Syntax:	`template <typename G>` `bool CheckError (G &&error) const noexcept;`	
Template param:	G	the type of the error argument `error`
Parameters (in):	error	the error to check

▽

△

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

⌋

8.23.4.2.3.3 EmplaceError

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	
Syntax:	`template <typename... Args>` `void EmplaceError (Args &&... args) noexcept(std::is_nothrow_constructible< E, Args... >::value);`	
Template param:	Args...	the types of arguments given to this function
Parameters (in):	args	the arguments used for constructing the error
Return value:	None	
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-safe	
Description:	Put a new error into this instance, constructed in-place from the given arguments.	

⌋

8.23.4.2.3.4 EmplaceValue

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	
Syntax:	`template <typename... Args>` `void EmplaceValue (Args &&... args) noexcept;`	
Template param:	Args...	the types of arguments given to this function

▽

△

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

⌋

8.23.4.2.3.5 Err() const &

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.4.2.3.6 Err() &&

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.4.2.3.7 Error() &&

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.4.2.3.8 Error() &

[SWS_CORE_00876] Definition of API function ara::core::Result< void, E >::Error

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.4.2.3.9 Error() const &

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.4.2.3.10 ErrorOr(G&&) &&

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	
Syntax:	`template <typename G>` `E ErrorOr (G &&defaultError) &&noexcept(std::is_nothrow_move_constructible< E >::value &&std::is_nothrow_constructible< E, G && >::value);`	
Template param:	G	the type of defaultError
Parameters (in):	defaultError	the error to use if `*this` does not contain an error
Return value:	E	the error
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return the contained error or the given default error. If `*this` contains an error, it is `std::move`'d into the return value. Otherwise, the specified default error is returned, `static_cast`'d to `E`	

⌋

8.23.4.2.3.11 ErrorOr(G&&) const &

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	
Syntax:	`template <typename G>` `E ErrorOr (G &&defaultError) const &noexcept(std::is_nothrow_copy_constructible< E >::value &&std::is_nothrow_constructible< E, G &&>::value);`	
Template param:	G	the type of defaultError
Parameters (in):	defaultError	the error to use if `*this` does not contain an error
Return value:	E	the error
Exception Safety:	conditionally exception safe	
Thread Safety:	thread-safe	
Description:	Return the contained error or the given default error. If `*this` contains an error, it is returned. Otherwise, the specified default error is returned, `static_cast`'d to `E`	

⌋

8.23.4.2.3.12 HasValue

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.4.2.3.13 Resolve

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	
Syntax:	`template <typename F>` `void Resolve (F &&f) const;`	
Template param:	F	the type of the Callable f
Parameters (in):	f	the Callable
Return value:	None	
Exception Safety:	not exception safe	
Thread Safety:	conditional,	depends on thread safety of f
Description:	Do nothing or call a function. If *this contains a value, this function does nothing. Otherwise, the specified callable is invoked. The Callable is expected to be compatible to this interface: `void f(const E&);` This function only exists for helping with generic programming.	

⌋

8.23.4.2.3.14 Swap

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.4.2.3.15 Value

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	
Syntax:	`void Value () const noexcept;`	
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`ResultWithNoValue-Violation`	If *this does not contain a value
Description:	This function only exists for helping with generic programming	

⌋

8.23.4.2.3.16 ValueOr

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	
Syntax:	`template <typename U>` `void ValueOr (U &&defaultValue) const noexcept;`	
Template param:	U	the type of defaultValue
Parameters (in):	defaultValue	the value to use if *this does not contain a value
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Do nothing. This function only exists for helping with generic programming.	

⌋

8.23.4.2.3.17 ValueOrThrow() const

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function
Header file:	#include "ara/core/result.h"
Scope:	`class ara::core::Result< void, E >`
Syntax:	`void ValueOrThrow () const noexcept(false);`
Return value:	None
Exceptions:	The exception type associated with the contained error
Exception Safety:	not exception safe
Thread Safety:	thread-safe
Description:	Return the contained value or throw an exception.
Notes:	This function shall not participate in overload resolution when C++ exceptions are disabled in the compiler toolchain

⌋

8.23.4.2.3.18 operator bool

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.4.2.3.19 operator*() const

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	
Syntax:	`void operator* () const noexcept;`	
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`ResultWithNoValue-Violation`	If `*this` does not contain a value.
Description:	Access the contained value.	

⌋

8.23.4.2.4 Named Constructors

8.23.4.2.4.1 FromError

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

Upstream requirements: RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/core/result.h"	
Scope:	`class ara::core::Result< void, E >`	
Syntax:	`template <typename... Args>` `static Result FromError (Args &&... args) noexcept(std::is_nothrow_constructible< E, Args... >::value);`	
Template param:	Args...	the types of arguments given to this function
Parameters (in):	args	the parameter pack used for constructing the error
Return value:	Result	a Result that contains an error
Exception Safety:	conditionally exception safe	
Thread Safety:	not thread-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	

⌋

8.23.4.2.4.2 FromError(const E&)

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.4.2.4.3 FromError(E&&)

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

8.23.4.2.4.4 FromValue

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

Upstream requirements: RS_AP_00130

⌈

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

⌋

9 Service Interfaces

This functional cluster does not define any provided or required service interfaces.

10 Configuration

The configuration model of this functional cluster is defined in [17]. This chapter defines the default values for attributes and semantic constraints for elements specified in [17] that are part of the configuration model of this functional cluster.

10.1 Default Values

This functional cluster does not define any default values for attributes specified in [17].

10.2 Semantic Constraints

This functional cluster does not define any semantic constraints for elements specified in [17].

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**			
Note	This meta-class represents the ability to define a global error domain for an ApApplicationError. **Tags:** atp.recommendedPackage=ApplicationErrorDomains This Class is only used by the AUTOSAR Adaptive Platform.			
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	**CompositionSwComponentType**			
Note	A `CompositionSwComponentType` aggregates `SwComponentPrototype`s (that in turn are typed by `SwComponentType`)s as well as `SwConnector`s for primarily connecting `SwComponentPrototype`s among each others and towards the surface of the `CompositionSwComponentType`. By this means, a hierarchical structures of software-components can be created. **Tags:** atp.recommendedPackage=SwComponentTypes			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
component	SwComponent Prototype	*	aggr	The instantiated components that are part of this composition. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=component.shortName, component.variation Point.shortLabel vh.latestBindingTime=postBuild
connector	SwConnector	*	aggr	`SwConnector`s have the principal ability to establish a connection among `PortPrototype`s. They can have many roles in the context of a `CompositionSwComponentType`. Details are refined by subclasses. The aggregation of `SwConnector`s is subject to variability with the purpose to support variant data flow. The aggregation is marked as atpSplitable in order to allow the extension of the ECU extract with `AssemblySwConnector`s between `ApplicationSwComponentType`s and `ServiceSwComponentType`s during the ECU integration. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=connector.shortName, connector.variation Point.shortLabel vh.latestBindingTime=postBuild

▽

△

Class	**CompositionSwComponentType**			
constantValue Mapping	ConstantSpecification MappingSet	*	ref	Reference to the `ConstantSpecificationMapping` to be applied for initValues of `PPortComSpec`s and `RPortComSpec`. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=constantValueMapping
dataType Mapping	DataTypeMappingSet	*	ref	Reference to the `DataTypeMapping` to be applied for the used `ApplicationDataType`s in `ServiceInterface`s. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=dataTypeMapping
physical Dimension Mapping	PhysicalDimension MappingSet	0..1	ref	This reference identifies the `PhysicalDimensionMappingSet` that is applicable in the context of the enclosing `CompositionSwComponentType`. The `PhysicalDimensionMapping`s contained in the `PhysicalDimensionMappingSet` shall be taken into account for the assessment of the compatibility of `PhysicalDimension`s in the context of creation of a `PortInterfaceMapping` in the scope of the `CompositionSwComponentType`.

Table A.2: CompositionSwComponentType

Class	***CppImplementationDataType*** (abstract)			
Note	This meta-class represents the way to specify a reusable data type definition taken as a the basis for a C++ language binding This Class is only used by the AUTOSAR Adaptive Platform.			
Base	*ARElement*, *ARObject*, *AbstractImplementationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *CppImplementationDataTypeContextTarget*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	CustomCppImplementationDataType, StdCppImplementationDataType			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
arraySize	PositiveInteger	0..1	attr	This attribute can be used to specify the array size if the enclosing CppImplementationDataType has array semantics. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
headerFile	String	0..1	attr	Configuration of the Header File with the custom class declaration.
namespace (ordered)	SymbolProps	*	aggr	This aggregation allows for the definition an own namespace for the enclosing CppImplementationData Type.
subElement (ordered)	CppImplementation DataTypeElement	*	aggr	This represents the collection of sub-elements of the enclosing CppImplementationDataType
template Argument (ordered)	CppTemplateArgument	*	aggr	This aggregation allows for the specification of properties of template arguments
typeEmitter	NameToken	0..1	attr	This attribute can be taken to control how the respective CppImplementationDataType is contributed to the language binding.
typeReference	CppImplementation DataType	0..1	ref	This reference shall be defined to define a type reference (a.k.a. typedef).

Table A.3: CppImplementationDataType

Class	**Executable**			
Note	This meta-class represents an executable program. **Tags:** atp.recommendedPackage=Executables This Class is only used by the AUTOSAR Adaptive Platform.			
Base	*ARElement*, *ARObject*, *AtpClassifier*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDesignElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
implementation Props	Executable ImplementationProps	*	aggr	This aggregation contains the collection of implementation-specific properties necessary to properly build the enclosing Executable.
minimumTimer Granularity	TimeValue	0..1	attr	This attribute describes the minimum timer resolution (TimeValue of one tick) that is required by the Executable.
reporting Behavior	ExecutionState ReportingBehavior Enum	0..1	attr	this attribute controls the execution state reporting behavior of the enclosing Executable.
rootSw Component Prototype	RootSwComponent Prototype	0..1	aggr	This represents the root SwCompositionPrototype of the Executable. This aggregation is required (in contrast to a direct reference of a SwComponentType) in order to support the definition of instanceRefs in Executable context.
suspendToRam Awareness	SuspendToRam AwarenessEnum	0..1	attr	This attribute describes the type of awareness of the enclosing Executable to suspend-to-RAM functionality. **Tags:** atp.Status=candidate
version	StrongRevisionLabel String	0..1	attr	Version of the executable.

Table A.4: Executable

Class	***FunctionalClusterInteractsWithFunctionalClusterMapping*** (abstract)			
Note	This meta-class identifies a relation between functional clusters on the adaptive platform such one functional cluster can call APIs of the other functional cluster. This Class is only used by the AUTOSAR Adaptive Platform.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableDeploymentElement*, *UploadablePackageElement*			
Subclasses	ArtifactChecksumToCryptoProviderMapping, ComCertificateToCryptoCertificateMapping, ComKeyTo CryptoKeySlotMapping, ComSecOcToCryptoKeySlotMapping, FunctionalClusterInteractsWithDiagnostic EventMapping, FunctionalClusterInteractsWithPersistencyDeploymentMapping, FunctionalClusterToDlt LogSinkMapping, FunctionalClusterToSecurityEventDefinitionMapping, NmInteractsWithSmMapping, PersistencyDeploymentElementToCryptoKeySlotMapping, PersistencyDeploymentToCryptoKeySlot Mapping, PersistencyDeploymentToDltLogSinkMapping, SmInteractsWithNmMapping, TimeBase ProviderToPersistencyMapping, UcmToTimeBaseResourceMapping			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table A.5: FunctionalClusterInteractsWithFunctionalClusterMapping

Class	***Identifiable*** (abstract)
Note	Instances of this class can be referred to by their identifier (within the namespace borders). In addition to this, Identifiables are objects which contribute significantly to the overall structure of an AUTOSAR description. In particular, Identifiables might contain Identifiables.
Base	*ARObject*, *MultilanguageReferrable*, *Referrable*

▽

△

Class	**Identifiable** (abstract)			
Subclasses	ARPackage, *AbstractDoIpLogicAddressProps*, *AbstractEvent*, *AbstractFunctionalClusterDesign*, *AbstractImplementationDataTypeElement*, *AbstractSecurityEventFilter*, *AbstractSecurityIdsmInstance Filter*, *AbstractServiceInstance*, *AbstractSignalBasedToISignalTriggeringMapping*, AdaptiveSwcInternal Behavior, ApApplicationEndpoint, *ApmcAbstractDefinition*, *ApmcConfigurationElementDef*, *Apmc ContainerElementValue*, ApmcContainerValue, ApmcEnumerationLiteralDef, ApplicationEndpoint, ApplicationError, AppliedStandard, ArtifactChecksum, ArtifactLocator, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpFeature*, AutosarOperationArgumentInstance, AutosarVariableInstance, *BuildAction Entity*, BuildActionEnvironment, Chapter, CheckpointTransition, ClientIdDefinition, ClientServer Operation, Code, *CollectableElement*, ComManagementMapping, *CommConnectorPort*, *CommunicationConnector*, *CommunicationController*, Compiler, ConsistencyNeeds, ConsumedEvent Group, CouplingPort, *CouplingPortAbstractShaper*, *CouplingPortStructuralElement*, CryptoCertificate, CryptoCertificateGroup, CryptoKeySlot, CryptoKeySlotDesign, CryptoKeySlotUsageDesign, Crypto Provider, *CryptoServiceMapping*, DataPrototypeGroup, DataPrototypeTransformationPropsIdent, Data Transformation, *DdsAbstractServiceInstanceElementCp*, DdsCpDomain, DdsCpPartition, DdsCpQos Profile, DdsCpTopic, DdsDomainRange, DependencyOnArtifact, *DiagEventDebounceAlgorithm*, *DiagnosticAbstractSovdContent*, DiagnosticAuthTransmitCertificateEvaluation, DiagnosticConnected Indicator, DiagnosticDataElement, DiagnosticDebounceAlgorithmProps, DiagnosticExtendedDataRecord Element, DiagnosticFunctionInhibitSource, DiagnosticParameterElement, *DiagnosticRoutineSubfunction*, DiagnosticSovdMethodPrimitive, DltApplication, DltArgument, DltArgumentProps, DltMessage, DoIp Interface, DoIpLogicAddress, DoIpLogicalAddress, DoIpNetworkConfigurationDesign, DoIpRouting Activation, E2EProfileConfiguration, End2EndEventProtectionProps, End2EndMethodProtectionProps, EthernetWakeupSleepOnDatalineConfig, EventHandler, EventMapping, ExclusiveArea, *Executable Entity*, *ExecutionTime*, FMAttributeDef, FMFeatureMapAssertion, FMFeatureMapCondition, FMFeature MapElement, FMFeatureRelation, FMFeatureRestriction, FMFeatureSelection, FieldMapping, FireAnd ForgetMethodMapping, FlexrayArTpNode, FlexrayTpPduPool, *FrameTriggering*, GeneralParameter, GlobalSupervision, GlobalTimeGateway, *GlobalTimeMaster*, *GlobalTimeSlave*, *HealthChannel*, *Heap Usage*, HwAttributeDef, HwAttributeLiteralDef, HwPin, HwPinGroup, *IEEE1722TpAcfBus*, *IEEE1722Tp AcfBusPart*, IPSecRule, IPv6ExtHeaderFilterList, ISignalToIPduMapping, ISignalTriggering, *Ident Caption*, ImpositionTime, InternalTriggeringPoint, Keyword, LifeCycleState, Linker, MacAddressVlan Membership, MacMulticastGroup, MacSecKayParticipant, McDataInstance, MemorySection, Memory Usage, MethodMapping, ModeDeclaration, ModeDeclarationMapping, ModeSwitchPoint, ModeSwitch SenderComSpecProps, NetworkEndpoint, *NmCluster*, *NmNode*, *PackageableElement*, Parameter Access, PduActivationRoutingGroup, PduToFrameMapping, PduTriggering, PerInstanceMemory, *PersistencyDeploymentElement*, *PersistencyInterfaceElement*, *PhmSupervision*, *PhysicalChannel*, Port Group, *PortInterfaceMapping*, ProcessToMachineMapping, Processor, ProcessorCore, PskIdentityToKey SlotMapping, QueuedReceiverComSpecProps, ResourceConsumption, ResourceGroup, RootSwCluster DesignComponentPrototype, RootSwComponentPrototype, RootSwCompositionPrototype, Rpt Component, RptContainer, RptExecutableEntity, RptExecutableEntityEvent, RptExecutionContext, Rpt Profile, RptServicePoint, RunnableEntityGroup, *SdgAttribute*, SdgClass, SecOcJobMapping, SecOcJob Requirement, SecureCommunicationAuthenticationProps, *SecureCommunicationDeployment*, Secure CommunicationFreshnessProps, SecurityEventContextDataElement, SecurityEventContextProps, Server ComSpecProps, *ServiceInterfaceDeploymentElement*, ServiceInterfaceElementSecureComConfig, *ServiceNeeds*, SignalServiceTranslationEventProps, SignalServiceTranslationProps, SocketAddress, SoftwarePackageStep, SomeipEventGroup, SomeipProvidedEventGroup, SomeipTpChannel, *Stack Usage*, *StateManagementActionItem*, StateManagementActionList, StateManagementStateNotification, *StateManagementStateRequest*, StaticSocketConnection, StructuredReq, SupervisionCheckpoint, SupervisionMode, SupervisionModeCondition, SwGenericAxisParamType, SwServiceArg, SwcService Dependency, SystemMapping, *TimeBaseResource*, *TimingClock*, TimingClockSyncAccuracy, Timing Condition, *TimingConstraint*, *TimingDescription*, TimingExtensionResource, TimingModeInstance, Tls CryptoCipherSuite, TlsCryptoCipherSuiteProps, TlsJobMapping, Topic1, TpAddress, TraceableTable, TraceableText, *TracedFailure*, TransformationISignalPropsIdent, *TransformationProps*, Transformation Technology, Trigger, UcmDescription, UcmRetryStrategy, UcmStep, UriDescription, VariableAccess, VariationPointProxy, VehicleRolloutStep, ViewMap, VlanConfig, WaitPoint			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
adminData	AdminData	0..1	aggr	This represents the administrative data for the identifiable object. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=adminData xml.sequenceOffset=-40

▽

△

Class	***Identifiable*** (abstract)			
annotation	Annotation	*	aggr	Possibility to provide additional notes while defining a model element (e.g. the ECU Configuration Parameter Values). These are not intended as documentation but are mere design notes. **Tags:** xml.sequenceOffset=-25
category	CategoryString	0..1	attr	The category is a keyword that specializes the semantics of the Identifiable. It affects the expected existence of attributes and the applicability of constraints. **Tags:** xml.sequenceOffset=-50
desc	MultiLanguageOverview Paragraph	0..1	aggr	This represents a general but brief (one paragraph) description what the object in question is about. It is only one paragraph! Desc is intended to be collected into overview tables. This property helps a human reader to identify the object in question. More elaborate documentation, (in particular how the object is built or used) should go to "introduction". **Tags:** xml.sequenceOffset=-60
introduction	DocumentationBlock	0..1	aggr	This represents more information about how the object in question is built or is used. Therefore it is a DocumentationBlock. **Tags:** xml.sequenceOffset=-30
uuid	String	0..1	attr	The purpose of this attribute is to provide a globally unique identifier for an instance of a meta-class. The values of this attribute should be globally unique strings prefixed by the type of identifier. For example, to include a DCE UUID as defined by The Open Group, the UUID would be preceded by "DCE:". The values of this attribute may be used to support merging of different AUTOSAR models. The form of the UUID (Universally Unique Identifier) is taken from a standard defined by the Open Group (was Open Software Foundation). This standard is widely used, including by Microsoft for COM (GUIDs) and by many companies for DCE, which is based on CORBA. The method for generating these 128-bit IDs is published in the standard and the effectiveness and uniqueness of the IDs is not in practice disputed. If the id namespace is omitted, DCE is assumed. An example is "DCE:2fac1234-31f8-11b4-a222-08002b34c003". The uuid attribute has no semantic meaning for an AUTOSAR model and there is no requirement for AUTOSAR tools to manage the timestamp. **Tags:** xml.attribute=true

Table A.6: Identifiable

Class	***PortInterface*** (abstract)			
Note	Abstract base class for an interface that is either provided or required by a port of a software component.			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	*AbstractRawDataStreamInterface*, *AbstractSuspendToRamInterface*, *AbstractSynchronizedTimeBase Interface*, ClientServerInterface, *CryptoInterface*, *DataInterface*, *DiagnosticPortInterface*, FirewallState SwitchInterface, *IdsmAbstractPortInterface*, LogAndTraceInterface, ModeSwitchInterface, Network ManagementPortInterface, *PersistencyInterface*, *PlatformHealthManagementInterface*, ServiceInterface, *StateManagementPortInterface*, TriggerInterface			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	***PortInterface*** (abstract)			
namespace (ordered)	SymbolProps	*	aggr	This represents the SymbolProps used for the definition of a hierarchical namespace applicable for the generation of code artifacts out of the definition of a ServiceInterface. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=namespace.shortName This Attribute is only used by the AUTOSAR Adaptive Platform.

Table A.7: PortInterface

Class	***PortPrototype*** (abstract)			
Note	Base class for the ports of an AUTOSAR software component. The aggregation of PortPrototypes is subject to variability with the purpose to support the conditional existence of ports.			
Base	*ARObject*, *AtpBlueprintable*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	*AbstractProvidedPortPrototype*, *AbstractRequiredPortPrototype*			
Aggregated by	*AtpClassifier*.atpFeature, *SwComponentType*.port			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
clientServer Annotation	ClientServerAnnotation	*	aggr	Annotation of this `PortPrototype` with respect to client/ server communication.
delegatedPort Annotation	DelegatedPort Annotation	0..1	aggr	Annotations on this delegated port.
ioHwAbstraction Server Annotation	IoHwAbstractionServer Annotation	*	aggr	Annotations on this IO Hardware Abstraction port.
modePort Annotation	ModePortAnnotation	*	aggr	Annotations on this mode port.
nvDataPort Annotation	NvDataPortAnnotation	*	aggr	Annotations on this non voilatile data port.
parameterPort Annotation	ParameterPort Annotation	*	aggr	Annotations on this parameter port.
portPrototype Props	PortPrototypeProps	0..1	aggr	This attribute allows for the definition of further qualification of the semantics of a PortPrototype. This Attribute is only used by the AUTOSAR Adaptive Platform.
senderReceiver Annotation	SenderReceiver Annotation	*	aggr	Collection of annotations of this ports sender/receiver communication. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=senderReceiverAnnotation
triggerPort Annotation	TriggerPortAnnotation	*	aggr	Annotations on this trigger port.

Table A.8: PortPrototype

Class	***Referrable*** (abstract)			
Note	Instances of this class can be referred to by their identifier (while adhering to namespace borders).			
Base	*ARObject*			
Subclasses	*AtpDefinition*, BswDistinguishedPartition, *BswModuleCallPoint*, BswModuleClientServerEntry, Bsw VariableAccess, CouplingPortTrafficClassAssignment, *CppImplementationDataTypeContextTarget*, *DiagnosticEnvModeElement*, EthernetPriorityRegeneration, ExclusiveAreaNestingOrder, *HwDescription Entity*, *ImplementationProps*, ModeTransition, *MultilanguageReferrable*, NmNetworkHandle, Pnc MappingIdent, *SingleLanguageReferrable*, SoConIPduIdentifier, SomeipRequiredEventGroup, Tp ConnectionIdent			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	***Referrable*** (abstract)			
shortName	Identifier	1	attr	This specifies an identifying shortName for the object. It needs to be unique within its context and is intended for humans but even more for technical reference. **Stereotypes:** atpIdentityContributor **Tags:** xml.enforceMinMultiplicity=true xml.sequenceOffset=-100
shortName Fragment	ShortNameFragment	*	aggr	This specifies how the Referrable.shortName is composed of several shortNameFragments. **Tags:** xml.sequenceOffset=-90

Table A.9: Referrable

Class	**RootSwComponentPrototype**			
Note	The RootSwCompositionPrototype represents the top-level-composition of software components within an Executable. The contained SwComponentPrototypes are fully specified by their SwComponentTypes (including Port Prototypes, PortInterfaces, VariableDataPrototypes, etc.). This Class is only used by the AUTOSAR Adaptive Platform.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, Executable.rootSwComponentPrototype			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
applicationType	SwComponentType	0..1	tref	This SwComponentType acts as the Type of the RootSw ComponentPrototype. **Stereotypes:** isOfType

Table A.10: RootSwComponentPrototype

Class	**StdCppImplementationDataType**			
Note	This meta-class represents the way to specify a data type definition that is taken as the basis for a C++ language binding to a C++ Standard Library feature. **Tags:** atp.recommendedPackage=CppImplementationDataTypes This Class is only used by the AUTOSAR Adaptive Platform.			
Base	*ARElement*, *ARObject*, *AbstractImplementationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *CppImplementationDataType*, *CppImplementationData TypeContextTarget*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.11: StdCppImplementationDataType

Class	**SwComponentPrototype**			
Note	Role of a software component within a composition.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, CompositionSwComponentType.component			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
type	SwComponentType	0..1	tref	Type of the instance. **Stereotypes:** isOfType

Table A.12: SwComponentPrototype

B Demands and constraints on Base Software (normative)

This functional cluster defines no demands or constraints for the Base Software on which the AUTOSAR Adaptive Platform is running on (usually a POSIX-compatible operating system).

C Platform Extension Interfaces (normative)

This functional cluster does not specify any Platform Extension Interfaces.

D Implementation-defined and undefined behavior (informative)

This chapter shows the summary of implementation-defined behavior (IDB) and undefined behavior (UB) from chapters 7 and 8

D.1 Implementation-defined behavior (IDB)

Specification Item	Title	Description
[SWS_CORE_15005]	ARA behavior before `ara::core::Initialize`	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.
[SWS_CORE_90022]	ARA behavior after `ara::core::Deinitialize`	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.
[SWS_CORE_00090]	Handling of `Standardized Violations`	Which Log and Trace Context Identifier ([SWS_CORE_90024]) is chosen is implementation-defined
[SWS_CORE_00004]	Handling of Corruptions	If a `Corruption` is detected, it shall result in unsuccessful process termination, in an implementation-defined way.
[SWS_CORE_01414]	Definition of API type ara::core::Map::const_iterator	`{<const-iterator-type>}` shall be an implementation-defined type
[SWS_CORE_01412]	Definition of API type ara::core::Map::difference_type	`{<difference-type>}` shall be an implementation-defined type
[SWS_CORE_01418]	Definition of API type ara::core::Map::insert_return_type	`{<insert-return-type>}` shall be an implementation-defined type
[SWS_CORE_01413]	Definition of API type ara::core::Map::iterator	`{<iterator-type>}` shall be an implementation-defined type
[SWS_CORE_01417]	Definition of API type ara::core::Map::node_type	`{<node-type>}` shall be an implementation-defined type
[SWS_CORE_01411]	Definition of API type ara::core::Map::size_type	`{<size-type>}` shall be an implementation-defined type
[SWS_CORE_01918]	Definition of API type ara::core::Span::const_iterator	`{<const-iterator-type>}` shall be an implementation-defined type
[SWS_CORE_01917]	Definition of API type ara::core::Span::iterator	`{<iterator-type>}` shall be an implementation-defined type
[SWS_CORE_02105]	Definition of API type ara::core::StringView::const_iterator	`{<const-iterator-type>}` shall be an implementation-defined type
[SWS_CORE_00072]	Definition of API type ara::core::BasicString::const_iterator	`{<const-iterator-type>}` shall be an implementation-defined type
[SWS_CORE_00071]	Definition of API type ara::core::BasicString::iterator	`{<iterator-type>}` shall be an implementation-defined type
[SWS_CORE_01311]	Definition of API type ara::core::Vector::const_iterator	`{<const-iterator-type>}` shall be an implementation-defined type

▽

△

Specification Item	Title	Description
[SWS_CORE_01309]	Definition of API type ara::core::Vector::difference_type	{<difference-type>} shall be an implementation-defined type
[SWS_CORE_01310]	Definition of API type ara::core::Vector::iterator	{<iterator-type>} shall be an implementation-defined type
[SWS_CORE_01308]	Definition of API type ara::core::Vector::size_type	{<size-type>} shall be an implementation-defined type
[SWS_CORE_01315]	Definition of API function ara::core::Vector::operator=	{<may-throw-on-allocate>} shall be an implementation-defined type
[SWS_CORE_00054]	Definition of API function ara::core::AddAbortHandler	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
[SWS_CORE_00051]	Definition of API function ara::core::SetAbortHandler	This function can be called from multiple threads simultaneously; these calls are performed in an implementation-defined sequence
[SWS_CORE_00361]	Definition of API enum ara::core::FutureStatus	The numerical values of the enum literals are implementation-defined
[SWS_CORE_04200]	Definition of API type ara::core::Byte	{<byte-type>} shall be an implementation-defined type

Table D.1: Summary of implementation defined behavior

D.2 Undefined behavior (UB)

Specification Item	Title	Description
[SWS_CORE_13203]	Behavior of a concurrently executed non-thread-safe member function	If a non-thread-safe member function is called concurrently on the same object to itself or other functions of that object, the behavior is undefined
[SWS_CORE_13204]	Behavior of a concurrently executed non-thread-safe non-member function	If a non-thread-safe non-member function is called concurrently, the behavior is undefined
[SWS_CORE_13210]	ARA API usage within a signal handler	Usage of any ARA API within a signal handler will result in undefined behavior of the application, unless otherwise specified
[SWS_CORE_12409]	Usage of ARA API within AbortHandlers	Usage of any ARA API within an ara::core::AbortHandler may result in undefined behavior of the application, unless otherwise specified
[SWS_CORE_10103]	Creation of ara::core::Byte instances	If the initializer value is outside the value range of type ara::core::Byte (see [SWS_CORE_10102]), the behavior is undefined
[SWS_CORE_01251]	Definition of API function ara::core::Array::begin	De-referencing the returned iterator on an empty() array is undefined behavior
[SWS_CORE_01250]	Definition of API function ara::core::Array::begin	De-referencing the returned iterator on an empty() array is undefined behavior
[SWS_CORE_01962]	Definition of API function ara::core::Span::first	The behavior of this function is undefined if: (count > size())
[SWS_CORE_01961]	Definition of API function ara::core::Span::first	The behavior of this function is undefined if: (Count > size())

▽

△

Specification Item	Title	Description
[SWS_CORE_01964]	Definition of API function ara::core::Span::last	The behavior of this function is undefined if: `(count > size())`
[SWS_CORE_01963]	Definition of API function ara::core::Span::last	The behavior of this function is undefined if: `(Count > size())`
[SWS_CORE_01966]	Definition of API function ara::core::Span::subspan	The behavior of this function is undefined unless: `(offset <= size() && (count == dynamic_extent || count <= size() - offset)) == true`
[SWS_CORE_02135]	Definition of API function ara::core::String View::back	The behavior of this function is undefined if: (`empty()` == true)
[SWS_CORE_02134]	Definition of API function ara::core::String View::front	The behavior of this function is undefined if: (`empty()` == true)
[SWS_CORE_02132]	Definition of API function ara::core::String View::operator[]	The behavior of this function is undefined if: (pos >= `size()`)
[SWS_CORE_02137]	Definition of API function ara::core::String View::remove_prefix	The behavior of this function is undefined if: (n > `size()`)
[SWS_CORE_02138]	Definition of API function ara::core::String View::remove_suffix	The behavior of this function is undefined if: (n > `size()`)
[SWS_CORE_02139]	Definition of API function ara::core::String View::swap	The behavior of this function is undefined if: (n > `size()`)

Table D.2: Summary of undefined behavior

E Not implemented requirements

This functional cluster implements all functional requirements specified in the corresponding requirement specifications.

F Change History

Please note that the lists in this chapter also include specification items that have been removed from the specification in a later version. These specification items do not appear as hyperlinks in the document.

F.1 Change History of this document according to AUTOSAR Release R19-11

F.1.1 Added Specification Items in R19-11

Number	Heading
[SWS_CORE_00003]	Handling of Violations
[SWS_CORE_00004]	Handling of Corruptions
[SWS_CORE_00005]	Handling of failed default allocations
[SWS_CORE_00014]	The Core error domain
[SWS_CORE_00050]	
[SWS_CORE_00051]	
[SWS_CORE_00052]	
[SWS_CORE_00131]	
[SWS_CORE_00132]	
[SWS_CORE_00133]	
[SWS_CORE_00134]	
[SWS_CORE_00135]	
[SWS_CORE_00136]	
[SWS_CORE_00137]	
[SWS_CORE_00138]	
[SWS_CORE_00151]	
[SWS_CORE_00152]	
[SWS_CORE_00153]	
[SWS_CORE_00154]	
[SWS_CORE_00322]	
[SWS_CORE_00323]	
[SWS_CORE_00325]	
[SWS_CORE_00326]	
[SWS_CORE_00327]	
[SWS_CORE_00328]	
[SWS_CORE_00329]	
[SWS_CORE_00330]	

▽

△

Number	Heading
[SWS_CORE_00331]	
[SWS_CORE_00332]	
[SWS_CORE_00333]	
[SWS_CORE_00334]	
[SWS_CORE_00335]	
[SWS_CORE_00336]	
[SWS_CORE_00341]	
[SWS_CORE_00342]	
[SWS_CORE_00343]	
[SWS_CORE_00344]	
[SWS_CORE_00345]	
[SWS_CORE_00346]	
[SWS_CORE_00349]	
[SWS_CORE_00350]	
[SWS_CORE_00351]	
[SWS_CORE_00352]	
[SWS_CORE_00353]	
[SWS_CORE_00354]	
[SWS_CORE_00412]	
[SWS_CORE_00441]	
[SWS_CORE_00442]	
[SWS_CORE_00443]	
[SWS_CORE_00444]	
[SWS_CORE_00480]	
[SWS_CORE_00490]	
[SWS_CORE_00512]	
[SWS_CORE_00513]	
[SWS_CORE_00514]	
[SWS_CORE_00515]	
[SWS_CORE_00516]	
[SWS_CORE_00518]	
[SWS_CORE_00519]	
[SWS_CORE_00571]	
[SWS_CORE_00572]	
[SWS_CORE_00611]	
[SWS_CORE_00612]	
[SWS_CORE_00613]	
[SWS_CORE_00721]	

▽

△

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_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]	

▽

△

Number	Heading
[SWS_CORE_00784]	
[SWS_CORE_00785]	
[SWS_CORE_00786]	
[SWS_CORE_00787]	
[SWS_CORE_00788]	
[SWS_CORE_00789]	
[SWS_CORE_00796]	
[SWS_CORE_00821]	
[SWS_CORE_00823]	
[SWS_CORE_00824]	
[SWS_CORE_00825]	
[SWS_CORE_00826]	
[SWS_CORE_00827]	
[SWS_CORE_00831]	
[SWS_CORE_00834]	
[SWS_CORE_00835]	
[SWS_CORE_00836]	
[SWS_CORE_00841]	
[SWS_CORE_00842]	
[SWS_CORE_00843]	
[SWS_CORE_00844]	
[SWS_CORE_00845]	
[SWS_CORE_00851]	
[SWS_CORE_00852]	
[SWS_CORE_00853]	
[SWS_CORE_00855]	
[SWS_CORE_00857]	
[SWS_CORE_00858]	
[SWS_CORE_00861]	
[SWS_CORE_00863]	
[SWS_CORE_00865]	
[SWS_CORE_00866]	
[SWS_CORE_00867]	
[SWS_CORE_01941]	
[SWS_CORE_01942]	
[SWS_CORE_01943]	
[SWS_CORE_01944]	
[SWS_CORE_01945]	

▽

△

Number	Heading
[SWS_CORE_01946]	
[SWS_CORE_01947]	
[SWS_CORE_01948]	
[SWS_CORE_01949]	
[SWS_CORE_01950]	
[SWS_CORE_01951]	
[SWS_CORE_01952]	
[SWS_CORE_01961]	
[SWS_CORE_01962]	
[SWS_CORE_01963]	
[SWS_CORE_01964]	
[SWS_CORE_01965]	
[SWS_CORE_01966]	
[SWS_CORE_01967]	
[SWS_CORE_01968]	
[SWS_CORE_01969]	
[SWS_CORE_01970]	
[SWS_CORE_01971]	
[SWS_CORE_01972]	
[SWS_CORE_01973]	
[SWS_CORE_01974]	
[SWS_CORE_01975]	
[SWS_CORE_01976]	
[SWS_CORE_01977]	
[SWS_CORE_01978]	
[SWS_CORE_01979]	
[SWS_CORE_01990]	
[SWS_CORE_01991]	
[SWS_CORE_01992]	
[SWS_CORE_01993]	
[SWS_CORE_01994]	
[SWS_CORE_03000]	`BasicString` type
[SWS_CORE_04012]	
[SWS_CORE_04022]	
[SWS_CORE_04032]	
[SWS_CORE_04110]	
[SWS_CORE_04111]	
[SWS_CORE_04112]	

▽

△

Number	Heading
[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_06340]	
[SWS_CORE_06341]	
[SWS_CORE_06342]	
[SWS_CORE_06343]	

▽

△

Number	Heading
[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_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 F.1: Added Specification Items in R19-11

F.1.2 Changed Specification Items in R19-11

Number	Heading
[SWS_CORE_00002]	Handling of Errors
[SWS_CORE_00040]	Errors originating from C++ standard classes
[SWS_CORE_03001]	`String` type
[SWS_CORE_03296]	`swap` overload for `BasicString`
[SWS_CORE_03301]	Implicit conversion to `StringView`
[SWS_CORE_03302]	Constructor from `StringView`
[SWS_CORE_03303]	Constructor from implicit `StringView`
[SWS_CORE_03304]	operator= from `StringView`
[SWS_CORE_03305]	Assignment from `StringView`
[SWS_CORE_03306]	Assignment from implicit `StringView`
[SWS_CORE_03307]	operator+ from `StringView`
[SWS_CORE_03308]	Concatenation of `StringView`
[SWS_CORE_03309]	Concatenation of implicit `StringView`
[SWS_CORE_03310]	Insertion of `StringView`
[SWS_CORE_03311]	Insertion of implicit `StringView`
[SWS_CORE_03312]	Replacement with `StringView`
[SWS_CORE_03313]	Replacement with implicit `StringView`
[SWS_CORE_03314]	Replacement of iterator range with `StringView`
[SWS_CORE_03315]	Forward-find a `StringView`
[SWS_CORE_03316]	Reverse-find a `StringView`
[SWS_CORE_03317]	Forward-find of character set within a `StringView`
[SWS_CORE_03318]	Reverse-find of character set within a `StringView`
[SWS_CORE_03319]	Forward-find of character set not within a `StringView`
[SWS_CORE_03320]	Reverse-find of character set not within a `StringView`
[SWS_CORE_03321]	Comparison with a `StringView`
[SWS_CORE_03322]	Comparison of subsequence with a `StringView`
[SWS_CORE_03323]	Comparison of subsequence with a subsequence of a `StringView`

Table F.2: Changed Specification Items in R19-11

F.1.3 Deleted Specification Items in R19-11

Number	Heading
[SWS_CORE_00001]	Handling of Fatal Errors
[SWS_CORE_00012]	The POSIX error domain

Table F.3: Deleted Specification Items in R19-11

F.2 Change History of this document according to AUTOSAR Release R20-11

F.2.1 Added Specification Items in R20-11

Number	Heading
[SWS_CORE_00011]	AUTOSAR error domain range
[SWS_CORE_00016]	Vendor-defined error domain range
[SWS_CORE_00053]	
[SWS_CORE_00337]	
[SWS_CORE_00355]	
[SWS_CORE_00356]	
[SWS_CORE_00614]	
[SWS_CORE_00764]	
[SWS_CORE_00770]	
[SWS_CORE_00771]	
[SWS_CORE_00772]	
[SWS_CORE_00773]	
[SWS_CORE_00864]	
[SWS_CORE_00868]	
[SWS_CORE_00869]	
[SWS_CORE_00870]	
[SWS_CORE_01210]	
[SWS_CORE_01211]	
[SWS_CORE_01212]	
[SWS_CORE_01213]	
[SWS_CORE_01214]	
[SWS_CORE_01215]	
[SWS_CORE_01216]	
[SWS_CORE_01217]	
[SWS_CORE_01218]	
[SWS_CORE_01219]	
[SWS_CORE_01220]	
[SWS_CORE_01241]	
[SWS_CORE_01242]	
[SWS_CORE_01250]	
[SWS_CORE_01251]	
[SWS_CORE_01252]	
[SWS_CORE_01253]	

▽

△

Number	Heading
[SWS_CORE_01254]	
[SWS_CORE_01255]	
[SWS_CORE_01256]	
[SWS_CORE_01257]	
[SWS_CORE_01258]	
[SWS_CORE_01259]	
[SWS_CORE_01260]	
[SWS_CORE_01261]	
[SWS_CORE_01262]	
[SWS_CORE_01263]	
[SWS_CORE_01264]	
[SWS_CORE_01265]	
[SWS_CORE_01266]	
[SWS_CORE_01267]	
[SWS_CORE_01268]	
[SWS_CORE_01269]	
[SWS_CORE_01270]	
[SWS_CORE_01271]	
[SWS_CORE_01272]	
[SWS_CORE_01280]	
[SWS_CORE_01281]	
[SWS_CORE_01282]	
[SWS_CORE_01283]	
[SWS_CORE_01284]	
[SWS_CORE_01285]	
[SWS_CORE_01290]	
[SWS_CORE_01291]	
[SWS_CORE_01292]	
[SWS_CORE_01293]	
[SWS_CORE_01294]	
[SWS_CORE_01295]	
[SWS_CORE_01980]	
[SWS_CORE_01981]	
[SWS_CORE_04023]	
[SWS_CORE_04033]	
[SWS_CORE_06237]	
[SWS_CORE_06355]	
[SWS_CORE_06356]	

▽

△

Number	Heading
[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
[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

▽

△

Number	Heading
[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 F.4: Added Specification Items in R20-11

F.2.2 Changed Specification Items in R20-11

Number	Heading
[SWS_CORE_00010]	Error domain identifier
[SWS_CORE_00050]	
[SWS_CORE_00051]	
[SWS_CORE_00052]	
[SWS_CORE_00110]	
[SWS_CORE_00121]	
[SWS_CORE_00122]	
[SWS_CORE_00123]	
[SWS_CORE_00131]	
[SWS_CORE_00132]	
[SWS_CORE_00133]	
[SWS_CORE_00134]	
[SWS_CORE_00135]	
[SWS_CORE_00136]	
[SWS_CORE_00137]	
[SWS_CORE_00138]	
[SWS_CORE_00151]	
[SWS_CORE_00152]	
[SWS_CORE_00153]	
[SWS_CORE_00154]	

▽

△

Number	Heading
[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]	
[SWS_CORE_00432]	
[SWS_CORE_00441]	
[SWS_CORE_00442]	
[SWS_CORE_00443]	

▽

△

Number	Heading
[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]	
[SWS_CORE_00743]	
[SWS_CORE_00744]	
[SWS_CORE_00745]	

▽

△

Number	Heading
[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]	
[SWS_CORE_00827]	
[SWS_CORE_00831]	

▽

△

Number	Heading
[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]	
[SWS_CORE_01919]	

▽

△

Number	Heading
[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]	

▽

△

Number	Heading
[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]	
[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

▽

△

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

Table F.5: Changed Specification Items in R20-11

F.2.3 Deleted Specification Items in R20-11

none

F.3 Change History of this document according to AUTOSAR Release R21-11

F.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]	

▽

△

Number	Heading
[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_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 F.6: Added Specification Items in R21-11

F.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]	
[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]	

△

Number	Heading
[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]	

▽

△

Number	Heading
[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]	
[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]	

▽

△

Number	Heading
[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_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]	

▽

△

Number	Heading
[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_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]	

▽

△

Number	Heading
[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_01296]	
[SWS_CORE_01900]	
[SWS_CORE_01901]	
[SWS_CORE_01911]	
[SWS_CORE_01912]	
[SWS_CORE_01914]	

▽

△

Number	Heading
[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]	
[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]	

▽

△

Number	Heading
[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]	
[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]	

▽

△

Number	Heading
[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]	

▽

△

Number	Heading
[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
[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

▽

△

Number	Heading
[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 F.7: Changed Specification Items in R21-11

F.3.3 Deleted Specification Items in R21-11

Number	Heading
[SWS_CORE_01913]	

Table F.8: Deleted Specification Items in R21-11

F.4 Change History of this document according to AUTOSAR Release R22-11

F.4.1 Added Specification Items in R22-11

Number	Heading
[SWS_CORE_00054]	
[SWS_CORE_00615]	
[SWS_CORE_00616]	
[SWS_CORE_00617]	
[SWS_CORE_00618]	
[SWS_CORE_03012]	
[SWS_CORE_10203]	Valid InstanceSpecifier representations - functional cluster interaction
[SWS_CORE_11000]	Optional base behavior
[SWS_CORE_11300]	Vector base behavior
[SWS_CORE_11400]	Map base behavior
[SWS_CORE_11600]	Variant base behavior
[SWS_CORE_11900]	Span base behavior
[SWS_CORE_12000]	String base behavior
[SWS_CORE_12200]	StringView base behavior
[SWS_CORE_90005]	Custom declarations and definitions
[SWS_CORE_90006]	

Table F.9: Added Specification Items in R22-11

F.4.2 Changed Specification Items in R22-11

Number	Heading
[SWS_CORE_00051]	
[SWS_CORE_00052]	
[SWS_CORE_00340]	
[SWS_CORE_00341]	
[SWS_CORE_00342]	
[SWS_CORE_00343]	
[SWS_CORE_00344]	
[SWS_CORE_00345]	
[SWS_CORE_00346]	
[SWS_CORE_00349]	
[SWS_CORE_00350]	
[SWS_CORE_00351]	

▽

△

Number	Heading
[SWS_CORE_00352]	
[SWS_CORE_00353]	
[SWS_CORE_00354]	
[SWS_CORE_00355]	
[SWS_CORE_00356]	
[SWS_CORE_00571]	
[SWS_CORE_00572]	
[SWS_CORE_00614]	
[SWS_CORE_01033]	
[SWS_CORE_01096]	
[SWS_CORE_01301]	
[SWS_CORE_01390]	
[SWS_CORE_01391]	
[SWS_CORE_01392]	
[SWS_CORE_01393]	
[SWS_CORE_01394]	
[SWS_CORE_01395]	
[SWS_CORE_01396]	
[SWS_CORE_01400]	
[SWS_CORE_01496]	
[SWS_CORE_01601]	
[SWS_CORE_01696]	
[SWS_CORE_02001]	
[SWS_CORE_03000]	
[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]	

▽

△

Number	Heading
[SWS_CORE_03313]	
[SWS_CORE_03314]	
[SWS_CORE_03315]	
[SWS_CORE_03316]	
[SWS_CORE_03317]	
[SWS_CORE_03318]	
[SWS_CORE_03319]	
[SWS_CORE_03320]	
[SWS_CORE_03321]	
[SWS_CORE_03322]	
[SWS_CORE_03323]	
[SWS_CORE_05244]	
[SWS_CORE_06340]	
[SWS_CORE_06341]	
[SWS_CORE_06342]	
[SWS_CORE_06343]	
[SWS_CORE_06344]	
[SWS_CORE_06345]	
[SWS_CORE_06349]	
[SWS_CORE_06350]	
[SWS_CORE_06351]	
[SWS_CORE_06352]	
[SWS_CORE_06353]	
[SWS_CORE_06354]	
[SWS_CORE_06355]	
[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

▽

△

Number	Heading
[SWS_CORE_90003]	

Table F.10: Changed Specification Items in R22-11

F.4.3 Deleted Specification Items in R22-11

none

F.5 Change History of this document according to AUTOSAR Release R23-11

F.5.1 Added Specification Items in R23-11

Number	Heading
[SWS_CORE_00774]	Definition of API function ara::core::Result::operator*
[SWS_CORE_00775]	Definition of API function ara::core::Result::Value
[SWS_CORE_00776]	Definition of API function ara::core::Result::Error
[SWS_CORE_00876]	Definition of API function ara::core::Result< void, E >::Error
[SWS_CORE_01273]	Definition of API function ara::core::Array::at
[SWS_CORE_01274]	Definition of API function ara::core::Array::at
[SWS_CORE_06500]	Definition of API class ara::core::MemoryResource
[SWS_CORE_06501]	Definition of API function ara::core::MemoryResource::MemoryResource
[SWS_CORE_06502]	Definition of API function ara::core::MemoryResource::MemoryResource
[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

▽

△

Number	Heading
[SWS_CORE_06526]	Definition of API function ara::core::MonotonicBufferResource::Monotonic BufferResource
[SWS_CORE_06527]	Definition of API function ara::core::MonotonicBufferResource::Monotonic BufferResource
[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

▽

△

Number	Heading
[SWS_CORE_11952]	Resolution of macro ARA_COMPILER_DEFINED_NODISCARD
[SWS_CORE_15005]	
[SWS_CORE_90021]	
[SWS_CORE_90022]	

Table F.11: Added Specification Items in R23-11

F.5.2 Changed Specification Items in R23-11

Number	Heading
[SWS_CORE_00122]	Definition of API type ara::core::ErrorDomain::CodeType
[SWS_CORE_00123]	Definition of API type ara::core::ErrorDomain::SupportDataType
[SWS_CORE_00151]	Definition of API function ara::core::ErrorDomain::Id
[SWS_CORE_00152]	Definition of API function ara::core::ErrorDomain::Name
[SWS_CORE_00154]	Definition of API function ara::core::ErrorDomain::ThrowAsException
[SWS_CORE_00326]	Definition of API function ara::core::Future::get
[SWS_CORE_00328]	Definition of API function ara::core::Future::wait
[SWS_CORE_00329]	Definition of API function ara::core::Future::wait_for
[SWS_CORE_00330]	Definition of API function ara::core::Future::wait_until
[SWS_CORE_00331]	Definition of API function ara::core::Future::then
[SWS_CORE_00332]	Definition of API function ara::core::Future::is_ready
[SWS_CORE_00333]	Definition of API function ara::core::Future::~Future
[SWS_CORE_00336]	Definition of API function ara::core::Future::GetResult
[SWS_CORE_00337]	Definition of API function ara::core::Future::then
[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==

▽

△

Number	Heading
[SWS_CORE_00572]	Definition of API function ara::core::operator!=
[SWS_CORE_00711]	Definition of API type ara::core::Result::value_type
[SWS_CORE_00712]	Definition of API type ara::core::Result::error_type
[SWS_CORE_00721]	Definition of API function ara::core::Result::Result
[SWS_CORE_00722]	Definition of API function ara::core::Result::Result
[SWS_CORE_00723]	Definition of API function ara::core::Result::Result
[SWS_CORE_00724]	Definition of API function ara::core::Result::Result
[SWS_CORE_00725]	Definition of API function ara::core::Result::Result
[SWS_CORE_00726]	Definition of API function ara::core::Result::Result
[SWS_CORE_00731]	Definition of API function ara::core::Result::FromValue
[SWS_CORE_00732]	Definition of API function ara::core::Result::FromValue
[SWS_CORE_00733]	Definition of API function ara::core::Result::FromValue
[SWS_CORE_00734]	Definition of API function ara::core::Result::FromError
[SWS_CORE_00735]	Definition of API function ara::core::Result::FromError
[SWS_CORE_00736]	Definition of API function ara::core::Result::FromError
[SWS_CORE_00741]	Definition of API function ara::core::Result::operator=
[SWS_CORE_00742]	Definition of API function ara::core::Result::operator=
[SWS_CORE_00743]	Definition of API function ara::core::Result::EmplaceValue
[SWS_CORE_00744]	Definition of API function ara::core::Result::EmplaceError
[SWS_CORE_00745]	Definition of API function ara::core::Result::Swap
[SWS_CORE_00751]	Definition of API function ara::core::Result::HasValue
[SWS_CORE_00752]	Definition of API function ara::core::Result::operator bool
[SWS_CORE_00753]	Definition of API function ara::core::Result::operator*
[SWS_CORE_00754]	Definition of API function ara::core::Result::operator->
[SWS_CORE_00755]	Definition of API function ara::core::Result::Value
[SWS_CORE_00756]	Definition of API function ara::core::Result::Value
[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

▽

△

Number	Heading
[SWS_CORE_00770]	Definition of API function ara::core::Result::Ok
[SWS_CORE_00771]	Definition of API function ara::core::Result::Ok
[SWS_CORE_00772]	Definition of API function ara::core::Result::Err
[SWS_CORE_00773]	Definition of API function ara::core::Result::Err
[SWS_CORE_00853]	Definition of API function ara::core::Result< void, E >::operator*
[SWS_CORE_00855]	Definition of API function ara::core::Result< void, E >::Value
[SWS_CORE_00857]	Definition of API function ara::core::Result< void, E >::Error
[SWS_CORE_00858]	Definition of API function ara::core::Result< void, E >::Error
[SWS_CORE_01265]	Definition of API function ara::core::Array::operator[]
[SWS_CORE_01266]	Definition of API function ara::core::Array::operator[]
[SWS_CORE_05244]	Definition of API function ara::core::CoreErrorDomain::ThrowAsException
[SWS_CORE_06226]	Definition of API function ara::core::Future< void, E >::get
[SWS_CORE_06228]	Definition of API function ara::core::Future< void, E >::wait
[SWS_CORE_06229]	Definition of API function ara::core::Future< void, E >::wait_for
[SWS_CORE_06230]	Definition of API function ara::core::Future< void, E >::wait_until
[SWS_CORE_06231]	Definition of API function ara::core::Future< void, E >::then
[SWS_CORE_06232]	Definition of API function ara::core::Future< void, E >::is_ready
[SWS_CORE_06233]	Definition of API function ara::core::Future< void, E >::~Future
[SWS_CORE_06236]	Definition of API function ara::core::Future< void, E >::GetResult
[SWS_CORE_06237]	Definition of API function ara::core::Future< void, E >::then
[SWS_CORE_06343]	Definition of API function ara::core::Promise< void, E >::operator=
[SWS_CORE_06344]	Definition of API function ara::core::Promise< void, E >::get_future
[SWS_CORE_06345]	Definition of API function ara::core::Promise< void, E >::set_value
[SWS_CORE_06349]	Definition of API function ara::core::Promise< void, E >::~Promise
[SWS_CORE_06353]	Definition of API function ara::core::Promise< void, E >::SetError
[SWS_CORE_06354]	Definition of API function ara::core::Promise< void, E >::SetError
[SWS_CORE_06355]	Definition of API function ara::core::Promise< void, E >::SetResult
[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 F.12: Changed Specification Items in R23-11

F.5.3 Deleted Specification Items in R23-11

Number	Heading
[SWS_CORE_08101]	
[SWS_CORE_08111]	
[SWS_CORE_08121]	
[SWS_CORE_08122]	
[SWS_CORE_08123]	
[SWS_CORE_08124]	
[SWS_CORE_08125]	
[SWS_CORE_08126]	
[SWS_CORE_08127]	
[SWS_CORE_08128]	
[SWS_CORE_08129]	
[SWS_CORE_08141]	
[SWS_CORE_08180]	
[SWS_CORE_08181]	
[SWS_CORE_08182]	
[SWS_CORE_08183]	
[SWS_CORE_08184]	
[SWS_CORE_08185]	
[SWS_CORE_08186]	
[SWS_CORE_08187]	
[SWS_CORE_08188]	
[SWS_CORE_08189]	
[SWS_CORE_08190]	
[SWS_CORE_08191]	
[SWS_CORE_08192]	
[SWS_CORE_08193]	
[SWS_CORE_08194]	
[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 F.13: Deleted Specification Items in R23-11

F.6 Change History of this document according to AUTOSAR Release R24-11

F.6.1 Added Specification Items in R24-11

Number	Heading
[SWS_CORE_00006]	Handling of exception-based Violations
[SWS_CORE_00007]	Handling of exception-based Failed Default Allocations
[SWS_CORE_00008]	Definition of API class ara::core::Executor
[SWS_CORE_00009]	Definition of API function ara::core::Executor::Executor
[SWS_CORE_00015]	Definition of API function ara::core::Executor::Executor
[SWS_CORE_00017]	Definition of API function ara::core::Executor::operator=
[SWS_CORE_00018]	Definition of API function ara::core::Executor::operator=
[SWS_CORE_00019]	Definition of API function ara::core::Executor::~Executor
[SWS_CORE_00024]	Definition of API function ara::core::Executor::execute
[SWS_CORE_00025]	Definition of API function ara::core::Executor::oneway_execute
[SWS_CORE_00026]	Definition of API class ara::core::PoolOptions
[SWS_CORE_00027]	Definition of API variable ara::core::PoolOptions::max_blocks_per_chunk
[SWS_CORE_00028]	Definition of API variable ara::core::PoolOptions::largest_required_pool_block
[SWS_CORE_00029]	Definition of API class ara::core::SynchronizedPoolResource
[SWS_CORE_00030]	Definition of API function ara::core::SynchronizedPoolResource::SynchronizedPoolResource
[SWS_CORE_00031]	Definition of API function ara::core::SynchronizedPoolResource::SynchronizedPoolResource
[SWS_CORE_00032]	Definition of API function ara::core::SynchronizedPoolResource::SynchronizedPoolResource
[SWS_CORE_00033]	Definition of API function ara::core::SynchronizedPoolResource::SynchronizedPoolResource
[SWS_CORE_00034]	Definition of API function ara::core::SynchronizedPoolResource::SynchronizedPoolResource
[SWS_CORE_00035]	Definition of API function ara::core::SynchronizedPoolResource::~SynchronizedPoolResource
[SWS_CORE_00036]	Definition of API function ara::core::SynchronizedPoolResource::operator=
[SWS_CORE_00037]	Definition of API function ara::core::SynchronizedPoolResource::release
[SWS_CORE_00038]	Definition of API function ara::core::SynchronizedPoolResource::upstream_resource
[SWS_CORE_00039]	Definition of API function ara::core::SynchronizedPoolResource::options
[SWS_CORE_00041]	Definition of API function ara::core::SynchronizedPoolResource::do_deallocate
[SWS_CORE_00042]	Definition of API function ara::core::SynchronizedPoolResource::do_is_equal

△

Number	Heading
[SWS_CORE_00043]	Definition of API function ara::core::SynchronizedPool Resource::SynchronizedPoolResource
[SWS_CORE_00044]	Definition of API function ara::core::SynchronizedPoolResource::operator=
[SWS_CORE_00045]	Definition of API class ara::core::UnsynchronizedPoolResource
[SWS_CORE_00046]	Definition of API function ara::core::UnsynchronizedPool Resource::UnsynchronizedPoolResource
[SWS_CORE_00047]	Definition of API function ara::core::UnsynchronizedPool Resource::operator=
[SWS_CORE_00048]	Definition of API function ara::core::UnsynchronizedPool Resource::UnsynchronizedPoolResource
[SWS_CORE_00049]	Definition of API function ara::core::UnsynchronizedPool Resource::operator=
[SWS_CORE_00055]	Functional Cluster Log and Trace messages
[SWS_CORE_00056]	Definition of API function ara::core::UnsynchronizedPool Resource::upstream_resource
[SWS_CORE_00057]	Definition of API function ara::core::UnsynchronizedPoolResource::options
[SWS_CORE_00058]	Definition of API function ara::core::UnsynchronizedPoolResource::do_ allocate
[SWS_CORE_00059]	Definition of API function ara::core::UnsynchronizedPoolResource::do_ deallocate
[SWS_CORE_00060]	Definition of API function ara::core::UnsynchronizedPoolResource::do_is_ equal
[SWS_CORE_00061]	Definition of API function ara::core::SynchronizedPoolResource::do_allocate
[SWS_CORE_00062]	Definition of API function ara::core::UnsynchronizedPool Resource::~UnsynchronizedPoolResource
[SWS_CORE_00063]	Definition of API function ara::core::UnsynchronizedPool Resource::UnsynchronizedPoolResource
[SWS_CORE_00064]	Definition of API function ara::core::UnsynchronizedPool Resource::UnsynchronizedPoolResource
[SWS_CORE_00065]	Definition of API function ara::core::UnsynchronizedPoolResource::release
[SWS_CORE_00066]	Definition of API function ara::core::PolymorphicAllocator::operator=
[SWS_CORE_00067]	Definition of API function ara::core::UnsynchronizedPool Resource::UnsynchronizedPoolResource
[SWS_CORE_00068]	Definition of API function ara::core::UnsynchronizedPool Resource::UnsynchronizedPoolResource
[SWS_CORE_00069]	`ara::core::Result<T&>` may hold lvalue reference types
[SWS_CORE_00070]	Assigning to `ara::core::Result<T&>`
[SWS_CORE_00071]	Definition of API type ara::core::BasicString::iterator
[SWS_CORE_00072]	Definition of API type ara::core::BasicString::const_iterator
[SWS_CORE_00073]	Definition of API variable ara::core::BasicString::npos
[SWS_CORE_00090]	Handling of `Standardized Violations`
[SWS_CORE_00091]	Messages for `Violations`

▽

△

Number	Heading
[SWS_CORE_00092]	Vendor-specific errors
[SWS_CORE_00093]	Vendor header file
[SWS_CORE_00777]	Definition of API function ara::core::Result::operator->
[SWS_CORE_00901]	Definition of API class ara::core::Result< T &, E >
[SWS_CORE_00902]	Definition of API type ara::core::Result< T &, E >::value_type
[SWS_CORE_00903]	Definition of API type ara::core::Result< T &, E >::error_type
[SWS_CORE_00904]	Definition of API function ara::core::Result< T &, E >::FromValue
[SWS_CORE_00905]	Definition of API function ara::core::Result< T &, E >::FromError
[SWS_CORE_00906]	Definition of API function ara::core::Result< T &, E >::FromError
[SWS_CORE_00907]	Definition of API function ara::core::Result< T &, E >::FromError
[SWS_CORE_00908]	Definition of API function ara::core::Result< T &, E >::Result
[SWS_CORE_00909]	Definition of API function ara::core::Result< T &, E >::Result
[SWS_CORE_00910]	Definition of API function ara::core::Result< T &, E >::Result
[SWS_CORE_00911]	Definition of API function ara::core::Result< T &, E >::Result
[SWS_CORE_00912]	Definition of API function ara::core::Result< T &, E >::Result
[SWS_CORE_00913]	Definition of API function ara::core::Result< T &, E >::~Result
[SWS_CORE_00914]	Definition of API function ara::core::Result< T &, E >::operator=
[SWS_CORE_00915]	Definition of API function ara::core::Result< T &, E >::operator=
[SWS_CORE_00916]	Definition of API function ara::core::Result< T &, E >::EmplaceError
[SWS_CORE_00917]	Definition of API function ara::core::Result< T &, E >::Swap
[SWS_CORE_00918]	Definition of API function ara::core::Result< T &, E >::HasValue
[SWS_CORE_00919]	Definition of API function ara::core::Result< T &, E >::operator bool
[SWS_CORE_00920]	Definition of API function ara::core::Result< T &, E >::operator*
[SWS_CORE_00921]	Definition of API function ara::core::Result< T &, E >::operator*
[SWS_CORE_00922]	Definition of API function ara::core::Result< T &, E >::operator->
[SWS_CORE_00923]	Definition of API function ara::core::Result< T &, E >::operator->
[SWS_CORE_00924]	Definition of API function ara::core::Result< T &, E >::Value
[SWS_CORE_00925]	Definition of API function ara::core::Result< T &, E >::Value
[SWS_CORE_00926]	Definition of API function ara::core::Result< T &, E >::Ok
[SWS_CORE_00927]	Definition of API function ara::core::Result< T &, E >::Ok
[SWS_CORE_00928]	Definition of API function ara::core::Result< T &, E >::Error
[SWS_CORE_00929]	Definition of API function ara::core::Result< T &, E >::Error
[SWS_CORE_00930]	Definition of API function ara::core::Result< T &, E >::Error
[SWS_CORE_00931]	Definition of API function ara::core::Result< T &, E >::Err
[SWS_CORE_00932]	Definition of API function ara::core::Result< T &, E >::Err
[SWS_CORE_00933]	Definition of API function ara::core::Result< T &, E >::ValueOr
[SWS_CORE_00934]	Definition of API function ara::core::Result< T &, E >::ValueOr
[SWS_CORE_00935]	Definition of API function ara::core::Result< T &, E >::ErrorOr

▽

△

Number	Heading
[SWS_CORE_00936]	Definition of API function ara::core::Result< T &, E >::ErrorOr
[SWS_CORE_00937]	Definition of API function ara::core::Result< T &, E >::CheckError
[SWS_CORE_00938]	Definition of API function ara::core::Result< T &, E >::ValueOrThrow
[SWS_CORE_00939]	Definition of API function ara::core::Result< T &, E >::Resolve
[SWS_CORE_00940]	Definition of API function ara::core::Result< T &, E >::Resolve
[SWS_CORE_00941]	Definition of API function ara::core::Result< T &, E >::Bind
[SWS_CORE_01032]	Optional references
[SWS_CORE_01034]	Assignment behavior of Optional references
[SWS_CORE_01100]	Definition of API class ara::core::nullopt_t
[SWS_CORE_01101]	Definition of API variable ara::core::nullopt
[SWS_CORE_01102]	Definition of API type ara::core::Optional::value_type
[SWS_CORE_01103]	Definition of API function ara::core::Optional::Optional
[SWS_CORE_01104]	Definition of API function ara::core::Optional::Optional
[SWS_CORE_01105]	Definition of API function ara::core::Optional::Optional
[SWS_CORE_01106]	Definition of API function ara::core::Optional::Optional
[SWS_CORE_01107]	Definition of API function ara::core::Optional::Optional
[SWS_CORE_01108]	Definition of API function ara::core::Optional::Optional
[SWS_CORE_01109]	Definition of API function ara::core::Optional::Optional
[SWS_CORE_01110]	Definition of API function ara::core::Optional::Optional
[SWS_CORE_01111]	Definition of API function ara::core::Optional::Optional
[SWS_CORE_01112]	Definition of API function ara::core::Optional::~Optional
[SWS_CORE_01113]	Definition of API function ara::core::Optional::operator=
[SWS_CORE_01114]	Definition of API function ara::core::Optional::operator=
[SWS_CORE_01115]	Definition of API function ara::core::Optional::operator=
[SWS_CORE_01116]	Definition of API function ara::core::Optional::operator=
[SWS_CORE_01117]	Definition of API function ara::core::Optional::operator=
[SWS_CORE_01118]	Definition of API function ara::core::Optional::operator=
[SWS_CORE_01119]	Definition of API function ara::core::Optional::emplace
[SWS_CORE_01120]	Definition of API function ara::core::Optional::emplace
[SWS_CORE_01121]	Definition of API function ara::core::Optional::swap
[SWS_CORE_01122]	Definition of API function ara::core::Optional::operator->
[SWS_CORE_01123]	Definition of API function ara::core::Optional::operator->
[SWS_CORE_01124]	Definition of API function ara::core::Optional::operator*
[SWS_CORE_01125]	Definition of API function ara::core::Optional::operator*
[SWS_CORE_01126]	Definition of API function ara::core::Optional::operator*
[SWS_CORE_01127]	Definition of API function ara::core::Optional::operator*
[SWS_CORE_01128]	Definition of API function ara::core::Optional::operator bool
[SWS_CORE_01129]	Definition of API function ara::core::Optional::has_value

▽

△

Number	Heading
[SWS_CORE_01130]	Definition of API function ara::core::Optional::value
[SWS_CORE_01131]	Definition of API function ara::core::Optional::value
[SWS_CORE_01132]	Definition of API function ara::core::Optional::value
[SWS_CORE_01133]	Definition of API function ara::core::Optional::value
[SWS_CORE_01134]	Definition of API function ara::core::Optional::value_or
[SWS_CORE_01135]	Definition of API function ara::core::Optional::value_or
[SWS_CORE_01136]	Definition of API function ara::core::Optional::reset
[SWS_CORE_01138]	Definition of API function ara::core::make_optional
[SWS_CORE_01139]	Definition of API function ara::core::make_optional
[SWS_CORE_01140]	Definition of API function ara::core::make_optional
[SWS_CORE_01150]	Definition of API class ara::core::Optional< T & >
[SWS_CORE_01151]	Definition of API type ara::core::Optional< T & >::value_type
[SWS_CORE_01152]	Definition of API function ara::core::Optional< T & >::Optional
[SWS_CORE_01153]	Definition of API function ara::core::Optional< T & >::Optional
[SWS_CORE_01154]	Definition of API function ara::core::Optional< T & >::Optional
[SWS_CORE_01155]	Definition of API function ara::core::Optional< T & >::Optional
[SWS_CORE_01156]	Definition of API function ara::core::Optional< T & >::Optional
[SWS_CORE_01157]	Definition of API function ara::core::Optional< T & >::Optional
[SWS_CORE_01158]	Definition of API function ara::core::Optional< T & >::~Optional
[SWS_CORE_01159]	Definition of API function ara::core::Optional< T & >::operator=
[SWS_CORE_01160]	Definition of API function ara::core::Optional< T & >::operator=
[SWS_CORE_01161]	Definition of API function ara::core::Optional< T & >::operator=
[SWS_CORE_01162]	Definition of API function ara::core::Optional< T & >::operator=
[SWS_CORE_01163]	Definition of API function ara::core::Optional< T & >::operator=
[SWS_CORE_01164]	Definition of API function ara::core::Optional< T & >::emplace
[SWS_CORE_01165]	Definition of API function ara::core::Optional< T & >::reset
[SWS_CORE_01166]	Definition of API function ara::core::Optional< T & >::swap
[SWS_CORE_01167]	Definition of API function ara::core::Optional< T & >::operator->
[SWS_CORE_01168]	Definition of API function ara::core::Optional< T & >::operator->
[SWS_CORE_01169]	Definition of API function ara::core::Optional< T & >::operator*
[SWS_CORE_01170]	Definition of API function ara::core::Optional< T & >::operator*
[SWS_CORE_01171]	Definition of API function ara::core::Optional< T & >::operator bool
[SWS_CORE_01172]	Definition of API function ara::core::Optional< T & >::has_value
[SWS_CORE_01173]	Definition of API function ara::core::Optional< T & >::value
[SWS_CORE_01174]	Definition of API function ara::core::Optional< T & >::value
[SWS_CORE_01175]	Definition of API function ara::core::Optional< T & >::value_or
[SWS_CORE_01600]	Definition of API class ara::core::variant_size
[SWS_CORE_01602]	Definition of API class ara::core::variant_size< const T >

▽

△

Number	Heading
[SWS_CORE_01603]	Definition of API class ara::core::variant_size< volatile T >
[SWS_CORE_01604]	Definition of API class ara::core::variant_size< const volatile T >
[SWS_CORE_01605]	Definition of API class ara::core::variant_size< Variant< Types... > >
[SWS_CORE_01606]	Definition of API variable ara::core::variant_size_v
[SWS_CORE_01607]	Definition of API class ara::core::variant_alternative
[SWS_CORE_01608]	Definition of API class ara::core::variant_alternative< I, const T >
[SWS_CORE_01609]	Definition of API class ara::core::variant_alternative< I, volatile T >
[SWS_CORE_01610]	Definition of API class ara::core::variant_alternative< I, const volatile T >
[SWS_CORE_01611]	Definition of API class ara::core::variant_alternative< I, Variant< Types... > >
[SWS_CORE_01612]	Definition of API type ara::core::variant_alternative_t
[SWS_CORE_01613]	Definition of API function ara::core::get
[SWS_CORE_01614]	Definition of API function ara::core::get
[SWS_CORE_01615]	Definition of API function ara::core::get
[SWS_CORE_01616]	Definition of API function ara::core::get
[SWS_CORE_01617]	Definition of API function ara::core::get
[SWS_CORE_01618]	Definition of API function ara::core::get
[SWS_CORE_01619]	Definition of API function ara::core::get
[SWS_CORE_01620]	Definition of API function ara::core::get
[SWS_CORE_01621]	Definition of API function ara::core::get_if
[SWS_CORE_01622]	Definition of API function ara::core::get_if
[SWS_CORE_01623]	Definition of API function ara::core::get_if
[SWS_CORE_01624]	Definition of API function ara::core::get_if
[SWS_CORE_01626]	Definition of API function ara::core::holds_alternative
[SWS_CORE_01627]	Definition of API function ara::core::operator==
[SWS_CORE_01628]	Definition of API function ara::core::operator!=
[SWS_CORE_01629]	Definition of API function ara::core::operator<
[SWS_CORE_01630]	Definition of API function ara::core::operator>
[SWS_CORE_01631]	Definition of API function ara::core::operator<=
[SWS_CORE_01632]	Definition of API function ara::core::operator>=
[SWS_CORE_01633]	Definition of API function ara::core::visit
[SWS_CORE_01634]	Definition of API function ara::core::visit
[SWS_CORE_01640]	Definition of API class ara::core::Monostate
[SWS_CORE_01641]	Definition of API function ara::core::operator==
[SWS_CORE_01642]	Definition of API function ara::core::operator!=
[SWS_CORE_01643]	Definition of API function ara::core::operator<
[SWS_CORE_01644]	Definition of API function ara::core::operator>
[SWS_CORE_01645]	Definition of API function ara::core::operator<=
[SWS_CORE_01646]	Definition of API function ara::core::operator>=

▽

△

Number	Heading
[SWS_CORE_01649]	Definition of API function ara::core::Variant::Variant
[SWS_CORE_01650]	Definition of API function ara::core::Variant::Variant
[SWS_CORE_01651]	Definition of API function ara::core::Variant::Variant
[SWS_CORE_01652]	Definition of API function ara::core::Variant::Variant
[SWS_CORE_01653]	Definition of API function ara::core::Variant::Variant
[SWS_CORE_01654]	Definition of API function ara::core::Variant::Variant
[SWS_CORE_01655]	Definition of API function ara::core::Variant::Variant
[SWS_CORE_01656]	Definition of API function ara::core::Variant::Variant
[SWS_CORE_01657]	Definition of API function ara::core::Variant::~Variant
[SWS_CORE_01658]	Definition of API function ara::core::Variant::operator=
[SWS_CORE_01659]	Definition of API function ara::core::Variant::operator=
[SWS_CORE_01660]	Definition of API function ara::core::Variant::operator=
[SWS_CORE_01661]	Definition of API function ara::core::Variant::emplace
[SWS_CORE_01662]	Definition of API function ara::core::Variant::emplace
[SWS_CORE_01663]	Definition of API function ara::core::Variant::emplace
[SWS_CORE_01664]	Definition of API function ara::core::Variant::emplace
[SWS_CORE_01665]	Definition of API function ara::core::Variant::index
[SWS_CORE_01666]	Definition of API function ara::core::Variant::valueless_by_exception
[SWS_CORE_01667]	Definition of API function ara::core::Variant::swap
[SWS_CORE_01668]	Definition of API class std::hash<::ara::core::Variant< Types... > >
[SWS_CORE_01669]	Definition of API function std::hash<::ara::core::Variant< Types... > >::operator()
[SWS_CORE_01670]	Definition of API class std::hash<::ara::core::Monostate >
[SWS_CORE_01671]	Definition of API function std::hash<::ara::core::Monostate >::operator()
[SWS_CORE_02100]	Definition of API type ara::core::StringView::value_type
[SWS_CORE_02101]	Definition of API type ara::core::StringView::pointer
[SWS_CORE_02102]	Definition of API type ara::core::StringView::const_pointer
[SWS_CORE_02103]	Definition of API type ara::core::StringView::reference
[SWS_CORE_02104]	Definition of API type ara::core::StringView::const_reference
[SWS_CORE_02105]	Definition of API type ara::core::StringView::const_iterator
[SWS_CORE_02106]	Definition of API type ara::core::StringView::iterator
[SWS_CORE_02107]	Definition of API type ara::core::StringView::const_reverse_iterator
[SWS_CORE_02108]	Definition of API type ara::core::StringView::reverse_iterator
[SWS_CORE_02109]	Definition of API type ara::core::StringView::size_type
[SWS_CORE_02110]	Definition of API type ara::core::StringView::difference_type
[SWS_CORE_02111]	Definition of API variable ara::core::StringView::npos
[SWS_CORE_02112]	Definition of API function ara::core::StringView::StringView
[SWS_CORE_02113]	Definition of API function ara::core::StringView::StringView

▽

△

Number	Heading
[SWS_CORE_02114]	Definition of API function ara::core::StringView::StringView
[SWS_CORE_02115]	Definition of API function ara::core::StringView::StringView
[SWS_CORE_02116]	Definition of API function ara::core::StringView::operator=
[SWS_CORE_02117]	Definition of API function ara::core::StringView::StringView
[SWS_CORE_02118]	Definition of API function ara::core::StringView::operator=
[SWS_CORE_02119]	Definition of API function ara::core::StringView::~StringView
[SWS_CORE_02120]	Definition of API function ara::core::StringView::begin
[SWS_CORE_02121]	Definition of API function ara::core::StringView::end
[SWS_CORE_02122]	Definition of API function ara::core::StringView::cbegin
[SWS_CORE_02123]	Definition of API function ara::core::StringView::cend
[SWS_CORE_02124]	Definition of API function ara::core::StringView::rbegin
[SWS_CORE_02125]	Definition of API function ara::core::StringView::rend
[SWS_CORE_02126]	Definition of API function ara::core::StringView::crbegin
[SWS_CORE_02127]	Definition of API function ara::core::StringView::crend
[SWS_CORE_02128]	Definition of API function ara::core::StringView::size
[SWS_CORE_02129]	Definition of API function ara::core::StringView::length
[SWS_CORE_02130]	Definition of API function ara::core::StringView::max_size
[SWS_CORE_02131]	Definition of API function ara::core::StringView::empty
[SWS_CORE_02132]	Definition of API function ara::core::StringView::operator[]
[SWS_CORE_02133]	Definition of API function ara::core::StringView::at
[SWS_CORE_02134]	Definition of API function ara::core::StringView::front
[SWS_CORE_02135]	Definition of API function ara::core::StringView::back
[SWS_CORE_02136]	Definition of API function ara::core::StringView::data
[SWS_CORE_02137]	Definition of API function ara::core::StringView::remove_prefix
[SWS_CORE_02138]	Definition of API function ara::core::StringView::remove_suffix
[SWS_CORE_02139]	Definition of API function ara::core::StringView::swap
[SWS_CORE_02140]	Definition of API function ara::core::StringView::copy
[SWS_CORE_02141]	Definition of API function ara::core::StringView::substr
[SWS_CORE_02142]	Definition of API function ara::core::StringView::compare
[SWS_CORE_02143]	Definition of API function ara::core::StringView::compare
[SWS_CORE_02144]	Definition of API function ara::core::StringView::compare
[SWS_CORE_02145]	Definition of API function ara::core::StringView::compare
[SWS_CORE_02146]	Definition of API function ara::core::StringView::compare
[SWS_CORE_02147]	Definition of API function ara::core::StringView::compare
[SWS_CORE_02148]	Definition of API function ara::core::StringView::starts_with
[SWS_CORE_02149]	Definition of API function ara::core::StringView::starts_with
[SWS_CORE_02150]	Definition of API function ara::core::StringView::starts_with
[SWS_CORE_02151]	Definition of API function ara::core::StringView::ends_with

▽

△

Number	Heading
[SWS_CORE_02152]	Definition of API function ara::core::StringView::ends_with
[SWS_CORE_02153]	Definition of API function ara::core::StringView::ends_with
[SWS_CORE_02154]	Definition of API function ara::core::StringView::contains
[SWS_CORE_02155]	Definition of API function ara::core::StringView::contains
[SWS_CORE_02156]	Definition of API function ara::core::StringView::contains
[SWS_CORE_02157]	Definition of API function ara::core::StringView::find
[SWS_CORE_02158]	Definition of API function ara::core::StringView::find
[SWS_CORE_02159]	Definition of API function ara::core::StringView::find
[SWS_CORE_02160]	Definition of API function ara::core::StringView::find
[SWS_CORE_02161]	Definition of API function ara::core::StringView::rfind
[SWS_CORE_02162]	Definition of API function ara::core::StringView::rfind
[SWS_CORE_02163]	Definition of API function ara::core::StringView::rfind
[SWS_CORE_02164]	Definition of API function ara::core::StringView::rfind
[SWS_CORE_02165]	Definition of API function ara::core::StringView::find_first_of
[SWS_CORE_02166]	Definition of API function ara::core::StringView::find_first_of
[SWS_CORE_02167]	Definition of API function ara::core::StringView::find_first_of
[SWS_CORE_02168]	Definition of API function ara::core::StringView::find_first_of
[SWS_CORE_02169]	Definition of API function ara::core::StringView::find_last_of
[SWS_CORE_02170]	Definition of API function ara::core::StringView::find_last_of
[SWS_CORE_02171]	Definition of API function ara::core::StringView::find_last_of
[SWS_CORE_02172]	Definition of API function ara::core::StringView::find_last_of
[SWS_CORE_02173]	Definition of API function ara::core::StringView::find_first_not_of
[SWS_CORE_02174]	Definition of API function ara::core::StringView::find_first_not_of
[SWS_CORE_02175]	Definition of API function ara::core::StringView::find_first_not_of
[SWS_CORE_02176]	Definition of API function ara::core::StringView::find_first_not_of
[SWS_CORE_02177]	Definition of API function ara::core::StringView::find_last_not_of
[SWS_CORE_02178]	Definition of API function ara::core::StringView::find_last_not_of
[SWS_CORE_02179]	Definition of API function ara::core::StringView::find_last_not_of
[SWS_CORE_02180]	Definition of API function ara::core::StringView::find_last_not_of
[SWS_CORE_02181]	Definition of API function ara::core::operator==
[SWS_CORE_02182]	Definition of API function ara::core::operator!=
[SWS_CORE_02183]	Definition of API function ara::core::operator<
[SWS_CORE_02184]	Definition of API function ara::core::operator>
[SWS_CORE_02185]	Definition of API function ara::core::operator<=
[SWS_CORE_02186]	Definition of API function ara::core::operator>=
[SWS_CORE_02187]	Definition of API function ara::core::operator<<
[SWS_CORE_02188]	Definition of API function ara::core::literals::operator""_SV
[SWS_CORE_02189]	Definition of API class std::hash< ara::core::StringView >

▽

△

Number	Heading
[SWS_CORE_02190]	Definition of API function std::hash< ara::core::StringView >::operator()
[SWS_CORE_06566]	Definition of API function ara::core::MemoryResource::do_allocate
[SWS_CORE_06567]	Definition of API function ara::core::MemoryResource::do_deallocate
[SWS_CORE_06568]	Definition of API function ara::core::MemoryResource::do_is_equal
[SWS_CORE_06569]	Definition of API function ara::core::MonotonicBufferResource::do_allocate
[SWS_CORE_06570]	Definition of API function ara::core::MonotonicBufferResource::do_ deallocate
[SWS_CORE_06571]	Definition of API function ara::core::MonotonicBufferResource::do_is_equal
[SWS_CORE_06572]	Definition of API type ara::core::PolymorphicAllocator::value_type
[SWS_CORE_06573]	Definition of API function ara::core::PolymorphicAllocator::select_on_ container_copy_construction
[SWS_CORE_06574]	Definition of API function ara::core::MemoryResource::MemoryResource
[SWS_CORE_06575]	Definition of API function ara::core::MemoryResource::operator=
[SWS_CORE_06576]	Definition of API function ara::core::MonotonicBufferResource::Monotonic BufferResource
[SWS_CORE_06577]	Definition of API function ara::core::MonotonicBufferResource::operator=
[SWS_CORE_10003]	Definition of API function ara::core::Initialize
[SWS_CORE_10304]	Availability of `ara::core::ErrorDomain::ThrowAsException` and overriding functions
[SWS_CORE_12408]	DLT Logging of `Explicitly aborting an Operation`
[SWS_CORE_12409]	Usage of ARA API within AbortHandlers
[SWS_CORE_13018]	LogMessage FailedDefaultAllocation
[SWS_CORE_13019]	LogMessage AbortMessage
[SWS_CORE_13200]	AUTOSAR definition of a thread-safe member function
[SWS_CORE_13201]	AUTOSAR definition of multiple thread-safe member functions
[SWS_CORE_13202]	AUTOSAR definition of a thread-safe non-member function
[SWS_CORE_13203]	Behavior of a concurrently executed non-thread-safe member function
[SWS_CORE_13204]	Behavior of a concurrently executed non-thread-safe non-member function
[SWS_CORE_13205]	Behavior of concurrently executed functions on different objects
[SWS_CORE_13206]	AUTOSAR `callable` definition
[SWS_CORE_13207]	Behavior of thread-safe `callable`
[SWS_CORE_13208]	Behavior of non-thread-safe `callable`
[SWS_CORE_13209]	Behavior of conditionally thread-safe `callable`
[SWS_CORE_13210]	ARA API usage within a signal handler
[SWS_CORE_15006]	Command line argument injection in `ara::core::Initialize`
[SWS_CORE_90007]	Potentially throwing constructors
[SWS_CORE_90023]	AUTOSAR-standardized Functional Cluster Error Domain Identifiers
[SWS_CORE_90024]	AUTOSAR-standardized Functional Cluster Log and Trace settings
[SWS_CORE_90025]	AUTOSAR-standardized Functional Cluster names

▽

△

Number	Heading
[SWS_CORE_90026]	Framework initialization
[SWS_CORE_90027]	Clean deinitialization
[SWS_CORE_90028]	Re-try of initialization
[SWS_CORE_90029]	Double initialization
[SWS_CORE_90030]	Double de-initialization

Table F.14: Added Specification Items in R24-11

F.6.2 Changed Specification Items in R24-11

Number	Heading
[SWS_CORE_00003]	Handling of `Non-Standardized Violations`
[SWS_CORE_00005]	Handling of failed default allocations
[SWS_CORE_00135]	Definition of API function ara::core::ErrorDomain::ErrorDomain
[SWS_CORE_00136]	Definition of API function ara::core::ErrorDomain::~ErrorDomain
[SWS_CORE_00137]	Definition of API function ara::core::ErrorDomain::operator==
[SWS_CORE_00138]	Definition of API function ara::core::ErrorDomain::operator!=
[SWS_CORE_00151]	Definition of API function ara::core::ErrorDomain::Id
[SWS_CORE_00152]	Definition of API function ara::core::ErrorDomain::Name
[SWS_CORE_00153]	Definition of API function ara::core::ErrorDomain::Message
[SWS_CORE_00154]	Definition of API function ara::core::ErrorDomain::ThrowAsException
[SWS_CORE_00321]	Definition of API class ara::core::Future
[SWS_CORE_00322]	Definition of API function ara::core::Future::Future
[SWS_CORE_00323]	Definition of API function ara::core::Future::Future
[SWS_CORE_00325]	Definition of API function ara::core::Future::operator=
[SWS_CORE_00326]	Definition of API function ara::core::Future::get
[SWS_CORE_00327]	Definition of API function ara::core::Future::valid
[SWS_CORE_00328]	Definition of API function ara::core::Future::wait
[SWS_CORE_00329]	Definition of API function ara::core::Future::wait_for
[SWS_CORE_00330]	Definition of API function ara::core::Future::wait_until
[SWS_CORE_00331]	Definition of API function ara::core::Future::then
[SWS_CORE_00332]	Definition of API function ara::core::Future::is_ready
[SWS_CORE_00333]	Definition of API function ara::core::Future::~Future
[SWS_CORE_00336]	Definition of API function ara::core::Future::GetResult
[SWS_CORE_00337]	Definition of API function ara::core::Future::then
[SWS_CORE_00340]	Definition of API class ara::core::Promise
[SWS_CORE_00341]	Definition of API function ara::core::Promise::Promise

▽

△

Number	Heading
[SWS_CORE_00342]	Definition of API function ara::core::Promise::Promise
[SWS_CORE_00343]	Definition of API function ara::core::Promise::operator=
[SWS_CORE_00344]	Definition of API function ara::core::Promise::get_future
[SWS_CORE_00345]	Definition of API function ara::core::Promise::set_value
[SWS_CORE_00346]	Definition of API function ara::core::Promise::set_value
[SWS_CORE_00349]	Definition of API function ara::core::Promise::~Promise
[SWS_CORE_00352]	Definition of API function ara::core::Promise::swap
[SWS_CORE_00353]	Definition of API function ara::core::Promise::SetError
[SWS_CORE_00354]	Definition of API function ara::core::Promise::SetError
[SWS_CORE_00355]	Definition of API function ara::core::Promise::SetResult
[SWS_CORE_00356]	Definition of API function ara::core::Promise::SetResult
[SWS_CORE_00361]	Definition of API enum ara::core::FutureStatus
[SWS_CORE_00400]	Definition of API enum ara::core::FutureErrc
[SWS_CORE_00412]	Definition of API function ara::core::FutureException::FutureException
[SWS_CORE_00421]	Definition of API class ara::core::FutureErrorDomain
[SWS_CORE_00431]	Definition of API type ara::core::FutureErrorDomain::Errc
[SWS_CORE_00441]	Definition of API function ara::core::FutureErrorDomain::FutureErrorDomain
[SWS_CORE_00442]	Definition of API function ara::core::FutureErrorDomain::Name
[SWS_CORE_00443]	Definition of API function ara::core::FutureErrorDomain::Message
[SWS_CORE_00444]	Definition of API function ara::core::FutureErrorDomain::ThrowAsException
[SWS_CORE_00480]	Definition of API function ara::core::GetFutureErrorDomain
[SWS_CORE_00490]	Definition of API function ara::core::MakeErrorCode
[SWS_CORE_00512]	Definition of API function ara::core::ErrorCode::ErrorCode
[SWS_CORE_00513]	Definition of API function ara::core::ErrorCode::ErrorCode
[SWS_CORE_00514]	Definition of API function ara::core::ErrorCode::Value
[SWS_CORE_00515]	Definition of API function ara::core::ErrorCode::Domain
[SWS_CORE_00516]	Definition of API function ara::core::ErrorCode::SupportData
[SWS_CORE_00518]	Definition of API function ara::core::ErrorCode::Message
[SWS_CORE_00519]	Definition of API function ara::core::ErrorCode::ThrowAsException
[SWS_CORE_00571]	Definition of API function ara::core::operator==
[SWS_CORE_00572]	Definition of API function ara::core::operator!=
[SWS_CORE_00611]	Definition of API function ara::core::Exception::Exception
[SWS_CORE_00612]	Definition of API function ara::core::Exception::what
[SWS_CORE_00613]	Definition of API function ara::core::Exception::Error
[SWS_CORE_00614]	Definition of API function ara::core::Exception::operator=
[SWS_CORE_00615]	Definition of API function ara::core::Exception::Exception
[SWS_CORE_00616]	Definition of API function ara::core::Exception::operator=
[SWS_CORE_00617]	Definition of API function ara::core::Exception::~Exception

▽

△

Number	Heading
[SWS_CORE_00618]	Definition of API function ara::core::Exception::Exception
[SWS_CORE_00701]	Definition of API class ara::core::Result
[SWS_CORE_00721]	Definition of API function ara::core::Result::Result
[SWS_CORE_00722]	Definition of API function ara::core::Result::Result
[SWS_CORE_00723]	Definition of API function ara::core::Result::Result
[SWS_CORE_00724]	Definition of API function ara::core::Result::Result
[SWS_CORE_00725]	Definition of API function ara::core::Result::Result
[SWS_CORE_00726]	Definition of API function ara::core::Result::Result
[SWS_CORE_00727]	Definition of API function ara::core::Result::~Result
[SWS_CORE_00731]	Definition of API function ara::core::Result::FromValue
[SWS_CORE_00732]	Definition of API function ara::core::Result::FromValue
[SWS_CORE_00733]	Definition of API function ara::core::Result::FromValue
[SWS_CORE_00734]	Definition of API function ara::core::Result::FromError
[SWS_CORE_00735]	Definition of API function ara::core::Result::FromError
[SWS_CORE_00736]	Definition of API function ara::core::Result::FromError
[SWS_CORE_00741]	Definition of API function ara::core::Result::operator=
[SWS_CORE_00742]	Definition of API function ara::core::Result::operator=
[SWS_CORE_00743]	Definition of API function ara::core::Result::EmplaceValue
[SWS_CORE_00744]	Definition of API function ara::core::Result::EmplaceError
[SWS_CORE_00745]	Definition of API function ara::core::Result::Swap
[SWS_CORE_00751]	Definition of API function ara::core::Result::HasValue
[SWS_CORE_00752]	Definition of API function ara::core::Result::operator bool
[SWS_CORE_00753]	Definition of API function ara::core::Result::operator*
[SWS_CORE_00754]	Definition of API function ara::core::Result::operator->
[SWS_CORE_00755]	Definition of API function ara::core::Result::Value
[SWS_CORE_00756]	Definition of API function ara::core::Result::Value
[SWS_CORE_00757]	Definition of API function ara::core::Result::Error
[SWS_CORE_00758]	Definition of API function ara::core::Result::Error
[SWS_CORE_00759]	Definition of API function ara::core::Result::operator*
[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

▽

△

Number	Heading
[SWS_CORE_00770]	Definition of API function ara::core::Result::Ok
[SWS_CORE_00771]	Definition of API function ara::core::Result::Ok
[SWS_CORE_00772]	Definition of API function ara::core::Result::Err
[SWS_CORE_00773]	Definition of API function ara::core::Result::Err
[SWS_CORE_00774]	Definition of API function ara::core::Result::operator*
[SWS_CORE_00775]	Definition of API function ara::core::Result::Value
[SWS_CORE_00776]	Definition of API function ara::core::Result::Error
[SWS_CORE_00780]	Definition of API function ara::core::operator==
[SWS_CORE_00781]	Definition of API function ara::core::operator!=
[SWS_CORE_00782]	Definition of API function ara::core::operator==
[SWS_CORE_00783]	Definition of API function ara::core::operator==
[SWS_CORE_00784]	Definition of API function ara::core::operator!=
[SWS_CORE_00785]	Definition of API function ara::core::operator!=
[SWS_CORE_00786]	Definition of API function ara::core::operator==
[SWS_CORE_00787]	Definition of API function ara::core::operator==
[SWS_CORE_00788]	Definition of API function ara::core::operator!=
[SWS_CORE_00789]	Definition of API function ara::core::operator!=
[SWS_CORE_00796]	Definition of API function ara::core::swap
[SWS_CORE_00801]	Definition of API class ara::core::Result< void, E >
[SWS_CORE_00821]	Definition of API function ara::core::Result< void, E >::Result
[SWS_CORE_00823]	Definition of API function ara::core::Result< void, E >::Result
[SWS_CORE_00824]	Definition of API function ara::core::Result< void, E >::Result
[SWS_CORE_00825]	Definition of API function ara::core::Result< void, E >::Result
[SWS_CORE_00826]	Definition of API function ara::core::Result< void, E >::Result
[SWS_CORE_00827]	Definition of API function ara::core::Result< void, E >::~Result
[SWS_CORE_00831]	Definition of API function ara::core::Result< void, E >::FromValue
[SWS_CORE_00834]	Definition of API function ara::core::Result< void, E >::FromError
[SWS_CORE_00835]	Definition of API function ara::core::Result< void, E >::FromError
[SWS_CORE_00836]	Definition of API function ara::core::Result< void, E >::FromError
[SWS_CORE_00841]	Definition of API function ara::core::Result< void, E >::operator=
[SWS_CORE_00842]	Definition of API function ara::core::Result< void, E >::operator=
[SWS_CORE_00843]	Definition of API function ara::core::Result< void, E >::EmplaceValue
[SWS_CORE_00844]	Definition of API function ara::core::Result< void, E >::EmplaceError
[SWS_CORE_00845]	Definition of API function ara::core::Result< void, E >::Swap
[SWS_CORE_00851]	Definition of API function ara::core::Result< void, E >::HasValue
[SWS_CORE_00852]	Definition of API function ara::core::Result< void, E >::operator bool
[SWS_CORE_00853]	Definition of API function ara::core::Result< void, E >::operator*
[SWS_CORE_00855]	Definition of API function ara::core::Result< void, E >::Value

▽

△

Number	Heading
[SWS_CORE_00857]	Definition of API function ara::core::Result< void, E >::Error
[SWS_CORE_00858]	Definition of API function ara::core::Result< void, E >::Error
[SWS_CORE_00861]	Definition of API function ara::core::Result< void, E >::ValueOr
[SWS_CORE_00863]	Definition of API function ara::core::Result< void, E >::ErrorOr
[SWS_CORE_00864]	Definition of API function ara::core::Result< void, E >::ErrorOr
[SWS_CORE_00865]	Definition of API function ara::core::Result< void, E >::CheckError
[SWS_CORE_00866]	Definition of API function ara::core::Result< void, E >::ValueOrThrow
[SWS_CORE_00867]	Definition of API function ara::core::Result< void, E >::Resolve
[SWS_CORE_00868]	Definition of API function ara::core::Result< void, E >::Err
[SWS_CORE_00869]	Definition of API function ara::core::Result< void, E >::Err
[SWS_CORE_00870]	Definition of API function ara::core::Result< void, E >::Bind
[SWS_CORE_00876]	Definition of API function ara::core::Result< void, E >::Error
[SWS_CORE_01096]	Definition of API function ara::core::swap
[SWS_CORE_01241]	Definition of API function ara::core::Array::fill
[SWS_CORE_01242]	Definition of API function ara::core::Array::swap
[SWS_CORE_01250]	Definition of API function ara::core::Array::begin
[SWS_CORE_01251]	Definition of API function ara::core::Array::begin
[SWS_CORE_01252]	Definition of API function ara::core::Array::end
[SWS_CORE_01253]	Definition of API function ara::core::Array::end
[SWS_CORE_01254]	Definition of API function ara::core::Array::rbegin
[SWS_CORE_01255]	Definition of API function ara::core::Array::rbegin
[SWS_CORE_01256]	Definition of API function ara::core::Array::rend
[SWS_CORE_01257]	Definition of API function ara::core::Array::rend
[SWS_CORE_01258]	Definition of API function ara::core::Array::cbegin
[SWS_CORE_01259]	Definition of API function ara::core::Array::cend
[SWS_CORE_01260]	Definition of API function ara::core::Array::crbegin
[SWS_CORE_01261]	Definition of API function ara::core::Array::crend
[SWS_CORE_01262]	Definition of API function ara::core::Array::size
[SWS_CORE_01263]	Definition of API function ara::core::Array::max_size
[SWS_CORE_01264]	Definition of API function ara::core::Array::empty
[SWS_CORE_01265]	Definition of API function ara::core::Array::operator[]
[SWS_CORE_01266]	Definition of API function ara::core::Array::operator[]
[SWS_CORE_01267]	Definition of API function ara::core::Array::front
[SWS_CORE_01268]	Definition of API function ara::core::Array::front
[SWS_CORE_01269]	Definition of API function ara::core::Array::back
[SWS_CORE_01270]	Definition of API function ara::core::Array::back
[SWS_CORE_01271]	Definition of API function ara::core::Array::data
[SWS_CORE_01272]	Definition of API function ara::core::Array::data

▽

△

Number	Heading
[SWS_CORE_01273]	Definition of API function ara::core::Array::at
[SWS_CORE_01274]	Definition of API function ara::core::Array::at
[SWS_CORE_01282]	Definition of API function ara::core::get
[SWS_CORE_01283]	Definition of API function ara::core::get
[SWS_CORE_01284]	Definition of API function ara::core::get
[SWS_CORE_01290]	Definition of API function ara::core::operator==
[SWS_CORE_01291]	Definition of API function ara::core::operator!=
[SWS_CORE_01292]	Definition of API function ara::core::operator<
[SWS_CORE_01293]	Definition of API function ara::core::operator>
[SWS_CORE_01294]	Definition of API function ara::core::operator<=
[SWS_CORE_01295]	Definition of API function ara::core::operator>=
[SWS_CORE_01296]	Definition of API function ara::core::swap
[SWS_CORE_01390]	Definition of API function ara::core::operator==
[SWS_CORE_01391]	Definition of API function ara::core::operator!=
[SWS_CORE_01392]	Definition of API function ara::core::operator<
[SWS_CORE_01393]	Definition of API function ara::core::operator<=
[SWS_CORE_01394]	Definition of API function ara::core::operator>
[SWS_CORE_01395]	Definition of API function ara::core::operator>=
[SWS_CORE_01396]	Definition of API function ara::core::swap
[SWS_CORE_01496]	Definition of API function ara::core::swap
[SWS_CORE_01601]	Definition of API class ara::core::Variant
[SWS_CORE_01696]	Definition of API function ara::core::swap
[SWS_CORE_01941]	Definition of API function ara::core::Span::Span
[SWS_CORE_01942]	Definition of API function ara::core::Span::Span
[SWS_CORE_01943]	Definition of API function ara::core::Span::Span
[SWS_CORE_01944]	Definition of API function ara::core::Span::Span
[SWS_CORE_01945]	Definition of API function ara::core::Span::Span
[SWS_CORE_01946]	Definition of API function ara::core::Span::Span
[SWS_CORE_01947]	Definition of API function ara::core::Span::Span
[SWS_CORE_01948]	Definition of API function ara::core::Span::Span
[SWS_CORE_01949]	Definition of API function ara::core::Span::Span
[SWS_CORE_01950]	Definition of API function ara::core::Span::Span
[SWS_CORE_01951]	Definition of API function ara::core::Span::~Span
[SWS_CORE_01952]	Definition of API function ara::core::Span::operator=
[SWS_CORE_01953]	Definition of API function ara::core::Span::Span
[SWS_CORE_01954]	Definition of API function ara::core::Span::Span
[SWS_CORE_01959]	Definition of API function ara::core::Span::front
[SWS_CORE_01960]	Definition of API function ara::core::Span::back

▽

△

Number	Heading
[SWS_CORE_01961]	Definition of API function ara::core::Span::first
[SWS_CORE_01962]	Definition of API function ara::core::Span::first
[SWS_CORE_01963]	Definition of API function ara::core::Span::last
[SWS_CORE_01964]	Definition of API function ara::core::Span::last
[SWS_CORE_01965]	Definition of API function ara::core::Span::subspan
[SWS_CORE_01966]	Definition of API function ara::core::Span::subspan
[SWS_CORE_01967]	Definition of API function ara::core::Span::size
[SWS_CORE_01968]	Definition of API function ara::core::Span::size_bytes
[SWS_CORE_01969]	Definition of API function ara::core::Span::empty
[SWS_CORE_01970]	Definition of API function ara::core::Span::operator[]
[SWS_CORE_01971]	Definition of API function ara::core::Span::data
[SWS_CORE_01972]	Definition of API function ara::core::Span::begin
[SWS_CORE_01973]	Definition of API function ara::core::Span::end
[SWS_CORE_01974]	Definition of API function ara::core::Span::cbegin
[SWS_CORE_01975]	Definition of API function ara::core::Span::cend
[SWS_CORE_01976]	Definition of API function ara::core::Span::rbegin
[SWS_CORE_01977]	Definition of API function ara::core::Span::rend
[SWS_CORE_01978]	Definition of API function ara::core::Span::crbegin
[SWS_CORE_01979]	Definition of API function ara::core::Span::crend
[SWS_CORE_01980]	Definition of API function ara::core::as_bytes
[SWS_CORE_01981]	Definition of API function ara::core::as_writable_bytes
[SWS_CORE_01990]	Definition of API function ara::core::MakeSpan
[SWS_CORE_01991]	Definition of API function ara::core::MakeSpan
[SWS_CORE_01992]	Definition of API function ara::core::MakeSpan
[SWS_CORE_01993]	Definition of API function ara::core::MakeSpan
[SWS_CORE_01994]	Definition of API function ara::core::MakeSpan
[SWS_CORE_02001]	Definition of API class ara::core::StringView
[SWS_CORE_04012]	Definition of API function ara::core::in_place_t::in_place_t
[SWS_CORE_04022]	Definition of API function ara::core::in_place_type_t::in_place_type_t
[SWS_CORE_04032]	Definition of API function ara::core::in_place_index_t::in_place_index_t
[SWS_CORE_04110]	Definition of API function ara::core::data
[SWS_CORE_04111]	Definition of API function ara::core::data
[SWS_CORE_04112]	Definition of API function ara::core::data
[SWS_CORE_04113]	Definition of API function ara::core::data
[SWS_CORE_04120]	Definition of API function ara::core::size
[SWS_CORE_04121]	Definition of API function ara::core::size
[SWS_CORE_04130]	Definition of API function ara::core::empty
[SWS_CORE_04131]	Definition of API function ara::core::empty

▽

△

Number	Heading
[SWS_CORE_04132]	Definition of API function ara::core::empty
[SWS_CORE_04200]	Definition of API type ara::core::Byte
[SWS_CORE_05200]	Definition of API enum ara::core::CoreErrc
[SWS_CORE_05212]	Definition of API function ara::core::CoreException::CoreException
[SWS_CORE_05221]	Definition of API class ara::core::CoreErrorDomain
[SWS_CORE_05241]	Definition of API function ara::core::CoreErrorDomain::CoreErrorDomain
[SWS_CORE_05242]	Definition of API function ara::core::CoreErrorDomain::Name
[SWS_CORE_05243]	Definition of API function ara::core::CoreErrorDomain::Message
[SWS_CORE_05244]	Definition of API function ara::core::CoreErrorDomain::ThrowAsException
[SWS_CORE_05280]	Definition of API function ara::core::GetCoreErrorDomain
[SWS_CORE_05290]	Definition of API function ara::core::MakeErrorCode
[SWS_CORE_06221]	Definition of API class ara::core::Future< void, E >
[SWS_CORE_06222]	Definition of API function ara::core::Future< void, E >::Future
[SWS_CORE_06223]	Definition of API function ara::core::Future< void, E >::Future
[SWS_CORE_06225]	Definition of API function ara::core::Future< void, E >::operator=
[SWS_CORE_06226]	Definition of API function ara::core::Future< void, E >::get
[SWS_CORE_06227]	Definition of API function ara::core::Future< void, E >::valid
[SWS_CORE_06228]	Definition of API function ara::core::Future< void, E >::wait
[SWS_CORE_06229]	Definition of API function ara::core::Future< void, E >::wait_for
[SWS_CORE_06230]	Definition of API function ara::core::Future< void, E >::wait_until
[SWS_CORE_06231]	Definition of API function ara::core::Future< void, E >::then
[SWS_CORE_06232]	Definition of API function ara::core::Future< void, E >::is_ready
[SWS_CORE_06233]	Definition of API function ara::core::Future< void, E >::~Future
[SWS_CORE_06236]	Definition of API function ara::core::Future< void, E >::GetResult
[SWS_CORE_06237]	Definition of API function ara::core::Future< void, E >::then
[SWS_CORE_06340]	Definition of API class ara::core::Promise< void, E >
[SWS_CORE_06341]	Definition of API function ara::core::Promise< void, E >::Promise
[SWS_CORE_06342]	Definition of API function ara::core::Promise< void, E >::Promise
[SWS_CORE_06343]	Definition of API function ara::core::Promise< void, E >::operator=
[SWS_CORE_06344]	Definition of API function ara::core::Promise< void, E >::get_future
[SWS_CORE_06345]	Definition of API function ara::core::Promise< void, E >::set_value
[SWS_CORE_06349]	Definition of API function ara::core::Promise< void, E >::~Promise
[SWS_CORE_06352]	Definition of API function ara::core::Promise< void, E >::swap
[SWS_CORE_06353]	Definition of API function ara::core::Promise< void, E >::SetError
[SWS_CORE_06354]	Definition of API function ara::core::Promise< void, E >::SetError
[SWS_CORE_06355]	Definition of API function ara::core::Promise< void, E >::SetResult
[SWS_CORE_06356]	Definition of API function ara::core::Promise< void, E >::SetResult
[SWS_CORE_06432]	Definition of API function ara::core::SteadyClock::now

▽

△

Number	Heading
[SWS_CORE_06500]	Definition of API class ara::core::MemoryResource
[SWS_CORE_06501]	Definition of API function ara::core::MemoryResource::MemoryResource
[SWS_CORE_06502]	Definition of API function ara::core::MemoryResource::MemoryResource
[SWS_CORE_06503]	Definition of API function ara::core::MemoryResource::allocate
[SWS_CORE_06504]	Definition of API function ara::core::MemoryResource::deallocate
[SWS_CORE_06505]	Definition of API function ara::core::MemoryResource::is_equal
[SWS_CORE_06506]	Definition of API function ara::core::MemoryResource::~MemoryResource
[SWS_CORE_06507]	Definition of API function ara::core::MemoryResource::operator=
[SWS_CORE_06521]	Definition of API function ara::core::MonotonicBufferResource::Monotonic BufferResource
[SWS_CORE_06522]	Definition of API function ara::core::MonotonicBufferResource::Monotonic BufferResource
[SWS_CORE_06523]	Definition of API function ara::core::MonotonicBufferResource::Monotonic BufferResource
[SWS_CORE_06524]	Definition of API function ara::core::MonotonicBufferResource::Monotonic BufferResource
[SWS_CORE_06525]	Definition of API function ara::core::MonotonicBufferResource::Monotonic BufferResource
[SWS_CORE_06526]	Definition of API function ara::core::MonotonicBufferResource::Monotonic BufferResource
[SWS_CORE_06528]	Definition of API function ara::core::MonotonicBufferResource::~Monotonic BufferResource
[SWS_CORE_06530]	Definition of API function ara::core::MonotonicBufferResource::release
[SWS_CORE_06531]	Definition of API function ara::core::MonotonicBufferResource::upstream_ resource
[SWS_CORE_06541]	Definition of API function ara::core::PolymorphicAllocator::Polymorphic Allocator
[SWS_CORE_06542]	Definition of API function ara::core::PolymorphicAllocator::Polymorphic Allocator
[SWS_CORE_06543]	Definition of API function ara::core::PolymorphicAllocator::Polymorphic Allocator
[SWS_CORE_06544]	Definition of API function ara::core::PolymorphicAllocator::Polymorphic Allocator
[SWS_CORE_06546]	Definition of API function ara::core::PolymorphicAllocator::Polymorphic Allocator
[SWS_CORE_06547]	Definition of API function ara::core::PolymorphicAllocator::allocate
[SWS_CORE_06548]	Definition of API function ara::core::PolymorphicAllocator::deallocate
[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

▽

△

Number	Heading
[SWS_CORE_06555]	Definition of API function ara::core::PolymorphicAllocator::construct
[SWS_CORE_06556]	Definition of API function ara::core::PolymorphicAllocator::destroy
[SWS_CORE_06557]	Definition of API function ara::core::PolymorphicAllocator::resource
[SWS_CORE_06560]	Definition of API function ara::core::operator==
[SWS_CORE_06561]	Definition of API function ara::core::operator==
[SWS_CORE_06562]	Definition of API function ara::core::NewDeleteResource
[SWS_CORE_06563]	Definition of API function ara::core::NullMemoryResource
[SWS_CORE_06564]	Definition of API function ara::core::SetDefaultResource
[SWS_CORE_06565]	Definition of API function ara::core::GetDefaultResource
[SWS_CORE_08021]	Definition of API function ara::core::InstanceSpecifier::InstanceSpecifier
[SWS_CORE_08022]	Definition of API function ara::core::InstanceSpecifier::InstanceSpecifier
[SWS_CORE_08023]	Definition of API function ara::core::InstanceSpecifier::InstanceSpecifier
[SWS_CORE_08024]	Definition of API function ara::core::InstanceSpecifier::operator=
[SWS_CORE_08025]	Definition of API function ara::core::InstanceSpecifier::operator=
[SWS_CORE_08029]	Definition of API function ara::core::InstanceSpecifier::~InstanceSpecifier
[SWS_CORE_08032]	Definition of API function ara::core::InstanceSpecifier::Create
[SWS_CORE_08041]	Definition of API function ara::core::InstanceSpecifier::ToString
[SWS_CORE_08042]	Definition of API function ara::core::InstanceSpecifier::operator==
[SWS_CORE_08043]	Definition of API function ara::core::InstanceSpecifier::operator==
[SWS_CORE_08044]	Definition of API function ara::core::InstanceSpecifier::operator!=
[SWS_CORE_08045]	Definition of API function ara::core::InstanceSpecifier::operator!=
[SWS_CORE_08046]	Definition of API function ara::core::InstanceSpecifier::operator<
[SWS_CORE_08081]	Definition of API function ara::core::operator==
[SWS_CORE_08082]	Definition of API function ara::core::operator!=
[SWS_CORE_10001]	Definition of API function ara::core::Initialize
[SWS_CORE_10002]	Definition of API function ara::core::Deinitialize
[SWS_CORE_10200]	Valid InstanceSpecifier representations - application interaction
[SWS_CORE_10203]	Valid InstanceSpecifier representations - functional cluster interaction
[SWS_CORE_10600]	Semantics of `ara::core::Result`
[SWS_CORE_10800]	Semantics of `ara::core::Future` and `ara::core::Promise`
[SWS_CORE_12403]	Standard Error Stream Logging of `Explicitly aborting an Operation`
[SWS_CORE_90001]	Include folder structure
[SWS_CORE_90005]	Custom declarations and definitions
[SWS_CORE_90021]	Pre-conditions for `ara::core::InstanceSpecifier`

Table F.15: Changed Specification Items in R24-11

F.6.3 Deleted Specification Items in R24-11

Number	Heading
[SWS_CORE_01030]	`value` member function overloads
[SWS_CORE_11950]	MemoryResource base behavior
[SWS_CORE_11951]	MemoryResource error behavior
[SWS_CORE_12402]	"Noreturn" property for Abort
[SWS_CORE_90006]	

Table F.16: Deleted Specification Items in R24-11

F.7 Change History of this document according to AUTOSAR Release R25-11

F.7.1 Added Specification Items in R25-11

Number	Heading
[SWS_CORE_00094]	Error domain name
[SWS_CORE_00095]	Error domain message
[SWS_CORE_00778]	Definition of API function ara::core::Result::ValueOrThrow
[SWS_CORE_00949]	Definition of API function ara::core::Result< T &, E >::ValueOrThrow
[SWS_CORE_01141]	Definition of Header File ara/core/optional.h
[SWS_CORE_01142]	Definition of API function ara::core::operator==
[SWS_CORE_01143]	Definition of API function ara::core::operator!=
[SWS_CORE_01144]	Definition of API function ara::core::operator<
[SWS_CORE_01145]	Definition of API function ara::core::operator>
[SWS_CORE_01146]	Definition of API function ara::core::operator<=
[SWS_CORE_01147]	Definition of API function ara::core::operator>=
[SWS_CORE_01148]	Definition of API function ara::core::operator==
[SWS_CORE_01149]	Definition of API function ara::core::operator==
[SWS_CORE_01176]	Definition of API function ara::core::operator!=
[SWS_CORE_01177]	Definition of API function ara::core::operator!=
[SWS_CORE_01178]	Definition of API function ara::core::operator<
[SWS_CORE_01179]	Definition of API function ara::core::operator<
[SWS_CORE_01180]	Definition of API function ara::core::operator<=
[SWS_CORE_01181]	Definition of API function ara::core::operator<=
[SWS_CORE_01182]	Definition of API function ara::core::operator>
[SWS_CORE_01183]	Definition of API function ara::core::operator>
[SWS_CORE_01184]	Definition of API function ara::core::operator>=

▽

△

Number	Heading
[SWS_CORE_01185]	Definition of API function ara::core::operator>=
[SWS_CORE_01186]	Definition of API function ara::core::operator==
[SWS_CORE_01187]	Definition of API function ara::core::operator==
[SWS_CORE_01188]	Definition of API function ara::core::operator!=
[SWS_CORE_01189]	Definition of API function ara::core::operator!=
[SWS_CORE_01190]	Definition of API function ara::core::operator<
[SWS_CORE_01191]	Definition of API function ara::core::operator<
[SWS_CORE_01192]	Definition of API function ara::core::operator<=
[SWS_CORE_01193]	Definition of API function ara::core::operator<=
[SWS_CORE_01194]	Definition of API function ara::core::operator>
[SWS_CORE_01195]	Definition of API function ara::core::operator>
[SWS_CORE_01196]	Definition of API function ara::core::operator>=
[SWS_CORE_01197]	Definition of API function ara::core::operator>=
[SWS_CORE_01198]	Definition of API class std::hash< ara::core::Optional< T > >
[SWS_CORE_01199]	Definition of API function std::hash< ara::core::Optional< T > >::operator()
[SWS_CORE_01200]	Definition of Header File ara/core/array.h
[SWS_CORE_01286]	Definition of API function ara::core::get
[SWS_CORE_01300]	Definition of Header File ara/core/vector.h
[SWS_CORE_01302]	Definition of API type ara::core::Vector::value_type
[SWS_CORE_01303]	Definition of API type ara::core::Vector::allocator_type
[SWS_CORE_01304]	Definition of API type ara::core::Vector::pointer
[SWS_CORE_01305]	Definition of API type ara::core::Vector::const_pointer
[SWS_CORE_01306]	Definition of API type ara::core::Vector::reference
[SWS_CORE_01307]	Definition of API type ara::core::Vector::const_reference
[SWS_CORE_01308]	Definition of API type ara::core::Vector::size_type
[SWS_CORE_01309]	Definition of API type ara::core::Vector::difference_type
[SWS_CORE_01310]	Definition of API type ara::core::Vector::iterator
[SWS_CORE_01311]	Definition of API type ara::core::Vector::const_iterator
[SWS_CORE_01312]	Definition of API type ara::core::Vector::reverse_iterator
[SWS_CORE_01313]	Definition of API type ara::core::Vector::const_reverse_iterator
[SWS_CORE_01314]	Definition of API function ara::core::Vector::Vector
[SWS_CORE_01315]	Definition of API function ara::core::Vector::operator=
[SWS_CORE_01316]	Definition of API function ara::core::Vector::Vector
[SWS_CORE_01317]	Definition of API function ara::core::Vector::operator=
[SWS_CORE_01318]	Definition of API function ara::core::Vector::Vector
[SWS_CORE_01319]	Definition of API function ara::core::Vector::Vector
[SWS_CORE_01320]	Definition of API function ara::core::Vector::Vector
[SWS_CORE_01321]	Definition of API function ara::core::Vector::Vector

▽

△

Number	Heading
[SWS_CORE_01322]	Definition of API function ara::core::Vector::Vector
[SWS_CORE_01323]	Definition of API function ara::core::Vector::Vector
[SWS_CORE_01324]	Definition of API function ara::core::Vector::Vector
[SWS_CORE_01325]	Definition of API function ara::core::Vector::Vector
[SWS_CORE_01326]	Definition of API function ara::core::Vector::~Vector
[SWS_CORE_01327]	Definition of API function ara::core::Vector::operator=
[SWS_CORE_01328]	Definition of API function ara::core::Vector::assign
[SWS_CORE_01329]	Definition of API function ara::core::Vector::assign
[SWS_CORE_01330]	Definition of API function ara::core::Vector::assign
[SWS_CORE_01331]	Definition of API function ara::core::Vector::get_allocator
[SWS_CORE_01332]	Definition of API function ara::core::Vector::begin
[SWS_CORE_01333]	Definition of API function ara::core::Vector::begin
[SWS_CORE_01334]	Definition of API function ara::core::Vector::end
[SWS_CORE_01335]	Definition of API function ara::core::Vector::end
[SWS_CORE_01336]	Definition of API function ara::core::Vector::rbegin
[SWS_CORE_01337]	Definition of API function ara::core::Vector::rbegin
[SWS_CORE_01338]	Definition of API function ara::core::Vector::rend
[SWS_CORE_01339]	Definition of API function ara::core::Vector::rend
[SWS_CORE_01340]	Definition of API function ara::core::Vector::cbegin
[SWS_CORE_01341]	Definition of API function ara::core::Vector::cend
[SWS_CORE_01342]	Definition of API function ara::core::Vector::crbegin
[SWS_CORE_01343]	Definition of API function ara::core::Vector::crend
[SWS_CORE_01344]	Definition of API function ara::core::Vector::empty
[SWS_CORE_01345]	Definition of API function ara::core::Vector::size
[SWS_CORE_01346]	Definition of API function ara::core::Vector::max_size
[SWS_CORE_01347]	Definition of API function ara::core::Vector::reserve
[SWS_CORE_01348]	Definition of API function ara::core::Vector::capacity
[SWS_CORE_01349]	Definition of API function ara::core::Vector::shrink_to_fit
[SWS_CORE_01350]	Definition of API function ara::core::Vector::resize
[SWS_CORE_01351]	Definition of API function ara::core::Vector::resize
[SWS_CORE_01352]	Definition of API function ara::core::Vector::operator[]
[SWS_CORE_01353]	Definition of API function ara::core::Vector::operator[]
[SWS_CORE_01354]	Definition of API function ara::core::Vector::at
[SWS_CORE_01355]	Definition of API function ara::core::Vector::at
[SWS_CORE_01356]	Definition of API function ara::core::Vector::front
[SWS_CORE_01357]	Definition of API function ara::core::Vector::front
[SWS_CORE_01358]	Definition of API function ara::core::Vector::back
[SWS_CORE_01359]	Definition of API function ara::core::Vector::back

▽

△

Number	Heading
[SWS_CORE_01360]	Definition of API function ara::core::Vector::data
[SWS_CORE_01361]	Definition of API function ara::core::Vector::data
[SWS_CORE_01362]	Definition of API function ara::core::Vector::emplace_back
[SWS_CORE_01363]	Definition of API function ara::core::Vector::push_back
[SWS_CORE_01364]	Definition of API function ara::core::Vector::push_back
[SWS_CORE_01365]	Definition of API function ara::core::Vector::pop_back
[SWS_CORE_01366]	Definition of API function ara::core::Vector::emplace
[SWS_CORE_01367]	Definition of API function ara::core::Vector::insert
[SWS_CORE_01368]	Definition of API function ara::core::Vector::insert
[SWS_CORE_01369]	Definition of API function ara::core::Vector::insert
[SWS_CORE_01370]	Definition of API function ara::core::Vector::insert
[SWS_CORE_01371]	Definition of API function ara::core::Vector::insert
[SWS_CORE_01372]	Definition of API function ara::core::Vector::erase
[SWS_CORE_01373]	Definition of API function ara::core::Vector::erase
[SWS_CORE_01374]	Definition of API function ara::core::Vector::swap
[SWS_CORE_01375]	Definition of API function ara::core::Vector::clear
[SWS_CORE_01376]	Definition of API class std::hash< ara::core::Vector< T, Allocator > >
[SWS_CORE_01377]	Definition of API function std::hash< ara::core::Vector< T, Allocator > >::operator()
[SWS_CORE_01401]	Definition of Header File ara/core/map.h
[SWS_CORE_01402]	Definition of API type ara::core::Map::key_type
[SWS_CORE_01403]	Definition of API type ara::core::Map::mapped_type
[SWS_CORE_01404]	Definition of API type ara::core::Map::value_type
[SWS_CORE_01405]	Definition of API type ara::core::Map::key_compare
[SWS_CORE_01406]	Definition of API type ara::core::Map::allocator_type
[SWS_CORE_01407]	Definition of API type ara::core::Map::pointer
[SWS_CORE_01408]	Definition of API type ara::core::Map::const_pointer
[SWS_CORE_01409]	Definition of API type ara::core::Map::reference
[SWS_CORE_01410]	Definition of API type ara::core::Map::const_reference
[SWS_CORE_01411]	Definition of API type ara::core::Map::size_type
[SWS_CORE_01412]	Definition of API type ara::core::Map::difference_type
[SWS_CORE_01413]	Definition of API type ara::core::Map::iterator
[SWS_CORE_01414]	Definition of API type ara::core::Map::const_iterator
[SWS_CORE_01415]	Definition of API type ara::core::Map::reverse_iterator
[SWS_CORE_01416]	Definition of API type ara::core::Map::const_reverse_iterator
[SWS_CORE_01417]	Definition of API type ara::core::Map::node_type
[SWS_CORE_01418]	Definition of API type ara::core::Map::insert_return_type
[SWS_CORE_01419]	Definition of API class ara::core::Map::value_compare

▽

△

Number	Heading
[SWS_CORE_01421]	Definition of API variable ara::core::Map::value_compare::comp
[SWS_CORE_01422]	Definition of API function ara::core::Map::value_compare::value_compare
[SWS_CORE_01423]	Definition of API function ara::core::Map::value_compare::operator()
[SWS_CORE_01424]	Definition of API function ara::core::Map::Map
[SWS_CORE_01425]	Definition of API function ara::core::Map::operator=
[SWS_CORE_01426]	Definition of API function ara::core::Map::Map
[SWS_CORE_01427]	Definition of API function ara::core::Map::operator=
[SWS_CORE_01428]	Definition of API function ara::core::Map::Map
[SWS_CORE_01429]	Definition of API function ara::core::Map::Map
[SWS_CORE_01430]	Definition of API function ara::core::Map::Map
[SWS_CORE_01431]	Definition of API function ara::core::Map::Map
[SWS_CORE_01432]	Definition of API function ara::core::Map::Map
[SWS_CORE_01433]	Definition of API function ara::core::Map::Map
[SWS_CORE_01434]	Definition of API function ara::core::Map::Map
[SWS_CORE_01435]	Definition of API function ara::core::Map::Map
[SWS_CORE_01436]	Definition of API function ara::core::Map::Map
[SWS_CORE_01437]	Definition of API function ara::core::Map::~Map
[SWS_CORE_01438]	Definition of API function ara::core::Map::operator=
[SWS_CORE_01439]	Definition of API function ara::core::Map::get_allocator
[SWS_CORE_01440]	Definition of API function ara::core::Map::begin
[SWS_CORE_01441]	Definition of API function ara::core::Map::begin
[SWS_CORE_01442]	Definition of API function ara::core::Map::end
[SWS_CORE_01443]	Definition of API function ara::core::Map::end
[SWS_CORE_01444]	Definition of API function ara::core::Map::rbegin
[SWS_CORE_01445]	Definition of API function ara::core::Map::rbegin
[SWS_CORE_01446]	Definition of API function ara::core::Map::rend
[SWS_CORE_01447]	Definition of API function ara::core::Map::rend
[SWS_CORE_01448]	Definition of API function ara::core::Map::cbegin
[SWS_CORE_01449]	Definition of API function ara::core::Map::cend
[SWS_CORE_01450]	Definition of API function ara::core::Map::crbegin
[SWS_CORE_01451]	Definition of API function ara::core::Map::crend
[SWS_CORE_01452]	Definition of API function ara::core::Map::size
[SWS_CORE_01453]	Definition of API function ara::core::Map::max_size
[SWS_CORE_01454]	Definition of API function ara::core::Map::operator[]
[SWS_CORE_01455]	Definition of API function ara::core::Map::operator[]
[SWS_CORE_01456]	Definition of API function ara::core::Map::at
[SWS_CORE_01457]	Definition of API function ara::core::Map::at
[SWS_CORE_01458]	Definition of API function ara::core::Map::emplace

▽

△

Number	Heading
[SWS_CORE_01459]	Definition of API function ara::core::Map::emplace_hint
[SWS_CORE_01460]	Definition of API function ara::core::Map::insert
[SWS_CORE_01461]	Definition of API function ara::core::Map::insert
[SWS_CORE_01462]	Definition of API function ara::core::Map::insert
[SWS_CORE_01463]	Definition of API function ara::core::Map::insert
[SWS_CORE_01464]	Definition of API function ara::core::Map::insert
[SWS_CORE_01465]	Definition of API function ara::core::Map::insert
[SWS_CORE_01466]	Definition of API function ara::core::Map::insert
[SWS_CORE_01467]	Definition of API function ara::core::Map::insert
[SWS_CORE_01468]	Definition of API function ara::core::Map::insert
[SWS_CORE_01469]	Definition of API function ara::core::Map::insert
[SWS_CORE_01470]	Definition of API function ara::core::Map::extract
[SWS_CORE_01471]	Definition of API function ara::core::Map::extract
[SWS_CORE_01472]	Definition of API function ara::core::Map::try_emplace
[SWS_CORE_01473]	Definition of API function ara::core::Map::try_emplace
[SWS_CORE_01474]	Definition of API function ara::core::Map::try_emplace
[SWS_CORE_01475]	Definition of API function ara::core::Map::try_emplace
[SWS_CORE_01476]	Definition of API function ara::core::Map::insert_or_assign
[SWS_CORE_01477]	Definition of API function ara::core::Map::insert_or_assign
[SWS_CORE_01478]	Definition of API function ara::core::Map::insert_or_assign
[SWS_CORE_01479]	Definition of API function ara::core::Map::insert_or_assign
[SWS_CORE_01480]	Definition of API function ara::core::Map::erase
[SWS_CORE_01481]	Definition of API function ara::core::Map::erase
[SWS_CORE_01482]	Definition of API function ara::core::Map::erase
[SWS_CORE_01483]	Definition of API function ara::core::Map::erase
[SWS_CORE_01484]	Definition of API function ara::core::Map::swap
[SWS_CORE_01485]	Definition of API function ara::core::Map::clear
[SWS_CORE_01486]	Definition of API function ara::core::Map::merge
[SWS_CORE_01487]	Definition of API function ara::core::Map::merge
[SWS_CORE_01488]	Definition of API function ara::core::Map::key_comp
[SWS_CORE_01489]	Definition of API function ara::core::Map::value_comp
[SWS_CORE_01490]	Definition of API function ara::core::Map::find
[SWS_CORE_01491]	Definition of API function ara::core::Map::find
[SWS_CORE_01492]	Definition of API function ara::core::Map::find
[SWS_CORE_01493]	Definition of API function ara::core::Map::find
[SWS_CORE_01494]	Definition of API function ara::core::Map::count
[SWS_CORE_01495]	Definition of API function ara::core::Map::count
[SWS_CORE_01497]	Definition of API function ara::core::Map::contains

▽

△

Number	Heading
[SWS_CORE_01498]	Definition of API function ara::core::Map::lower_bound
[SWS_CORE_01499]	Definition of API function ara::core::Map::lower_bound
[SWS_CORE_01500]	Definition of API function ara::core::Map::lower_bound
[SWS_CORE_01501]	Definition of API function ara::core::Map::lower_bound
[SWS_CORE_01502]	Definition of API function ara::core::Map::upper_bound
[SWS_CORE_01503]	Definition of API function ara::core::Map::upper_bound
[SWS_CORE_01504]	Definition of API function ara::core::Map::upper_bound
[SWS_CORE_01505]	Definition of API function ara::core::Map::upper_bound
[SWS_CORE_01506]	Definition of API function ara::core::Map::equal_range
[SWS_CORE_01507]	Definition of API function ara::core::Map::equal_range
[SWS_CORE_01508]	Definition of API function ara::core::Map::equal_range
[SWS_CORE_01509]	Definition of API function ara::core::Map::equal_range
[SWS_CORE_01510]	Definition of API function ara::core::Map::contains
[SWS_CORE_01511]	Definition of API function ara::core::Map::empty
[SWS_CORE_01672]	Definition of Header File ara/core/variant.h
[SWS_CORE_01700]	Definition of Header File ara/core/bitset.h
[SWS_CORE_01701]	Definition of API class ara::core::Bitset
[SWS_CORE_01703]	Definition of API function ara::core::Bitset::reference::reference
[SWS_CORE_01704]	Definition of API function ara::core::Bitset::reference::reference
[SWS_CORE_01705]	Definition of API function ara::core::Bitset::reference::operator=
[SWS_CORE_01706]	Definition of API function ara::core::Bitset::reference::reference
[SWS_CORE_01707]	Definition of API function ara::core::Bitset::reference::operator=
[SWS_CORE_01708]	Definition of API function ara::core::Bitset::reference::~reference
[SWS_CORE_01709]	Definition of API function ara::core::Bitset::reference::operator=
[SWS_CORE_01710]	Definition of API function ara::core::Bitset::reference::operator~
[SWS_CORE_01711]	Definition of API function ara::core::Bitset::reference::operator bool
[SWS_CORE_01712]	Definition of API function ara::core::Bitset::reference::flip
[SWS_CORE_01713]	Definition of API function ara::core::Bitset::Bitset
[SWS_CORE_01714]	Definition of API function ara::core::Bitset::Bitset
[SWS_CORE_01715]	Definition of API function ara::core::Bitset::operator=
[SWS_CORE_01716]	Definition of API function ara::core::Bitset::Bitset
[SWS_CORE_01717]	Definition of API function ara::core::Bitset::operator=
[SWS_CORE_01718]	Definition of API function ara::core::Bitset::~Bitset
[SWS_CORE_01719]	Definition of API function ara::core::Bitset::Bitset
[SWS_CORE_01720]	Definition of API function ara::core::Bitset::operator&=
[SWS_CORE_01721]	Definition of API function ara::core::Bitset::operator|=
[SWS_CORE_01722]	Definition of API function ara::core::Bitset::operatorˆ=
[SWS_CORE_01723]	Definition of API function ara::core::Bitset::operator<<=

▽

△

Number	Heading
[SWS_CORE_01724]	Definition of API function ara::core::Bitset::operator>>=
[SWS_CORE_01725]	Definition of API function ara::core::Bitset::set
[SWS_CORE_01726]	Definition of API function ara::core::Bitset::set
[SWS_CORE_01727]	Definition of API function ara::core::Bitset::reset
[SWS_CORE_01728]	Definition of API function ara::core::Bitset::reset
[SWS_CORE_01729]	Definition of API function ara::core::Bitset::operator~
[SWS_CORE_01730]	Definition of API function ara::core::Bitset::flip
[SWS_CORE_01731]	Definition of API function ara::core::Bitset::flip
[SWS_CORE_01732]	Definition of API function ara::core::Bitset::operator[]
[SWS_CORE_01733]	Definition of API function ara::core::Bitset::operator[]
[SWS_CORE_01734]	Definition of API function ara::core::Bitset::to_ulong
[SWS_CORE_01735]	Definition of API function ara::core::Bitset::to_ullong
[SWS_CORE_01736]	Definition of API function ara::core::Bitset::to_uint8t
[SWS_CORE_01737]	Definition of API function ara::core::Bitset::to_uint16t
[SWS_CORE_01738]	Definition of API function ara::core::Bitset::to_uint32t
[SWS_CORE_01739]	Definition of API function ara::core::Bitset::to_uint64t
[SWS_CORE_01740]	Definition of API function ara::core::Bitset::count
[SWS_CORE_01741]	Definition of API function ara::core::Bitset::size
[SWS_CORE_01742]	Definition of API function ara::core::Bitset::operator==
[SWS_CORE_01743]	Definition of API function ara::core::Bitset::test
[SWS_CORE_01744]	Definition of API function ara::core::Bitset::all
[SWS_CORE_01745]	Definition of API function ara::core::Bitset::any
[SWS_CORE_01746]	Definition of API function ara::core::Bitset::none
[SWS_CORE_01747]	Definition of API function ara::core::Bitset::operator<<
[SWS_CORE_01748]	Definition of API function ara::core::Bitset::operator>>
[SWS_CORE_01749]	Definition of API function ara::core::Bitset::Bitset
[SWS_CORE_01750]	Definition of API function ara::core::Bitset::Bitset
[SWS_CORE_01751]	Definition of API function ara::core::Bitset::to_string
[SWS_CORE_01752]	Definition of API function ara::core::operator&
[SWS_CORE_01753]	Definition of API function ara::core::operator|
[SWS_CORE_01754]	Definition of API function ara::core::operatorˆ
[SWS_CORE_01755]	Definition of API function ara::core::operator>>
[SWS_CORE_01756]	Definition of API function ara::core::operator<<
[SWS_CORE_01757]	Definition of API class ara::core::hash
[SWS_CORE_01759]	Definition of API class ara::core::Bitset::reference
[SWS_CORE_01902]	Definition of Header File ara/core/span.h
[SWS_CORE_02002]	Definition of Header File ara/core/string_view.h
[SWS_CORE_03002]	Definition of Header File ara/core/string.h

▽

△

Number	Heading
[SWS_CORE_03003]	Definition of API type ara::core::BasicString::traits_type
[SWS_CORE_03004]	Definition of API type ara::core::BasicString::value_type
[SWS_CORE_03005]	Definition of API type ara::core::BasicString::allocator_type
[SWS_CORE_03013]	Definition of API type ara::core::BasicString::difference_type
[SWS_CORE_03014]	Definition of API type ara::core::BasicString::pointer
[SWS_CORE_03015]	Definition of API type ara::core::BasicString::const_pointer
[SWS_CORE_03016]	Definition of API type ara::core::BasicString::reference
[SWS_CORE_03017]	Definition of API type ara::core::BasicString::const_reference
[SWS_CORE_03018]	Definition of API type ara::core::BasicString::reverse_iterator
[SWS_CORE_03019]	Definition of API type ara::core::BasicString::const_reverse_iterator
[SWS_CORE_03020]	Definition of API function ara::core::BasicString::BasicString
[SWS_CORE_03021]	Definition of API function ara::core::BasicString::operator=
[SWS_CORE_03022]	Definition of API function ara::core::BasicString::BasicString
[SWS_CORE_03023]	Definition of API function ara::core::BasicString::operator=
[SWS_CORE_03024]	Definition of API function ara::core::BasicString::BasicString
[SWS_CORE_03025]	Definition of API function ara::core::BasicString::BasicString
[SWS_CORE_03026]	Definition of API function ara::core::BasicString::BasicString
[SWS_CORE_03027]	Definition of API function ara::core::BasicString::BasicString
[SWS_CORE_03028]	Definition of API function ara::core::BasicString::BasicString
[SWS_CORE_03029]	Definition of API function ara::core::BasicString::BasicString
[SWS_CORE_03030]	Definition of API function ara::core::BasicString::BasicString
[SWS_CORE_03031]	Definition of API function ara::core::BasicString::BasicString
[SWS_CORE_03032]	Definition of API function ara::core::BasicString::BasicString
[SWS_CORE_03033]	Definition of API function ara::core::BasicString::BasicString
[SWS_CORE_03034]	Definition of API function ara::core::BasicString::BasicString
[SWS_CORE_03035]	Definition of API function ara::core::BasicString::operator=
[SWS_CORE_03036]	Definition of API function ara::core::BasicString::operator=
[SWS_CORE_03037]	Definition of API function ara::core::BasicString::operator=
[SWS_CORE_03038]	Definition of API function ara::core::BasicString::~BasicString
[SWS_CORE_03039]	Definition of API function ara::core::BasicString::begin
[SWS_CORE_03040]	Definition of API function ara::core::BasicString::begin
[SWS_CORE_03041]	Definition of API function ara::core::BasicString::end
[SWS_CORE_03042]	Definition of API function ara::core::BasicString::end
[SWS_CORE_03043]	Definition of API function ara::core::BasicString::rbegin
[SWS_CORE_03044]	Definition of API function ara::core::BasicString::rbegin
[SWS_CORE_03045]	Definition of API function ara::core::BasicString::rend
[SWS_CORE_03046]	Definition of API function ara::core::BasicString::rend
[SWS_CORE_03047]	Definition of API function ara::core::BasicString::cbegin

▽

△

Number	Heading
[SWS_CORE_03048]	Definition of API function ara::core::BasicString::cend
[SWS_CORE_03049]	Definition of API function ara::core::BasicString::crbegin
[SWS_CORE_03050]	Definition of API function ara::core::BasicString::crend
[SWS_CORE_03051]	Definition of API function ara::core::BasicString::empty
[SWS_CORE_03052]	Definition of API function ara::core::BasicString::size
[SWS_CORE_03053]	Definition of API function ara::core::BasicString::max_size
[SWS_CORE_03054]	Definition of API function ara::core::BasicString::reserve
[SWS_CORE_03055]	Definition of API function ara::core::BasicString::capacity
[SWS_CORE_03056]	Definition of API function ara::core::BasicString::shrink_to_fit
[SWS_CORE_03057]	Definition of API function ara::core::BasicString::clear
[SWS_CORE_03058]	Definition of API function ara::core::BasicString::resize
[SWS_CORE_03059]	Definition of API function ara::core::BasicString::resize
[SWS_CORE_03060]	Definition of API function ara::core::BasicString::length
[SWS_CORE_03061]	Definition of API function ara::core::BasicString::operator[]
[SWS_CORE_03062]	Definition of API function ara::core::BasicString::operator[]
[SWS_CORE_03063]	Definition of API function ara::core::BasicString::at
[SWS_CORE_03064]	Definition of API function ara::core::BasicString::at
[SWS_CORE_03065]	Definition of API function ara::core::BasicString::front
[SWS_CORE_03066]	Definition of API function ara::core::BasicString::front
[SWS_CORE_03067]	Definition of API function ara::core::BasicString::back
[SWS_CORE_03068]	Definition of API function ara::core::BasicString::back
[SWS_CORE_03069]	Definition of API function ara::core::BasicString::c_str
[SWS_CORE_03070]	Definition of API function ara::core::BasicString::data
[SWS_CORE_03071]	Definition of API function ara::core::BasicString::data
[SWS_CORE_03072]	Definition of API function ara::core::BasicString::operator+=
[SWS_CORE_03073]	Definition of API function ara::core::BasicString::operator+=
[SWS_CORE_03074]	Definition of API function ara::core::BasicString::operator+=
[SWS_CORE_03075]	Definition of API function ara::core::BasicString::operator+=
[SWS_CORE_03076]	Definition of API function ara::core::BasicString::append
[SWS_CORE_03077]	Definition of API function ara::core::BasicString::append
[SWS_CORE_03078]	Definition of API function ara::core::BasicString::append
[SWS_CORE_03079]	Definition of API function ara::core::BasicString::append
[SWS_CORE_03080]	Definition of API function ara::core::BasicString::append
[SWS_CORE_03081]	Definition of API function ara::core::BasicString::append
[SWS_CORE_03082]	Definition of API function ara::core::BasicString::append
[SWS_CORE_03083]	Definition of API function ara::core::BasicString::push_back
[SWS_CORE_03084]	Definition of API function ara::core::BasicString::assign
[SWS_CORE_03085]	Definition of API function ara::core::BasicString::assign

▽

△

Number	Heading
[SWS_CORE_03086]	Definition of API function ara::core::BasicString::assign
[SWS_CORE_03087]	Definition of API function ara::core::BasicString::assign
[SWS_CORE_03088]	Definition of API function ara::core::BasicString::assign
[SWS_CORE_03089]	Definition of API function ara::core::BasicString::assign
[SWS_CORE_03090]	Definition of API function ara::core::BasicString::assign
[SWS_CORE_03091]	Definition of API function ara::core::BasicString::assign
[SWS_CORE_03092]	Definition of API function ara::core::BasicString::assign
[SWS_CORE_03093]	Definition of API function ara::core::BasicString::insert
[SWS_CORE_03094]	Definition of API function ara::core::BasicString::insert
[SWS_CORE_03095]	Definition of API function ara::core::BasicString::insert
[SWS_CORE_03096]	Definition of API function ara::core::BasicString::insert
[SWS_CORE_03097]	Definition of API function ara::core::BasicString::insert
[SWS_CORE_03098]	Definition of API function ara::core::BasicString::insert
[SWS_CORE_03099]	Definition of API function ara::core::BasicString::insert
[SWS_CORE_03100]	Definition of API function ara::core::BasicString::insert
[SWS_CORE_03101]	Definition of API function ara::core::BasicString::insert
[SWS_CORE_03102]	Definition of API function ara::core::BasicString::erase
[SWS_CORE_03103]	Definition of API function ara::core::BasicString::erase
[SWS_CORE_03104]	Definition of API function ara::core::BasicString::erase
[SWS_CORE_03105]	Definition of API function ara::core::BasicString::pop_back
[SWS_CORE_03106]	Definition of API function ara::core::BasicString::replace
[SWS_CORE_03107]	Definition of API function ara::core::BasicString::replace
[SWS_CORE_03108]	Definition of API function ara::core::BasicString::replace
[SWS_CORE_03109]	Definition of API function ara::core::BasicString::replace
[SWS_CORE_03110]	Definition of API function ara::core::BasicString::replace
[SWS_CORE_03111]	Definition of API function ara::core::BasicString::replace
[SWS_CORE_03112]	Definition of API function ara::core::BasicString::replace
[SWS_CORE_03324]	Definition of API function ara::core::BasicString::find
[SWS_CORE_03325]	Definition of API function ara::core::BasicString::rfind
[SWS_CORE_03326]	Definition of API function ara::core::BasicString::rfind
[SWS_CORE_03327]	Definition of API function ara::core::BasicString::rfind
[SWS_CORE_03328]	Definition of API function ara::core::BasicString::rfind
[SWS_CORE_03329]	Definition of API function ara::core::BasicString::find_first_of
[SWS_CORE_03330]	Definition of API function ara::core::BasicString::find_first_of
[SWS_CORE_03331]	Definition of API function ara::core::BasicString::find_first_of
[SWS_CORE_03332]	Definition of API function ara::core::BasicString::find_first_of
[SWS_CORE_03333]	Definition of API function ara::core::BasicString::find_last_of
[SWS_CORE_03334]	Definition of API function ara::core::BasicString::find_last_of

▽

△

Number	Heading
[SWS_CORE_03335]	Definition of API function ara::core::BasicString::find_last_of
[SWS_CORE_03336]	Definition of API function ara::core::BasicString::find_last_of
[SWS_CORE_03337]	Definition of API function ara::core::BasicString::find_first_not_of
[SWS_CORE_03338]	Definition of API function ara::core::BasicString::find_first_not_of
[SWS_CORE_03339]	Definition of API function ara::core::BasicString::find_first_not_of
[SWS_CORE_03340]	Definition of API function ara::core::BasicString::find_first_not_of
[SWS_CORE_03341]	Definition of API function ara::core::BasicString::find_last_not_of
[SWS_CORE_03342]	Definition of API function ara::core::BasicString::find_last_not_of
[SWS_CORE_03343]	Definition of API function ara::core::BasicString::find_last_not_of
[SWS_CORE_03344]	Definition of API function ara::core::BasicString::find_last_not_of
[SWS_CORE_03345]	Definition of API function ara::core::BasicString::substr
[SWS_CORE_03346]	Definition of API function ara::core::BasicString::compare
[SWS_CORE_03347]	Definition of API function ara::core::BasicString::compare
[SWS_CORE_03348]	Definition of API function ara::core::BasicString::compare
[SWS_CORE_03349]	Definition of API function ara::core::BasicString::compare
[SWS_CORE_03350]	Definition of API function ara::core::BasicString::compare
[SWS_CORE_03351]	Definition of API function ara::core::BasicString::compare
[SWS_CORE_03352]	Definition of API function ara::core::operator+
[SWS_CORE_03353]	Definition of API function ara::core::operator+
[SWS_CORE_03354]	Definition of API function ara::core::operator+
[SWS_CORE_03355]	Definition of API function ara::core::operator+
[SWS_CORE_03356]	Definition of API function ara::core::operator+
[SWS_CORE_03357]	Definition of API function ara::core::operator+
[SWS_CORE_03358]	Definition of API function ara::core::operator+
[SWS_CORE_03359]	Definition of API function ara::core::operator+
[SWS_CORE_03360]	Definition of API function ara::core::operator+
[SWS_CORE_03361]	Definition of API function ara::core::operator+
[SWS_CORE_03362]	Definition of API function ara::core::operator+
[SWS_CORE_03363]	Definition of API function ara::core::operator+
[SWS_CORE_03364]	Definition of API function ara::core::operator==
[SWS_CORE_03365]	Definition of API function ara::core::operator==
[SWS_CORE_03366]	Definition of API function ara::core::operator==
[SWS_CORE_03367]	Definition of API function ara::core::operator!=
[SWS_CORE_03368]	Definition of API function ara::core::operator!=
[SWS_CORE_03369]	Definition of API function ara::core::operator!=
[SWS_CORE_03370]	Definition of API function ara::core::operator<
[SWS_CORE_03371]	Definition of API function ara::core::operator<
[SWS_CORE_03372]	Definition of API function ara::core::operator<

▽

△

Number	Heading
[SWS_CORE_03373]	Definition of API function ara::core::operator>
[SWS_CORE_03374]	Definition of API function ara::core::operator>
[SWS_CORE_03375]	Definition of API function ara::core::operator>
[SWS_CORE_03376]	Definition of API function ara::core::operator<=
[SWS_CORE_03377]	Definition of API function ara::core::operator<=
[SWS_CORE_03378]	Definition of API function ara::core::operator<=
[SWS_CORE_03379]	Definition of API function ara::core::operator>=
[SWS_CORE_03380]	Definition of API function ara::core::operator>=
[SWS_CORE_03381]	Definition of API function ara::core::operator>=
[SWS_CORE_03382]	Definition of API function ara::core::to_string
[SWS_CORE_03383]	Definition of API function ara::core::to_string
[SWS_CORE_03384]	Definition of API function ara::core::to_string
[SWS_CORE_03385]	Definition of API function ara::core::to_string
[SWS_CORE_03386]	Definition of API function ara::core::to_string
[SWS_CORE_03387]	Definition of API function ara::core::to_string
[SWS_CORE_03388]	Definition of API function ara::core::to_string
[SWS_CORE_03389]	Definition of API function ara::core::to_string
[SWS_CORE_03390]	Definition of API function ara::core::to_string
[SWS_CORE_03391]	Definition of API function ara::core::stoi
[SWS_CORE_03392]	Definition of API function ara::core::stol
[SWS_CORE_03393]	Definition of API function ara::core::stoul
[SWS_CORE_03394]	Definition of API function ara::core::stoll
[SWS_CORE_03395]	Definition of API function ara::core::stoull
[SWS_CORE_03396]	Definition of API function ara::core::stof
[SWS_CORE_03397]	Definition of API function ara::core::stod
[SWS_CORE_03398]	Definition of API function ara::core::stold
[SWS_CORE_03399]	Definition of API function ara::core::operator>>
[SWS_CORE_03400]	Definition of API function ara::core::operator<<
[SWS_CORE_03401]	Definition of API function ara::core::getline
[SWS_CORE_03402]	Definition of API function ara::core::getline
[SWS_CORE_03403]	Definition of API function ara::core::getline
[SWS_CORE_03404]	Definition of API function ara::core::getline
[SWS_CORE_03405]	Definition of API class std::hash< ara::core::String >
[SWS_CORE_03406]	Definition of API function ara::core::BasicString::replace
[SWS_CORE_03407]	Definition of API function ara::core::BasicString::find
[SWS_CORE_03408]	Definition of API function ara::core::BasicString::find
[SWS_CORE_03409]	Definition of API function ara::core::BasicString::find
[SWS_CORE_03410]	Definition of API function ara::core::BasicString::copy

▽

△

Number	Heading
[SWS_CORE_03411]	Definition of API function ara::core::BasicString::replace
[SWS_CORE_03412]	Definition of API function ara::core::BasicString::swap
[SWS_CORE_03413]	Definition of API function ara::core::BasicString::replace
[SWS_CORE_03414]	Definition of API function ara::core::BasicString::replace
[SWS_CORE_03415]	Definition of API function ara::core::BasicString::get_allocator
[SWS_CORE_04014]	Definition of API class ara::core::is_nothrow_swappable
[SWS_CORE_06578]	Definition of Header File ara/core/memory_resource.h
[SWS_CORE_13000]	ViolationMessage InsufficientPermissionsViolation
[SWS_CORE_13001]	ViolationMessage IsoStdBadVariantAccessViolation
[SWS_CORE_13003]	ViolationMessage InstanceSpecifierMappingIntegrityViolation
[SWS_CORE_13004]	ViolationMessage PortInterfaceMappingViolation
[SWS_CORE_13005]	ViolationMessage ProcessMappingViolation
[SWS_CORE_13006]	ViolationMessage InstanceSpecifierAlreadyInUseViolation
[SWS_CORE_13007]	ViolationMessage AraNotInitializedViolation
[SWS_CORE_13008]	ViolationMessage WrongDomainOfAnErrorCodeViolation
[SWS_CORE_13009]	ViolationMessage ResultWithNoValueViolation
[SWS_CORE_13010]	ViolationMessage ResultWithNoErrorViolation
[SWS_CORE_13011]	ViolationMessage FutureInvalidViolation
[SWS_CORE_13012]	ViolationMessage FutureContinuationHasThrownViolation
[SWS_CORE_13013]	ViolationMessage IsoStdFutureErrcFutureAlreadyRetrievedViolation
[SWS_CORE_13014]	ViolationMessage IsoStdFutureErrcNoStateViolation
[SWS_CORE_13015]	ViolationMessage IsoStdFutureErrcPromiseAlreadySatisfiedViolation
[SWS_CORE_13016]	ViolationMessage PromiseNoSharedStateToSetViolation
[SWS_CORE_13017]	ViolationMessage IsoStdOutOfRangeViolation
[SWS_CORE_13020]	ViolationMessage IsoStdBadOptionalAccessViolation
[SWS_CORE_13021]	ViolationMessage IsoStdBadAllocViolation
[SWS_CORE_13022]	ViolationMessage IsoStdInvalidArgumentViolation
[SWS_CORE_13023]	ViolationMessage IsoStdOverflowErrorViolation
[SWS_CORE_13024]	ViolationMessage IsoStdLengthErrorViolation
[SWS_CORE_13211]	AUTOSAR definition of a thread-safe static member function
[SWS_CORE_13212]	Definition of `functions implemented by the user of the API`
[SWS_CORE_13213]	Usage of thread-safe user-implemented functions
[SWS_CORE_13214]	Usage of non-thread-safe user-implemented functions
[SWS_CORE_90031]	Undefined behavior in AUTOSAR APIs
[SWS_CORE_90032]	Implementation-defined behavior in AUTOSAR APIs
[SWS_CORE_90033]	Vendor-specific behavior in AUTOSAR APIs

Table F.17: Added Specification Items in R25-11

F.7.2 Changed Specification Items in R25-11

Number	Heading
[SWS_CORE_00003]	Handling of Non-Standardized Violations
[SWS_CORE_00009]	Definition of API function ara::core::Executor::Executor
[SWS_CORE_00010]	Error domain identifier
[SWS_CORE_00015]	Definition of API function ara::core::Executor::Executor
[SWS_CORE_00017]	Definition of API function ara::core::Executor::operator=
[SWS_CORE_00018]	Definition of API function ara::core::Executor::operator=
[SWS_CORE_00019]	Definition of API function ara::core::Executor::~Executor
[SWS_CORE_00024]	Definition of API function ara::core::Executor::execute
[SWS_CORE_00025]	Definition of API function ara::core::Executor::oneway_execute
[SWS_CORE_00030]	Definition of API function ara::core::SynchronizedPool Resource::SynchronizedPoolResource
[SWS_CORE_00031]	Definition of API function ara::core::SynchronizedPool Resource::SynchronizedPoolResource
[SWS_CORE_00032]	Definition of API function ara::core::SynchronizedPool Resource::SynchronizedPoolResource
[SWS_CORE_00033]	Definition of API function ara::core::SynchronizedPool Resource::SynchronizedPoolResource
[SWS_CORE_00034]	Definition of API function ara::core::SynchronizedPool Resource::SynchronizedPoolResource
[SWS_CORE_00035]	Definition of API function ara::core::SynchronizedPool Resource::~SynchronizedPoolResource
[SWS_CORE_00037]	Definition of API function ara::core::SynchronizedPoolResource::release
[SWS_CORE_00038]	Definition of API function ara::core::SynchronizedPoolResource::upstream_ resource
[SWS_CORE_00039]	Definition of API function ara::core::SynchronizedPoolResource::options
[SWS_CORE_00040]	Errors originating from C++ standard classes
[SWS_CORE_00041]	Definition of API function ara::core::SynchronizedPoolResource::do_ deallocate
[SWS_CORE_00042]	Definition of API function ara::core::SynchronizedPoolResource::do_is_ equal
[SWS_CORE_00043]	Definition of API function ara::core::SynchronizedPool Resource::SynchronizedPoolResource
[SWS_CORE_00044]	Definition of API function ara::core::SynchronizedPoolResource::operator=
[SWS_CORE_00048]	Definition of API function ara::core::UnsynchronizedPool Resource::UnsynchronizedPoolResource
[SWS_CORE_00049]	Definition of API function ara::core::UnsynchronizedPool Resource::operator=
[SWS_CORE_00051]	Definition of API function ara::core::SetAbortHandler
[SWS_CORE_00052]	Definition of API function ara::core::Abort

▽

△

Number	Heading
[SWS_CORE_00054]	Definition of API function ara::core::AddAbortHandler
[SWS_CORE_00056]	Definition of API function ara::core::UnsynchronizedPool Resource::upstream_resource
[SWS_CORE_00057]	Definition of API function ara::core::UnsynchronizedPoolResource::options
[SWS_CORE_00058]	Definition of API function ara::core::UnsynchronizedPoolResource::do_ allocate
[SWS_CORE_00059]	Definition of API function ara::core::UnsynchronizedPoolResource::do_ deallocate
[SWS_CORE_00060]	Definition of API function ara::core::UnsynchronizedPoolResource::do_is_ equal
[SWS_CORE_00061]	Definition of API function ara::core::SynchronizedPoolResource::do_allocate
[SWS_CORE_00062]	Definition of API function ara::core::UnsynchronizedPool Resource::~UnsynchronizedPoolResource
[SWS_CORE_00063]	Definition of API function ara::core::UnsynchronizedPool Resource::UnsynchronizedPoolResource
[SWS_CORE_00064]	Definition of API function ara::core::UnsynchronizedPool Resource::UnsynchronizedPoolResource
[SWS_CORE_00065]	Definition of API function ara::core::UnsynchronizedPoolResource::release
[SWS_CORE_00066]	Definition of API function ara::core::PolymorphicAllocator::operator=
[SWS_CORE_00067]	Definition of API function ara::core::UnsynchronizedPool Resource::UnsynchronizedPoolResource
[SWS_CORE_00068]	Definition of API function ara::core::UnsynchronizedPool Resource::UnsynchronizedPoolResource
[SWS_CORE_00071]	Definition of API type ara::core::BasicString::iterator
[SWS_CORE_00072]	Definition of API type ara::core::BasicString::const_iterator
[SWS_CORE_00073]	Definition of API variable ara::core::BasicString::npos
[SWS_CORE_00123]	Definition of API type ara::core::ErrorDomain::SupportDataType
[SWS_CORE_00136]	Definition of API function ara::core::ErrorDomain::~ErrorDomain
[SWS_CORE_00322]	Definition of API function ara::core::Future::Future
[SWS_CORE_00323]	Definition of API function ara::core::Future::Future
[SWS_CORE_00326]	Definition of API function ara::core::Future::get
[SWS_CORE_00331]	Definition of API function ara::core::Future::then
[SWS_CORE_00336]	Definition of API function ara::core::Future::GetResult
[SWS_CORE_00337]	Definition of API function ara::core::Future::then
[SWS_CORE_00342]	Definition of API function ara::core::Promise::Promise
[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_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

▽

△

Number	Heading
[SWS_CORE_00356]	Definition of API function ara::core::Promise::SetResult
[SWS_CORE_00361]	Definition of API enum ara::core::FutureStatus
[SWS_CORE_00442]	Definition of API function ara::core::FutureErrorDomain::Name
[SWS_CORE_00513]	Definition of API function ara::core::ErrorCode::ErrorCode
[SWS_CORE_00614]	Definition of API function ara::core::Exception::operator=
[SWS_CORE_00615]	Definition of API function ara::core::Exception::Exception
[SWS_CORE_00616]	Definition of API function ara::core::Exception::operator=
[SWS_CORE_00617]	Definition of API function ara::core::Exception::~Exception
[SWS_CORE_00618]	Definition of API function ara::core::Exception::Exception
[SWS_CORE_00721]	Definition of API function ara::core::Result::Result
[SWS_CORE_00722]	Definition of API function ara::core::Result::Result
[SWS_CORE_00723]	Definition of API function ara::core::Result::Result
[SWS_CORE_00724]	Definition of API function ara::core::Result::Result
[SWS_CORE_00725]	Definition of API function ara::core::Result::Result
[SWS_CORE_00726]	Definition of API function ara::core::Result::Result
[SWS_CORE_00731]	Definition of API function ara::core::Result::FromValue
[SWS_CORE_00732]	Definition of API function ara::core::Result::FromValue
[SWS_CORE_00733]	Definition of API function ara::core::Result::FromValue
[SWS_CORE_00734]	Definition of API function ara::core::Result::FromError
[SWS_CORE_00735]	Definition of API function ara::core::Result::FromError
[SWS_CORE_00736]	Definition of API function ara::core::Result::FromError
[SWS_CORE_00823]	Definition of API function ara::core::Result< void, E >::Result
[SWS_CORE_00824]	Definition of API function ara::core::Result< void, E >::Result
[SWS_CORE_00825]	Definition of API function ara::core::Result< void, E >::Result
[SWS_CORE_00826]	Definition of API function ara::core::Result< void, E >::Result
[SWS_CORE_00834]	Definition of API function ara::core::Result< void, E >::FromError
[SWS_CORE_00835]	Definition of API function ara::core::Result< void, E >::FromError
[SWS_CORE_00836]	Definition of API function ara::core::Result< void, E >::FromError
[SWS_CORE_00904]	Definition of API function ara::core::Result< T &, E >::FromValue
[SWS_CORE_00905]	Definition of API function ara::core::Result< T &, E >::FromError
[SWS_CORE_00906]	Definition of API function ara::core::Result< T &, E >::FromError
[SWS_CORE_00907]	Definition of API function ara::core::Result< T &, E >::FromError
[SWS_CORE_00908]	Definition of API function ara::core::Result< T &, E >::Result
[SWS_CORE_00909]	Definition of API function ara::core::Result< T &, E >::Result
[SWS_CORE_00910]	Definition of API function ara::core::Result< T &, E >::Result
[SWS_CORE_00911]	Definition of API function ara::core::Result< T &, E >::Result
[SWS_CORE_00912]	Definition of API function ara::core::Result< T &, E >::Result
[SWS_CORE_01096]	Definition of API function ara::core::swap

▽

△

Number	Heading
[SWS_CORE_01100]	Definition of API class ara::core::nullopt_t
[SWS_CORE_01101]	Definition of API variable ara::core::nullopt
[SWS_CORE_01103]	Definition of API function ara::core::Optional::Optional
[SWS_CORE_01104]	Definition of API function ara::core::Optional::Optional
[SWS_CORE_01105]	Definition of API function ara::core::Optional::Optional
[SWS_CORE_01106]	Definition of API function ara::core::Optional::Optional
[SWS_CORE_01107]	Definition of API function ara::core::Optional::Optional
[SWS_CORE_01108]	Definition of API function ara::core::Optional::Optional
[SWS_CORE_01109]	Definition of API function ara::core::Optional::Optional
[SWS_CORE_01110]	Definition of API function ara::core::Optional::Optional
[SWS_CORE_01111]	Definition of API function ara::core::Optional::Optional
[SWS_CORE_01112]	Definition of API function ara::core::Optional::~Optional
[SWS_CORE_01113]	Definition of API function ara::core::Optional::operator=
[SWS_CORE_01114]	Definition of API function ara::core::Optional::operator=
[SWS_CORE_01115]	Definition of API function ara::core::Optional::operator=
[SWS_CORE_01116]	Definition of API function ara::core::Optional::operator=
[SWS_CORE_01117]	Definition of API function ara::core::Optional::operator=
[SWS_CORE_01118]	Definition of API function ara::core::Optional::operator=
[SWS_CORE_01119]	Definition of API function ara::core::Optional::emplace
[SWS_CORE_01120]	Definition of API function ara::core::Optional::emplace
[SWS_CORE_01121]	Definition of API function ara::core::Optional::swap
[SWS_CORE_01122]	Definition of API function ara::core::Optional::operator->
[SWS_CORE_01123]	Definition of API function ara::core::Optional::operator->
[SWS_CORE_01124]	Definition of API function ara::core::Optional::operator*
[SWS_CORE_01125]	Definition of API function ara::core::Optional::operator*
[SWS_CORE_01126]	Definition of API function ara::core::Optional::operator*
[SWS_CORE_01127]	Definition of API function ara::core::Optional::operator*
[SWS_CORE_01128]	Definition of API function ara::core::Optional::operator bool
[SWS_CORE_01129]	Definition of API function ara::core::Optional::has_value
[SWS_CORE_01130]	Definition of API function ara::core::Optional::value
[SWS_CORE_01131]	Definition of API function ara::core::Optional::value
[SWS_CORE_01132]	Definition of API function ara::core::Optional::value
[SWS_CORE_01133]	Definition of API function ara::core::Optional::value
[SWS_CORE_01134]	Definition of API function ara::core::Optional::value_or
[SWS_CORE_01135]	Definition of API function ara::core::Optional::value_or
[SWS_CORE_01136]	Definition of API function ara::core::Optional::reset
[SWS_CORE_01138]	Definition of API function ara::core::make_optional
[SWS_CORE_01139]	Definition of API function ara::core::make_optional

▽

△

Number	Heading
[SWS_CORE_01140]	Definition of API function ara::core::make_optional
[SWS_CORE_01150]	Definition of API class ara::core::Optional< T & >
[SWS_CORE_01152]	Definition of API function ara::core::Optional< T & >::Optional
[SWS_CORE_01153]	Definition of API function ara::core::Optional< T & >::Optional
[SWS_CORE_01154]	Definition of API function ara::core::Optional< T & >::Optional
[SWS_CORE_01155]	Definition of API function ara::core::Optional< T & >::Optional
[SWS_CORE_01156]	Definition of API function ara::core::Optional< T & >::Optional
[SWS_CORE_01157]	Definition of API function ara::core::Optional< T & >::Optional
[SWS_CORE_01158]	Definition of API function ara::core::Optional< T & >::~Optional
[SWS_CORE_01159]	Definition of API function ara::core::Optional< T & >::operator=
[SWS_CORE_01160]	Definition of API function ara::core::Optional< T & >::operator=
[SWS_CORE_01161]	Definition of API function ara::core::Optional< T & >::operator=
[SWS_CORE_01162]	Definition of API function ara::core::Optional< T & >::operator=
[SWS_CORE_01163]	Definition of API function ara::core::Optional< T & >::operator=
[SWS_CORE_01164]	Definition of API function ara::core::Optional< T & >::emplace
[SWS_CORE_01165]	Definition of API function ara::core::Optional< T & >::reset
[SWS_CORE_01166]	Definition of API function ara::core::Optional< T & >::swap
[SWS_CORE_01167]	Definition of API function ara::core::Optional< T & >::operator->
[SWS_CORE_01168]	Definition of API function ara::core::Optional< T & >::operator->
[SWS_CORE_01169]	Definition of API function ara::core::Optional< T & >::operator*
[SWS_CORE_01170]	Definition of API function ara::core::Optional< T & >::operator*
[SWS_CORE_01171]	Definition of API function ara::core::Optional< T & >::operator bool
[SWS_CORE_01172]	Definition of API function ara::core::Optional< T & >::has_value
[SWS_CORE_01173]	Definition of API function ara::core::Optional< T & >::value
[SWS_CORE_01174]	Definition of API function ara::core::Optional< T & >::value
[SWS_CORE_01175]	Definition of API function ara::core::Optional< T & >::value_or
[SWS_CORE_01241]	Definition of API function ara::core::Array::fill
[SWS_CORE_01242]	Definition of API function ara::core::Array::swap
[SWS_CORE_01250]	Definition of API function ara::core::Array::begin
[SWS_CORE_01251]	Definition of API function ara::core::Array::begin
[SWS_CORE_01252]	Definition of API function ara::core::Array::end
[SWS_CORE_01253]	Definition of API function ara::core::Array::end
[SWS_CORE_01254]	Definition of API function ara::core::Array::rbegin
[SWS_CORE_01255]	Definition of API function ara::core::Array::rbegin
[SWS_CORE_01256]	Definition of API function ara::core::Array::rend
[SWS_CORE_01257]	Definition of API function ara::core::Array::rend
[SWS_CORE_01258]	Definition of API function ara::core::Array::cbegin
[SWS_CORE_01259]	Definition of API function ara::core::Array::cend

▽

△

Number	Heading
[SWS_CORE_01260]	Definition of API function ara::core::Array::crbegin
[SWS_CORE_01261]	Definition of API function ara::core::Array::crend
[SWS_CORE_01262]	Definition of API function ara::core::Array::size
[SWS_CORE_01263]	Definition of API function ara::core::Array::max_size
[SWS_CORE_01264]	Definition of API function ara::core::Array::empty
[SWS_CORE_01265]	Definition of API function ara::core::Array::operator[]
[SWS_CORE_01266]	Definition of API function ara::core::Array::operator[]
[SWS_CORE_01267]	Definition of API function ara::core::Array::front
[SWS_CORE_01268]	Definition of API function ara::core::Array::front
[SWS_CORE_01269]	Definition of API function ara::core::Array::back
[SWS_CORE_01270]	Definition of API function ara::core::Array::back
[SWS_CORE_01271]	Definition of API function ara::core::Array::data
[SWS_CORE_01272]	Definition of API function ara::core::Array::data
[SWS_CORE_01273]	Definition of API function ara::core::Array::at
[SWS_CORE_01274]	Definition of API function ara::core::Array::at
[SWS_CORE_01282]	Definition of API function ara::core::get
[SWS_CORE_01283]	Definition of API function ara::core::get
[SWS_CORE_01284]	Definition of API function ara::core::get
[SWS_CORE_01290]	Definition of API function ara::core::operator==
[SWS_CORE_01291]	Definition of API function ara::core::operator!=
[SWS_CORE_01292]	Definition of API function ara::core::operator<
[SWS_CORE_01293]	Definition of API function ara::core::operator>
[SWS_CORE_01294]	Definition of API function ara::core::operator<=
[SWS_CORE_01295]	Definition of API function ara::core::operator>=
[SWS_CORE_01296]	Definition of API function ara::core::swap
[SWS_CORE_01301]	Definition of API class ara::core::Vector
[SWS_CORE_01390]	Definition of API function ara::core::operator==
[SWS_CORE_01391]	Definition of API function ara::core::operator!=
[SWS_CORE_01392]	Definition of API function ara::core::operator<
[SWS_CORE_01393]	Definition of API function ara::core::operator<=
[SWS_CORE_01394]	Definition of API function ara::core::operator>
[SWS_CORE_01395]	Definition of API function ara::core::operator>=
[SWS_CORE_01396]	Definition of API function ara::core::swap
[SWS_CORE_01400]	Definition of API class ara::core::Map
[SWS_CORE_01496]	Definition of API function ara::core::swap
[SWS_CORE_01613]	Definition of API function ara::core::get
[SWS_CORE_01614]	Definition of API function ara::core::get
[SWS_CORE_01615]	Definition of API function ara::core::get

▽

△

Number	Heading
[SWS_CORE_01616]	Definition of API function ara::core::get
[SWS_CORE_01617]	Definition of API function ara::core::get
[SWS_CORE_01618]	Definition of API function ara::core::get
[SWS_CORE_01619]	Definition of API function ara::core::get
[SWS_CORE_01620]	Definition of API function ara::core::get
[SWS_CORE_01621]	Definition of API function ara::core::get_if
[SWS_CORE_01622]	Definition of API function ara::core::get_if
[SWS_CORE_01623]	Definition of API function ara::core::get_if
[SWS_CORE_01624]	Definition of API function ara::core::get_if
[SWS_CORE_01626]	Definition of API function ara::core::holds_alternative
[SWS_CORE_01627]	Definition of API function ara::core::operator==
[SWS_CORE_01628]	Definition of API function ara::core::operator!=
[SWS_CORE_01629]	Definition of API function ara::core::operator<
[SWS_CORE_01630]	Definition of API function ara::core::operator>
[SWS_CORE_01631]	Definition of API function ara::core::operator<=
[SWS_CORE_01632]	Definition of API function ara::core::operator>=
[SWS_CORE_01633]	Definition of API function ara::core::visit
[SWS_CORE_01634]	Definition of API function ara::core::visit
[SWS_CORE_01640]	Definition of API class ara::core::Monostate
[SWS_CORE_01641]	Definition of API function ara::core::operator==
[SWS_CORE_01642]	Definition of API function ara::core::operator!=
[SWS_CORE_01643]	Definition of API function ara::core::operator<
[SWS_CORE_01644]	Definition of API function ara::core::operator>
[SWS_CORE_01645]	Definition of API function ara::core::operator<=
[SWS_CORE_01646]	Definition of API function ara::core::operator>=
[SWS_CORE_01649]	Definition of API function ara::core::Variant::Variant
[SWS_CORE_01650]	Definition of API function ara::core::Variant::Variant
[SWS_CORE_01651]	Definition of API function ara::core::Variant::Variant
[SWS_CORE_01652]	Definition of API function ara::core::Variant::Variant
[SWS_CORE_01653]	Definition of API function ara::core::Variant::Variant
[SWS_CORE_01654]	Definition of API function ara::core::Variant::Variant
[SWS_CORE_01655]	Definition of API function ara::core::Variant::Variant
[SWS_CORE_01656]	Definition of API function ara::core::Variant::Variant
[SWS_CORE_01657]	Definition of API function ara::core::Variant::~Variant
[SWS_CORE_01658]	Definition of API function ara::core::Variant::operator=
[SWS_CORE_01659]	Definition of API function ara::core::Variant::operator=
[SWS_CORE_01660]	Definition of API function ara::core::Variant::operator=
[SWS_CORE_01661]	Definition of API function ara::core::Variant::emplace

▽

△

Number	Heading
[SWS_CORE_01662]	Definition of API function ara::core::Variant::emplace
[SWS_CORE_01663]	Definition of API function ara::core::Variant::emplace
[SWS_CORE_01664]	Definition of API function ara::core::Variant::emplace
[SWS_CORE_01665]	Definition of API function ara::core::Variant::index
[SWS_CORE_01666]	Definition of API function ara::core::Variant::valueless_by_exception
[SWS_CORE_01667]	Definition of API function ara::core::Variant::swap
[SWS_CORE_01668]	Definition of API class std::hash< ara::core::Variant< Types... > >
[SWS_CORE_01669]	Definition of API function std::hash< ara::core::Variant< Types... > >::operator()
[SWS_CORE_01671]	Definition of API function std::hash< ara::core::Monostate >::operator()
[SWS_CORE_01696]	Definition of API function ara::core::swap
[SWS_CORE_01917]	Definition of API type ara::core::Span::iterator
[SWS_CORE_01918]	Definition of API type ara::core::Span::const_iterator
[SWS_CORE_01941]	Definition of API function ara::core::Span::Span
[SWS_CORE_01942]	Definition of API function ara::core::Span::Span
[SWS_CORE_01943]	Definition of API function ara::core::Span::Span
[SWS_CORE_01944]	Definition of API function ara::core::Span::Span
[SWS_CORE_01945]	Definition of API function ara::core::Span::Span
[SWS_CORE_01946]	Definition of API function ara::core::Span::Span
[SWS_CORE_01947]	Definition of API function ara::core::Span::Span
[SWS_CORE_01948]	Definition of API function ara::core::Span::Span
[SWS_CORE_01949]	Definition of API function ara::core::Span::Span
[SWS_CORE_01950]	Definition of API function ara::core::Span::Span
[SWS_CORE_01951]	Definition of API function ara::core::Span::~Span
[SWS_CORE_01952]	Definition of API function ara::core::Span::operator=
[SWS_CORE_01959]	Definition of API function ara::core::Span::front
[SWS_CORE_01960]	Definition of API function ara::core::Span::back
[SWS_CORE_01961]	Definition of API function ara::core::Span::first
[SWS_CORE_01962]	Definition of API function ara::core::Span::first
[SWS_CORE_01963]	Definition of API function ara::core::Span::last
[SWS_CORE_01964]	Definition of API function ara::core::Span::last
[SWS_CORE_01965]	Definition of API function ara::core::Span::subspan
[SWS_CORE_01966]	Definition of API function ara::core::Span::subspan
[SWS_CORE_01967]	Definition of API function ara::core::Span::size
[SWS_CORE_01968]	Definition of API function ara::core::Span::size_bytes
[SWS_CORE_01969]	Definition of API function ara::core::Span::empty
[SWS_CORE_01970]	Definition of API function ara::core::Span::operator[]
[SWS_CORE_01971]	Definition of API function ara::core::Span::data

▽

△

Number	Heading
[SWS_CORE_01972]	Definition of API function ara::core::Span::begin
[SWS_CORE_01973]	Definition of API function ara::core::Span::end
[SWS_CORE_01974]	Definition of API function ara::core::Span::cbegin
[SWS_CORE_01975]	Definition of API function ara::core::Span::cend
[SWS_CORE_01976]	Definition of API function ara::core::Span::rbegin
[SWS_CORE_01977]	Definition of API function ara::core::Span::rend
[SWS_CORE_01978]	Definition of API function ara::core::Span::crbegin
[SWS_CORE_01979]	Definition of API function ara::core::Span::crend
[SWS_CORE_01980]	Definition of API function ara::core::as_bytes
[SWS_CORE_01981]	Definition of API function ara::core::as_writable_bytes
[SWS_CORE_01991]	Definition of API function ara::core::MakeSpan
[SWS_CORE_02105]	Definition of API type ara::core::StringView::const_iterator
[SWS_CORE_02112]	Definition of API function ara::core::StringView::StringView
[SWS_CORE_02113]	Definition of API function ara::core::StringView::StringView
[SWS_CORE_02114]	Definition of API function ara::core::StringView::StringView
[SWS_CORE_02115]	Definition of API function ara::core::StringView::StringView
[SWS_CORE_02116]	Definition of API function ara::core::StringView::operator=
[SWS_CORE_02117]	Definition of API function ara::core::StringView::StringView
[SWS_CORE_02118]	Definition of API function ara::core::StringView::operator=
[SWS_CORE_02119]	Definition of API function ara::core::StringView::~StringView
[SWS_CORE_02120]	Definition of API function ara::core::StringView::begin
[SWS_CORE_02121]	Definition of API function ara::core::StringView::end
[SWS_CORE_02122]	Definition of API function ara::core::StringView::cbegin
[SWS_CORE_02123]	Definition of API function ara::core::StringView::cend
[SWS_CORE_02124]	Definition of API function ara::core::StringView::rbegin
[SWS_CORE_02125]	Definition of API function ara::core::StringView::rend
[SWS_CORE_02126]	Definition of API function ara::core::StringView::crbegin
[SWS_CORE_02127]	Definition of API function ara::core::StringView::crend
[SWS_CORE_02128]	Definition of API function ara::core::StringView::size
[SWS_CORE_02129]	Definition of API function ara::core::StringView::length
[SWS_CORE_02130]	Definition of API function ara::core::StringView::max_size
[SWS_CORE_02131]	Definition of API function ara::core::StringView::empty
[SWS_CORE_02132]	Definition of API function ara::core::StringView::operator[]
[SWS_CORE_02133]	Definition of API function ara::core::StringView::at
[SWS_CORE_02134]	Definition of API function ara::core::StringView::front
[SWS_CORE_02135]	Definition of API function ara::core::StringView::back
[SWS_CORE_02136]	Definition of API function ara::core::StringView::data
[SWS_CORE_02137]	Definition of API function ara::core::StringView::remove_prefix

▽

△

Number	Heading
[SWS_CORE_02138]	Definition of API function ara::core::StringView::remove_suffix
[SWS_CORE_02139]	Definition of API function ara::core::StringView::swap
[SWS_CORE_02140]	Definition of API function ara::core::StringView::copy
[SWS_CORE_02141]	Definition of API function ara::core::StringView::substr
[SWS_CORE_02142]	Definition of API function ara::core::StringView::compare
[SWS_CORE_02143]	Definition of API function ara::core::StringView::compare
[SWS_CORE_02144]	Definition of API function ara::core::StringView::compare
[SWS_CORE_02145]	Definition of API function ara::core::StringView::compare
[SWS_CORE_02146]	Definition of API function ara::core::StringView::compare
[SWS_CORE_02147]	Definition of API function ara::core::StringView::compare
[SWS_CORE_02148]	Definition of API function ara::core::StringView::starts_with
[SWS_CORE_02149]	Definition of API function ara::core::StringView::starts_with
[SWS_CORE_02150]	Definition of API function ara::core::StringView::starts_with
[SWS_CORE_02151]	Definition of API function ara::core::StringView::ends_with
[SWS_CORE_02152]	Definition of API function ara::core::StringView::ends_with
[SWS_CORE_02153]	Definition of API function ara::core::StringView::ends_with
[SWS_CORE_02154]	Definition of API function ara::core::StringView::contains
[SWS_CORE_02155]	Definition of API function ara::core::StringView::contains
[SWS_CORE_02156]	Definition of API function ara::core::StringView::contains
[SWS_CORE_02157]	Definition of API function ara::core::StringView::find
[SWS_CORE_02158]	Definition of API function ara::core::StringView::find
[SWS_CORE_02159]	Definition of API function ara::core::StringView::find
[SWS_CORE_02160]	Definition of API function ara::core::StringView::find
[SWS_CORE_02161]	Definition of API function ara::core::StringView::rfind
[SWS_CORE_02162]	Definition of API function ara::core::StringView::rfind
[SWS_CORE_02163]	Definition of API function ara::core::StringView::rfind
[SWS_CORE_02164]	Definition of API function ara::core::StringView::rfind
[SWS_CORE_02165]	Definition of API function ara::core::StringView::find_first_of
[SWS_CORE_02166]	Definition of API function ara::core::StringView::find_first_of
[SWS_CORE_02167]	Definition of API function ara::core::StringView::find_first_of
[SWS_CORE_02168]	Definition of API function ara::core::StringView::find_first_of
[SWS_CORE_02169]	Definition of API function ara::core::StringView::find_last_of
[SWS_CORE_02170]	Definition of API function ara::core::StringView::find_last_of
[SWS_CORE_02171]	Definition of API function ara::core::StringView::find_last_of
[SWS_CORE_02172]	Definition of API function ara::core::StringView::find_last_of
[SWS_CORE_02173]	Definition of API function ara::core::StringView::find_first_not_of
[SWS_CORE_02174]	Definition of API function ara::core::StringView::find_first_not_of
[SWS_CORE_02175]	Definition of API function ara::core::StringView::find_first_not_of

▽

△

Number	Heading
[SWS_CORE_02176]	Definition of API function ara::core::StringView::find_first_not_of
[SWS_CORE_02177]	Definition of API function ara::core::StringView::find_last_not_of
[SWS_CORE_02178]	Definition of API function ara::core::StringView::find_last_not_of
[SWS_CORE_02179]	Definition of API function ara::core::StringView::find_last_not_of
[SWS_CORE_02180]	Definition of API function ara::core::StringView::find_last_not_of
[SWS_CORE_02181]	Definition of API function ara::core::operator==
[SWS_CORE_02182]	Definition of API function ara::core::operator!=
[SWS_CORE_02183]	Definition of API function ara::core::operator<
[SWS_CORE_02184]	Definition of API function ara::core::operator>
[SWS_CORE_02185]	Definition of API function ara::core::operator<=
[SWS_CORE_02186]	Definition of API function ara::core::operator>=
[SWS_CORE_02187]	Definition of API function ara::core::operator<<
[SWS_CORE_02188]	Definition of API function ara::core::literals::string_view_literals::operator""_SV
[SWS_CORE_02189]	Definition of API class std::hash< ara::core::StringView >
[SWS_CORE_02190]	Definition of API function std::hash< ara::core::StringView >::operator()
[SWS_CORE_03000]	Definition of API class ara::core::BasicString
[SWS_CORE_03012]	Definition of API type ara::core::BasicString::size_type
[SWS_CORE_03296]	Definition of API function ara::core::swap
[SWS_CORE_03301]	Definition of API function ara::core::BasicString::operator StringView
[SWS_CORE_03302]	Definition of API function ara::core::BasicString::BasicString
[SWS_CORE_03303]	Definition of API function ara::core::BasicString::BasicString
[SWS_CORE_03304]	Definition of API function ara::core::BasicString::operator=
[SWS_CORE_03306]	Definition of API function ara::core::BasicString::assign
[SWS_CORE_03307]	Definition of API function ara::core::BasicString::operator+=
[SWS_CORE_03308]	Definition of API function ara::core::BasicString::append
[SWS_CORE_03309]	Definition of API function ara::core::BasicString::append
[SWS_CORE_03310]	Definition of API function ara::core::BasicString::insert
[SWS_CORE_03311]	Definition of API function ara::core::BasicString::insert
[SWS_CORE_03312]	Definition of API function ara::core::BasicString::replace
[SWS_CORE_03313]	Definition of API function ara::core::BasicString::replace
[SWS_CORE_03314]	Definition of API function ara::core::BasicString::replace
[SWS_CORE_03315]	Definition of API function ara::core::BasicString::find
[SWS_CORE_03316]	Definition of API function ara::core::BasicString::rfind
[SWS_CORE_03317]	Definition of API function ara::core::BasicString::find_first_of
[SWS_CORE_03318]	Definition of API function ara::core::BasicString::find_last_of
[SWS_CORE_03319]	Definition of API function ara::core::BasicString::find_first_not_of
[SWS_CORE_03320]	Definition of API function ara::core::BasicString::find_last_not_of

▽

△

Number	Heading
[SWS_CORE_03321]	Definition of API function ara::core::BasicString::compare
[SWS_CORE_03322]	Definition of API function ara::core::BasicString::compare
[SWS_CORE_03323]	Definition of API function ara::core::BasicString::compare
[SWS_CORE_04012]	Definition of API function ara::core::in_place_t::in_place_t
[SWS_CORE_04022]	Definition of API function ara::core::in_place_type_t::in_place_type_t
[SWS_CORE_04032]	Definition of API function ara::core::in_place_index_t::in_place_index_t
[SWS_CORE_04200]	Definition of API type ara::core::Byte
[SWS_CORE_05200]	Definition of API enum ara::core::CoreErrc
[SWS_CORE_05242]	Definition of API function ara::core::CoreErrorDomain::Name
[SWS_CORE_06223]	Definition of API function ara::core::Future< void, E >::Future
[SWS_CORE_06226]	Definition of API function ara::core::Future< void, E >::get
[SWS_CORE_06231]	Definition of API function ara::core::Future< void, E >::then
[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_06342]	Definition of API function ara::core::Promise< void, E >::Promise
[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_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_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_06507]	Definition of API function ara::core::MemoryResource::operator=
[SWS_CORE_06521]	Definition of API function ara::core::MonotonicBufferResource::Monotonic BufferResource
[SWS_CORE_06522]	Definition of API function ara::core::MonotonicBufferResource::Monotonic BufferResource
[SWS_CORE_06523]	Definition of API function ara::core::MonotonicBufferResource::Monotonic BufferResource
[SWS_CORE_06525]	Definition of API function ara::core::MonotonicBufferResource::Monotonic BufferResource
[SWS_CORE_06526]	Definition of API function ara::core::MonotonicBufferResource::Monotonic BufferResource
[SWS_CORE_06528]	Definition of API function ara::core::MonotonicBufferResource::~Monotonic BufferResource
[SWS_CORE_06542]	Definition of API function ara::core::PolymorphicAllocator::Polymorphic Allocator
[SWS_CORE_06543]	Definition of API function ara::core::PolymorphicAllocator::Polymorphic Allocator

▽

△

Number	Heading
[SWS_CORE_06544]	Definition of API function ara::core::PolymorphicAllocator::Polymorphic Allocator
[SWS_CORE_06546]	Definition of API function ara::core::PolymorphicAllocator::Polymorphic Allocator
[SWS_CORE_06547]	Definition of API function ara::core::PolymorphicAllocator::allocate
[SWS_CORE_06549]	Definition of API function ara::core::PolymorphicAllocator::allocate_bytes
[SWS_CORE_06551]	Definition of API function ara::core::PolymorphicAllocator::allocate_object
[SWS_CORE_06553]	Definition of API function ara::core::PolymorphicAllocator::new_object
[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_06566]	Definition of API function ara::core::MemoryResource::do_allocate
[SWS_CORE_06569]	Definition of API function ara::core::MonotonicBufferResource::do_allocate
[SWS_CORE_06571]	Definition of API function ara::core::MonotonicBufferResource::do_is_equal
[SWS_CORE_06574]	Definition of API function ara::core::MemoryResource::MemoryResource
[SWS_CORE_06575]	Definition of API function ara::core::MemoryResource::operator=
[SWS_CORE_06576]	Definition of API function ara::core::MonotonicBufferResource::Monotonic BufferResource
[SWS_CORE_06577]	Definition of API function ara::core::MonotonicBufferResource::operator=
[SWS_CORE_08021]	Definition of API function ara::core::InstanceSpecifier::InstanceSpecifier
[SWS_CORE_08022]	Definition of API function ara::core::InstanceSpecifier::InstanceSpecifier
[SWS_CORE_08023]	Definition of API function ara::core::InstanceSpecifier::InstanceSpecifier
[SWS_CORE_08032]	Definition of API function ara::core::InstanceSpecifier::Create
[SWS_CORE_10002]	Definition of API function ara::core::Deinitialize
[SWS_CORE_10003]	Definition of API function ara::core::Initialize
[SWS_CORE_10201]	Validation of meta-model paths
[SWS_CORE_13018]	LogMessage FailedDefaultAllocation
[SWS_CORE_13019]	LogMessage AbortMessage
[SWS_CORE_13200]	AUTOSAR definition of a thread-safe member function
[SWS_CORE_90023]	AUTOSAR-standardized Functional Cluster Error Domain Identifiers
[SWS_CORE_90024]	AUTOSAR-standardized Functional Cluster Log and Trace settings
[SWS_CORE_90025]	AUTOSAR-standardized Functional Cluster names

Table F.18: Changed Specification Items in R25-11

F.7.3 Deleted Specification Items in R25-11

Number	Heading
[SWS_CORE_00091]	Messages for `Violations`
[SWS_CORE_00093]	Vendor header file
[SWS_CORE_01031]	class bad_optional_access
[SWS_CORE_01953]	Definition of API function ara::core::Span::Span
[SWS_CORE_01954]	Definition of API function ara::core::Span::Span
[SWS_CORE_03305]	Definition of API function ara::core::BasicString::assign
[SWS_CORE_10001]	Definition of API function ara::core::Initialize
[SWS_CORE_10952]	ErrorDomain sub-class unique identifier retrieval
[SWS_CORE_11000]	Optional base behavior
[SWS_CORE_11200]	Array 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_11952]	Resolution of macro `ARA_COMPILER_DEFINED_NODISCARD`
[SWS_CORE_12000]	String base behavior
[SWS_CORE_12200]	StringView base behavior
[SWS_CORE_13208]	Behavior of non-thread-safe `callable`
[SWS_CORE_13209]	Behavior of conditionally thread-safe `callable`
[SWS_CORE_90021]	Pre-conditions for `ara::core::InstanceSpecifier`

Table F.19: Deleted Specification Items in R25-11