

Writing Regexprs 2021-22 / Regex Parser

Generated by Doxygen 1.9.3

1 Namespace Index	1
1.1 Namespace List	1
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Class Index	5
3.1 Class List	5
4 File Index	7
4.1 File List	7
5 Namespace Documentation	9
5.1 wr22 Namespace Reference	9
5.2 wr22::regex_parser Namespace Reference	9
5.3 wr22::regex_parser::parser Namespace Reference	9
5.3.1 Function Documentation	10
5.3.1.1 parse_regex()	10
5.3.1.2 Parser()	10
5.4 wr22::regex_parser::parser::errors Namespace Reference	10
5.5 wr22::regex_parser::regex Namespace Reference	11
5.5.1 Enumeration Type Documentation	11
5.5.1.1 NamedCaptureFlavor	12
5.5.2 Function Documentation	13
5.5.2.1 operator<<() [1/5]	13
5.5.2.2 operator<<() [2/5]	13
5.5.2.3 operator<<() [3/5]	13
5.5.2.4 operator<<() [4/5]	13
5.5.2.5 operator<<() [5/5]	14
5.5.2.6 to_json() [1/6]	14
5.5.2.7 to_json() [2/6]	14
5.5.2.8 to_json() [3/6]	14
5.5.2.9 to_json() [4/6]	14
5.5.2.10 to_json() [5/6]	14
5.5.2.11 to_json() [6/6]	15
5.6 wr22::regex_parser::regex::capture Namespace Reference	15
5.6.1 Typedef Documentation	15
5.6.1.1 Adt	15
5.6.2 Function Documentation	15
5.6.2.1 to_json() [1/3]	15
5.6.2.2 to_json() [2/3]	16
5.6.2.3 to_json() [3/3]	16
5.7 wr22::regex_parser::regex::part Namespace Reference	16
5.7.1 Detailed Description	17

5.7.2 Typedef Documentation	17
5.7.2.1 Adt	17
5.7.3 Function Documentation	17
5.7.3.1 to_json() [1/10]	17
5.7.3.2 to_json() [2/10]	17
5.7.3.3 to_json() [3/10]	18
5.7.3.4 to_json() [4/10]	18
5.7.3.5 to_json() [5/10]	18
5.7.3.6 to_json() [6/10]	18
5.7.3.7 to_json() [7/10]	18
5.7.3.8 to_json() [8/10]	18
5.7.3.9 to_json() [9/10]	19
5.7.3.10 to_json() [10/10]	19
5.8 wr22::regex_parser::span Namespace Reference	19
5.8.1 Function Documentation	19
5.8.1.1 operator<<()	19
5.8.1.2 to_json()	19
5.9 wr22::regex_parser::utils Namespace Reference	20
5.9.1 Function Documentation	20
5.9.1.1 Box() [1/2]	20
5.9.1.2 Box() [2/2]	21
5.9.1.3 operator!=(()) [1/2]	21
5.9.1.4 operator!=(()) [2/2]	21
5.9.1.5 operator==(()) [1/2]	21
5.9.1.6 operator==(()) [2/2]	21
5.10 wr22::regex_parser::utils::detail Namespace Reference	22
5.11 wr22::regex_parser::utils::detail::adt Namespace Reference	22
6 Class Documentation	23
6.1 wr22::regex_parser::utils::Adt< Variants > Class Template Reference	23
6.1.1 Detailed Description	24
6.1.2 Member Typedef Documentation	24
6.1.2.1 VariantType	24
6.1.3 Constructor & Destructor Documentation	24
6.1.3.1 Adt()	24
6.1.4 Member Function Documentation	25
6.1.4.1 as_variant() [1/2]	25
6.1.4.2 as_variant() [2/2]	25
6.1.4.3 visit() [1/2]	25
6.1.4.4 visit() [2/2]	25
6.1.5 Member Data Documentation	26
6.1.5.1 m_variant	26

6.2 wr22::regex_parser::regex::part::Alternatives Struct Reference	26
6.2.1 Detailed Description	26
6.2.2 Constructor & Destructor Documentation	26
6.2.2.1 Alternatives()	27
6.2.3 Member Function Documentation	27
6.2.3.1 operator==()	27
6.2.4 Member Data Documentation	27
6.2.4.1 alternatives	27
6.2.4.2 code_name	27
6.3 wr22::regex_parser::utils::Box< T > Class Template Reference	27
6.3.1 Detailed Description	28
6.3.2 Constructor & Destructor Documentation	28
6.3.2.1 Box() [1 / 3]	28
6.3.2.2 Box() [2 / 3]	29
6.3.2.3 Box() [3 / 3]	29
6.3.3 Member Function Documentation	29
6.3.3.1 construct_in_place()	29
6.3.3.2 operator*() [1 / 2]	30
6.3.3.3 operator*() [2 / 2]	30
6.4 wr22::regex_parser::utils::BoxIsEmpty Struct Reference	30
6.4.1 Member Function Documentation	30
6.4.1.1 what()	31
6.5 wr22::regex_parser::regex::Capture Class Reference	31
6.5.1 Detailed Description	31
6.6 wr22::regex_parser::regex::part::CharacterClass Struct Reference	31
6.6.1 Detailed Description	32
6.6.2 Constructor & Destructor Documentation	32
6.6.2.1 CharacterClass()	32
6.6.3 Member Function Documentation	32
6.6.3.1 operator==()	32
6.6.4 Member Data Documentation	32
6.6.4.1 code_name	33
6.6.4.2 data	33
6.7 wr22::regex_parser::regex::CharacterClassData Struct Reference	33
6.7.1 Detailed Description	33
6.7.2 Member Function Documentation	33
6.7.2.1 operator==()	34
6.7.3 Member Data Documentation	34
6.7.3.1 inverted	34
6.7.3.2 ranges	34
6.8 wr22::regex_parser::regex::CharacterRange Class Reference	34
6.8.1 Detailed Description	35

6.8.2 Member Function Documentation	35
6.8.2.1 contains()	35
6.8.2.2 first()	35
6.8.2.3 from_endpoints()	35
6.8.2.4 from_single_character()	36
6.8.2.5 is_single_character()	36
6.8.2.6 last()	36
6.8.2.7 operator==()	36
6.9 wr22::regex_parser::regex::part::Empty Struct Reference	36
6.9.1 Detailed Description	37
6.9.2 Constructor & Destructor Documentation	37
6.9.2.1 Empty()	37
6.9.3 Member Function Documentation	37
6.9.3.1 operator==()	37
6.9.4 Member Data Documentation	37
6.9.4.1 code_name	37
6.10 wr22::regex_parser::parser::errors::ExpectedEnd Class Reference	38
6.10.1 Detailed Description	38
6.10.2 Constructor & Destructor Documentation	38
6.10.2.1 ExpectedEnd()	38
6.10.3 Member Function Documentation	39
6.10.3.1 char_got()	39
6.10.3.2 position()	39
6.11 wr22::regex_parser::regex::part::Group Struct Reference	39
6.11.1 Detailed Description	40
6.11.2 Constructor & Destructor Documentation	40
6.11.2.1 Group()	40
6.11.3 Member Function Documentation	40
6.11.3.1 operator==()	40
6.11.4 Member Data Documentation	40
6.11.4.1 capture	40
6.11.4.2 code_name	41
6.11.4.3 inner	41
6.12 wr22::regex_parser::regex::capture::Index Struct Reference	41
6.12.1 Detailed Description	41
6.12.2 Constructor & Destructor Documentation	41
6.12.2.1 Index()	41
6.12.3 Member Function Documentation	42
6.12.3.1 operator==()	42
6.12.4 Member Data Documentation	42
6.12.4.1 code_name	42
6.13 wr22::regex_parser::regex::InvalidCharacterRange Struct Reference	42

6.13.1 Constructor & Destructor Documentation	43
6.13.1.1 InvalidCharacterRange()	43
6.13.2 Member Data Documentation	43
6.13.2.1 first	43
6.13.2.2 last	43
6.14 wr22::regex_parser::parser::errors::InvalidRange Class Reference	43
6.14.1 Detailed Description	44
6.14.2 Constructor & Destructor Documentation	44
6.14.2.1 InvalidRange()	44
6.14.3 Member Function Documentation	44
6.14.3.1 first()	44
6.14.3.2 last()	45
6.14.3.3 span()	45
6.15 wr22::regex_parser::span::InvalidSpan Struct Reference	45
6.15.1 Detailed Description	45
6.15.2 Constructor & Destructor Documentation	46
6.15.2.1 InvalidSpan()	46
6.15.3 Member Data Documentation	46
6.15.3.1 begin	46
6.15.3.2 end	46
6.16 wr22::regex_parser::regex::part::Literal Struct Reference	46
6.16.1 Detailed Description	47
6.16.2 Constructor & Destructor Documentation	47
6.16.2.1 Literal()	47
6.16.3 Member Function Documentation	47
6.16.3.1 operator==()	47
6.16.4 Member Data Documentation	47
6.16.4.1 character	47
6.16.4.2 code_name	48
6.17 wr22::regex_parser::utils::detail::adt::MultiCallable< Fs > Struct Template Reference	48
6.17.1 Constructor & Destructor Documentation	48
6.17.1.1 MultiCallable()	48
6.18 wr22::regex_parser::regex::capture::Name Struct Reference	48
6.18.1 Detailed Description	49
6.18.2 Constructor & Destructor Documentation	49
6.18.2.1 Name()	49
6.18.3 Member Function Documentation	49
6.18.3.1 operator==()	49
6.18.4 Member Data Documentation	49
6.18.4.1 code_name	50
6.18.4.2 flavor	50
6.18.4.3 name	50

6.19 wr22::regex_parser::regex::capture::None Struct Reference	50
6.19.1 Detailed Description	50
6.19.2 Constructor & Destructor Documentation	50
6.19.2.1 None()	51
6.19.3 Member Function Documentation	51
6.19.3.1 operator==()	51
6.19.4 Member Data Documentation	51
6.19.4.1 code_name	51
6.20 wr22::regex_parser::regex::part::Optional Struct Reference	51
6.20.1 Detailed Description	52
6.20.2 Constructor & Destructor Documentation	52
6.20.2.1 Optional()	52
6.20.3 Member Function Documentation	52
6.20.3.1 operator==()	52
6.20.4 Member Data Documentation	52
6.20.4.1 code_name	52
6.20.4.2 inner	53
6.21 wr22::regex_parser::parser::errors::ParseError Struct Reference	53
6.21.1 Detailed Description	53
6.22 wr22::regex_parser::parser::Parser< Iter, Sentinel > Class Template Reference	53
6.22.1 Detailed Description	54
6.22.2 Constructor & Destructor Documentation	54
6.22.2.1 Parser()	54
6.22.3 Member Function Documentation	55
6.22.3.1 expect_end()	55
6.22.3.2 parse_alternatives()	55
6.22.3.3 parse_atom()	55
6.22.3.4 parse_char_class()	56
6.22.3.5 parse_char_literal()	56
6.22.3.6 parse_group()	56
6.22.3.7 parse_group_name()	57
6.22.3.8 parse_regex()	57
6.22.3.9 parse_sequence()	57
6.22.3.10 parse_sequence_or_empty()	58
6.22.3.11 parse_wildcard()	58
6.23 wr22::regex_parser::regex::Part Class Reference	59
6.23.1 Detailed Description	59
6.24 wr22::regex_parser::regex::part::Plus Struct Reference	60
6.24.1 Detailed Description	60
6.24.2 Constructor & Destructor Documentation	60
6.24.2.1 Plus()	60
6.24.3 Member Function Documentation	60

6.24.3.1 operator==()	60
6.24.4 Member Data Documentation	61
6.24.4.1 code_name	61
6.24.4.2 inner	61
6.25 wr22::regex_parser::regex::part::Sequence Struct Reference	61
6.25.1 Detailed Description	62
6.25.2 Constructor & Destructor Documentation	62
6.25.2.1 Sequence()	62
6.25.3 Member Function Documentation	62
6.25.3.1 operator==()	62
6.25.4 Member Data Documentation	62
6.25.4.1 code_name	62
6.25.4.2 items	62
6.26 wr22::regex_parser::span::Span Class Reference	63
6.26.1 Detailed Description	63
6.26.2 Member Function Documentation	63
6.26.2.1 begin()	64
6.26.2.2 end()	64
6.26.2.3 extend_right()	64
6.26.2.4 length()	64
6.26.2.5 make_empty()	64
6.26.2.6 make_from_positions()	65
6.26.2.7 make_single_position()	65
6.26.2.8 make_with_length()	65
6.26.2.9 operator!=(())	66
6.26.2.10 operator==(())	66
6.27 wr22::regex_parser::regex::SpannedCharacterRange Struct Reference	66
6.27.1 Detailed Description	66
6.27.2 Member Function Documentation	66
6.27.2.1 operator==(())	66
6.27.3 Member Data Documentation	67
6.27.3.1 range	67
6.27.3.2 span	67
6.28 wr22::regex_parser::regex::SpannedPart Class Reference	67
6.28.1 Detailed Description	67
6.28.2 Constructor & Destructor Documentation	67
6.28.2.1 SpannedPart()	68
6.28.3 Member Function Documentation	68
6.28.3.1 operator!=(())	68
6.28.3.2 operator==(())	68
6.28.3.3 part() [1/2]	68
6.28.3.4 part() [2/2]	68

6.28.3.5 <code>span()</code>	68
6.29 <code>wr22::regex_parser::regex::part::Star</code> Struct Reference	69
6.29.1 Detailed Description	69
6.29.2 Constructor & Destructor Documentation	69
6.29.2.1 <code>Star()</code>	69
6.29.3 Member Function Documentation	69
6.29.3.1 <code>operator==()</code>	69
6.29.4 Member Data Documentation	70
6.29.4.1 <code>code_name</code>	70
6.29.4.2 <code>inner</code>	70
6.30 <code>wr22::regex_parser::parser::errors::UnexpectedChar</code> Class Reference	70
6.30.1 Detailed Description	71
6.30.2 Constructor & Destructor Documentation	71
6.30.2.1 <code>UnexpectedChar()</code>	71
6.30.3 Member Function Documentation	71
6.30.3.1 <code>char_got()</code>	71
6.30.3.2 <code>expected()</code>	71
6.30.3.3 <code>position()</code>	72
6.31 <code>wr22::regex_parser::parser::errors::UnexpectedEnd</code> Class Reference	72
6.31.1 Detailed Description	72
6.31.2 Constructor & Destructor Documentation	72
6.31.2.1 <code>UnexpectedEnd()</code>	72
6.31.3 Member Function Documentation	73
6.31.3.1 <code>expected()</code>	73
6.31.3.2 <code>position()</code>	73
6.32 <code>wr22::regex_parser::regex::part::Wildcard</code> Struct Reference	73
6.32.1 Detailed Description	74
6.32.2 Constructor & Destructor Documentation	74
6.32.2.1 <code>Wildcard()</code>	74
6.32.3 Member Function Documentation	74
6.32.3.1 <code>operator==()</code>	74
6.32.4 Member Data Documentation	74
6.32.4.1 <code>code_name</code>	74
7 File Documentation	75
7.1 <code>include/wr22/regex_parser/parser/errors.hpp</code> File Reference	75
7.2 <code>errors.hpp</code>	76
7.3 <code>include/wr22/regex_parser/parser/regex.hpp</code> File Reference	76
7.4 <code>regex.hpp</code>	77
7.5 <code>include/wr22/regex_parser/regex/capture.hpp</code> File Reference	77
7.6 <code>capture.hpp</code>	78
7.7 <code>include/wr22/regex_parser/regex/character_class_data.hpp</code> File Reference	79

7.8 character_class_data.hpp	79
7.9 include/wr22/regex_parser/regex/character_range.hpp File Reference	79
7.10 character_range.hpp	80
7.11 include/wr22/regex_parser/regex/named_capture_flavor.hpp File Reference	80
7.12 named_capture_flavor.hpp	81
7.13 include/wr22/regex_parser/regex/part.hpp File Reference	81
7.14 part.hpp	83
7.15 include/wr22/regex_parser/regex/spanned_character_range.hpp File Reference	85
7.16 spanned_character_range.hpp	85
7.17 include/wr22/regex_parser/span/span.hpp File Reference	86
7.18 span.hpp	86
7.19 include/wr22/regex_parser/utils/adt.hpp File Reference	87
7.20 adt.hpp	88
7.21 include/wr22/regex_parser/utils/box.hpp File Reference	88
7.22 box.hpp	89
7.23 src/parser/capture.cpp File Reference	90
7.24 src/parser/errors.cpp File Reference	90
7.25 src/parser/regex.cpp File Reference	91
7.26 src/regex/character_range.cpp File Reference	91
7.27 src/regex/named_capture_flavor.cpp File Reference	92
7.28 src/regex/part.cpp File Reference	92
7.29 src/regex/spanned_character_range.cpp File Reference	93
7.30 src/span/span.cpp File Reference	93
7.31 src/utils/box.cpp File Reference	94
Index	95

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

wr22	9
wr22::regex_parser	9
wr22::regex_parser::parser	9
wr22::regex_parser::parser::errors	10
wr22::regex_parser::regex	11
wr22::regex_parser::regex::capture	15
wr22::regex_parser::regex::part The namespace with the variants of Part	16
wr22::regex_parser::span	19
wr22::regex_parser::utils	20
wr22::regex_parser::utils::detail	22
wr22::regex_parser::utils::detail::adt	22

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

wr22::regex_parser::utils::Adt< Variants >	23
wr22::regex_parser::regex::Capture	31
wr22::regex_parser::regex::Part	59
wr22::regex_parser::regex::part::Alternatives	26
wr22::regex_parser::utils::Box< T >	27
wr22::regex_parser::utils::Box< wr22::regex_parser::regex::SpannedPart >	27
wr22::regex_parser::regex::part::CharacterClass	31
wr22::regex_parser::regex::CharacterClassData	33
wr22::regex_parser::regex::CharacterRange	34
wr22::regex_parser::regex::part::Empty	36
std::exception	
wr22::regex_parser::utils::BoxIsEmpty	30
wr22::regex_parser::regex::part::Group	39
wr22::regex_parser::regex::capture::Index	41
wr22::regex_parser::regex::part::Literal	46
wr22::regex_parser::regex::capture::Name	48
wr22::regex_parser::regex::capture::None	50
wr22::regex_parser::regex::part::Optional	51
wr22::regex_parser::parser::Parser< Iter, Sentinel >	53
wr22::regex_parser::regex::part::Plus	60
std::runtime_error	
wr22::regex_parser::parser::errors::ParseError	53
wr22::regex_parser::parser::errors::ExpectedEnd	38
wr22::regex_parser::parser::errors::InvalidRange	43
wr22::regex_parser::parser::errors::UnexpectedChar	70
wr22::regex_parser::parser::errors::UnexpectedEnd	72
wr22::regex_parser::regex::InvalidCharacterRange	42
wr22::regex_parser::span::InvalidSpan	45
wr22::regex_parser::regex::part::Sequence	61
wr22::regex_parser::span::Span	63
wr22::regex_parser::regex::SpannedCharacterRange	66
wr22::regex_parser::regex::SpannedPart	67
wr22::regex_parser::regex::part::Star	69
wr22::regex_parser::regex::part::Wildcard	73
wr22::regex_parser::utils::detail::adt::Fs	
wr22::regex_parser::utils::detail::adt::MultiCallable< Fs >	48

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

wr22::regex_parser::utils::Adt< Variants >	23
A helper class that simplifies creation of algebraic data types	
wr22::regex_parser::regex::part::Alternatives	26
A regex part with the list of alternatives to be matched	
wr22::regex_parser::utils::Box< T >	27
A copyable and equality-comparable wrapper around <code>std::unique_ptr</code>	
wr22::regex_parser::utils::BoxIsEmpty	30
wr22::regex_parser::regex::Capture	31
Group capture behavior	
wr22::regex_parser::regex::part::CharacterClass	31
A regex part specifying a character class (e.g. <code>[a-z_]</code>)	
wr22::regex_parser::regex::CharacterClassData	33
A character class representation: a list of character ranges plus some additional properties	
wr22::regex_parser::regex::CharacterRange	34
A non-empty character range, possibly containing only one character	
wr22::regex_parser::regex::part::Empty	36
An empty regex part	
wr22::regex_parser::parser::errors::ExpectedEnd	38
The error when the parser expected the input to end, but it did not	
wr22::regex_parser::regex::part::Group	39
A regex part that represents a group in parentheses	
wr22::regex_parser::regex::capture::Index	41
Denotes a group captured by index	
wr22::regex_parser::regex::InvalidCharacterRange	42
wr22::regex_parser::parser::errors::InvalidRange	43
The error indicating that a character range in a character class is invalid	
wr22::regex_parser::span::InvalidSpan	45
The exception thrown on an attempt to construct an invalid span	
wr22::regex_parser::regex::part::Literal	46
An regex part that matches a single character literally	
wr22::regex_parser::utils::detail::adt::MultiCallable< Fs >	48
wr22::regex_parser::regex::capture::Name	48
Denotes a group captured by name	
wr22::regex_parser::regex::capture::None	50
Denotes an non-capturing group	

wr22::regex_parser::regex::part::Optional	
A regex part specifying an optional quantifier ((expression)?)	51
wr22::regex_parser::parser::errors::ParseError	
The base class for parse errors	53
wr22::regex_parser::parser::Parser< Iter, Sentinel >	
A regex parser	53
wr22::regex_parser::regex::Part	
A part of a regular expression and its AST node type	59
wr22::regex_parser::regex::part::Plus	
A regex part specifying an "at least one" quantifier ((expression)+)	60
wr22::regex_parser::regex::part::Sequence	
A regex part with the list of items to be matched one after another	61
wr22::regex_parser::span::Span	
Character position range in the input string	63
wr22::regex_parser::regex::SpannedCharacterRange	
A CharacterRange with its span	66
wr22::regex_parser::regex::SpannedPart	
A version of Part including the span information (position in the input) of the root AST node (child nodes always contain it because they are represented as SpannedParts themselves)	67
wr22::regex_parser::regex::part::Star	
A regex part specifying an "at least zero" quantifier ((expression)*)	69
wr22::regex_parser::parser::errors::UnexpectedChar	
The error when the parser got a character it didn't expect at the current position	70
wr22::regex_parser::parser::errors::UnexpectedEnd	
The error when the parser hit the end of the input earlier than it expected	72
wr22::regex_parser::regex::part::Wildcard	
A regex part specifying any single character (.)	73

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

include/wr22/regex_parser/parser/errors.hpp	75
include/wr22/regex_parser/parser/regex.hpp	76
include/wr22/regex_parser/regex/capture.hpp	77
include/wr22/regex_parser/regex/character_class_data.hpp	79
include/wr22/regex_parser/regex/character_range.hpp	79
include/wr22/regex_parser/regex/named_capture_flavor.hpp	80
include/wr22/regex_parser/regex/part.hpp	81
include/wr22/regex_parser/regex/spanned_character_range.hpp	85
include/wr22/regex_parser/span/span.hpp	86
include/wr22/regex_parser/utils/adt.hpp	87
include/wr22/regex_parser/utils/box.hpp	88
src/parser/capture.cpp	90
src/parser/errors.cpp	90
src/parser/regex.cpp	91
src/regex/character_range.cpp	91
src/regex/named_capture_flavor.cpp	92
src/regex/part.cpp	92
src/regex/spanned_character_range.cpp	93
src/span/span.cpp	93
src/utils/box.cpp	94

Chapter 5

Namespace Documentation

5.1 wr22 Namespace Reference

Namespaces

- namespace [regex_parser](#)

5.2 wr22::regex_parser Namespace Reference

Namespaces

- namespace [parser](#)
- namespace [regex](#)
- namespace [span](#)
- namespace [utils](#)

5.3 wr22::regex_parser::parser Namespace Reference

Namespaces

- namespace [errors](#)

Classes

- class [Parser](#)
A regex parser.

Functions

- `template<typename Iter , typename Sentinel >
Parser (Iter begin, Sentinel end) -> Parser< Iter, Sentinel >`
The type deduction guideline for [Parser](#).
- `regex::SpannedPart parse_regex (const std::u32string_view ®ex)`
Parse a regular expression into its AST.

5.3.1 Function Documentation

5.3.1.1 `parse_regex()`

```
regex::SpannedPart wr22::regex_parser::parser::parse_regex (
    const std::u32string_view & regex )
```

Parse a regular expression into its AST.

The regular expression is a string view in the UTF-32 encoding. It is parsed and its object representation (see the docs for `regex::SpannedPart`) is built. The returned representation is an owned object and its lifetime does not depend on the lifetime of the `regex` argument.

If the parsing fails, an exception is thrown. `errors::ParseError` is the base class for all exceptions thrown from this function, but more specific exceptions may be caught and handled separately. See the docs for the `errors.hpp` file for details.

Returns

the parsed regex AST if the parsing succeeds.

Exceptions

<code>errors::ParseError</code>	if the parsing fails.
---------------------------------	-----------------------

5.3.1.2 `Parser()`

```
template<typename Iter , typename Sentinel >
wr22::regex_parser::parser::Parser (
    Iter begin,
    Sentinel end ) -> Parser< Iter, Sentinel >
```

The type deduction guideline for `Parser`.

5.4 `wr22::regex_parser::parser::errors` Namespace Reference

Classes

- class `ExpectedEnd`
The error when the parser expected the input to end, but it did not.
- class `InvalidRange`
The error indicating that a character range in a character class is invalid.
- struct `ParseError`
The base class for parse errors.
- class `UnexpectedChar`
The error when the parser got a character it didn't expect at the current position.
- class `UnexpectedEnd`
The error when the parser hit the end of the input earlier than it expected.

5.5 wr22::regex_parser::regex Namespace Reference

Namespaces

- namespace [capture](#)
- namespace [part](#)

The namespace with the variants of [Part](#).

Classes

- class [Capture](#)
Group capture behavior.
- struct [CharacterClassData](#)
A character class representation: a list of character ranges plus some additional properties.
- class [CharacterRange](#)
A non-empty character range, possibly containing only one character.
- struct [InvalidCharacterRange](#)
- class [Part](#)
A part of a regular expression and its AST node type.
- struct [SpannedCharacterRange](#)
A [CharacterRange](#) with its span.
- class [SpannedPart](#)
A version of [Part](#) including the span information (position in the input) of the root AST node (child nodes always contain it because they are represented as [SpannedParts](#) themselves).

Enumerations

- enum class [NamedCaptureFlavor](#) { [Apostrophes](#) , [Angles](#) , [AnglesWithP](#) }
- The flavor (dialect) of a named group capture.*

Functions

- `std::ostream & operator<< (std::ostream &out, const Capture &capture)`
- `void to_json (nlohmann::json &j, const Capture &capture)`
- `std::ostream & operator<< (std::ostream &out, const CharacterRange &range)`
- `void to_json (nlohmann::json &j, const CharacterRange &range)`
- `std::ostream & operator<< (std::ostream &out, NamedCaptureFlavor flavor)`
- `void to_json (nlohmann::json &j, NamedCaptureFlavor flavor)`
- `std::ostream & operator<< (std::ostream &out, const SpannedPart &part)`
Convert a [SpannedPart](#) to a textual representation and write it to an `std::ostream`.
- `void to_json (nlohmann::json &j, const Part &part)`
- `void to_json (nlohmann::json &j, const SpannedPart &part)`
- `std::ostream & operator<< (std::ostream &out, const SpannedCharacterRange &range)`
- `void to_json (nlohmann::json &j, const SpannedCharacterRange &range)`

5.5.1 Enumeration Type Documentation

5.5.1.1 NamedCaptureFlavor

```
enum class wr22::regex_parser::regex::NamedCaptureFlavor [strong]
```

The flavor (dialect) of a named group capture.

The most common variants are included. This list is subject to extension if deemed necessary. The source used as a reference is <https://www.regular-expressions.info/named.html>.

Enumerator

Apostrophes	The flavor (? 'name' contents) . Mostly used in C# and other .NET-oriented languages, although can also be found in certain versions Perl, Boost and elsewhere.
Angles	The flavor (<name>contents) . Mostly used in C# and other .NET-oriented languages, although can also be found in certain versions Perl, Boost and elsewhere.
AnglesWithP	The flavor (?P<name>contents) . Found in Python, PCRE and elsewhere.

5.5.2 Function Documentation

5.5.2.1 operator<<() [1/5]

```
std::ostream & wr22::regex_parser::regex::operator<< (
    std::ostream & out,
    const Capture & capture )
```

5.5.2.2 operator<<() [2/5]

```
std::ostream & wr22::regex_parser::regex::operator<< (
    std::ostream & out,
    const CharacterRange & range )
```

5.5.2.3 operator<<() [3/5]

```
std::ostream & wr22::regex_parser::regex::operator<< (
    std::ostream & out,
    const SpannedCharacterRange & range )
```

5.5.2.4 operator<<() [4/5]

```
std::ostream & wr22::regex_parser::regex::operator<< (
    std::ostream & out,
    const SpannedPart & spanned_part )
```

Convert a [SpannedPart](#) to a textual representation and write it to an `std::ostream`.

5.5.2.5 operator<<() [5/5]

```
std::ostream & wr22::regex_parser::regex::operator<< (
    std::ostream & out,
    NamedCaptureFlavor flavor )
```

5.5.2.6 to_json() [1/6]

```
void wr22::regex_parser::regex::to_json (
    nlohmann::json & j,
    const Capture & capture )
```

5.5.2.7 to_json() [2/6]

```
void wr22::regex_parser::regex::to_json (
    nlohmann::json & j,
    const CharacterRange & range )
```

5.5.2.8 to_json() [3/6]

```
void wr22::regex_parser::regex::to_json (
    nlohmann::json & j,
    const Part & part )
```

5.5.2.9 to_json() [4/6]

```
void wr22::regex_parser::regex::to_json (
    nlohmann::json & j,
    const SpannedCharacterRange & range )
```

5.5.2.10 to_json() [5/6]

```
void wr22::regex_parser::regex::to_json (
    nlohmann::json & j,
    const SpannedPart & part )
```

5.5.2.11 to_json() [6/6]

```
void wr22::regex_parser::regex::to_json (
    nlohmann::json & j,
    NamedCaptureFlavor flavor )
```

5.6 wr22::regex_parser::regex::capture Namespace Reference

Classes

- struct [Index](#)
Denotes a group captured by index.
- struct [Name](#)
Denotes a group captured by name.
- struct [None](#)
Denotes an non-capturing group.

Typedefs

- using [Adt](#) = [utils::Adt](#)< [None](#), [Index](#), [Name](#) >

Functions

- void [to_json](#) (nlohmann::json &j, const [None](#) &capture)
- void [to_json](#) (nlohmann::json &j, const [Index](#) &capture)
- void [to_json](#) (nlohmann::json &j, const [Name](#) &capture)

5.6.1 Typedef Documentation

5.6.1.1 Adt

```
using wr22::regex_parser::regex::capture::Adt = typedef utils::Adt<None, Index, Name>
```

5.6.2 Function Documentation

5.6.2.1 to_json() [1/3]

```
void wr22::regex_parser::regex::capture::to_json (
    nlohmann::json & j,
    const Index & capture )
```

5.6.2.2 to_json() [2/3]

```
void wr22::regex_parser::regex::capture::to_json (
    nlohmann::json & j,
    const Name & capture )
```

5.6.2.3 to_json() [3/3]

```
void wr22::regex_parser::regex::capture::to_json (
    nlohmann::json & j,
    const None & capture )
```

5.7 wr22::regex_parser::regex::part Namespace Reference

The namespace with the variants of [Part](#).

Classes

- struct [Alternatives](#)
A regex part with the list of alternatives to be matched.
- struct [CharacterClass](#)
A regex part specifying a character class (e.g. `[a-z_]`).
- struct [Empty](#)
An empty regex part.
- struct [Group](#)
A regex part that represents a group in parentheses.
- struct [Literal](#)
A regex part that matches a single character literally.
- struct [Optional](#)
A regex part specifying an optional quantifier `((expression)?)`.
- struct [Plus](#)
A regex part specifying an "at least one" quantifier `((expression)+)`.
- struct [Sequence](#)
A regex part with the list of items to be matched one after another.
- struct [Star](#)
A regex part specifying an "at least zero" quantifier `((expression)*)`.
- struct [Wildcard](#)
A regex part specifying any single character `(.)`.

Typedefs

- using [Adt](#) = [utils::Adt](#)< [Empty](#), [Literal](#), [Alternatives](#), [Sequence](#), [Group](#), [Optional](#), [Plus](#), [Star](#), [Wildcard](#), [CharacterClass](#) >

Functions

- void [to_json](#) (nlohmann::json &j, const [part::Empty](#) &part)
- void [to_json](#) (nlohmann::json &j, const [part::Literal](#) &part)
- void [to_json](#) (nlohmann::json &j, const [part::Alternatives](#) &part)
- void [to_json](#) (nlohmann::json &j, const [part::Sequence](#) &part)
- void [to_json](#) (nlohmann::json &j, const [part::Group](#) &part)
- void [to_json](#) (nlohmann::json &j, const [part::Optional](#) &part)
- void [to_json](#) (nlohmann::json &j, const [part::Plus](#) &part)
- void [to_json](#) (nlohmann::json &j, const [part::Star](#) &part)
- void [to_json](#) (nlohmann::json &j, const [part::Wildcard](#) &part)
- void [to_json](#) (nlohmann::json &j, const [part::CharacterClass](#) &part)

5.7.1 Detailed Description

The namespace with the variants of [Part](#).

See the docs for the [Part](#) type for additional information.

5.7.2 Typedef Documentation

5.7.2.1 Adt

```
using wr22::regex_parser::regex::part::Adt = typedef utils:: Adt<Empty, Literal, Alternatives,  
Sequence, Group, Optional, Plus, Star, Wildcard, CharacterClass>
```

5.7.3 Function Documentation

5.7.3.1 to_json() [1/10]

```
void wr22::regex_parser::regex::part::to_json (  
    nlohmann::json & j,  
    const part::Alternatives & part )
```

5.7.3.2 to_json() [2/10]

```
void wr22::regex_parser::regex::part::to_json (  
    nlohmann::json & j,  
    const part::CharacterClass & part )
```

5.7.3.3 to_json() [3/10]

```
void wr22::regex_parser::regex::part::to_json (
    nlohmann::json & j,
    const part::Empty & part )
```

5.7.3.4 to_json() [4/10]

```
void wr22::regex_parser::regex::part::to_json (
    nlohmann::json & j,
    const part::Group & part )
```

5.7.3.5 to_json() [5/10]

```
void wr22::regex_parser::regex::part::to_json (
    nlohmann::json & j,
    const part::Literal & part )
```

5.7.3.6 to_json() [6/10]

```
void wr22::regex_parser::regex::part::to_json (
    nlohmann::json & j,
    const part::Optional & part )
```

5.7.3.7 to_json() [7/10]

```
void wr22::regex_parser::regex::part::to_json (
    nlohmann::json & j,
    const part::Plus & part )
```

5.7.3.8 to_json() [8/10]

```
void wr22::regex_parser::regex::part::to_json (
    nlohmann::json & j,
    const part::Sequence & part )
```

5.7.3.9 to_json() [9/10]

```
void wr22::regex_parser::regex::part::to_json (
    nlohmann::json & j,
    const part::Star & part )
```

5.7.3.10 to_json() [10/10]

```
void wr22::regex_parser::regex::part::to_json (
    nlohmann::json & j,
    const part::Wildcard & part )
```

5.8 wr22::regex_parser::span Namespace Reference

Classes

- struct [InvalidSpan](#)
The exception thrown on an attempt to construct an invalid span.
- class [Span](#)
Character position range in the input string.

Functions

- std::ostream & [operator<<](#) (std::ostream &out, [Span](#) span)
- void [to_json](#) (nlohmann::json &j, [Span](#) span)

5.8.1 Function Documentation

5.8.1.1 operator<<()

```
std::ostream & wr22::regex_parser::span::operator<< (
    std::ostream & out,
    Span span )
```

5.8.1.2 to_json()

```
void wr22::regex_parser::span::to_json (
    nlohmann::json & j,
    Span span )
```

5.9 wr22::regex_parser::utils Namespace Reference

Namespaces

- namespace [detail](#)

Classes

- class [Adt](#)
A helper class that simplifies creation of algebraic data types.
- class [Box](#)
A copyable and equality-comparable wrapper around `std::unique_ptr`.
- struct [BoxIsEmpty](#)

Functions

- `template<typename... Variants>`
`bool operator== (const Adt< Variants... > &lhs, const Adt< Variants... > &rhs)`
Compare two compatible ADTs for equality.
- `template<typename... Variants>`
`bool operator!= (const Adt< Variants... > &lhs, const Adt< Variants... > &rhs)`
Compare two compatible ADTs for non-equality.
- `template<typename T >`
`Box (T &&value) -> Box< T >`
Type deduction guideline for [Box](#) (value initialization).
- `template<typename T >`
`Box (std::unique_ptr< T > ptr) -> Box< T >`
Type deduction guideline for [Box](#) (std::unique_ptr adoption).
- `template<typename T, typename U >`
`bool operator== (const Box< T > &lhs, const Box< U > &rhs)`
- `template<typename T, typename U >`
`bool operator!= (const Box< T > &lhs, const Box< U > &rhs)`

5.9.1 Function Documentation

5.9.1.1 [Box\(\)](#) [1/2]

```
template<typename T >
wr22::regex_parser::utils::Box (
    std::unique_ptr< T > ptr ) -> Box< T >
```

Type deduction guideline for [Box](#) (std::unique_ptr adoption).

5.9.1.2 Box() [2/2]

```
template<typename T >
wr22::regex_parser::utils::Box (
    T && value ) -> Box< T >
```

Type deduction guideline for `Box` (value initialization).

5.9.1.3 operator!=() [1/2]

```
template<typename... Variants>
bool wr22::regex_parser::utils::operator!= (
    const Adt< Variants... > & lhs,
    const Adt< Variants... > & rhs )
```

Compare two compatible ADTs for non-equality.

5.9.1.4 operator!=() [2/2]

```
template<typename T , typename U >
bool wr22::regex_parser::utils::operator!= (
    const Box< T > & lhs,
    const Box< U > & rhs )
```

5.9.1.5 operator==() [1/2]

```
template<typename... Variants>
bool wr22::regex_parser::utils::operator== (
    const Adt< Variants... > & lhs,
    const Adt< Variants... > & rhs )
```

Compare two compatible ADTs for equality.

5.9.1.6 operator==() [2/2]

```
template<typename T , typename U >
bool wr22::regex_parser::utils::operator== (
    const Box< T > & lhs,
    const Box< U > & rhs )
```

5.10 `wr22::regex_parser::utils::detail` Namespace Reference

Namespaces

- namespace [adt](#)

5.11 `wr22::regex_parser::utils::detail::adt` Namespace Reference

Classes

- struct [MultiCallable](#)

Chapter 6

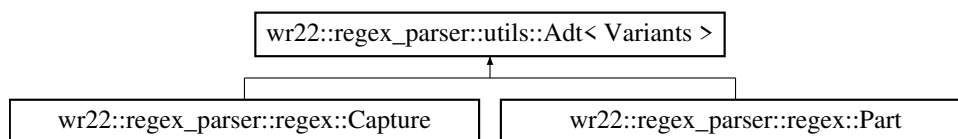
Class Documentation

6.1 wr22::regex_parser::utils::Adt< Variants > Class Template Reference

A helper class that simplifies creation of algebraic data types.

```
#include <adt.hpp>
```

Inheritance diagram for wr22::regex_parser::utils::Adt< Variants >:



Public Types

- using `VariantType` = `std::variant< Variants... >`
A convenience type alias for the concrete `std::variant` type used.

Public Member Functions

- template<typename V >
`Adt` (V variant)
Constructor for each of the variants.
- template<typename... Fs>
`decltype(auto)` `visit` (Fs &&... visitors) `const`
Visit the ADT, applying the suitable function from the list of visitors on the variant held.
- template<typename... Fs>
`decltype(auto)` `visit` (Fs &&... visitors)
Visit the ADT, applying the suitable function from the list of visitors on the variant held.
- `const` `VariantType` & `as_variant` () `const`
Access the underlying `std::variant` type (constant version).
- `VariantType` & `as_variant` ()
Access the underlying `std::variant` type (non-constant version).

Protected Attributes

- [VariantType m_variant](#)

6.1.1 Detailed Description

```
template<typename... Variants>
class wr22::regex_parser::utils::Adt< Variants >
```

A helper class that simplifies creation of algebraic data types.

Algebraic data types are data types that can have one type of a predefined set of variants, but be stored and represented as values of one common type. In C++, `std::variant` serves exactly this purpose. It is, however, not very convenient to work with or build upon, so this class is designed to simplify building new algebraic data types. It still uses `std::variant` under the hood.

The template type parameters are the types that the variants may hold (must be distinct types).

6.1.2 Member Typedef Documentation

6.1.2.1 VariantType

```
template<typename... Variants>
using wr22::regex_parser::utils::Adt< Variants >::VariantType = std::variant<Variants...>
```

A convenience type alias for the concrete `std::variant` type used.

6.1.3 Constructor & Destructor Documentation

6.1.3.1 Adt()

```
template<typename... Variants>
template<typename V >
wr22::regex_parser::utils::Adt< Variants >::Adt (
    V variant ) [inline]
```

Constructor for each of the variants.

Construct an instance holding a specified variant. The type `V` of the variant provided must be one of the types from `Variants`. Note that this constructor is purposefully implicit, so that the variants as separate types are transparently converted to this common type when necessary.

The variant is taken by value and moved thereafter, so that, when constructing the common type, the variant may be either copied or moved, depending on the user's intentions.

6.1.4 Member Function Documentation

6.1.4.1 as_variant() [1/2]

```
template<typename... Variants>
VariantType & wr22::regex_parser::utils::Adt< Variants >::as_variant ( ) [inline]
```

Access the underlying `std::variant` type (non-constant version).

6.1.4.2 as_variant() [2/2]

```
template<typename... Variants>
const VariantType & wr22::regex_parser::utils::Adt< Variants >::as_variant ( ) const [inline]
```

Access the underlying `std::variant` type (constant version).

6.1.4.3 visit() [1/2]

```
template<typename... Variants>
template<typename... Fs>
decltype(auto) wr22::regex_parser::utils::Adt< Variants >::visit (
    Fs &&... visitors ) [inline]
```

Visit the ADT, applying the suitable function from the list of visitors on the variant held.

This is the non-constant version of the method. See the docs for the constant version for a detailed description and code examples. The only thing different in this version of the method is that the visitors get called with a non-const lvalue reference to the variants instead of a const reference.

6.1.4.4 visit() [2/2]

```
template<typename... Variants>
template<typename... Fs>
decltype(auto) wr22::regex_parser::utils::Adt< Variants >::visit (
    Fs &&... visitors ) const [inline]
```

Visit the ADT, applying the suitable function from the list of visitors on the variant held.

Using this method is essentially the same as using `std::visit` on the variant, except that, for convenience, multiple visitors are joined into one big visitor. That is, a typical `Adt` usage might look like this:

```
struct MyAdt : public Adt<int, double> {
    // Make the constructor available in the derived class.
    using Adt<int, double>::Adt;
};
// <...>
void func() {
    // Variant type: double.
    MyAdt my_adt = 3.14;
    // Prints "Double: 3.14".
    my_adt.visit(
        [] (int x) { std::cout << "Int: " << x << std::endl; },
        [] (double x) { std::cout << "Double: " << x << std::endl; }
    );
}
```

This is the constant version of the method. Visitors must be callable with the const reference to variant types.

6.1.5 Member Data Documentation

6.1.5.1 m_variant

```
template<typename... Variants>
VariantType wr22::regex_parser::utils::Adt< Variants >::m_variant [protected]
```

The documentation for this class was generated from the following file:

- include/wr22/regex_parser/utils/[adt.hpp](#)

6.2 wr22::regex_parser::regex::part::Alternatives Struct Reference

A regex part with the list of alternatives to be matched.

```
#include <part.hpp>
```

Public Member Functions

- [Alternatives](#) (std::vector< [SpannedPart](#) > [alternatives](#))
- bool [operator==](#) (const [Alternatives](#) &rhs) const =default

Public Attributes

- std::vector< [SpannedPart](#) > [alternatives](#)
The list of the alternatives.

Static Public Attributes

- static constexpr const char * [code_name](#) = "alternatives"

6.2.1 Detailed Description

A regex part with the list of alternatives to be matched.

[Alternatives](#) in regular expressions are subexpressions by `|`. For the whole expression part's match to succeed, at least one of the subexpressions must match the input successfully.

As an example, `a| (b) |cde` would be represented as an [Alternatives](#) part with 3 alternatives. The alternatives themselves are represented recursively as [SpannedParts](#).

6.2.2 Constructor & Destructor Documentation

6.2.2.1 Alternatives()

```
wr22::regex_parser::regex::part::Alternatives::Alternatives (
    std::vector< SpannedPart > alternatives ) [explicit]
```

6.2.3 Member Function Documentation

6.2.3.1 operator==()

```
bool wr22::regex_parser::regex::part::Alternatives::operator== (
    const Alternatives & rhs ) const [default]
```

6.2.4 Member Data Documentation

6.2.4.1 alternatives

```
std::vector<SpannedPart> wr22::regex_parser::regex::part::Alternatives::alternatives
```

The list of the alternatives.

6.2.4.2 code_name

```
constexpr const char* wr22::regex_parser::regex::part::Alternatives::code_name = "alternatives"
[static], [constexpr]
```

The documentation for this struct was generated from the following files:

- include/wr22/regex_parser/regex/part.hpp
- src/regex/part.cpp

6.3 wr22::regex_parser::utils::Box< T > Class Template Reference

A copyable and equality-comparable wrapper around `std::unique_ptr`.

```
#include <box.hpp>
```

Public Member Functions

- `Box (T &&value)`
Constructor that places a value inside the wrapped `std::unique_ptr`.
- `Box (std::unique_ptr< T > ptr)`
Constructor that adopts an existing `std::unique_ptr`.
- `template<typename Dummy = T>`
`Box (const Box &other)`
Copy constructor.
- `const T & operator* () const`
Dereferencing operator: obtain a const reference to the stored value.
- `T & operator* ()`
Dereferencing operator: obtain a reference to the stored value.

Static Public Member Functions

- `template<typename... Args>`
`static Box< T > construct_in_place (Args &&... args)`
Construct a value on the heap in place.

6.3.1 Detailed Description

```
template<typename T>
class wr22::regex_parser::utils::Box< T >
```

A copyable and equality-comparable wrapper around `std::unique_ptr`.

The behavior of this wrapper regarding copying and equality comparison are akin to that of Rust's `std::boxed::Box`, and hence the class's name. Namely, when testing for (in)equality, the wrapped values are compared instead of raw pointers, and, when wrapped values are copyable, copying a `Box` creates another `std::unique_ptr` with a copy of the wrapped value.

A `Box` usually contains a value. However, it may become empty when it is moved from. To ensure safety, most operations on an empty box will throw a `BoxIsEmpty` exception instead of causing undefined behavior.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 `Box()` [1/3]

```
template<typename T >
wr22::regex_parser::utils::Box< T >::Box (
    T && value ) [inline], [explicit]
```

Constructor that places a value inside the wrapped `std::unique_ptr`.

Takes the value by a universal reference and, due to perfect forwarding, both copy and move initialization is possible.

6.3.2.2 Box() [2/3]

```
template<typename T >
wr22::regex_parser::utils::Box< T >::Box (
    std::unique_ptr< T > ptr ) [inline], [explicit]
```

Constructor that adopts an existing `std::unique_ptr`.

Takes the `std::unique_ptr` by value, so the latter must be either passed directly as an rvalue or `std::move()` d into the argument. However, please note that if your code snippet looks like this:

```
Box<T>::construct_in_place(args...)
```

Then you should take a look at the `construct_in_place` method:

```
Box<T>::construct_in_place(args...)
```

6.3.2.3 Box() [3/3]

```
template<typename T >
template<typename Dummy = T>
wr22::regex_parser::utils::Box< T >::Box (
    const Box< T > & other ) [inline]
```

Copy constructor.

Creates another `std::unique_ptr` with a copy of the currently wrapped value.

Parameters

<code>`other`</code>	the <code>Box</code> from which to copy.
----------------------	--

Exceptions

<code>BoxIsEmpty</code>	if <code>other</code> is empty.
-------------------------	---------------------------------

6.3.3 Member Function Documentation

6.3.3.1 construct_in_place()

```
template<typename T >
template<typename... Args>
static Box< T > wr22::regex_parser::utils::Box< T >::construct_in_place (
    Args &&... args ) [inline], [static]
```

Construct a value on the heap in place.

Forwards the arguments to `std::make_unique` and wraps the resulting `std::unique_ptr`.

6.3.3.2 operator*() [1/2]

```
template<typename T >
T & wr22::regex_parser::utils::Box< T >::operator* ( ) [inline]
```

Dereferencing operator: obtain a reference to the stored value.

Exceptions

BoxIsEmpty	if this Box does not contain a value at the moment.
----------------------------	---

6.3.3.3 operator*() [2/2]

```
template<typename T >
const T & wr22::regex_parser::utils::Box< T >::operator* ( ) const [inline]
```

Dereferencing operator: obtain a const reference to the stored value.

Exceptions

BoxIsEmpty	if this Box does not contain a value at the moment.
----------------------------	---

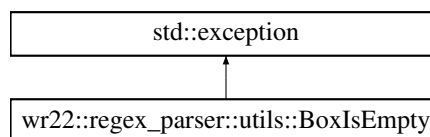
The documentation for this class was generated from the following file:

- include/wr22/regex_parser/utils/[box.hpp](#)

6.4 wr22::regex_parser::utils::BoxIsEmpty Struct Reference

```
#include <box.hpp>
```

Inheritance diagram for wr22::regex_parser::utils::BoxIsEmpty:



Public Member Functions

- const char * [what](#) () const noexcept override

6.4.1 Member Function Documentation

6.4.1.1 what()

```
const char * wr22::regex_parser::utils::BoxIsEmpty::what ( ) const [override], [noexcept]
```

The documentation for this struct was generated from the following files:

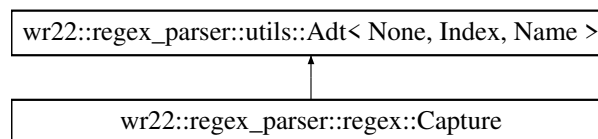
- include/wr22/regex_parser/utils/[box.hpp](#)
- src/utils/[box.cpp](#)

6.5 wr22::regex_parser::regex::Capture Class Reference

Group capture behavior.

```
#include <capture.hpp>
```

Inheritance diagram for wr22::regex_parser::regex::Capture:



Additional Inherited Members

6.5.1 Detailed Description

Group capture behavior.

A group can be captured by index (when one writes `(contents)`), by name (e.g. `(?<name>contents)` in some dialects) or not captured at all (`(?:contents)`). Objects of this type determine how exactly a certain group is going to be captured. This is a variant type (see [Part](#) and [utils::Adt](#) for a more detailed explanation of the concept). The variants for this class (explicitly or implicitly convertible to this type) are located in the `capture` namespace.

The documentation for this class was generated from the following file:

- include/wr22/regex_parser/regex/[capture.hpp](#)

6.6 wr22::regex_parser::regex::part::CharacterClass Struct Reference

A regex part specifying a character class (e.g. `[a-z_]`).

```
#include <part.hpp>
```

Public Member Functions

- [CharacterClass](#) ([CharacterClassData](#) data)
- bool [operator==](#) (const [CharacterClass](#) &rhs) const =default

Public Attributes

- [CharacterClassData](#) data
The list of character ranges.

Static Public Attributes

- static constexpr const char * [code_name](#) = "character_class"

6.6.1 Detailed Description

A regex part specifying a character class (e.g. `[a-z_]`).

6.6.2 Constructor & Destructor Documentation

6.6.2.1 [CharacterClass\(\)](#)

```
wr22::regex_parser::regex::part::CharacterClass::CharacterClass (  
    CharacterClassData data ) [explicit]
```

6.6.3 Member Function Documentation

6.6.3.1 [operator==\(\)](#)

```
bool wr22::regex_parser::regex::part::CharacterClass::operator== (  
    const CharacterClass & rhs ) const [default]
```

6.6.4 Member Data Documentation

6.6.4.1 code_name

```
constexpr const char* wr22::regex_parser::regex::part::CharacterClass::code_name = "character↵_class" [static], [constexpr]
```

6.6.4.2 data

```
CharacterClassData wr22::regex_parser::regex::part::CharacterClass::data
```

The list of character ranges.

The documentation for this struct was generated from the following files:

- include/wr22/regex_parser/regex/[part.hpp](#)
- src/regex/[part.cpp](#)

6.7 wr22::regex_parser::regex::CharacterClassData Struct Reference

A character class representation: a list of character ranges plus some additional properties.

```
#include <character_class_data.hpp>
```

Public Member Functions

- bool [operator==](#) (const [CharacterClassData](#) &rhs) const =default

Public Attributes

- std::vector< [SpannedCharacterRange](#) > [ranges](#)
List of character ranges and their spans.
- bool [inverted](#)
True if the match is inverted (i.e. `[^something]`), false otherwise.

6.7.1 Detailed Description

A character class representation: a list of character ranges plus some additional properties.

6.7.2 Member Function Documentation

6.7.2.1 operator==(

```
bool wr22::regex_parser::regex::CharacterClassData::operator== (
    const CharacterClassData & rhs ) const [default]
```

6.7.3 Member Data Documentation

6.7.3.1 inverted

```
bool wr22::regex_parser::regex::CharacterClassData::inverted
```

True if the match is inverted (i.e. `[^something]`), false otherwise.

6.7.3.2 ranges

```
std::vector<SpannedCharacterRange> wr22::regex_parser::regex::CharacterClassData::ranges
```

List of character ranges and their spans.

Example: for `[a-z123]` it is `['a'..'z' [1..4], '1'..'1' [4..4], '2'..'2' [5..5], '3'..'3' [6..6]]` (both ends in character ranges are included; `[A..B]` are spans with the left end included and the right one excluded).

The documentation for this struct was generated from the following file:

- `include/wr22/regex_parser/regex/character_class_data.hpp`

6.8 wr22::regex_parser::regex::CharacterRange Class Reference

A non-empty character range, possibly containing only one character.

```
#include <character_range.hpp>
```

Public Member Functions

- `char32_t first ()` const noexcept
Get the left range bound (inclusive).
- `char32_t last ()` const noexcept
Get the right range bound (inclusive).
- `bool contains (char32_t character)` const noexcept
Check if this range contains a given character.
- `bool is_single_character ()` const noexcept
Check if this range contains exactly one character.
- `bool operator== (const CharacterRange &rhs)` const noexcept=default

Static Public Member Functions

- static [CharacterRange from_endpoints](#) (char32_t *first*, char32_t *last*)
Construct a character range given its two inclusive endpoints.
- static [CharacterRange from_single_character](#) (char32_t *character*) noexcept
Construct a character range containing one given character.

6.8.1 Detailed Description

A non-empty character range, possibly containing only one character.

6.8.2 Member Function Documentation

6.8.2.1 contains()

```
bool wr22::regex_parser::regex::CharacterRange::contains (
    char32_t character ) const [noexcept]
```

Check if this range contains a given character.

6.8.2.2 first()

```
char32_t wr22::regex_parser::regex::CharacterRange::first ( ) const [noexcept]
```

Get the left range bound (inclusive).

6.8.2.3 from_endpoints()

```
CharacterRange wr22::regex_parser::regex::CharacterRange::from_endpoints (
    char32_t first,
    char32_t last ) [static]
```

Construct a character range given its two inclusive endpoints.

Exceptions

<code>`InvalidCharacterRange`</code>	if <code>last < first</code> .
--------------------------------------	-----------------------------------

6.8.2.4 from_single_character()

```
CharacterRange wr22::regex_parser::regex::CharacterRange::from_single_character (
    char32_t character ) [static], [noexcept]
```

Construct a character range containing one given character.

6.8.2.5 is_single_character()

```
bool wr22::regex_parser::regex::CharacterRange::is_single_character ( ) const [noexcept]
```

Check if this range contains exactly one character.

6.8.2.6 last()

```
char32_t wr22::regex_parser::regex::CharacterRange::last ( ) const [noexcept]
```

Get the right range bound (inclusive).

6.8.2.7 operator==()

```
bool wr22::regex_parser::regex::CharacterRange::operator== (
    const CharacterRange & rhs ) const [default], [noexcept]
```

The documentation for this class was generated from the following files:

- include/wr22/regex_parser/regex/character_range.hpp
- src/regex/character_range.cpp

6.9 wr22::regex_parser::regex::part::Empty Struct Reference

An empty regex part.

```
#include <part.hpp>
```

Public Member Functions

- `Empty` ()=default
- bool `operator==` (const `Empty` &rhs) const =default

Static Public Attributes

- static constexpr const char * `code_name` = "empty"

6.9.1 Detailed Description

An empty regex part.

Corresponds to an empty regular expression ("") or the contents of an empty parenthesized group (" () ").

6.9.2 Constructor & Destructor Documentation

6.9.2.1 Empty()

```
wr22::regex_parser::regex::part::Empty::Empty ( ) [explicit], [default]
```

6.9.3 Member Function Documentation

6.9.3.1 operator==()

```
bool wr22::regex_parser::regex::part::Empty::operator== (
    const Empty & rhs ) const [default]
```

6.9.4 Member Data Documentation

6.9.4.1 code_name

```
constexpr const char* wr22::regex_parser::regex::part::Empty::code_name = "empty" [static],
[constexpr]
```

The documentation for this struct was generated from the following file:

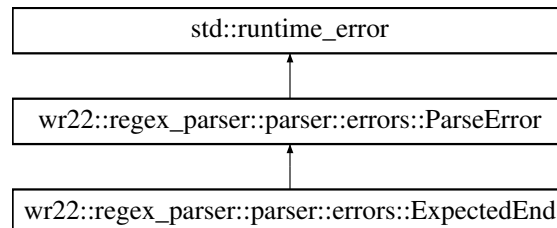
- include/wr22/regex_parser/regex/[part.hpp](#)

6.10 wr22::regex_parser::parser::errors::ExpectedEnd Class Reference

The error when the parser expected the input to end, but it did not.

```
#include <errors.hpp>
```

Inheritance diagram for wr22::regex_parser::parser::errors::ExpectedEnd:



Public Member Functions

- [ExpectedEnd](#) (size_t [position](#), char32_t [char_got](#))
Constructor.
- size_t [position](#) () const
Get the input position. See the constructor docs for a more detailed description.
- char32_t [char_got](#) () const
Get the character the parser has received.

6.10.1 Detailed Description

The error when the parser expected the input to end, but it did not.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 ExpectedEnd()

```
wr22::regex_parser::parser::errors::ExpectedEnd::ExpectedEnd (
    size_t position,
    char32_t char_got )
```

Constructor.

Parameters

<i>position</i>	the 0-based position in the input when the parser has encountered the end of input.
<i>char_got</i>	the character that the parser has received instead of the end of input.

6.10.3 Member Function Documentation

6.10.3.1 char_got()

```
char32_t wr22::regex_parser::parser::errors::ExpectedEnd::char_got ( ) const
```

Get the character the parser has received.

See the constructor docs for a more detailed description.

6.10.3.2 position()

```
size_t wr22::regex_parser::parser::errors::ExpectedEnd::position ( ) const
```

Get the input position. See the constructor docs for a more detailed description.

The documentation for this class was generated from the following files:

- include/wr22/regex_parser/parser/errors.hpp
- src/parser/errors.cpp

6.11 wr22::regex_parser::regex::part::Group Struct Reference

A regex part that represents a group in parentheses.

```
#include <part.hpp>
```

Public Member Functions

- [Group](#) ([Capture capture](#), [SpannedPart inner](#))
Convenience constructor.
- bool [operator==](#) (const [Group](#) &rhs) const =default

Public Attributes

- [Capture capture](#)
Capture behavior.
- [utils::Box< SpannedPart > inner](#)
The smart pointer to the group contents.

Static Public Attributes

- static constexpr const char * [code_name](#) = "group"

6.11.1 Detailed Description

A regex part that represents a group in parentheses.

A group in regular expressions is virtually everything that is enclosed with parentheses: `(some group)`, `(?<:blablabla)` and `(?P<group_name>group contents)` are all groups.

A group has two main attributes: (1) how it is captured during matching and (2) the contents of the group. The contents is simply another [SpannedPart](#). The capture behavior is expressed by a separate type [Capture](#). See its docs for additional info, and take a look at <https://www.regular-expressions.info/brackets.html> for an introduction to or a recap of regex groups and capturing.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 Group()

```
wr22::regex_parser::regex::part::Group::Group (
    Capture capture,
    SpannedPart inner ) [explicit]
```

Convenience constructor.

6.11.3 Member Function Documentation

6.11.3.1 operator==()

```
bool wr22::regex_parser::regex::part::Group::operator== (
    const Group & rhs ) const [default]
```

6.11.4 Member Data Documentation

6.11.4.1 capture

[Capture](#) wr22::regex_parser::regex::part::Group::capture

[Capture](#) behavior.

6.11.4.2 code_name

```
constexpr const char* wr22::regex_parser::regex::part::Group::code_name = "group" [static],
[constexpr]
```

6.11.4.3 inner

```
utils::Box<SpannedPart> wr22::regex_parser::regex::part::Group::inner
```

The smart pointer to the group contents.

The documentation for this struct was generated from the following files:

- include/wr22/regex_parser/regex/part.hpp
- src/regex/part.cpp

6.12 wr22::regex_parser::regex::capture::Index Struct Reference

Denotes a group captured by index.

```
#include <capture.hpp>
```

Public Member Functions

- [Index](#) ()=default
- bool [operator==](#) (const [Index](#) &rhs) const =default

Static Public Attributes

- static constexpr const char * [code_name](#) = "index"

6.12.1 Detailed Description

Denotes a group captured by index.

6.12.2 Constructor & Destructor Documentation

6.12.2.1 Index()

```
wr22::regex_parser::regex::capture::Index::Index ( ) [explicit], [default]
```

6.12.3 Member Function Documentation

6.12.3.1 operator==()

```
bool wr22::regex_parser::regex::capture::Index::operator== (
    const Index & rhs ) const [default]
```

6.12.4 Member Data Documentation

6.12.4.1 code_name

```
constexpr const char* wr22::regex_parser::regex::capture::Index::code_name = "index" [static],
[constexpr]
```

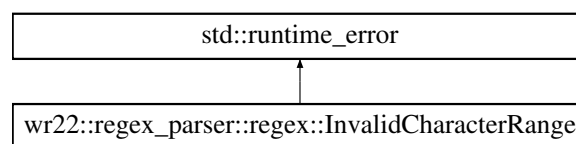
The documentation for this struct was generated from the following file:

- include/wr22/regex_parser/regex/capture.hpp

6.13 wr22::regex_parser::regex::InvalidCharacterRange Struct Reference

```
#include <character_range.hpp>
```

Inheritance diagram for wr22::regex_parser::regex::InvalidCharacterRange:



Public Member Functions

- [InvalidCharacterRange](#) (char32_t first, char32_t last)

Public Attributes

- char32_t first
- char32_t last

6.13.1 Constructor & Destructor Documentation

6.13.1.1 InvalidCharacterRange()

```
wr22::regex_parser::regex::InvalidCharacterRange::InvalidCharacterRange (
    char32_t first,
    char32_t last ) [explicit]
```

6.13.2 Member Data Documentation

6.13.2.1 first

```
char32_t wr22::regex_parser::regex::InvalidCharacterRange::first
```

6.13.2.2 last

```
char32_t wr22::regex_parser::regex::InvalidCharacterRange::last
```

The documentation for this struct was generated from the following files:

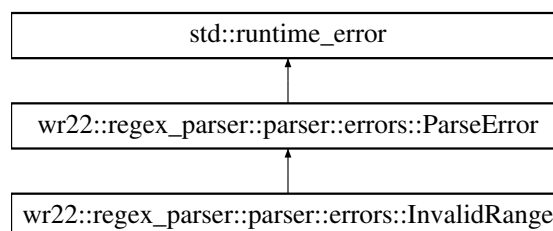
- include/wr22/regex_parser/regex/[character_range.hpp](#)
- src/regex/[character_range.cpp](#)

6.14 wr22::regex_parser::parser::errors::InvalidRange Class Reference

The error indicating that a character range in a character class is invalid.

```
#include <errors.hpp>
```

Inheritance diagram for wr22::regex_parser::parser::errors::InvalidRange:



Public Member Functions

- `InvalidRange (span::Span span, char32_t first, char32_t last)`
Constructor.
- `span::Span span () const`
Get the span of the character range. See the constructor docs for a more detailed explanation.
- `char32_t first () const`
Get the first character in the character range.
- `char32_t last () const`
Get the last character in the character range.

6.14.1 Detailed Description

The error indicating that a character range in a character class is invalid.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 InvalidRange()

```
wr22::regex_parser::parser::errors::InvalidRange::InvalidRange (
    span::Span span,
    char32_t first,
    char32_t last )
```

Constructor.

Parameters

<i>span</i>	the span of the character range considered.
<i>first</i>	the first character in (left bound of) the range, as in the regex.
<i>last</i>	the last character in (right bound of) the range, as in the regex.

6.14.3 Member Function Documentation

6.14.3.1 first()

```
char32_t wr22::regex_parser::parser::errors::InvalidRange::first ( ) const
```

Get the first character in the character range.

See the constructor docs for a more detailed explanation.

6.14.3.2 last()

```
char32_t wr22::regex_parser::parser::errors::InvalidRange::last ( ) const
```

Get the last character in the character range.

See the constructor docs for a more detailed explanation.

6.14.3.3 span()

```
span::Span wr22::regex_parser::parser::errors::InvalidRange::span ( ) const
```

Get the span of the character range. See the constructor docs for a more detailed explanation.

The documentation for this class was generated from the following files:

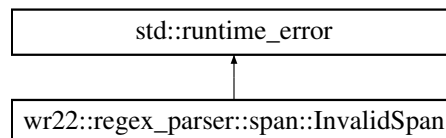
- include/wr22/regex_parser/parser/errors.hpp
- src/parser/errors.cpp

6.15 wr22::regex_parser::span::InvalidSpan Struct Reference

The exception thrown on an attempt to construct an invalid span.

```
#include <span.hpp>
```

Inheritance diagram for wr22::regex_parser::span::InvalidSpan:



Public Member Functions

- [InvalidSpan](#) (size_t [begin](#), size_t [end](#))

Public Attributes

- size_t [begin](#)
- size_t [end](#)

6.15.1 Detailed Description

The exception thrown on an attempt to construct an invalid span.

See the documentation for [Span](#) for additional information.

6.15.2 Constructor & Destructor Documentation

6.15.2.1 InvalidSpan()

```
wr22::regex_parser::span::InvalidSpan::InvalidSpan (
    size_t begin,
    size_t end )
```

6.15.3 Member Data Documentation

6.15.3.1 begin

```
size_t wr22::regex_parser::span::InvalidSpan::begin
```

6.15.3.2 end

```
size_t wr22::regex_parser::span::InvalidSpan::end
```

The documentation for this struct was generated from the following files:

- include/wr22/regex_parser/span/[span.hpp](#)
- src/span/[span.cpp](#)

6.16 wr22::regex_parser::regex::part::Literal Struct Reference

An regex part that matches a single character literally.

```
#include <part.hpp>
```

Public Member Functions

- [Literal](#) (char32_t [character](#))
- bool [operator==](#) (const [Literal](#) &rhs) const =default

Public Attributes

- char32_t [character](#)

Static Public Attributes

- static constexpr const char * `code_name` = "literal"

6.16.1 Detailed Description

An regex part that matches a single character literally.

Corresponds to a plain character in a regular expression. E.g. the regex "foo" contains three character literals: f, o and o.

6.16.2 Constructor & Destructor Documentation

6.16.2.1 Literal()

```
wr22::regex_parser::regex::part::Literal::Literal (  
    char32_t character ) [explicit]
```

6.16.3 Member Function Documentation

6.16.3.1 operator==()

```
bool wr22::regex_parser::regex::part::Literal::operator== (  
    const Literal & rhs ) const [default]
```

6.16.4 Member Data Documentation

6.16.4.1 character

```
char32_t wr22::regex_parser::regex::part::Literal::character
```

6.16.4.2 code_name

```
constexpr const char* wr22::regex_parser::regex::part::Literal::code_name = "literal" [static],
[constexpr]
```

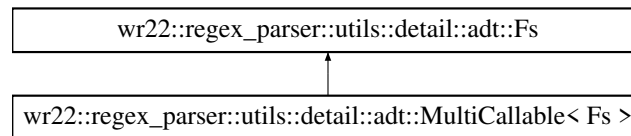
The documentation for this struct was generated from the following files:

- [include/wr22/regex_parser/regex/part.hpp](#)
- [src/regex/part.cpp](#)

6.17 wr22::regex_parser::utils::detail::adt::MultiCallable< Fs > Struct Template Reference

```
#include <adt.hpp>
```

Inheritance diagram for wr22::regex_parser::utils::detail::adt::MultiCallable< Fs >:



Public Member Functions

- [MultiCallable](#) (Fs &&... fs)

6.17.1 Constructor & Destructor Documentation

6.17.1.1 MultiCallable()

```
template<typename... Fs>
wr22::regex_parser::utils::detail::adt::MultiCallable< Fs >::MultiCallable (
    Fs &&... fs ) [inline]
```

The documentation for this struct was generated from the following file:

- [include/wr22/regex_parser/utils/adt.hpp](#)

6.18 wr22::regex_parser::regex::capture::Name Struct Reference

Denotes a group captured by name.

```
#include <capture.hpp>
```

Public Member Functions

- [Name](#) (std::string [name](#), [NamedCaptureFlavor](#) [flavor](#))
- bool [operator==](#) (const [Name](#) &[rhs](#)) const =default

Public Attributes

- std::string [name](#)
- [NamedCaptureFlavor](#) [flavor](#)

Static Public Attributes

- static constexpr const char * [code_name](#) = "name"

6.18.1 Detailed Description

Denotes a group captured by name.

A specific name and the syntax variant for this name's specification (see [NamedCaptureFlavor](#)) are stored.

6.18.2 Constructor & Destructor Documentation

6.18.2.1 Name()

```
wr22::regex_parser::regex::capture::Name::Name (
    std::string name,
    NamedCaptureFlavor flavor ) [explicit]
```

6.18.3 Member Function Documentation

6.18.3.1 operator==()

```
bool wr22::regex_parser::regex::capture::Name::operator== (
    const Name & rhs ) const [default]
```

6.18.4 Member Data Documentation

6.18.4.1 code_name

```
constexpr const char* wr22::regex_parser::regex::capture::Name::code_name = "name" [static],  
[constexpr]
```

6.18.4.2 flavor

```
NamedCaptureFlavor wr22::regex_parser::regex::capture::Name::flavor
```

6.18.4.3 name

```
std::string wr22::regex_parser::regex::capture::Name::name
```

The documentation for this struct was generated from the following files:

- include/wr22/regex_parser/regex/capture.hpp
- src/parser/capture.cpp

6.19 wr22::regex_parser::regex::capture::None Struct Reference

Denotes an non-capturing group.

```
#include <capture.hpp>
```

Public Member Functions

- [None](#) ()=default
- bool [operator==](#) (const [None](#) &rhs) const =default

Static Public Attributes

- static constexpr const char * [code_name](#) = "none"

6.19.1 Detailed Description

Denotes an non-capturing group.

6.19.2 Constructor & Destructor Documentation

6.19.2.1 None()

```
wr22::regex_parser::regex::capture::None::None ( ) [explicit], [default]
```

6.19.3 Member Function Documentation

6.19.3.1 operator==(

```
bool wr22::regex_parser::regex::capture::None::operator== (
    const None & rhs ) const [default]
```

6.19.4 Member Data Documentation

6.19.4.1 code_name

```
constexpr const char* wr22::regex_parser::regex::capture::None::code_name = "none" [static],
[constexpr]
```

The documentation for this struct was generated from the following file:

- include/wr22/regex_parser/regex/capture.hpp

6.20 wr22::regex_parser::regex::part::Optional Struct Reference

A regex part specifying an optional quantifier (*expression*) ?).

```
#include <part.hpp>
```

Public Member Functions

- [Optional](#) ([SpannedPart](#) inner)
Convenience constructor.
- bool [operator==](#) (const [Optional](#) &rhs) const =default

Public Attributes

- [utils::Box](#)< [SpannedPart](#) > inner
The smart pointer to the subexpression under the quantifier.

Static Public Attributes

- static constexpr const char * `code_name` = "optional"

6.20.1 Detailed Description

A regex part specifying an optional quantifier (`expression` ?).

6.20.2 Constructor & Destructor Documentation

6.20.2.1 Optional()

```
wr22::regex_parser::regex::part::Optional::Optional (
    SpannedPart inner ) [explicit]
```

Convenience constructor.

6.20.3 Member Function Documentation

6.20.3.1 operator==()

```
bool wr22::regex_parser::regex::part::Optional::operator== (
    const Optional & rhs ) const [default]
```

6.20.4 Member Data Documentation

6.20.4.1 code_name

```
constexpr const char* wr22::regex_parser::regex::part::Optional::code_name = "optional" [static],
[constexpr]
```


6.20.4.2 inner

```
utils::Box<SpannedPart> wr22::regex_parser::regex::part::Optional::inner
```

The smart pointer to the subexpression under the quantifier.

The documentation for this struct was generated from the following files:

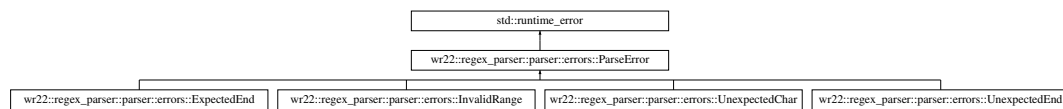
- include/wr22/regex_parser/regex/[part.hpp](#)
- src/regex/[part.cpp](#)

6.21 wr22::regex_parser::parser::errors::ParseError Struct Reference

The base class for parse errors.

```
#include <errors.hpp>
```

Inheritance diagram for wr22::regex_parser::parser::errors::ParseError:



6.21.1 Detailed Description

The base class for parse errors.

This exception type should be caught if it is desired to catch all parse errors. However, there are more specific exceptions deriving from this one that can be handled separately for greater flexibility.

The documentation for this struct was generated from the following file:

- include/wr22/regex_parser/parser/[errors.hpp](#)

6.22 wr22::regex_parser::parser::Parser< Iter, Sentinel > Class Template Reference

A regex parser.

Public Member Functions

- [Parser](#) (Iter begin, Sentinel end)
Constructor.
- void [expect_end](#) ()
Ensure that the parser has consumed all of the input.
- [regex::SpannedPart parse_regex](#) ()
Parse a regex consuming part of the remaining input.
- [regex::SpannedPart parse_alternatives](#) ()
Intermediate rule: parse a pipe-separated list of alternatives (e.g.
- [regex::SpannedPart parse_sequence](#) ()
Intermediate rule: parse a sequence of atoms (e.g.
- [regex::SpannedPart parse_sequence_or_empty](#) ()
Intermediate rule: parse a possibly empty sequence of atoms.
- [regex::SpannedPart parse_atom](#) ()
Intermediate rule: parse an atom.
- [regex::SpannedPart parse_wildcard](#) ()
Intermediate rule: parse a wildcard (.
- [regex::SpannedPart parse_char_literal](#) ()
Intermediate rule: parse a character literal.
- [regex::SpannedPart parse_group](#) ()
Intermediate rule: parse a parenthesized group (any capture variant).
- std::pair< std::string, [Span](#) > [parse_group_name](#) ()
Intermediate rule: parse a group name.
- [regex::SpannedPart parse_char_class](#) ()
Intermediate rule: parse a character class (e.g.

6.22.1 Detailed Description

```
template<typename Iter, typename Sentinel>
requires requires(Iter iter, Sentinel end) { ++iter; { *iter } -> std::convertible_to<char32_t>; { iter == end } -> std::convertible_to<bool>; { iter != end } -> std::convertible_to<bool>; }
class wr22::regex_parser::parser::Parser< Iter, Sentinel >
```

A regex parser.

For additional information see the methods' docs, particularly the constructor and the `parse_regex` method.

6.22.2 Constructor & Destructor Documentation

6.22.2.1 Parser()

```
template<typename Iter , typename Sentinel >
wr22::regex_parser::parser::Parser< Iter, Sentinel >::Parser (
    Iter begin,
    Sentinel end ) [inline]
```

Constructor.

This constructor stores a pair of forward iterators that should generate a sequence of Unicode code points (`char32_t`). The `begin` iterator and the `end` sentinel may have different types provided that the iterator can be equality comparable with the sentinel.

SAFETY: The iterators must not be invalidated as long as this [Parser](#) object is still alive.

6.22.3 Member Function Documentation

6.22.3.1 expect_end()

```
template<typename Iter , typename Sentinel >
void wr22::regex_parser::parser::Parser< Iter, Sentinel >::expect_end ( ) [inline]
```

Ensure that the parser has consumed all of the input.

Does nothing if all input has been consumed.

Exceptions

<code>errors::ExpectedEnd</code>	if this is not the case.
----------------------------------	--------------------------

6.22.3.2 parse_alternatives()

```
template<typename Iter , typename Sentinel >
regex::SpannedPart wr22::regex_parser::parser::Parser< Iter, Sentinel >::parse_alternatives (
) [inline]
```

Intermediate rule: parse a pipe-separated list of alternatives (e.g.

`a|bb|ccc`).

Returns

the list of parsed alternatives packed into `regex::part::Alternatives` or, if and only if the list of alternatives contains exactly 1 element, the only alternative unchanged.

Exceptions

<code>errors::ParseError</code>	if the input cannot be parsed.
---------------------------------	--------------------------------

6.22.3.3 parse_atom()

```
template<typename Iter , typename Sentinel >
regex::SpannedPart wr22::regex_parser::parser::Parser< Iter, Sentinel >::parse_atom ( ) [inline]
```

Intermediate rule: parse an atom.

Currently, this grammar only recognizes two kinds of atoms: character literals (individual plain characters in a regex) and parenthesized groups. As the project development goes on, new kinds of atoms will be added.

Returns

the parsed atom (some variant of `regex::SpannedPart` depending on the atom kind).

Exceptions

<code>errors::ParseError</code>	if the input cannot be parsed.
---------------------------------	--------------------------------

6.22.3.4 parse_char_class()

```
template<typename Iter , typename Sentinel >
regex::SpannedPart wr22::regex_parser::parser::Parser< Iter, Sentinel >::parse_char_class ( )
[inline]
```

Intermediate rule: parse a character class (e.g.

`[^a-z0-9_-]`).

Returns

the character class AST node.

– is found right after the character range started or right after another range. In either case, it is considered as a plain character.

6.22.3.5 parse_char_literal()

```
template<typename Iter , typename Sentinel >
regex::SpannedPart wr22::regex_parser::parser::Parser< Iter, Sentinel >::parse_char_literal (
) [inline]
```

Intermediate rule: parse a character literal.

Returns

the parsed character literal (`regex::part::Literal`).

Exceptions

<code>errors::UnexpectedEnd</code>	if all characters from the input have already been consumed.
------------------------------------	--

6.22.3.6 parse_group()

```
template<typename Iter , typename Sentinel >
```

```
regex::SpannedPart wr22::regex_parser::parser::Parser< Iter, Sentinel >::parse_group ( )
[inline]
```

Intermediate rule: parse a parenthesized group (any capture variant).

Returns

the parsed group ([regex::part::Group](#)).

6.22.3.7 parse_group_name()

```
template<typename Iter , typename Sentinel >
std::pair< std::string, Span > wr22::regex_parser::parser::Parser< Iter, Sentinel >::parse_group_name ( ) [inline]
```

Intermediate rule: parse a group name.

Returns

the UTF-8 encoded group name as an `std::string`.

6.22.3.8 parse_regex()

```
template<typename Iter , typename Sentinel >
regex::SpannedPart wr22::regex_parser::parser::Parser< Iter, Sentinel >::parse_regex ( )
[inline]
```

Parse a regex consuming part of the remaining input.

This is **the** method that should be called to parse a regular expression because it represents the root rule of the regex grammar. Please note that this method may not consume all of the parser's input. Hence, if a whole regex is to be parsed, the `expect_end` method should be called afterwards.

Returns

the parsed regex AST (some variant of [regex::SpannedPart](#) depending on the input).

Exceptions

errors::ParseError	if the input cannot be parsed.
------------------------------------	--------------------------------

6.22.3.9 parse_sequence()

```
template<typename Iter , typename Sentinel >
```

```
regex::SpannedPart wr22::regex_parser::parser::Parser< Iter, Sentinel >::parse_sequence ( )
[inline]
```

Intermediate rule: parse a sequence of atoms (e.g.

`a(?:b)[c-e]`).

Returns

the list of parsed atoms packed into `regex::part::Sequence` or, if and only if this list of contains exactly 1 element, the only atom unchanged.

Exceptions

<code>errors::ParseError</code>	if the input cannot be parsed.
---------------------------------	--------------------------------

6.22.3.10 parse_sequence_or_empty()

```
template<typename Iter , typename Sentinel >
regex::SpannedPart wr22::regex_parser::parser::Parser< Iter, Sentinel >::parse_sequence_or_↵
empty ( ) [inline]
```

Intermediate rule: parse a possibly empty sequence of atoms.

Returns

`regex::part::Empty` if the sequence is empty, or calls `parse_sequence` otherwise.

Exceptions

<code>errors::ParseError</code>	if the input cannot be parsed.
---------------------------------	--------------------------------

6.22.3.11 parse_wildcard()

```
template<typename Iter , typename Sentinel >
regex::SpannedPart wr22::regex_parser::parser::Parser< Iter, Sentinel >::parse_wildcard ( )
[inline]
```

Intermediate rule: parse a wildcard (.

).

Returns

the wildcard AST node.

Exceptions

<code>errors::UnexpectedEnd</code>	if all characters from the input have already been consumed.
<code>errors::UnexpectedChar</code>	if the next input character is not . .

The documentation for this class was generated from the following file:

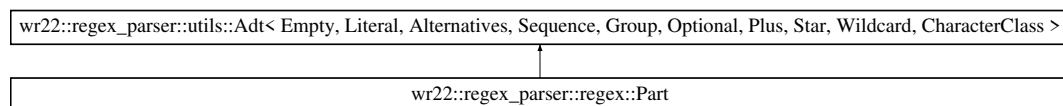
- [src/parser/regex.cpp](#)

6.23 wr22::regex_parser::regex::Part Class Reference

A part of a regular expression and its AST node type.

```
#include <part.hpp>
```

Inheritance diagram for wr22::regex_parser::regex::Part:



Additional Inherited Members

6.23.1 Detailed Description

A part of a regular expression and its AST node type.

The parsed regular expressions are represented as abstract syntax trees (ASTs). These are tree-like data structures where each node represents a regular expression part (or the whole regex), and, depending on their type, these nodes may have subexpressions. Subexpressions are [Parts](#) themselves, which also have child expressions and so on. For example, [`part::Sequence`](#) has a number of subexpressions, and each of them is of the type [Part](#) and is an AST node.

Each regex part has its own simple function. For example, [`part::Alternatives`](#) tries to match several alternative subexpressions against the input and succeeds if at least one of them does; and [`part::Sequence`](#) matches several subexpressions one after another, requiring them all to match respective parts of the input. By combining these simple nodes, it becomes possible to represent complex regular expressions. For example, the regex `aaa|bb` can be represented as a [`part::Alternatives`](#), where each of the alternatives is a [`part::Sequence`](#) of [`part::Literals`](#).

The [Part](#) itself is represented by [`std::variant`](#) via the helper class [`utils::Adt`](#). In a nutshell, it allows a regex part to "have" one of the several predefined types (the so-called variants, which are defined in the `part` namespace), but still be represented as a [Part](#). For the list of operations that can be performed on this type, e.g. to check if an instance of [Parts](#) has a specific variant and, if yes, access the value of this variant, see the documentation for the [`utils::Adt`](#) class, which [Part](#) inherits from.

Note that this type contains no span information for the root AST node. For a spanned version, see [SpannedPart](#).

The documentation for this class was generated from the following file:

- [include/wr22/regex_parser/regex/part.hpp](#)

6.24 wr22::regex_parser::regex::part::Plus Struct Reference

A regex part specifying an "at least one" quantifier ((expression) +).

```
#include <part.hpp>
```

Public Member Functions

- [Plus](#) ([SpannedPart](#) inner)
Convenience constructor.
- bool [operator==](#) (const [Plus](#) &rhs) const =default

Public Attributes

- [utils::Box](#)< [SpannedPart](#) > inner
The smart pointer to the subexpression under the quantifier.

Static Public Attributes

- static constexpr const char * [code_name](#) = "plus"

6.24.1 Detailed Description

A regex part specifying an "at least one" quantifier ((expression) +).

6.24.2 Constructor & Destructor Documentation

6.24.2.1 Plus()

```
wr22::regex_parser::regex::part::Plus::Plus (
    SpannedPart inner ) [explicit]
```

Convenience constructor.

6.24.3 Member Function Documentation

6.24.3.1 operator==()

```
bool wr22::regex_parser::regex::part::Plus::operator== (
    const Plus & rhs ) const [default]
```


6.24.4 Member Data Documentation

6.24.4.1 code_name

```
constexpr const char* wr22::regex_parser::regex::part::Plus::code_name = "plus" [static],
[constexpr]
```

6.24.4.2 inner

```
utils::Box<SpannedPart> wr22::regex_parser::regex::part::Plus::inner
```

The smart pointer to the subexpression under the quantifier.

The documentation for this struct was generated from the following files:

- include/wr22/regex_parser/regex/[part.hpp](#)
- src/regex/[part.cpp](#)

6.25 wr22::regex_parser::regex::part::Sequence Struct Reference

A regex part with the list of items to be matched one after another.

```
#include <part.hpp>
```

Public Member Functions

- [Sequence](#) (std::vector< [SpannedPart](#) > items)
- bool [operator==](#) (const [Sequence](#) &rhs) const =default

Public Attributes

- std::vector< [SpannedPart](#) > [items](#)
The list of the subexpressions.

Static Public Attributes

- static constexpr const char * [code_name](#) = "sequence"

6.25.1 Detailed Description

A regex part with the list of items to be matched one after another.

Sequences in regular expressions are just subexpressions going directly one after another. As an example, `a[b-e]` is a sequence of 3 subexpressions: `a`, `[b-e]` and `.`. As an another example, `ab` is a sequence of 2 subexpressions: `a` and `b`.

6.25.2 Constructor & Destructor Documentation

6.25.2.1 Sequence()

```
wr22::regex_parser::regex::part::Sequence::Sequence (
    std::vector< SpannedPart > items ) [explicit]
```

6.25.3 Member Function Documentation

6.25.3.1 operator==()

```
bool wr22::regex_parser::regex::part::Sequence::operator== (
    const Sequence & rhs ) const [default]
```

6.25.4 Member Data Documentation

6.25.4.1 code_name

```
constexpr const char* wr22::regex_parser::regex::part::Sequence::code_name = "sequence" [static],
[constexpr]
```

6.25.4.2 items

```
std::vector<SpannedPart> wr22::regex_parser::regex::part::Sequence::items
```

The list of the subexpressions.

The documentation for this struct was generated from the following files:

- include/wr22/regex_parser/regex/part.hpp
- src/regex/part.cpp

6.26 wr22::regex_parser::span::Span Class Reference

Character position range in the input string.

```
#include <span.hpp>
```

Public Member Functions

- [Span extend_right](#) (size_t num_positions) const
Construct a new span that equals the current span extended to the right by a specified number of positions.
- size_t [length](#) () const
Get the length of the span (the number of characters covered).
- size_t [begin](#) () const
*Get the *begin* position of the span.*
- size_t [end](#) () const
*Get the *end* position of the span.*
- bool [operator==](#) (const [Span](#) &other) const =default
- bool [operator!=](#) (const [Span](#) &other) const =default

Static Public Member Functions

- static [Span make_empty](#) (size_t position)
Construct an empty span that "starts" at a given position.
- static [Span make_single_position](#) (size_t position)
Construct a span that captures only one position.
- static [Span make_from_positions](#) (size_t [begin](#), size_t [end](#))
*Construct a span with given values of *begin* and *end* without any transformations.*
- static [Span make_with_length](#) (size_t [begin](#), size_t [length](#))
*Construct a span with a given value of *begin* and a given length.*

6.26.1 Detailed Description

Character position range in the input string.

The range is encoded by two numbers: *begin*, the position (0-based index) of the first character in the range, and *end*, the past-the-end position, or the 0-based index of the last character in the range **plus 1**. This is to be consistent with the behavior of C++ iterators and [begin\(\)/end\(\)](#) functions on STL containers. Please note, however, that the [begin\(\)/end\(\)](#) methods here are just accessors that are not used for iteration, they return plain indices which have no iterator semantics.

Invalid spans (*begin* > *end*) are not allowed and their construction will result in an error. See the documentation for the relevant methods for details.

6.26.2 Member Function Documentation

6.26.2.1 begin()

```
size_t wr22::regex_parser::span::Span::begin ( ) const
```

Get the `begin` position of the span.

6.26.2.2 end()

```
size_t wr22::regex_parser::span::Span::end ( ) const
```

Get the `end` position of the span.

6.26.2.3 extend_right()

```
Span wr22::regex_parser::span::Span::extend_right (
    size_t num_positions ) const
```

Construct a new span that equals the current span extended to the right by a specified number of positions.

Please note that this method creates a new span and does not modify the existing one.

Formally, `[begin..end].extend_right(n) = [begin..(end+n)]`.

Exceptions

<code>InvalidSpan</code>	if <code>end + length</code> overflows <code>size_t</code> . Note that the error message might not be precise enough.
--------------------------	---

6.26.2.4 length()

```
size_t wr22::regex_parser::span::Span::length ( ) const
```

Get the length of the span (the number of characters covered).

6.26.2.5 make_empty()

```
Span wr22::regex_parser::span::Span::make_empty (
    size_t position ) [static]
```

Construct an empty span that "starts" at a given position.

The resulting span will have `begin = position` and `end = position`.

6.26.2.6 make_from_positions()

```
Span wr22::regex_parser::span::Span::make_from_positions (
    size_t begin,
    size_t end ) [static]
```

Construct a span with given values of `begin` and `end` without any transformations.

Exceptions

<i>InvalidSpan</i>	if <code>end < begin</code> .
--------------------	----------------------------------

6.26.2.7 make_single_position()

```
Span wr22::regex_parser::span::Span::make_single_position (
    size_t position ) [static]
```

Construct a span that captures only one position.

The resulting span will have `begin = position` and `end = position + 1`.

Exceptions

<i>InvalidSpan</i>	if <code>position + 1</code> overflows <code>size_t</code> . Note that the error message might not be precise enough.
--------------------	---

6.26.2.8 make_with_length()

```
Span wr22::regex_parser::span::Span::make_with_length (
    size_t begin,
    size_t length ) [static]
```

Construct a span with a given value of `begin` and a given `length`.

The length is determined by the number of characters covered by this span, and, since `begin` and `end` form a half-interval, it equals `end - begin`.

Exceptions

<i>InvalidSpan</i>	if <code>begin + length</code> overflows <code>size_t</code> . Note that the error message might not be precise enough.
--------------------	---

6.26.2.9 operator"!=()

```
bool wr22::regex_parser::span::Span::operator!= (
    const Span & other ) const [default]
```

6.26.2.10 operator==(())

```
bool wr22::regex_parser::span::Span::operator== (
    const Span & other ) const [default]
```

The documentation for this class was generated from the following files:

- include/wr22/regex_parser/span/[span.hpp](#)
- src/span/[span.cpp](#)

6.27 wr22::regex_parser::regex::SpannedCharacterRange Struct Reference

A [CharacterRange](#) with its span.

```
#include <spanned_character_range.hpp>
```

Public Member Functions

- bool [operator==](#) (const [SpannedCharacterRange](#) &rhs) const =default

Public Attributes

- [CharacterRange](#) range
- [span::Span](#) span

6.27.1 Detailed Description

A [CharacterRange](#) with its span.

6.27.2 Member Function Documentation

6.27.2.1 operator==(())

```
bool wr22::regex_parser::regex::SpannedCharacterRange::operator== (
    const SpannedCharacterRange & rhs ) const [default]
```

6.27.3 Member Data Documentation

6.27.3.1 range

`CharacterRange` wr22::regex_parser::regex::SpannedCharacterRange::range

6.27.3.2 span

`span::Span` wr22::regex_parser::regex::SpannedCharacterRange::span

The documentation for this struct was generated from the following file:

- include/wr22/regex_parser/regex/spanned_character_range.hpp

6.28 wr22::regex_parser::regex::SpannedPart Class Reference

A version of `Part` including the span information (position in the input) of the root AST node (child nodes always contain it because they are represented as `SpannedParts` themselves).

```
#include <part.hpp>
```

Public Member Functions

- `SpannedPart` (`Part` part, `span::Span` span)
- `bool operator==` (const `SpannedPart` &other) const =default
- `bool operator!=` (const `SpannedPart` &other) const =default
- `const Part & part` () const
Access the wrapped `Part` (const version).
- `Part & part` ()
Access the wrapped `Part` (non-const version).
- `span::Span` span () const
Get the associated span.

6.28.1 Detailed Description

A version of `Part` including the span information (position in the input) of the root AST node (child nodes always contain it because they are represented as `SpannedParts` themselves).

6.28.2 Constructor & Destructor Documentation

6.28.2.1 SpannedPart()

```
wr22::regex_parser::regex::SpannedPart::SpannedPart (
    Part part,
    span::Span span ) [explicit]
```

6.28.3 Member Function Documentation

6.28.3.1 operator!=(())

```
bool wr22::regex_parser::regex::SpannedPart::operator!= (
    const SpannedPart & other ) const [default]
```

6.28.3.2 operator==(())

```
bool wr22::regex_parser::regex::SpannedPart::operator== (
    const SpannedPart & other ) const [default]
```

6.28.3.3 part() [1/2]

```
Part & wr22::regex_parser::regex::SpannedPart::part ( )
```

Access the wrapped [Part](#) (non-const version).

6.28.3.4 part() [2/2]

```
const Part & wr22::regex_parser::regex::SpannedPart::part ( ) const
```

Access the wrapped [Part](#) (const version).

6.28.3.5 span()

```
span::Span wr22::regex_parser::regex::SpannedPart::span ( ) const
```

Get the associated span.

The documentation for this class was generated from the following files:

- include/wr22/regex_parser/regex/[part.hpp](#)
- src/regex/[part.cpp](#)

6.29 wr22::regex_parser::regex::part::Star Struct Reference

A regex part specifying an "at least zero" quantifier ((expression) *).

```
#include <part.hpp>
```

Public Member Functions

- [Star](#) ([SpannedPart](#) inner)
Convenience constructor.
- bool [operator==](#) (const [Star](#) &rhs) const =default

Public Attributes

- [utils::Box](#)< [SpannedPart](#) > inner
The smart pointer to the subexpression under the quantifier.

Static Public Attributes

- static constexpr const char * [code_name](#) = "star"

6.29.1 Detailed Description

A regex part specifying an "at least zero" quantifier ((expression) *).

6.29.2 Constructor & Destructor Documentation

6.29.2.1 Star()

```
wr22::regex_parser::regex::part::Star::Star (  
    SpannedPart inner ) [explicit]
```

Convenience constructor.

6.29.3 Member Function Documentation

6.29.3.1 operator==()

```
bool wr22::regex_parser::regex::part::Star::operator== (  
    const Star & rhs ) const [default]
```

6.29.4 Member Data Documentation

6.29.4.1 code_name

```
constexpr const char* wr22::regex_parser::regex::part::Star::code_name = "star" [static],
[constexpr]
```

6.29.4.2 inner

```
utils::Box<SpannedPart> wr22::regex_parser::regex::part::Star::inner
```

The smart pointer to the subexpression under the quantifier.

The documentation for this struct was generated from the following files:

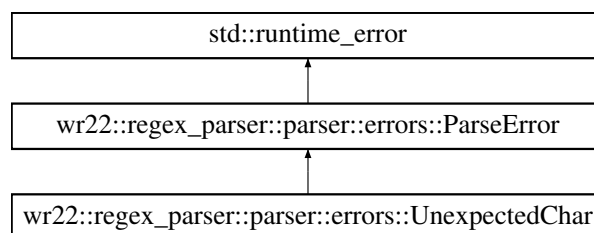
- include/wr22/regex_parser/regex/[part.hpp](#)
- src/regex/[part.cpp](#)

6.30 wr22::regex_parser::parser::errors::UnexpectedChar Class Reference

The error when the parser got a character it didn't expect at the current position.

```
#include <errors.hpp>
```

Inheritance diagram for wr22::regex_parser::parser::errors::UnexpectedChar:



Public Member Functions

- [UnexpectedChar](#) (size_t [position](#), char32_t [char_got](#), std::string [expected](#))
Constructor.
- size_t [position](#) () const
Get the input position. See the constructor docs for a more detailed description.
- char32_t [char_got](#) () const
Get the character the parser has received.
- const std::string & [expected](#) () const
Get the description of expected characters.

6.30.1 Detailed Description

The error when the parser got a character it didn't expect at the current position.

6.30.2 Constructor & Destructor Documentation

6.30.2.1 UnexpectedChar()

```
wr22::regex_parser::parser::errors::UnexpectedChar::UnexpectedChar (
    size_t position,
    char32_t char_got,
    std::string expected )
```

Constructor.

Parameters

<i>position</i>	the 0-based position in the input when the parser has encountered the unexpected character.
<i>char_got</i>	the character that the parser has received.
<i>expected</i>	a textual description of a class of characters expected instead.

6.30.3 Member Function Documentation

6.30.3.1 char_got()

```
char32_t wr22::regex_parser::parser::errors::UnexpectedChar::char_got ( ) const
```

Get the character the parser has received.

See the constructor docs for a more detailed description.

6.30.3.2 expected()

```
const std::string & wr22::regex_parser::parser::errors::UnexpectedChar::expected ( ) const
```

Get the description of expected characters.

See the constructor docs for a more detailed description.

6.30.3.3 position()

```
size_t wr22::regex_parser::parser::errors::UnexpectedChar::position ( ) const
```

Get the input position. See the constructor docs for a more detailed description.

The documentation for this class was generated from the following files:

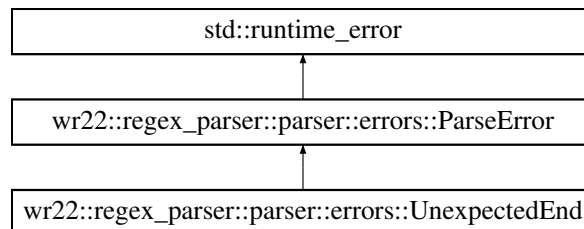
- include/wr22/regex_parser/parser/errors.hpp
- src/parser/errors.cpp

6.31 wr22::regex_parser::parser::errors::UnexpectedEnd Class Reference

The error when the parser hit the end of the input earlier than it expected.

```
#include <errors.hpp>
```

Inheritance diagram for wr22::regex_parser::parser::errors::UnexpectedEnd:



Public Member Functions

- [UnexpectedEnd](#) (size_t [position](#), std::string [expected](#))
Constructor.
- size_t [position](#) () const
Get the input position. See the constructor docs for a more detailed description.
- const std::string & [expected](#) () const
Get the description of expected characters.

6.31.1 Detailed Description

The error when the parser hit the end of the input earlier than it expected.

6.31.2 Constructor & Destructor Documentation

6.31.2.1 UnexpectedEnd()

```
wr22::regex_parser::parser::errors::UnexpectedEnd::UnexpectedEnd (
    size_t position,
    std::string expected )
```

Constructor.

Parameters

<i>position</i>	the 0-based position in the input when the parser has encountered the end of input.
<i>expected</i>	a textual description of a class of characters expected instead.

6.31.3 Member Function Documentation

6.31.3.1 expected()

```
const std::string & wr22::regex_parser::parser::errors::UnexpectedEnd::expected ( ) const
```

Get the description of expected characters.

See the constructor docs for a more detailed description.

6.31.3.2 position()

```
size_t wr22::regex_parser::parser::errors::UnexpectedEnd::position ( ) const
```

Get the input position. See the constructor docs for a more detailed description.

The documentation for this class was generated from the following files:

- include/wr22/regex_parser/parser/[errors.hpp](#)
- src/parser/[errors.cpp](#)

6.32 wr22::regex_parser::regex::part::Wildcard Struct Reference

A regex part specifying any single character (.).

```
#include <part.hpp>
```

Public Member Functions

- [Wildcard](#) ()=default
- bool [operator==](#) (const [Wildcard](#) &rhs) const =default

Static Public Attributes

- static constexpr const char * [code_name](#) = "wildcard"

6.32.1 Detailed Description

A regex part specifying any single character (.).

6.32.2 Constructor & Destructor Documentation

6.32.2.1 Wildcard()

```
wr22::regex_parser::regex::part::Wildcard::Wildcard ( ) [explicit], [default]
```

6.32.3 Member Function Documentation

6.32.3.1 operator==()

```
bool wr22::regex_parser::regex::part::Wildcard::operator== (
    const Wildcard & rhs ) const [default]
```

6.32.4 Member Data Documentation

6.32.4.1 code_name

```
constexpr const char* wr22::regex_parser::regex::part::Wildcard::code_name = "wildcard" [static],
[constexpr]
```

The documentation for this struct was generated from the following file:

- [include/wr22/regex_parser/regex/part.hpp](#)

Chapter 7

File Documentation

7.1 include/wr22/regex_parser/parser/errors.hpp File Reference

```
#include <wr22/regex_parser/span/span.hpp>
#include <exception>
#include <stdexcept>
#include <string>
```

Classes

- struct [wr22::regex_parser::parser::errors::ParseError](#)
The base class for parse errors.
- class [wr22::regex_parser::parser::errors::UnexpectedEnd](#)
The error when the parser hit the end of the input earlier than it expected.
- class [wr22::regex_parser::parser::errors::ExpectedEnd](#)
The error when the parser expected the input to end, but it did not.
- class [wr22::regex_parser::parser::errors::UnexpectedChar](#)
The error when the parser got a character it didn't expect at the current position.
- class [wr22::regex_parser::parser::errors::InvalidRange](#)
The error indicating that a character range in a character class is invalid.

Namespaces

- namespace [wr22](#)
- namespace [wr22::regex_parser](#)
- namespace [wr22::regex_parser::parser](#)
- namespace [wr22::regex_parser::parser::errors](#)

7.2 errors.hpp

[Go to the documentation of this file.](#)

```

1 #pragma once
2
3 // wr22
4 #include <wr22/regex_parser/span/span.hpp>
5
6 // STL
7 #include <exception>
8 #include <stdexcept>
9 #include <string>
10
11 namespace wr22::regex_parser::parser::errors {
12
13 struct ParseError : public std::runtime_error {
14     using std::runtime_error::runtime_error;
15 };
16
17 class UnexpectedEnd : public ParseError {
18 public:
19     UnexpectedEnd(size_t position, std::string expected);
20
21     size_t position() const;
22     const std::string& expected() const;
23 private:
24     size_t m_position;
25     std::string m_expected;
26 };
27
28 class ExpectedEnd : public ParseError {
29 public:
30     ExpectedEnd(size_t position, char32_t char_got);
31
32     size_t position() const;
33     char32_t char_got() const;
34 private:
35     size_t m_position;
36     char32_t m_char_got;
37 };
38
39 class UnexpectedChar : public ParseError {
40 public:
41     UnexpectedChar(size_t position, char32_t char_got, std::string expected);
42
43     size_t position() const;
44     char32_t char_got() const;
45     const std::string& expected() const;
46 private:
47     size_t m_position;
48     char32_t m_char_got;
49     std::string m_expected;
50 };
51
52 class InvalidRange : public ParseError {
53 public:
54     InvalidRange(span::Span span, char32_t first, char32_t last);
55
56     span::Span span() const;
57     char32_t first() const;
58     char32_t last() const;
59 private:
60     span::Span m_span;
61     char32_t m_first;
62     char32_t m_last;
63 };
64 } // namespace wr22::regex_parser::parser::errors

```

7.3 include/wr22/regex_parser/parser/regex.hpp File Reference

```

#include <wr22/regex_parser/regex/part.hpp>
#include <string_view>

```


Namespaces

- namespace [wr22](#)
- namespace [wr22::regex_parser](#)
- namespace [wr22::regex_parser::parser](#)

Functions

- `regex::SpannedPart wr22::regex_parser::parser::parse_regex` (const std::u32string_view ®ex)
Parse a regular expression into its AST.

7.4 regex.hpp

[Go to the documentation of this file.](#)

```
1 #pragma once
2
3 // wr22
4 #include <wr22/regex_parser/regex/part.hpp>
5
6 // stl
7 #include <string_view>
8
9 namespace wr22::regex_parser::parser {
10
11     regex::SpannedPart parse\_regex(const std::u32string_view& regex);
12
13 } // namespace wr22::regex_parser::parser
```

7.5 include/wr22/regex_parser/regex/capture.hpp File Reference

```
#include <wr22/regex_parser/regex/named_capture_flavor.hpp>
#include <wr22/regex_parser/regex/adt.hpp>
#include <iosfwd>
#include <string>
#include <nlohmann/json.hpp>
```

Classes

- struct [wr22::regex_parser::regex::capture::None](#)
Denotes an non-capturing group.
- struct [wr22::regex_parser::regex::capture::Index](#)
Denotes a group captured by index.
- struct [wr22::regex_parser::regex::capture::Name](#)
Denotes a group captured by name.
- class [wr22::regex_parser::regex::Capture](#)
Group capture behavior.

Namespaces

- namespace [wr22](#)
- namespace [wr22::regex_parser](#)
- namespace [wr22::regex_parser::regex](#)
- namespace [wr22::regex_parser::regex::capture](#)

Typedefs

- using `wr22::regex_parser::regex::capture::Adt` = `utils::Adt<None, Index, Name>`

Functions

- void `wr22::regex_parser::regex::capture::to_json` (`nlohmann::json& j`, `const None& capture`)
- void `wr22::regex_parser::regex::capture::to_json` (`nlohmann::json& j`, `const Index& capture`)
- void `wr22::regex_parser::regex::capture::to_json` (`nlohmann::json& j`, `const Name& capture`)
- `std::ostream&` `wr22::regex_parser::regex::operator<<` (`std::ostream& out`, `const Capture& capture`)
- void `wr22::regex_parser::regex::to_json` (`nlohmann::json& j`, `const Capture& capture`)

7.6 capture.hpp

[Go to the documentation of this file.](#)

```

1 #pragma once
2
3 // wr22
4 #include <wr22/regex_parser/regex/named_capture_flavor.hpp>
5 #include <wr22/regex_parser/regex/adt.hpp>
6
7 // stl
8 #include <iosfwd>
9 #include <string>
10
11 // nlohmann
12 #include <nlohmann/json.hpp>
13
14 namespace wr22::regex_parser::regex {
15
16 class Capture;
17
18 namespace capture {
19     struct None {
20         explicit None() = default;
21         bool operator==(const None& rhs) const = default;
22         static constexpr const char* code_name = "none";
23     };
24     void to_json(nlohmann::json& j, const None& capture);
25
26     struct Index {
27         explicit Index() = default;
28         bool operator==(const Index& rhs) const = default;
29         static constexpr const char* code_name = "index";
30     };
31     void to_json(nlohmann::json& j, const Index& capture);
32
33     struct Name {
34         explicit Name(std::string name, NamedCaptureFlavor flavor);
35         std::string name;
36         NamedCaptureFlavor flavor;
37         bool operator==(const Name& rhs) const = default;
38         static constexpr const char* code_name = "name";
39     };
40     void to_json(nlohmann::json& j, const Name& capture);
41
42     using Adt = utils::Adt<None, Index, Name>;
43 } // namespace capture
44
45 //
46 class Capture : public capture::Adt {
47 public:
48     using capture::Adt::Adt;
49 };
50
51 std::ostream& operator<<(std::ostream& out, const Capture& capture);
52 void to_json(nlohmann::json& j, const Capture& capture);
53
54 } // namespace wr22::regex_parser::regex

```

7.7 include/wr22/regex_parser/regex/character_class_data.hpp File Reference

```
#include <wr22/regex_parser/regex/spanned_character_range.hpp>
#include <vector>
```

Classes

- struct [wr22::regex_parser::regex::CharacterClassData](#)
A character class representation: a list of character ranges plus some additional properties.

Namespaces

- namespace [wr22](#)
- namespace [wr22::regex_parser](#)
- namespace [wr22::regex_parser::regex](#)

7.8 character_class_data.hpp

[Go to the documentation of this file.](#)

```
1 #pragma once
2
3 // wr22
4 #include <wr22/regex_parser/regex/spanned_character_range.hpp>
5
6 // stl
7 #include <vector>
8
9 namespace wr22::regex_parser::regex {
10
12 struct CharacterClassData {
13     std::vector<SpannedCharacterRange> ranges;
14
15     bool inverted;
16
17     bool operator==(const CharacterClassData& rhs) const = default;
18 };
19
20 }
```

7.9 include/wr22/regex_parser/regex/character_range.hpp File Reference

```
#include <stdexcept>
#include <iosfwd>
#include <fmt/core.h>
#include <nlohmann/json.hpp>
```

Classes

- struct [wr22::regex_parser::regex::InvalidCharacterRange](#)
- class [wr22::regex_parser::regex::CharacterRange](#)
A non-empty character range, possibly containing only one character.

Namespaces

- namespace [wr22](#)
- namespace [wr22::regex_parser](#)
- namespace [wr22::regex_parser::regex](#)

Functions

- `std::ostream & wr22::regex_parser::regex::operator<< (std::ostream &out, const CharacterRange &range)`
- `void wr22::regex_parser::regex::to_json (nlohmann::json &j, const CharacterRange &range)`

7.10 character_range.hpp

[Go to the documentation of this file.](#)

```

1 #pragma once
2
3 // stl
4 #include <stdexcept>
5 #include <iosfwd>
6
7 // fmt
8 #include <fmt/core.h>
9
10 // nlohmann
11 #include <nlohmann/json.hpp>
12
13 namespace wr22::regex_parser::regex {
14
15 struct InvalidCharacterRange : public std::runtime_error {
16     explicit InvalidCharacterRange(char32_t first, char32_t last);
17
18     char32_t first;
19     char32_t last;
20 };
21
22 class CharacterRange {
23 public:
24     static CharacterRange from_endpoints(char32_t first, char32_t last);
25
26     static CharacterRange from_single_character(char32_t character) noexcept;
27
28     char32_t first() const noexcept;
29
30     char32_t last() const noexcept;
31
32     bool contains(char32_t character) const noexcept;
33
34     bool is_single_character() const noexcept;
35
36     bool operator==(const CharacterRange& rhs) const noexcept = default;
37
38 private:
39     explicit CharacterRange(char32_t first, char32_t last);
40
41     char32_t m_first;
42     char32_t m_last;
43 };
44
45 std::ostream& operator<<(std::ostream& out, const CharacterRange& range);
46 void to_json(nlohmann::json& j, const CharacterRange& range);
47
48 } // namespace wr22::regex_parser::regex

```

7.11 include/wr22/regex_parser/regex/named_capture_flavor.hpp File Reference

```

#include <iosfwd>
#include <nlohmann/json.hpp>

```

Namespaces

- namespace [wr22](#)
- namespace [wr22::regex_parser](#)
- namespace [wr22::regex_parser::regex](#)

Enumerations

- enum class [wr22::regex_parser::regex::NamedCaptureFlavor](#) { [wr22::regex_parser::regex::Apostrophes](#) , [wr22::regex_parser::regex::Angles](#) , [wr22::regex_parser::regex::AnglesWithP](#) }

The flavor (dialect) of a named group capture.

Functions

- `std::ostream & wr22::regex_parser::regex::operator<< (std::ostream &out, NamedCaptureFlavor flavor)`
- `void wr22::regex_parser::regex::to_json (nlohmann::json &j, NamedCaptureFlavor flavor)`

7.12 named_capture_flavor.hpp

[Go to the documentation of this file.](#)

```
1 #pragma once
2
3 // stl
4 #include <iosfwd>
5
6 // nlohmann
7 #include <nlohmann/json.hpp>
8
9 namespace wr22::regex_parser::regex {
10
11     enum class NamedCaptureFlavor
12     {
13         Apostrophes,
14         Angles,
15         AnglesWithP,
16     };
17
18     std::ostream& operator<<(std::ostream& out, NamedCaptureFlavor flavor);
19     void to_json(nlohmann::json& j, NamedCaptureFlavor flavor);
20
21 } // namespace wr22::regex_parser::regex
```

7.13 include/wr22/regex_parser/regex/part.hpp File Reference

```
#include <nlohmann/json_fwd.hpp>
#include <wr22/regex_parser/regex/capture.hpp>
#include <wr22/regex_parser/regex/character_class_data.hpp>
#include <wr22/regex_parser/span/span.hpp>
#include <wr22/regex_parser/utils/adt.hpp>
#include <wr22/regex_parser/utils/box.hpp>
#include <iosfwd>
#include <memory>
#include <vector>
#include <nlohmann/json.hpp>
```

Classes

- struct [wr22::regex_parser::regex::part::Empty](#)
An empty regex part.
- struct [wr22::regex_parser::regex::part::Literal](#)
A regex part that matches a single character literally.
- struct [wr22::regex_parser::regex::part::Alternatives](#)
A regex part with the list of alternatives to be matched.
- struct [wr22::regex_parser::regex::part::Sequence](#)
A regex part with the list of items to be matched one after another.
- struct [wr22::regex_parser::regex::part::Group](#)
A regex part that represents a group in parentheses.
- struct [wr22::regex_parser::regex::part::Optional](#)
A regex part specifying an optional quantifier $((expression)?)$.
- struct [wr22::regex_parser::regex::part::Plus](#)
A regex part specifying an "at least one" quantifier $((expression)+)$.
- struct [wr22::regex_parser::regex::part::Star](#)
A regex part specifying an "at least zero" quantifier $((expression)*)$.
- struct [wr22::regex_parser::regex::part::Wildcard](#)
A regex part specifying any single character $(.)$.
- struct [wr22::regex_parser::regex::part::CharacterClass](#)
A regex part specifying a character class (e.g. $[a-z_]$).
- class [wr22::regex_parser::regex::Part](#)
A part of a regular expression and its AST node type.
- class [wr22::regex_parser::regex::SpannedPart](#)
A version of [Part](#) including the span information (position in the input) of the root AST node (child nodes always contain it because they are represented as [SpannedParts](#) themselves).

Namespaces

- namespace [wr22](#)
- namespace [wr22::regex_parser](#)
- namespace [wr22::regex_parser::regex](#)
- namespace [wr22::regex_parser::regex::part](#)
The namespace with the variants of [Part](#).

Typedefs

- using [wr22::regex_parser::regex::part::Adt](#) = `utils::Adt< Empty, Literal, Alternatives, Sequence, Group, Optional, Plus, Star, Wildcard, CharacterClass >`

Functions

- void `wr22::regex_parser::regex::part::to_json` (nlohmann::json &j, const part::Empty &part)
- void `wr22::regex_parser::regex::part::to_json` (nlohmann::json &j, const part::Literal &part)
- void `wr22::regex_parser::regex::part::to_json` (nlohmann::json &j, const part::Alternatives &part)
- void `wr22::regex_parser::regex::part::to_json` (nlohmann::json &j, const part::Sequence &part)
- void `wr22::regex_parser::regex::part::to_json` (nlohmann::json &j, const part::Group &part)
- void `wr22::regex_parser::regex::part::to_json` (nlohmann::json &j, const part::Optional &part)
- void `wr22::regex_parser::regex::part::to_json` (nlohmann::json &j, const part::Plus &part)
- void `wr22::regex_parser::regex::part::to_json` (nlohmann::json &j, const part::Star &part)
- void `wr22::regex_parser::regex::part::to_json` (nlohmann::json &j, const part::Wildcard &part)
- void `wr22::regex_parser::regex::part::to_json` (nlohmann::json &j, const part::CharacterClass &part)
- void `wr22::regex_parser::regex::to_json` (nlohmann::json &j, const Part &part)
- void `wr22::regex_parser::regex::to_json` (nlohmann::json &j, const SpannedPart &part)
- std::ostream & `wr22::regex_parser::regex::operator<<` (std::ostream &out, const SpannedPart &part)

Convert a *SpannedPart* to a textual representation and write it to an *std::ostream*.

7.14 part.hpp

[Go to the documentation of this file.](#)

```

1 #pragma once
2
3 // wr22
4 #include <nlohmann/json_fwd.hpp>
5 #include <wr22/regex_parser/regex/capture.hpp>
6 #include <wr22/regex_parser/regex/character_class_data.hpp>
7 #include <wr22/regex_parser/span/span.hpp>
8 #include <wr22/regex_parser/utils/adt.hpp>
9 #include <wr22/regex_parser/utils/box.hpp>
10
11 // stl
12 #include <iosfwd>
13 #include <memory>
14 #include <vector>
15
16 // nlohmann
17 #include <nlohmann/json.hpp>
18
19 namespace wr22::regex_parser::regex {
20
21 // Forward declarations.
22 class Part;
23 class SpannedPart;
24
25 namespace part {
26     struct Empty {
27         explicit Empty() = default;
28         bool operator==(const Empty& rhs) const = default;
29         static constexpr const char* code_name = "empty";
30     };
31     void to_json(nlohmann::json& j, const Empty& part);
32
33     struct Literal {
34         explicit Literal(char32_t character);
35         bool operator==(const Literal& rhs) const = default;
36         static constexpr const char* code_name = "literal";
37         char32_t character;
38     };
39     void to_json(nlohmann::json& j, const Literal& part);
40
41     struct Alternatives {
42         explicit Alternatives(std::vector<SpannedPart> alternatives);
43         bool operator==(const Alternatives& rhs) const = default;
44         static constexpr const char* code_name = "alternatives";
45         std::vector<SpannedPart> alternatives;
46     };
47     void to_json(nlohmann::json& j, const Alternatives& part);
48
49     struct Sequence {
50         explicit Sequence(std::vector<SpannedPart> items);

```

```

78     bool operator==(const Sequence& rhs) const = default;
79     static constexpr const char* code_name = "sequence";
80
81     std::vector<SpannedPart> items;
82 };
83 void to_json(nlohmann::json& j, const Sequence& part);
84
85 struct Group {
86     explicit Group(Capture capture, SpannedPart inner);
87     bool operator==(const Group& rhs) const = default;
88     static constexpr const char* code_name = "group";
89
90     Capture capture;
91     utils::Box<SpannedPart> inner;
92 };
93 void to_json(nlohmann::json& j, const Group& part);
94
95 struct Optional {
96     explicit Optional(SpannedPart inner);
97     bool operator==(const Optional& rhs) const = default;
98     static constexpr const char* code_name = "optional";
99
100    utils::Box<SpannedPart> inner;
101 };
102 void to_json(nlohmann::json& j, const Optional& part);
103
104 struct Plus {
105     explicit Plus(SpannedPart inner);
106     bool operator==(const Plus& rhs) const = default;
107     static constexpr const char* code_name = "plus";
108
109    utils::Box<SpannedPart> inner;
110 };
111 void to_json(nlohmann::json& j, const Plus& part);
112
113 struct Star {
114     explicit Star(SpannedPart inner);
115     bool operator==(const Star& rhs) const = default;
116     static constexpr const char* code_name = "star";
117
118    utils::Box<SpannedPart> inner;
119 };
120 void to_json(nlohmann::json& j, const Star& part);
121
122 struct Wildcard {
123     explicit Wildcard() = default;
124     bool operator==(const Wildcard& rhs) const = default;
125     static constexpr const char* code_name = "wildcard";
126 };
127 void to_json(nlohmann::json& j, const Wildcard& part);
128
129 struct CharacterClass {
130     explicit CharacterClass(CharacterClassData data);
131     bool operator==(const CharacterClass& rhs) const = default;
132     static constexpr const char* code_name = "character_class";
133
134    CharacterClassData data;
135 };
136 void to_json(nlohmann::json& j, const CharacterClass& part);
137
138 using Adt = utils::
139     Adt<Empty, Literal, Alternatives, Sequence, Group, Optional, Plus, Star, Wildcard,
140     CharacterClass>;
141 } // namespace part
142
143 class Part : public part::Adt {
144 public:
145     using part::Adt::Adt;
146 };
147 void to_json(nlohmann::json& j, const Part& part);
148
149 class SpannedPart {
150 public:
151     explicit SpannedPart(Part part, span::Span span);
152
153     bool operator==(const SpannedPart& other) const = default;
154     bool operator!=(const SpannedPart& other) const = default;
155
156     const Part& part() const;
157     Part& part();
158
159     span::Span span() const;
160
161 private:
162     Part m_part;
163     span::Span m_span;
164 };

```



```

221 void to_json(nlohmann::json& j, const SpannedPart& part);
222
224 std::ostream& operator<<(std::ostream& out, const SpannedPart& part);
225
226 } // namespace wr22::regex_parser::regex

```

7.15 include/wr22/regex_parser/regex/spanned_character_range.hpp File Reference

```

#include <wr22/regex_parser/regex/character_range.hpp>
#include <wr22/regex_parser/span/span.hpp>
#include <iosfwd>
#include <nlohmann/json.hpp>

```

Classes

- struct `wr22::regex_parser::regex::SpannedCharacterRange`
A *CharacterRange* with its span.

Namespaces

- namespace `wr22`
- namespace `wr22::regex_parser`
- namespace `wr22::regex_parser::regex`

Functions

- `std::ostream & wr22::regex_parser::regex::operator<<` (`std::ostream &out`, `const SpannedCharacterRange &range`)
- `void wr22::regex_parser::regex::to_json` (`nlohmann::json& j`, `const SpannedCharacterRange &range`)

7.16 spanned_character_range.hpp

[Go to the documentation of this file.](#)

```

1 #pragma once
2
3 // wr22
4 #include <wr22/regex_parser/regex/character_range.hpp>
5 #include <wr22/regex_parser/span/span.hpp>
6
7 // stl
8 #include <iosfwd>
9
10 // nlohmann
11 #include <nlohmann/json.hpp>
12
13 namespace wr22::regex_parser::regex {
14
15     struct SpannedCharacterRange {
16         CharacterRange range;
17         span::Span span;
18
19         bool operator==(const SpannedCharacterRange& rhs) const = default;
20     };
21
22     std::ostream& operator<<(std::ostream& out, const SpannedCharacterRange& range);
23 void to_json(nlohmann::json& j, const SpannedCharacterRange& range);
24
25 } // namespace wr22::regex_parser::regex

```

7.17 include/wr22/regex_parser/span/span.hpp File Reference

```
#include <cstdint>
#include <stdexcept>
#include <ostream>
#include <nlohmann/json.hpp>
```

Classes

- struct [wr22::regex_parser::span::InvalidSpan](#)
The exception thrown on an attempt to construct an invalid span.
- class [wr22::regex_parser::span::Span](#)
Character position range in the input string.

Namespaces

- namespace [wr22](#)
- namespace [wr22::regex_parser](#)
- namespace [wr22::regex_parser::span](#)

Functions

- [std::ostream & wr22::regex_parser::span::operator<<](#) (std::ostream &out, Span span)
- void [wr22::regex_parser::span::to_json](#) (nlohmann::json &j, Span span)

7.18 span.hpp

[Go to the documentation of this file.](#)

```
1 #pragma once
2
3 // stl
4 #include <cstdint>
5 #include <stdexcept>
6 #include <ostream>
7
8 // nlohmann
9 #include <nlohmann/json.hpp>
10
11 namespace wr22::regex_parser::span {
12
13     struct InvalidSpan : public std::runtime_error {
14         InvalidSpan(size_t begin, size_t end);
15
16         size_t begin;
17         size_t end;
18     };
19
20     class Span {
21     public:
22         static Span make_empty(size_t position);
23
24         static Span make_single_position(size_t position);
25
26         static Span make_from_positions(size_t begin, size_t end);
27
28         static Span make_with_length(size_t begin, size_t length);
29
30         Span extend_right(size_t num_positions) const;
31
32         size_t length() const;
```

```

73
74     size_t begin() const;
75
76     size_t end() const;
77
78     bool operator==(const Span& other) const = default;
79     bool operator!=(const Span& other) const = default;
80
81 private:
82     explicit Span(size_t begin, size_t end);
83
84     size_t m_begin;
85     size_t m_end;
86 };
87
88 std::ostream& operator<<(std::ostream& out, Span span);
89 void to_json(nlohmann::json& j, Span span);
90
91 } // namespace wr22::regex_parser::span

```

7.19 include/wr22/regex_parser/utils/adt.hpp File Reference

```

#include <utility>
#include <variant>

```

Classes

- struct [wr22::regex_parser::utils::detail::adt::MultiCallable< Fs >](#)
- class [wr22::regex_parser::utils::Adt< Variants >](#)

A helper class that simplifies creation of algebraic data types.

Namespaces

- namespace [wr22](#)
- namespace [wr22::regex_parser](#)
- namespace [wr22::regex_parser::utils](#)
- namespace [wr22::regex_parser::utils::detail](#)
- namespace [wr22::regex_parser::utils::detail::adt](#)

Functions

- template<typename... Variants>
bool [wr22::regex_parser::utils::operator==](#) (const Adt< Variants... > &lhs, const Adt< Variants... > &rhs)
Compare two compatible ADTs for equality.
- template<typename... Variants>
bool [wr22::regex_parser::utils::operator!=](#) (const Adt< Variants... > &lhs, const Adt< Variants... > &rhs)
Compare two compatible ADTs for non-equality.

7.20 adt.hpp

[Go to the documentation of this file.](#)

```

1 #pragma once
2
3 // stl
4 #include <utility>
5 #include <variant>
6
7 namespace wr22::regex_parser::utils {
8
9 namespace detail::adt {
10     // https://en.cppreference.com/w/cpp/utility/variant/visit#Example provides a very similar
11     // example of C++ template black magic.
12     template <typename... Fs>
13     struct MultiCallable : public Fs... {
14         MultiCallable(Fs&&... fs) : Fs(fs)... {}
15         using Fs::operator()...;
16     };
17 } // namespace detail::adt
18
19 template <typename... Variants>
20 class Adt {
21 public:
22     using VariantType = std::variant<Variants...>;
23
24     template <typename V>
25     Adt(V variant) : m_variant(std::move(variant)) {}
26
27     template <typename... Fs>
28     decltype(auto) visit(Fs&&... visitors) const {
29         return std::visit(
30             detail::adt::MultiCallable<Fs...>(std::forward<Fs>(visitors)...),
31             m_variant);
32     }
33
34     template <typename... Fs>
35     decltype(auto) visit(Fs&&... visitors) {
36         return std::visit(
37             detail::adt::MultiCallable<Fs...>(std::forward<Fs>(visitors)...),
38             m_variant);
39     }
40
41     const VariantType& as_variant() const {
42         return m_variant;
43     }
44
45     VariantType& as_variant() {
46         return m_variant;
47     }
48
49 protected:
50     VariantType m_variant;
51 };
52
53 template <typename... Variants>
54 bool operator==(const Adt<Variants...>& lhs, const Adt<Variants...>& rhs) {
55     return lhs.as_variant() == rhs.as_variant();
56 }
57
58 template <typename... Variants>
59 bool operator!=(const Adt<Variants...>& lhs, const Adt<Variants...>& rhs) {
60     return !(lhs == rhs);
61 }
62
63 } // namespace wr22::regex_parser::utils

```

7.21 include/wr22/regex_parser/utils/box.hpp File Reference

```

#include <exception>
#include <memory>
#include <utility>

```

Classes

- struct `wr22::regex_parser::utils::BoxIsEmpty`
- class `wr22::regex_parser::utils::Box< T >`

A copyable and equality-comparable wrapper around `std::unique_ptr`.

Namespaces

- namespace `wr22`
- namespace `wr22::regex_parser`
- namespace `wr22::regex_parser::utils`

Functions

- template<typename T >
`wr22::regex_parser::utils::Box` (T &&value) -> Box< T >
Type deduction guideline for `Box` (value initialization).
- template<typename T >
`wr22::regex_parser::utils::Box` (std::unique_ptr< T > ptr) -> Box< T >
Type deduction guideline for `Box` (std::unique_ptr adoption).
- template<typename T, typename U >
bool `wr22::regex_parser::utils::operator==` (const Box< T > &lhs, const Box< U > &rhs)
- template<typename T, typename U >
bool `wr22::regex_parser::utils::operator!=` (const Box< T > &lhs, const Box< U > &rhs)

7.22 box.hpp

[Go to the documentation of this file.](#)

```
1 #pragma once
2
3 // stl
4 #include <exception>
5 #include <memory>
6 #include <utility>
7
8 namespace wr22::regex_parser::utils {
9
10 struct BoxIsEmpty : public std::exception {
11     const char* what() const noexcept override;
12 };
13
14 template <typename T>
15 class Box {
16 public:
17     explicit Box(T&& value) : m_ptr(std::make_unique<T>(std::forward<T>(value))) {}
18
19     explicit Box(std::unique_ptr<T> ptr) : m_ptr(std::move(ptr)) {}
20
21     template <typename Dummy = T>
22     Box(const Box& other) : m_ptr(std::make_unique<T>(*other)) {}
23
24     template <typename... Args>
25     static Box<T> construct_in_place(Args&&... args) {
26         return Box(std::make_unique<T>(std::forward<Args>(args)...));
27     }
28
29     const T& operator*() const {
30         if (m_ptr == nullptr) {
31             throw BoxIsEmpty{};
32         }
33         return *m_ptr;
34     }
35
36     T& operator*() {
```

```

82         if (m_ptr == nullptr) {
83             throw BoxIsEmpty{};
84         }
85         return *m_ptr;
86     }
87
88 private:
89     std::unique_ptr<T> m_ptr;
90 };
91
92 template <typename T>
93 Box(T&& value) -> Box<T>;
94
95 template <typename T>
96 Box(std::unique_ptr<T> ptr) -> Box<T>;
97
98 template <typename T, typename U>
99 bool operator==(const Box<T>& lhs, const Box<U>& rhs) {
100     return *lhs == *rhs;
101 }
102
103 template <typename T, typename U>
104 bool operator!=(const Box<T>& lhs, const Box<U>& rhs) {
105     return !(lhs == rhs);
106 }
107
108 }
109
110 } // namespace wr22::regex_parser::utils

```

7.23 src/parser/capture.cpp File Reference

```

#include <wr22/regex_parser/regex/capture.hpp>
#include <wr22/regex_parser/regex/named_capture_flavor.hpp>
#include <iterator>
#include <ostream>
#include <fmt/core.h>
#include <fmt/ostream.h>

```

Namespaces

- namespace [wr22](#)
- namespace [wr22::regex_parser](#)
- namespace [wr22::regex_parser::regex](#)
- namespace [wr22::regex_parser::regex::capture](#)

Functions

- [std::ostream & wr22::regex_parser::regex::operator<< \(std::ostream &out, const Capture &capture\)](#)
- [void wr22::regex_parser::regex::to_json \(nlohmann::json &j, const Capture &capture\)](#)
- [void wr22::regex_parser::regex::capture::to_json \(nlohmann::json &j, const None &capture\)](#)
- [void wr22::regex_parser::regex::capture::to_json \(nlohmann::json &j, const Index &capture\)](#)
- [void wr22::regex_parser::regex::capture::to_json \(nlohmann::json &j, const Name &capture\)](#)

7.24 src/parser/errors.cpp File Reference

```

#include <wr22/regex_parser/parser/errors.hpp>
#include <wr22/unicode/conversion.hpp>
#include <fmt/core.h>
#include <fmt/ostream.h>

```

Namespaces

- namespace [wr22](#)
- namespace [wr22::regex_parser](#)
- namespace [wr22::regex_parser::parser](#)
- namespace [wr22::regex_parser::parser::errors](#)

7.25 src/parser/regex.cpp File Reference

```
#include "wr22/regex_parser/span/span.hpp"
#include <wr22/regex_parser/parser/errors.hpp>
#include <wr22/regex_parser/parser/regex.hpp>
#include <wr22/regex_parser/regex/part.hpp>
#include <wr22/unicode/conversion.hpp>
#include <optional>
#include <stdexcept>
#include <string>
#include <vector>
```

Classes

- class [wr22::regex_parser::parser::Parser< Iter, Sentinel >](#)
A regex parser.

Namespaces

- namespace [wr22](#)
- namespace [wr22::regex_parser](#)
- namespace [wr22::regex_parser::parser](#)

Functions

- template<typename Iter , typename Sentinel >
[wr22::regex_parser::parser::Parser](#) (Iter begin, Sentinel end) -> Parser< Iter, Sentinel >
The type deduction guideline for [Parser](#).
- regex::SpannedPart [wr22::regex_parser::parser::parse_regex](#) (const std::u32string_view ®ex)
Parse a regular expression into its AST.

7.26 src/regex/character_range.cpp File Reference

```
#include <wr22/regex_parser/regex/character_range.hpp>
#include <wr22/unicode/conversion.hpp>
#include <ostream>
#include <fmt/core.h>
#include <fmt/ostream.h>
```

Namespaces

- namespace [wr22](#)
- namespace [wr22::regex_parser](#)
- namespace [wr22::regex_parser::regex](#)

Functions

- `std::ostream & wr22::regex_parser::regex::operator<< (std::ostream &out, const CharacterRange &range)`
- `void wr22::regex_parser::regex::to_json (nlohmann::json &j, const CharacterRange &range)`

7.27 src/regex/named_capture_flavor.cpp File Reference

```
#include <wr22/regex_parser/regex/named_capture_flavor.hpp>
#include <ostream>
```

Namespaces

- namespace [wr22](#)
- namespace [wr22::regex_parser](#)
- namespace [wr22::regex_parser::regex](#)

Functions

- `std::ostream & wr22::regex_parser::regex::operator<< (std::ostream &out, NamedCaptureFlavor flavor)`
- `void wr22::regex_parser::regex::to_json (nlohmann::json &j, NamedCaptureFlavor flavor)`

7.28 src/regex/part.cpp File Reference

```
#include <wr22/regex_parser/regex/part.hpp>
#include <wr22/regex_parser/span/span.hpp>
#include <wr22/unicode/conversion.hpp>
#include <iterator>
#include <ostream>
#include <fmt/core.h>
#include <fmt/ostream.h>
```

Namespaces

- namespace [wr22](#)
- namespace [wr22::regex_parser](#)
- namespace [wr22::regex_parser::regex](#)
- namespace [wr22::regex_parser::regex::part](#)

The namespace with the variants of [Part](#).

Functions

- `std::ostream & wr22::regex_parser::regex::operator<<` (`std::ostream &out`, `const SpannedPart &part`)
Convert a *SpannedPart* to a textual representation and write it to an *std::ostream*.
- `void wr22::regex_parser::regex::part::to_json` (`nlohmann::json &j`, `const part::Empty &part`)
- `void wr22::regex_parser::regex::part::to_json` (`nlohmann::json &j`, `const part::Literal &part`)
- `void wr22::regex_parser::regex::part::to_json` (`nlohmann::json &j`, `const part::Alternatives &part`)
- `void wr22::regex_parser::regex::part::to_json` (`nlohmann::json &j`, `const part::Sequence &part`)
- `void wr22::regex_parser::regex::part::to_json` (`nlohmann::json &j`, `const part::Group &part`)
- `void wr22::regex_parser::regex::part::to_json` (`nlohmann::json &j`, `const part::Optional &part`)
- `void wr22::regex_parser::regex::part::to_json` (`nlohmann::json &j`, `const part::Plus &part`)
- `void wr22::regex_parser::regex::part::to_json` (`nlohmann::json &j`, `const part::Star &part`)
- `void wr22::regex_parser::regex::part::to_json` (`nlohmann::json &j`, `const part::Wildcard &part`)
- `void wr22::regex_parser::regex::part::to_json` (`nlohmann::json &j`, `const part::CharacterClass &part`)
- `void wr22::regex_parser::regex::to_json` (`nlohmann::json &j`, `const Part &part`)
- `void wr22::regex_parser::regex::to_json` (`nlohmann::json &j`, `const SpannedPart &part`)

7.29 src/regex/spanned_character_range.cpp File Reference

```
#include <wr22/regex_parser/regex/spanned_character_range.hpp>
#include <ostream>
#include <fmt/core.h>
#include <fmt/ostream.h>
```

Namespaces

- namespace `wr22`
- namespace `wr22::regex_parser`
- namespace `wr22::regex_parser::regex`

Functions

- `std::ostream & wr22::regex_parser::regex::operator<<` (`std::ostream &out`, `const SpannedCharacterRange &range`)
- `void wr22::regex_parser::regex::to_json` (`nlohmann::json &j`, `const SpannedCharacterRange &range`)

7.30 src/span/span.cpp File Reference

```
#include <stdexcept>
#include <wr22/regex_parser/span/span.hpp>
#include <fmt/core.h>
#include <fmt/ostream.h>
```

Namespaces

- namespace `wr22`
- namespace `wr22::regex_parser`
- namespace `wr22::regex_parser::span`

Functions

- `std::ostream & wr22::regex_parser::span::operator<<` (`std::ostream &out`, `Span span`)
- `void wr22::regex_parser::span::to_json` (`nlohmann::json &j`, `Span span`)

7.31 src/utils/box.cpp File Reference

```
#include <wr22/regex_parser/utils/box.hpp>
```

Namespaces

- namespace `wr22`
- namespace `wr22::regex_parser`
- namespace `wr22::regex_parser::utils`

Index

Adt

- wr22::regex_parser::regex::capture, [15](#)
- wr22::regex_parser::regex::part, [17](#)
- wr22::regex_parser::utils::Adt< Variants >, [24](#)

Alternatives

- wr22::regex_parser::regex::part::Alternatives, [26](#)

alternatives

- wr22::regex_parser::regex::part::Alternatives, [27](#)

Angles

- wr22::regex_parser::regex, [13](#)

AnglesWithP

- wr22::regex_parser::regex, [13](#)

Apostrophes

- wr22::regex_parser::regex, [13](#)

as_variant

- wr22::regex_parser::utils::Adt< Variants >, [25](#)

begin

- wr22::regex_parser::span::InvalidSpan, [46](#)
- wr22::regex_parser::span::Span, [63](#)

Box

- wr22::regex_parser::utils, [20](#)
- wr22::regex_parser::utils::Box< T >, [28](#), [29](#)

capture

- wr22::regex_parser::regex::part::Group, [40](#)

char_got

- wr22::regex_parser::parser::errors::ExpectedEnd, [39](#)
- wr22::regex_parser::parser::errors::UnexpectedChar, [71](#)

character

- wr22::regex_parser::regex::part::Literal, [47](#)

CharacterClass

- wr22::regex_parser::regex::part::CharacterClass, [32](#)

code_name

- wr22::regex_parser::regex::capture::Index, [42](#)
- wr22::regex_parser::regex::capture::Name, [49](#)
- wr22::regex_parser::regex::capture::None, [51](#)
- wr22::regex_parser::regex::part::Alternatives, [27](#)
- wr22::regex_parser::regex::part::CharacterClass, [32](#)
- wr22::regex_parser::regex::part::Empty, [37](#)
- wr22::regex_parser::regex::part::Group, [40](#)
- wr22::regex_parser::regex::part::Literal, [47](#)
- wr22::regex_parser::regex::part::Optional, [52](#)
- wr22::regex_parser::regex::part::Plus, [61](#)
- wr22::regex_parser::regex::part::Sequence, [62](#)
- wr22::regex_parser::regex::part::Star, [70](#)

- wr22::regex_parser::regex::part::Wildcard, [74](#)

construct_in_place

- wr22::regex_parser::utils::Box< T >, [29](#)

contains

- wr22::regex_parser::regex::CharacterRange, [35](#)

data

- wr22::regex_parser::regex::part::CharacterClass, [33](#)

Empty

- wr22::regex_parser::regex::part::Empty, [37](#)

end

- wr22::regex_parser::span::InvalidSpan, [46](#)
- wr22::regex_parser::span::Span, [64](#)

expect_end

- wr22::regex_parser::parser::Parser< Iter, Sentinel >, [55](#)

expected

- wr22::regex_parser::parser::errors::UnexpectedChar, [71](#)
- wr22::regex_parser::parser::errors::UnexpectedEnd, [73](#)

ExpectedEnd

- wr22::regex_parser::parser::errors::ExpectedEnd, [38](#)

extend_right

- wr22::regex_parser::span::Span, [64](#)

- wr22::regex_parser::parser::errors::InvalidRange, [44](#)

- wr22::regex_parser::regex::CharacterRange, [35](#)

- wr22::regex_parser::regex::InvalidCharacterRange, [43](#)

flavor

- wr22::regex_parser::regex::capture::Name, [50](#)

from_endpoints

- wr22::regex_parser::regex::CharacterRange, [35](#)

from_single_character

- wr22::regex_parser::regex::CharacterRange, [35](#)

Group

- wr22::regex_parser::regex::part::Group, [40](#)

- include/wr22/regex_parser/parser/errors.hpp, [75](#), [76](#)

- include/wr22/regex_parser/parser/regex.hpp, [76](#), [77](#)

- include/wr22/regex_parser/regex/capture.hpp, [77](#), [78](#)

- include/wr22/regex_parser/regex/character_class_data.hpp, [79](#)

- include/wr22/regex_parser/regex/character_range.hpp, 79, 80
- include/wr22/regex_parser/regex/named_capture_flavor.hpp, 80, 81
- include/wr22/regex_parser/regex/part.hpp, 81, 83
- include/wr22/regex_parser/regex/spanned_character_range.hpp, 85
- include/wr22/regex_parser/span/span.hpp, 86
- include/wr22/regex_parser/utils/adt.hpp, 87, 88
- include/wr22/regex_parser/utils/box.hpp, 88, 89
- Index
 - wr22::regex_parser::regex::capture::Index, 41
- inner
 - wr22::regex_parser::regex::part::Group, 41
 - wr22::regex_parser::regex::part::Optional, 52
 - wr22::regex_parser::regex::part::Plus, 61
 - wr22::regex_parser::regex::part::Star, 70
- InvalidCharacterRange
 - wr22::regex_parser::regex::InvalidCharacterRange, 43
- InvalidRange
 - wr22::regex_parser::parser::errors::InvalidRange, 44
- InvalidSpan
 - wr22::regex_parser::span::InvalidSpan, 46
- inverted
 - wr22::regex_parser::regex::CharacterClassData, 34
- is_single_character
 - wr22::regex_parser::regex::CharacterRange, 36
- items
 - wr22::regex_parser::regex::part::Sequence, 62
- last
 - wr22::regex_parser::parser::errors::InvalidRange, 44
 - wr22::regex_parser::regex::CharacterRange, 36
 - wr22::regex_parser::regex::InvalidCharacterRange, 43
- length
 - wr22::regex_parser::span::Span, 64
- Literal
 - wr22::regex_parser::regex::part::Literal, 47
- m_variant
 - wr22::regex_parser::utils::Adt< Variants >, 26
- make_empty
 - wr22::regex_parser::span::Span, 64
- make_from_positions
 - wr22::regex_parser::span::Span, 64
- make_single_position
 - wr22::regex_parser::span::Span, 65
- make_with_length
 - wr22::regex_parser::span::Span, 65
- MultiCallable
 - wr22::regex_parser::utils::detail::adt::MultiCallable< Fs >, 48
- Name
 - wr22::regex_parser::regex::capture::Name, 49
 - name
 - wr22::regex_parser::regex::capture::Name, 50
 - NamedCaptureFlavor
 - wr22::regex_parser::regex, 11
 - None
 - wr22::regex_parser::regex::capture::None, 50
- operator!=
 - wr22::regex_parser::regex::SpannedPart, 68
 - wr22::regex_parser::span::Span, 65
 - wr22::regex_parser::utils, 21
- operator<<
 - wr22::regex_parser::regex, 13
 - wr22::regex_parser::span, 19
- operator*
 - wr22::regex_parser::utils::Box< T >, 29, 30
- operator==
 - wr22::regex_parser::regex::capture::Index, 42
 - wr22::regex_parser::regex::capture::Name, 49
 - wr22::regex_parser::regex::capture::None, 51
 - wr22::regex_parser::regex::CharacterClassData, 33
 - wr22::regex_parser::regex::CharacterRange, 36
 - wr22::regex_parser::regex::part::Alternatives, 27
 - wr22::regex_parser::regex::part::CharacterClass, 32
 - wr22::regex_parser::regex::part::Empty, 37
 - wr22::regex_parser::regex::part::Group, 40
 - wr22::regex_parser::regex::part::Literal, 47
 - wr22::regex_parser::regex::part::Optional, 52
 - wr22::regex_parser::regex::part::Plus, 60
 - wr22::regex_parser::regex::part::Sequence, 62
 - wr22::regex_parser::regex::part::Star, 69
 - wr22::regex_parser::regex::part::Wildcard, 74
 - wr22::regex_parser::regex::SpannedCharacterRange, 66
 - wr22::regex_parser::regex::SpannedPart, 68
 - wr22::regex_parser::span::Span, 66
 - wr22::regex_parser::utils, 21
- Optional
 - wr22::regex_parser::regex::part::Optional, 52
- parse_alternatives
 - wr22::regex_parser::parser::Parser< Iter, Sentinel >, 55
- parse_atom
 - wr22::regex_parser::parser::Parser< Iter, Sentinel >, 55
- parse_char_class
 - wr22::regex_parser::parser::Parser< Iter, Sentinel >, 56
- parse_char_literal
 - wr22::regex_parser::parser::Parser< Iter, Sentinel >, 56
- parse_group
 - wr22::regex_parser::parser::Parser< Iter, Sentinel >, 56
- parse_group_name

- wr22::regex_parser::parser::Parser< Iter, Sentinel >, 57
- parse_regex
 - wr22::regex_parser::parser, 10
 - wr22::regex_parser::parser::Parser< Iter, Sentinel >, 57
- parse_sequence
 - wr22::regex_parser::parser::Parser< Iter, Sentinel >, 57
- parse_sequence_or_empty
 - wr22::regex_parser::parser::Parser< Iter, Sentinel >, 58
- parse_wildcard
 - wr22::regex_parser::parser::Parser< Iter, Sentinel >, 58
- Parser
 - wr22::regex_parser::parser, 10
 - wr22::regex_parser::parser::Parser< Iter, Sentinel >, 54
- part
 - wr22::regex_parser::regex::SpannedPart, 68
- Plus
 - wr22::regex_parser::regex::part::Plus, 60
- position
 - wr22::regex_parser::parser::errors::ExpectedEnd, 39
 - wr22::regex_parser::parser::errors::UnexpectedChar, 71
 - wr22::regex_parser::parser::errors::UnexpectedEnd, 73
- range
 - wr22::regex_parser::regex::SpannedCharacterRange, 67
- ranges
 - wr22::regex_parser::regex::CharacterClassData, 34
- Sequence
 - wr22::regex_parser::regex::part::Sequence, 62
- span
 - wr22::regex_parser::parser::errors::InvalidRange, 45
 - wr22::regex_parser::regex::SpannedCharacterRange, 67
 - wr22::regex_parser::regex::SpannedPart, 68
- SpannedPart
 - wr22::regex_parser::regex::SpannedPart, 67
- src/parser/capture.cpp, 90
- src/parser/errors.cpp, 90
- src/parser/regex.cpp, 91
- src/regex/character_range.cpp, 91
- src/regex/named_capture_flavor.cpp, 92
- src/regex/part.cpp, 92
- src/regex/spanned_character_range.cpp, 93
- src/span/span.cpp, 93
- src/utils/box.cpp, 94
- Star
 - wr22::regex_parser::regex::part::Star, 69
- to_json
 - wr22::regex_parser::regex, 14
 - wr22::regex_parser::regex::capture, 15, 16
 - wr22::regex_parser::regex::part, 17–19
 - wr22::regex_parser::span, 19
- UnexpectedChar
 - wr22::regex_parser::parser::errors::UnexpectedChar, 71
- UnexpectedEnd
 - wr22::regex_parser::parser::errors::UnexpectedEnd, 72
- VariantType
 - wr22::regex_parser::utils::Adt< Variants >, 24
- visit
 - wr22::regex_parser::utils::Adt< Variants >, 25
- what
 - wr22::regex_parser::utils::BoxIsEmpty, 30
- Wildcard
 - wr22::regex_parser::regex::part::Wildcard, 74
- wr22, 9
- wr22::regex_parser, 9
- wr22::regex_parser::parser, 9
 - parse_regex, 10
 - Parser, 10
 - wr22::regex_parser::parser::errors, 10
 - wr22::regex_parser::parser::errors::ExpectedEnd, 38
 - char_got, 39
 - ExpectedEnd, 38
 - position, 39
 - wr22::regex_parser::parser::errors::InvalidRange, 43
 - first, 44
 - InvalidRange, 44
 - last, 44
 - span, 45
 - wr22::regex_parser::parser::errors::ParseError, 53
 - wr22::regex_parser::parser::errors::UnexpectedChar, 70
 - char_got, 71
 - expected, 71
 - position, 71
 - UnexpectedChar, 71
 - wr22::regex_parser::parser::errors::UnexpectedEnd, 72
 - expected, 73
 - position, 73
 - UnexpectedEnd, 72
- wr22::regex_parser::parser::Parser< Iter, Sentinel >, 53
 - expect_end, 55
 - parse_alternatives, 55
 - parse_atom, 55
 - parse_char_class, 56
 - parse_char_literal, 56
 - parse_group, 56
 - parse_group_name, 57
 - parse_regex, 57
 - parse_sequence, 57

- parse_sequence_or_empty, 58
 - parse_wildcard, 58
 - Parser, 54
- wr22::regex_parser::regex, 11
 - Angles, 13
 - AnglesWithP, 13
 - Apostrophes, 13
 - NamedCaptureFlavor, 11
 - operator<<, 13
 - to_json, 14
- wr22::regex_parser::regex::Capture, 31
- wr22::regex_parser::regex::capture, 15
 - Adt, 15
 - to_json, 15, 16
- wr22::regex_parser::regex::capture::Index, 41
 - code_name, 42
 - Index, 41
 - operator==, 42
- wr22::regex_parser::regex::capture::Name, 48
 - code_name, 49
 - flavor, 50
 - Name, 49
 - name, 50
 - operator==, 49
- wr22::regex_parser::regex::capture::None, 50
 - code_name, 51
 - None, 50
 - operator==, 51
- wr22::regex_parser::regex::CharacterClassData, 33
 - inverted, 34
 - operator==, 33
 - ranges, 34
- wr22::regex_parser::regex::CharacterRange, 34
 - contains, 35
 - first, 35
 - from_endpoints, 35
 - from_single_character, 35
 - is_single_character, 36
 - last, 36
 - operator==, 36
- wr22::regex_parser::regex::InvalidCharacterRange, 42
 - first, 43
 - InvalidCharacterRange, 43
 - last, 43
- wr22::regex_parser::regex::Part, 59
- wr22::regex_parser::regex::part, 16
 - Adt, 17
 - to_json, 17–19
- wr22::regex_parser::regex::part::Alternatives, 26
 - Alternatives, 26
 - alternatives, 27
 - code_name, 27
 - operator==, 27
- wr22::regex_parser::regex::part::CharacterClass, 31
 - CharacterClass, 32
 - code_name, 32
 - data, 33
 - operator==, 32
- wr22::regex_parser::regex::part::Empty, 36
 - code_name, 37
 - Empty, 37
 - operator==, 37
- wr22::regex_parser::regex::part::Group, 39
 - capture, 40
 - code_name, 40
 - Group, 40
 - inner, 41
 - operator==, 40
- wr22::regex_parser::regex::part::Literal, 46
 - character, 47
 - code_name, 47
 - Literal, 47
 - operator==, 47
- wr22::regex_parser::regex::part::Optional, 51
 - code_name, 52
 - inner, 52
 - operator==, 52
 - Optional, 52
- wr22::regex_parser::regex::part::Plus, 60
 - code_name, 61
 - inner, 61
 - operator==, 60
 - Plus, 60
- wr22::regex_parser::regex::part::Sequence, 61
 - code_name, 62
 - items, 62
 - operator==, 62
 - Sequence, 62
- wr22::regex_parser::regex::part::Star, 69
 - code_name, 70
 - inner, 70
 - operator==, 69
 - Star, 69
- wr22::regex_parser::regex::part::Wildcard, 73
 - code_name, 74
 - operator==, 74
 - Wildcard, 74
- wr22::regex_parser::regex::SpannedCharacterRange, 66
 - operator==, 66
 - range, 67
 - span, 67
- wr22::regex_parser::regex::SpannedPart, 67
 - operator!=, 68
 - operator==, 68
 - part, 68
 - span, 68
 - SpannedPart, 67
- wr22::regex_parser::span, 19
 - operator<<, 19
 - to_json, 19
- wr22::regex_parser::span::InvalidSpan, 45
 - begin, 46
 - end, 46
 - InvalidSpan, 46
- wr22::regex_parser::span::Span, 63

- begin, [63](#)
- end, [64](#)
- extend_right, [64](#)
- length, [64](#)
- make_empty, [64](#)
- make_from_positions, [64](#)
- make_single_position, [65](#)
- make_with_length, [65](#)
- operator!=, [65](#)
- operator==, [66](#)
- wr22::regex_parser::utils, [20](#)
 - Box, [20](#)
 - operator!=, [21](#)
 - operator==, [21](#)
- wr22::regex_parser::utils::Adt< Variants >, [23](#)
 - Adt, [24](#)
 - as_variant, [25](#)
 - m_variant, [26](#)
 - VariantType, [24](#)
 - visit, [25](#)
- wr22::regex_parser::utils::Box< T >, [27](#)
 - Box, [28](#), [29](#)
 - construct_in_place, [29](#)
 - operator*, [29](#), [30](#)
- wr22::regex_parser::utils::BoxIsEmpty, [30](#)
 - what, [30](#)
- wr22::regex_parser::utils::detail, [22](#)
- wr22::regex_parser::utils::detail::adt, [22](#)
- wr22::regex_parser::utils::detail::adt::MultiCallable< Fs
>, [48](#)
 - MultiCallable, [48](#)