

## Writing Regexprs 2021-22 / Regex Parser

Generated by Doxygen 1.9.3



<b>1 Namespace Index</b>	<b>1</b>
1.1 Namespace List	1
<b>2 Hierarchical Index</b>	<b>3</b>
2.1 Class Hierarchy	3
<b>3 Class Index</b>	<b>5</b>
3.1 Class List	5
<b>4 File Index</b>	<b>7</b>
4.1 File List	7
<b>5 Namespace Documentation</b>	<b>9</b>
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	12
5.5.1.1 NamedCaptureFlavor	12
5.5.2 Function Documentation	12
5.5.2.1 operator<<() [1/5]	12
5.5.2.2 operator<<() [2/5]	12
5.5.2.3 operator<<() [3/5]	12
5.5.2.4 operator<<() [4/5]	13
5.5.2.5 operator<<() [5/5]	13
5.5.2.6 to_json() [1/6]	13
5.5.2.7 to_json() [2/6]	13
5.5.2.8 to_json() [3/6]	13
5.5.2.9 to_json() [4/6]	13
5.5.2.10 to_json() [5/6]	14
5.5.2.11 to_json() [6/6]	14
5.6 wr22::regex_parser::regex::capture Namespace Reference	14
5.6.1 Typedef Documentation	14
5.6.1.1 Adt	14
5.6.2 Function Documentation	15
5.6.2.1 to_json() [1/3]	15
5.6.2.2 to_json() [2/3]	15
5.6.2.3 to_json() [3/3]	15
5.7 wr22::regex_parser::regex::part Namespace Reference	15
5.7.1 Detailed Description	16

5.7.2 Typedef Documentation	16
5.7.2.1 Adt	16
5.7.3 Function Documentation	16
5.7.3.1 to_json() [1/10]	16
5.7.3.2 to_json() [2/10]	17
5.7.3.3 to_json() [3/10]	17
5.7.3.4 to_json() [4/10]	17
5.7.3.5 to_json() [5/10]	17
5.7.3.6 to_json() [6/10]	17
5.7.3.7 to_json() [7/10]	17
5.7.3.8 to_json() [8/10]	18
5.7.3.9 to_json() [9/10]	18
5.7.3.10 to_json() [10/10]	18
5.8 wr22::regex_parser::span Namespace Reference	18
5.8.1 Function Documentation	18
5.8.1.1 operator<<()	18
5.8.1.2 to_json()	18
<b>6 Class Documentation</b>	<b>19</b>
6.1 wr22::regex_parser::regex::part::Alternatives Struct Reference	19
6.1.1 Detailed Description	19
6.1.2 Constructor & Destructor Documentation	20
6.1.2.1 Alternatives()	20
6.1.3 Member Function Documentation	20
6.1.3.1 operator==(())	20
6.1.4 Member Data Documentation	20
6.1.4.1 alternatives	20
6.1.4.2 code_name	20
6.2 wr22::regex_parser::regex::Capture Class Reference	21
6.2.1 Detailed Description	21
6.3 wr22::regex_parser::regex::part::CharacterClass Struct Reference	21
6.3.1 Detailed Description	22
6.3.2 Constructor & Destructor Documentation	22
6.3.2.1 CharacterClass()	22
6.3.3 Member Function Documentation	22
6.3.3.1 operator==(())	22
6.3.4 Member Data Documentation	22
6.3.4.1 code_name	22
6.3.4.2 data	22
6.4 wr22::regex_parser::regex::CharacterClassData Struct Reference	23
6.4.1 Detailed Description	23
6.4.2 Member Function Documentation	23

6.4.2.1 operator==()	23
6.4.3 Member Data Documentation	23
6.4.3.1 inverted	23
6.4.3.2 ranges	24
6.5 wr22::regex_parser::regex::CharacterRange Class Reference	24
6.5.1 Detailed Description	24
6.5.2 Member Function Documentation	24
6.5.2.1 contains()	25
6.5.2.2 first()	25
6.5.2.3 from_endpoints()	25
6.5.2.4 from_single_character()	25
6.5.2.5 is_single_character()	25
6.5.2.6 last()	26
6.5.2.7 operator==()	26
6.6 wr22::regex_parser::regex::part::Empty Struct Reference	26
6.6.1 Detailed Description	26
6.6.2 Constructor & Destructor Documentation	26
6.6.2.1 Empty()	27
6.6.3 Member Function Documentation	27
6.6.3.1 operator==()	27
6.6.4 Member Data Documentation	27
6.6.4.1 code_name	27
6.7 wr22::regex_parser::parser::errors::ExpectedEnd Class Reference	27
6.7.1 Detailed Description	28
6.7.2 Constructor & Destructor Documentation	28
6.7.2.1 ExpectedEnd()	28
6.7.3 Member Function Documentation	28
6.7.3.1 char_got()	28
6.7.3.2 position()	29
6.8 wr22::regex_parser::regex::part::Group Struct Reference	29
6.8.1 Detailed Description	29
6.8.2 Constructor & Destructor Documentation	30
6.8.2.1 Group()	30
6.8.3 Member Function Documentation	30
6.8.3.1 operator==()	30
6.8.4 Member Data Documentation	30
6.8.4.1 capture	30
6.8.4.2 code_name	30
6.8.4.3 inner	31
6.9 wr22::regex_parser::regex::capture::Index Struct Reference	31
6.9.1 Detailed Description	31
6.9.2 Constructor & Destructor Documentation	31

6.9.2.1 Index()	31
6.9.3 Member Function Documentation	31
6.9.3.1 operator==()	32
6.9.4 Member Data Documentation	32
6.9.4.1 code_name	32
6.10 wr22::regex_parser::regex::InvalidCharacterRange Struct Reference	32
6.10.1 Constructor & Destructor Documentation	32
6.10.1.1 InvalidCharacterRange()	33
6.10.2 Member Data Documentation	33
6.10.2.1 first	33
6.10.2.2 last	33
6.11 wr22::regex_parser::parser::errors::InvalidRange Class Reference	33
6.11.1 Detailed Description	34
6.11.2 Constructor & Destructor Documentation	34
6.11.2.1 InvalidRange()	34
6.11.3 Member Function Documentation	34
6.11.3.1 first()	34
6.11.3.2 last()	35
6.11.3.3 span()	35
6.12 wr22::regex_parser::span::InvalidSpan Struct Reference	35
6.12.1 Detailed Description	35
6.12.2 Constructor & Destructor Documentation	36
6.12.2.1 InvalidSpan()	36
6.12.3 Member Data Documentation	36
6.12.3.1 begin	36
6.12.3.2 end	36
6.13 wr22::regex_parser::regex::part::Literal Struct Reference	36
6.13.1 Detailed Description	37
6.13.2 Constructor & Destructor Documentation	37
6.13.2.1 Literal()	37
6.13.3 Member Function Documentation	37
6.13.3.1 operator==()	37
6.13.4 Member Data Documentation	37
6.13.4.1 character	37
6.13.4.2 code_name	38
6.14 wr22::regex_parser::regex::capture::Name Struct Reference	38
6.14.1 Detailed Description	38
6.14.2 Constructor & Destructor Documentation	38
6.14.2.1 Name()	38
6.14.3 Member Function Documentation	39
6.14.3.1 operator==()	39
6.14.4 Member Data Documentation	39

6.14.4.1 code_name	39
6.14.4.2 flavor	39
6.14.4.3 name	39
6.15 wr22::regex_parser::regex::capture::None Struct Reference	39
6.15.1 Detailed Description	40
6.15.2 Constructor & Destructor Documentation	40
6.15.2.1 None()	40
6.15.3 Member Function Documentation	40
6.15.3.1 operator==( )	40
6.15.4 Member Data Documentation	40
6.15.4.1 code_name	40
6.16 wr22::regex_parser::regex::part::Optional Struct Reference	40
6.16.1 Detailed Description	41
6.16.2 Constructor & Destructor Documentation	41
6.16.2.1 Optional()	41
6.16.3 Member Function Documentation	41
6.16.3.1 operator==( )	41
6.16.4 Member Data Documentation	41
6.16.4.1 code_name	42
6.16.4.2 inner	42
6.17 wr22::regex_parser::parser::errors::ParseError Struct Reference	42
6.17.1 Detailed Description	42
6.18 wr22::regex_parser::parser::Parser< Iter, Sentinel > Class Template Reference	42
6.18.1 Detailed Description	43
6.18.2 Constructor & Destructor Documentation	43
6.18.2.1 Parser()	43
6.18.3 Member Function Documentation	44
6.18.3.1 expect_end()	44
6.18.3.2 parse_alternatives()	44
6.18.3.3 parse_atom()	44
6.18.3.4 parse_char_class()	45
6.18.3.5 parse_char_literal()	45
6.18.3.6 parse_group()	45
6.18.3.7 parse_group_name()	46
6.18.3.8 parse_regex()	46
6.18.3.9 parse_sequence()	47
6.18.3.10 parse_sequence_or_empty()	47
6.18.3.11 parse_wildcard()	47
6.19 wr22::regex_parser::regex::Part Class Reference	48
6.19.1 Detailed Description	48
6.20 wr22::regex_parser::regex::part::Plus Struct Reference	49
6.20.1 Detailed Description	49

6.20.2 Constructor & Destructor Documentation	49
6.20.2.1 Plus()	49
6.20.3 Member Function Documentation	49
6.20.3.1 operator==( )	49
6.20.4 Member Data Documentation	50
6.20.4.1 code_name	50
6.20.4.2 inner	50
6.21 wr22::regex_parser::regex::part::Sequence Struct Reference	50
6.21.1 Detailed Description	51
6.21.2 Constructor & Destructor Documentation	51
6.21.2.1 Sequence()	51
6.21.3 Member Function Documentation	51
6.21.3.1 operator==( )	51
6.21.4 Member Data Documentation	51
6.21.4.1 code_name	51
6.21.4.2 items	51
6.22 wr22::regex_parser::span::Span Class Reference	52
6.22.1 Detailed Description	52
6.22.2 Member Function Documentation	52
6.22.2.1 begin()	53
6.22.2.2 end()	53
6.22.2.3 extend_right()	53
6.22.2.4 length()	53
6.22.2.5 make_empty()	53
6.22.2.6 make_from_positions()	54
6.22.2.7 make_single_position()	54
6.22.2.8 make_with_length()	54
6.22.2.9 operator!=( )	55
6.22.2.10 operator==( )	55
6.23 wr22::regex_parser::regex::SpannedCharacterRange Struct Reference	55
6.23.1 Detailed Description	55
6.23.2 Member Function Documentation	55
6.23.2.1 operator==( )	55
6.23.3 Member Data Documentation	56
6.23.3.1 range	56
6.23.3.2 span	56
6.24 wr22::regex_parser::regex::SpannedPart Class Reference	56
6.24.1 Detailed Description	56
6.24.2 Constructor & Destructor Documentation	56
6.24.2.1 SpannedPart()	57
6.24.3 Member Function Documentation	57
6.24.3.1 operator!=( )	57



6.24.3.2 operator==()	57
6.24.3.3 part() [1/2]	57
6.24.3.4 part() [2/2]	57
6.24.3.5 span()	57
6.25 wr22::regex_parser::regex::part::Star Struct Reference	58
6.25.1 Detailed Description	58
6.25.2 Constructor & Destructor Documentation	58
6.25.2.1 Star()	58
6.25.3 Member Function Documentation	58
6.25.3.1 operator==()	58
6.25.4 Member Data Documentation	59
6.25.4.1 code_name	59
6.25.4.2 inner	59
6.26 wr22::regex_parser::parser::errors::TooStronglyNested Class Reference	59
6.26.1 Detailed Description	59
6.26.2 Constructor & Destructor Documentation	60
6.26.2.1 TooStronglyNested()	60
6.27 wr22::regex_parser::parser::errors::UnexpectedChar Class Reference	60
6.27.1 Detailed Description	61
6.27.2 Constructor & Destructor Documentation	61
6.27.2.1 UnexpectedChar()	61
6.27.3 Member Function Documentation	61
6.27.3.1 char_got()	61
6.27.3.2 expected()	61
6.27.3.3 needs_closing()	62
6.27.3.4 position()	62
6.28 wr22::regex_parser::parser::errors::UnexpectedEnd Class Reference	62
6.28.1 Detailed Description	63
6.28.2 Constructor & Destructor Documentation	63
6.28.2.1 UnexpectedEnd()	63
6.28.3 Member Function Documentation	63
6.28.3.1 expected()	63
6.28.3.2 needs_closing()	63
6.28.3.3 position()	64
6.29 wr22::regex_parser::regex::part::Wildcard Struct Reference	64
6.29.1 Detailed Description	64
6.29.2 Constructor & Destructor Documentation	64
6.29.2.1 Wildcard()	64
6.29.3 Member Function Documentation	64
6.29.3.1 operator==()	65
6.29.4 Member Data Documentation	65
6.29.4.1 code_name	65

<b>7 File Documentation</b>	<b>67</b>
7.1 include/wr22/regex_parser/parser/errors.hpp File Reference . . . . .	67
7.2 errors.hpp . . . . .	68
7.3 include/wr22/regex_parser/parser/regex.hpp File Reference . . . . .	69
7.4 regex.hpp . . . . .	69
7.5 include/wr22/regex_parser/regex/capture.hpp File Reference . . . . .	69
7.6 capture.hpp . . . . .	70
7.7 include/wr22/regex_parser/regex/character_class_data.hpp File Reference . . . . .	71
7.8 character_class_data.hpp . . . . .	71
7.9 include/wr22/regex_parser/regex/character_range.hpp File Reference . . . . .	71
7.10 character_range.hpp . . . . .	72
7.11 include/wr22/regex_parser/regex/named_capture_flavor.hpp File Reference . . . . .	73
7.12 named_capture_flavor.hpp . . . . .	73
7.13 include/wr22/regex_parser/regex/part.hpp File Reference . . . . .	74
7.14 part.hpp . . . . .	75
7.15 include/wr22/regex_parser/regex/spanned_character_range.hpp File Reference . . . . .	77
7.16 spanned_character_range.hpp . . . . .	77
7.17 include/wr22/regex_parser/span/span.hpp File Reference . . . . .	78
7.18 span.hpp . . . . .	78
7.19 src/parser/capture.cpp File Reference . . . . .	79
7.20 src/parser/errors.cpp File Reference . . . . .	80
7.21 src/parser/regex.cpp File Reference . . . . .	80
7.22 src/regex/character_range.cpp File Reference . . . . .	81
7.23 src/regex/named_capture_flavor.cpp File Reference . . . . .	81
7.24 src/regex/part.cpp File Reference . . . . .	81
7.25 src/regex/spanned_character_range.cpp File Reference . . . . .	82
7.26 src/span/span.cpp File Reference . . . . .	83
<b>Index</b>	<b>85</b>

# Chapter 1

## Namespace Index

### 1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">wr22</a> . . . . .	9
<a href="#">wr22::regex_parser</a> . . . . .	9
<a href="#">wr22::regex_parser::parser</a> . . . . .	9
<a href="#">wr22::regex_parser::parser::errors</a> . . . . .	10
<a href="#">wr22::regex_parser::regex</a> . . . . .	11
<a href="#">wr22::regex_parser::regex::capture</a> . . . . .	14
<a href="#">wr22::regex_parser::regex::part</a> The namespace with the variants of <a href="#">Part</a> . . . . .	15
<a href="#">wr22::regex_parser::span</a> . . . . .	18



## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

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

capture::Adt	
wr22::regex_parser::regex::Capture . . . . .	21
part::Adt	
wr22::regex_parser::regex::Part . . . . .	48
wr22::regex_parser::regex::part::Alternatives . . . . .	19
wr22::regex_parser::regex::part::CharacterClass . . . . .	21
wr22::regex_parser::regex::CharacterClassData . . . . .	23
wr22::regex_parser::regex::CharacterRange . . . . .	24
wr22::regex_parser::regex::part::Empty . . . . .	26
wr22::regex_parser::regex::part::Group . . . . .	29
wr22::regex_parser::regex::capture::Index . . . . .	31
wr22::regex_parser::regex::part::Literal . . . . .	36
wr22::regex_parser::regex::capture::Name . . . . .	38
wr22::regex_parser::regex::capture::None . . . . .	39
wr22::regex_parser::regex::part::Optional . . . . .	40
wr22::regex_parser::parser::Parser< Iter, Sentinel > . . . . .	42
wr22::regex_parser::regex::part::Plus . . . . .	49
std::runtime_error	
wr22::regex_parser::parser::errors::ParseError . . . . .	42
wr22::regex_parser::parser::errors::ExpectedEnd . . . . .	27
wr22::regex_parser::parser::errors::InvalidRange . . . . .	33
wr22::regex_parser::parser::errors::TooStronglyNested . . . . .	59
wr22::regex_parser::parser::errors::UnexpectedChar . . . . .	60
wr22::regex_parser::parser::errors::UnexpectedEnd . . . . .	62
wr22::regex_parser::regex::InvalidCharacterRange . . . . .	32
wr22::regex_parser::span::InvalidSpan . . . . .	35
wr22::regex_parser::regex::part::Sequence . . . . .	50
wr22::regex_parser::span::Span . . . . .	52
wr22::regex_parser::regex::SpannedCharacterRange . . . . .	55
wr22::regex_parser::regex::SpannedPart . . . . .	56
wr22::regex_parser::regex::part::Star . . . . .	58
wr22::regex_parser::regex::part::Wildcard . . . . .	64



## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">wr22::regex_parser::regex::part::Alternatives</a>	19
A regex part with the list of alternatives to be matched . . . . .	
<a href="#">wr22::regex_parser::regex::Capture</a>	21
Group capture behavior . . . . .	
<a href="#">wr22::regex_parser::regex::part::CharacterClass</a>	21
A regex part specifying a character class (e.g. [a-z_]) . . . . .	
<a href="#">wr22::regex_parser::regex::CharacterClassData</a>	23
A character class representation: a list of character ranges plus some additional properties . . .	
<a href="#">wr22::regex_parser::regex::CharacterRange</a>	24
A non-empty character range, possibly containing only one character . . . . .	
<a href="#">wr22::regex_parser::regex::part::Empty</a>	26
An empty regex part . . . . .	
<a href="#">wr22::regex_parser::parser::errors::ExpectedEnd</a>	27
The error when the parser expected the input to end, but it did not . . . . .	
<a href="#">wr22::regex_parser::regex::part::Group</a>	29
A regex part that represents a group in parentheses . . . . .	
<a href="#">wr22::regex_parser::regex::capture::Index</a>	31
Denotes a group captured by index . . . . .	
<a href="#">wr22::regex_parser::regex::InvalidCharacterRange</a>	32
<a href="#">wr22::regex_parser::parser::errors::InvalidRange</a>	33
The error indicating that a character range in a character class is invalid . . . . .	
<a href="#">wr22::regex_parser::span::InvalidSpan</a>	35
The exception thrown on an attempt to construct an invalid span . . . . .	
<a href="#">wr22::regex_parser::regex::part::Literal</a>	36
An regex part that matches a single character literally . . . . .	
<a href="#">wr22::regex_parser::regex::capture::Name</a>	38
Denotes a group captured by name . . . . .	
<a href="#">wr22::regex_parser::regex::capture::None</a>	39
Denotes an non-capturing group . . . . .	
<a href="#">wr22::regex_parser::regex::part::Optional</a>	40
A regex part specifying an optional quantifier ((expression)?) . . . . .	
<a href="#">wr22::regex_parser::parser::errors::ParseError</a>	42
The base class for parse errors . . . . .	
<a href="#">wr22::regex_parser::parser::Parser&lt; Iter, Sentinel &gt;</a>	42
A regex parser . . . . .	

<a href="#">wr22::regex_parser::regex::Part</a>	
A part of a regular expression and its AST node type . . . . .	48
<a href="#">wr22::regex_parser::regex::part::Plus</a>	
A regex part specifying an "at least one" quantifier ((expression)+) . . . . .	49
<a href="#">wr22::regex_parser::regex::part::Sequence</a>	
A regex part with the list of items to be matched one after another . . . . .	50
<a href="#">wr22::regex_parser::span::Span</a>	
Character position range in the input string . . . . .	52
<a href="#">wr22::regex_parser::regex::SpannedCharacterRange</a>	
A <a href="#">CharacterRange</a> with its span . . . . .	55
<a href="#">wr22::regex_parser::regex::SpannedPart</a>	
A version of <a href="#">Part</a> including the span information (position in the input) of the root AST node (child nodes always contain it because they are represented as <a href="#">SpannedPart</a> s themselves)	56
<a href="#">wr22::regex_parser::regex::part::Star</a>	
A regex part specifying an "at least zero" quantifier ((expression)*) . . . . .	58
<a href="#">wr22::regex_parser::parser::errors::TooStronglyNested</a>	
The error signalling that the regular expression has too many levels of nesting to be parsed . .	59
<a href="#">wr22::regex_parser::parser::errors::UnexpectedChar</a>	
The error when the parser got a character it didn't expect at the current position . . . . .	60
<a href="#">wr22::regex_parser::parser::errors::UnexpectedEnd</a>	
The error when the parser hit the end of the input earlier than it expected . . . . .	62
<a href="#">wr22::regex_parser::regex::part::Wildcard</a>	
A regex part specifying any single character (.) . . . . .	64



## 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 . . . . .	67
include/wr22/regex_parser/parser/regex.hpp . . . . .	69
include/wr22/regex_parser/regex/capture.hpp . . . . .	69
include/wr22/regex_parser/regex/character_class_data.hpp . . . . .	71
include/wr22/regex_parser/regex/character_range.hpp . . . . .	71
include/wr22/regex_parser/regex/named_capture_flavor.hpp . . . . .	73
include/wr22/regex_parser/regex/part.hpp . . . . .	74
include/wr22/regex_parser/regex/spanned_character_range.hpp . . . . .	77
include/wr22/regex_parser/span/span.hpp . . . . .	78
src/parser/capture.cpp . . . . .	79
src/parser/errors.cpp . . . . .	80
src/parser/regex.cpp . . . . .	80
src/regex/character_range.cpp . . . . .	81
src/regex/named_capture_flavor.cpp . . . . .	81
src/regex/part.cpp . . . . .	81
src/regex/spanned_character_range.cpp . . . . .	82
src/span/span.cpp . . . . .	83



## 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](#)

### 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 &regex)  
*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

<a href="#">errors::ParseError</a>	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 [TooStronglyNested](#)  
*The error signalling that the regular expression has too many levels of nesting to be parsed.*
- 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 [SpannedPart](#)s 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 ( <code>? 'name' contents</code> ). Mostly used in C# and other .NET-oriented languages, although can also be found in certain versions Perl, Boost and elsewhere.
Angles	The flavor ( <code>?&lt;name&gt;contents</code> ). Mostly used in C# and other .NET-oriented languages, although can also be found in certain versions Perl, Boost and elsewhere.
AnglesWithP	The flavor ( <code>?P&lt;name&gt;contents</code> ). 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](#)  
*An 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 )
```

## Chapter 6

# Class Documentation

### 6.1 `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"`

#### 6.1.1 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`.

## 6.1.2 Constructor & Destructor Documentation

### 6.1.2.1 Alternatives()

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

## 6.1.3 Member Function Documentation

### 6.1.3.1 operator==( )

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

## 6.1.4 Member Data Documentation

### 6.1.4.1 alternatives

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

The list of the alternatives.

### 6.1.4.2 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:

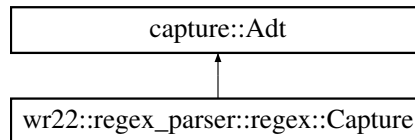
- include/wr22/regex\_parser/regex/part.hpp
- src/regex/part.cpp

## 6.2 wr22::regex\_parser::regex::Capture Class Reference

Group capture behavior.

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

Inheritance diagram for wr22::regex\_parser::regex::Capture:



### 6.2.1 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`

## 6.3 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"

### 6.3.1 Detailed Description

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

### 6.3.2 Constructor & Destructor Documentation

#### 6.3.2.1 CharacterClass()

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

### 6.3.3 Member Function Documentation

#### 6.3.3.1 operator==( )

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

### 6.3.4 Member Data Documentation

#### 6.3.4.1 code\_name

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

#### 6.3.4.2 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



## 6.4 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.*

#### 6.4.1 Detailed Description

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

#### 6.4.2 Member Function Documentation

##### 6.4.2.1 `operator==()`

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

#### 6.4.3 Member Data Documentation

##### 6.4.3.1 `inverted`

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

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

### 6.4.3.2 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`

## 6.5 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.*

### 6.5.1 Detailed Description

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

### 6.5.2 Member Function Documentation

### 6.5.2.1 contains()

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

Check if this range contains a given character.

### 6.5.2.2 first()

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

Get the left range bound (inclusive).

### 6.5.2.3 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 &lt; first</code> .
--------------------------------------	-----------------------------------

### 6.5.2.4 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.

### 6.5.2.5 is\_single\_character()

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

Check if this range contains exactly one character.

#### 6.5.2.6 last()

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

Get the right range bound (inclusive).

#### 6.5.2.7 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

## 6.6 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"

#### 6.6.1 Detailed Description

An empty regex part.

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

#### 6.6.2 Constructor & Destructor Documentation

### 6.6.2.1 Empty()

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

## 6.6.3 Member Function Documentation

### 6.6.3.1 operator==(

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

## 6.6.4 Member Data Documentation

### 6.6.4.1 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:

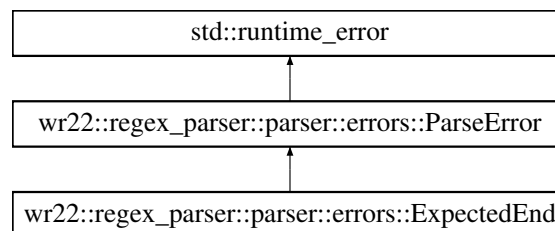
- include/wr22/regex\_parser/regex/part.hpp

## 6.7 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.*

### 6.7.1 Detailed Description

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

### 6.7.2 Constructor & Destructor Documentation

#### 6.7.2.1 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.

### 6.7.3 Member Function Documentation

#### 6.7.3.1 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.

### 6.7.3.2 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

## 6.8 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"

### 6.8.1 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.

## 6.8.2 Constructor & Destructor Documentation

### 6.8.2.1 Group()

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

Convenience constructor.

## 6.8.3 Member Function Documentation

### 6.8.3.1 operator==( )

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

## 6.8.4 Member Data Documentation

### 6.8.4.1 capture

Capture wr22::regex\_parser::regex::part::Group::capture

Capture behavior.

### 6.8.4.2 code\_name

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



### 6.8.4.3 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

## 6.9 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"

### 6.9.1 Detailed Description

Denotes a group captured by index.

### 6.9.2 Constructor & Destructor Documentation

#### 6.9.2.1 Index()

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

### 6.9.3 Member Function Documentation

### 6.9.3.1 operator==( )

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

## 6.9.4 Member Data Documentation

### 6.9.4.1 code\_name

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

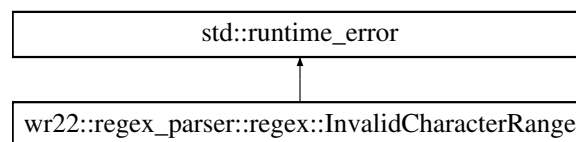
The documentation for this struct was generated from the following file:

- include/wr22/regex\_parser/regex/capture.hpp

## 6.10 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

### 6.10.1 Constructor & Destructor Documentation

### 6.10.1.1 InvalidCharacterRange()

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

## 6.10.2 Member Data Documentation

### 6.10.2.1 first

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

### 6.10.2.2 last

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

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

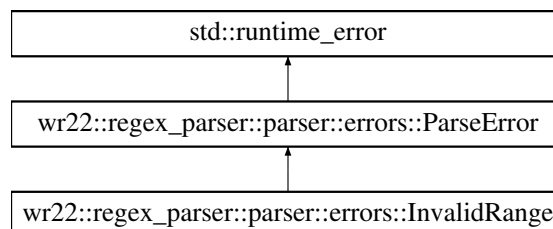
- include/wr22/regex\_parser/regex/[character\\_range.hpp](#)
- src/regex/[character\\_range.cpp](#)

## 6.11 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.*

### 6.11.1 Detailed Description

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

### 6.11.2 Constructor & Destructor Documentation

#### 6.11.2.1 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.

### 6.11.3 Member Function Documentation

#### 6.11.3.1 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.

### 6.11.3.2 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.

### 6.11.3.3 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:

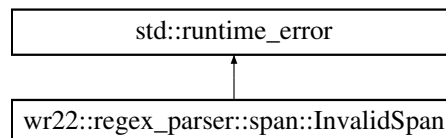
- include/wr22/regex\_parser/parser/errors.hpp
- src/parser/errors.cpp

## 6.12 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](#)

### 6.12.1 Detailed Description

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

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

## 6.12.2 Constructor & Destructor Documentation

### 6.12.2.1 InvalidSpan()

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

## 6.12.3 Member Data Documentation

### 6.12.3.1 begin

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

### 6.12.3.2 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](#)

## 6.13 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"

### 6.13.1 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.

### 6.13.2 Constructor & Destructor Documentation

#### 6.13.2.1 Literal()

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

### 6.13.3 Member Function Documentation

#### 6.13.3.1 operator==( )

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

### 6.13.4 Member Data Documentation

#### 6.13.4.1 character

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

#### 6.13.4.2 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

## 6.14 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"

#### 6.14.1 Detailed Description

Denotes a group captured by name.

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

#### 6.14.2 Constructor & Destructor Documentation

##### 6.14.2.1 Name()

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



### 6.14.3 Member Function Documentation

#### 6.14.3.1 operator==(

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

### 6.14.4 Member Data Documentation

#### 6.14.4.1 code\_name

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

#### 6.14.4.2 flavor

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

#### 6.14.4.3 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](#)

## 6.15 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"

### 6.15.1 Detailed Description

Denotes an non-capturing group.

### 6.15.2 Constructor & Destructor Documentation

#### 6.15.2.1 None()

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

### 6.15.3 Member Function Documentation

#### 6.15.3.1 operator==()

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

### 6.15.4 Member Data Documentation

#### 6.15.4.1 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

## 6.16 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"

### 6.16.1 Detailed Description

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

### 6.16.2 Constructor & Destructor Documentation

#### 6.16.2.1 Optional()

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

Convenience constructor.

### 6.16.3 Member Function Documentation

#### 6.16.3.1 operator==( )

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

### 6.16.4 Member Data Documentation

#### 6.16.4.1 code\_name

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

#### 6.16.4.2 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:

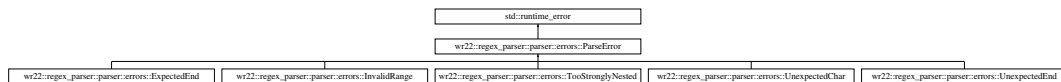
- include/wr22/regex\_parser/regex/[part.hpp](#)
- src/regex/[part.cpp](#)

### 6.17 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:



#### 6.17.1 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](#)

### 6.18 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](#) (char32\_t closing\_par)  
*Intermediate rule: parse a group name.*
- [regex::SpannedPart](#) [parse\\_char\\_class](#) ()  
*Intermediate rule: parse a character class (e.g.*

### 6.18.1 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.

### 6.18.2 Constructor & Destructor Documentation

#### 6.18.2.1 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.

### 6.18.3 Member Function Documentation

#### 6.18.3.1 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.
----------------------------------	--------------------------

#### 6.18.3.2 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.
---------------------------------	--------------------------------

#### 6.18.3.3 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.
---------------------------------	--------------------------------

**6.18.3.4 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.

**6.18.3.5 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.
------------------------------------	--

**6.18.3.6 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](#)).

### 6.18.3.7 parse\_group\_name()

```
template<typename Iter , typename Sentinel >
std::pair< std::string, Span > wr22::regex_parser::parser::Parser< Iter, Sentinel >::parse_group_name (
    char32_t closing_par ) [inline]
```

Intermediate rule: parse a group name.

#### Returns

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

### 6.18.3.8 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

<a href="#">errors::ParseError</a>	if the input cannot be parsed.
------------------------------------	--------------------------------



### 6.18.3.9 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.
---------------------------------	--------------------------------

### 6.18.3.10 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.
---------------------------------	--------------------------------

### 6.18.3.11 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

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

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

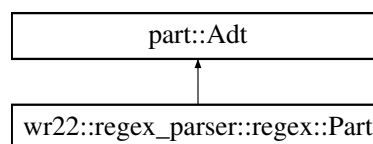
- [src/parser/regex.cpp](#)

## 6.19 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:



### 6.19.1 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](#)

## 6.20 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"

#### 6.20.1 Detailed Description

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

#### 6.20.2 Constructor & Destructor Documentation

##### 6.20.2.1 Plus()

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

Convenience constructor.

#### 6.20.3 Member Function Documentation

##### 6.20.3.1 operator==()

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

## 6.20.4 Member Data Documentation

### 6.20.4.1 `code_name`

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

### 6.20.4.2 `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](#)

## 6.21 `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"

### 6.21.1 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`.

### 6.21.2 Constructor & Destructor Documentation

#### 6.21.2.1 Sequence()

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

### 6.21.3 Member Function Documentation

#### 6.21.3.1 operator==( )

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

### 6.21.4 Member Data Documentation

#### 6.21.4.1 code\_name

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

#### 6.21.4.2 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

## 6.22 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.*

#### 6.22.1 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.

#### 6.22.2 Member Function Documentation

### 6.22.2.1 begin()

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

Get the `begin` position of the span.

### 6.22.2.2 end()

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

Get the `end` position of the span.

### 6.22.2.3 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

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

### 6.22.2.4 length()

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

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

### 6.22.2.5 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`.

### 6.22.2.6 `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 &lt; begin</code> .
--------------------	----------------------------------

### 6.22.2.7 `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.
--------------------	---

### 6.22.2.8 `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.
--------------------	---



### 6.22.2.9 operator!=(())

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

### 6.22.2.10 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](#)

## 6.23 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

### 6.23.1 Detailed Description

A [CharacterRange](#) with its span.

### 6.23.2 Member Function Documentation

#### 6.23.2.1 operator==(())

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

### 6.23.3 Member Data Documentation

#### 6.23.3.1 range

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

#### 6.23.3.2 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`

## 6.24 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.

#### 6.24.1 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).

#### 6.24.2 Constructor & Destructor Documentation

### 6.24.2.1 SpannedPart()

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

## 6.24.3 Member Function Documentation

### 6.24.3.1 operator!=(())

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

### 6.24.3.2 operator==(())

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

### 6.24.3.3 part() [1/2]

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

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

### 6.24.3.4 part() [2/2]

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

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

### 6.24.3.5 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](#)

## 6.25 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"

#### 6.25.1 Detailed Description

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

#### 6.25.2 Constructor & Destructor Documentation

##### 6.25.2.1 Star()

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

Convenience constructor.

#### 6.25.3 Member Function Documentation

##### 6.25.3.1 operator==()

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

## 6.25.4 Member Data Documentation

### 6.25.4.1 code\_name

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

### 6.25.4.2 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:

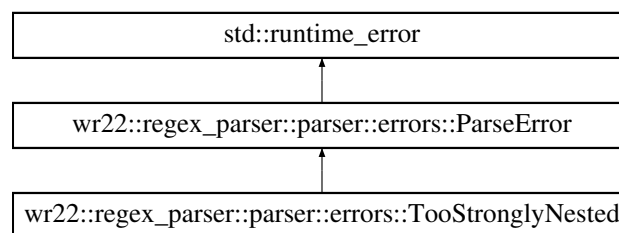
- include/wr22/regex\_parser/regex/[part.hpp](#)
- src/regex/[part.cpp](#)

## 6.26 wr22::regex\_parser::parser::errors::TooStronglyNested Class Reference

The error signalling that the regular expression has too many levels of nesting to be parsed.

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

Inheritance diagram for wr22::regex\_parser::parser::errors::TooStronglyNested:



## Public Member Functions

- [TooStronglyNested](#) ()  
*Constructor.*

### 6.26.1 Detailed Description

The error signalling that the regular expression has too many levels of nesting to be parsed.

## 6.26.2 Constructor & Destructor Documentation

### 6.26.2.1 TooStronglyNested()

```
wr22::regex_parser::parser::errors::TooStronglyNested::TooStronglyNested ( )
```

Constructor.

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

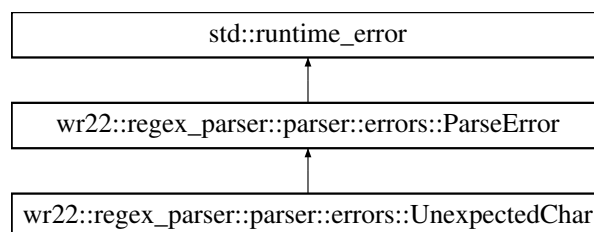
- include/wr22/regex\_parser/parser/errors.hpp
- src/parser/errors.cpp

## 6.27 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**, std::optional< char32\_t > **needs\_closing**)  
*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.*
- std::optional< char32\_t > **needs\_closing** () const  
*Get the bracket that needs to be closed.*

## 6.27.1 Detailed Description

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

## 6.27.2 Constructor & Destructor Documentation

### 6.27.2.1 UnexpectedChar()

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

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.
<i>needs_closing</i>	a character describing the type of bracket that needs to be closed, if any.

## 6.27.3 Member Function Documentation

### 6.27.3.1 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.

### 6.27.3.2 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.

### 6.27.3.3 needs\_closing()

```
std::optional< char32_t > wr22::regex_parser::parser::errors::UnexpectedChar::needs_closing (
) const
```

Get the bracket that needs to be closed.

### 6.27.3.4 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:

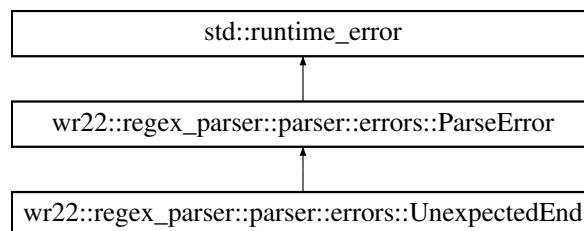
- include/wr22/regex\_parser/parser/[errors.hpp](#)
- src/parser/[errors.cpp](#)

## 6.28 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](#), std::optional< char32\_t > [needs\\_closing](#))  
*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.*
- std::optional< char32\_t > [needs\\_closing](#) () const  
*Get the bracket that needs to be closed.*



## 6.28.1 Detailed Description

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

## 6.28.2 Constructor & Destructor Documentation

### 6.28.2.1 UnexpectedEnd()

```
wr22::regex_parser::parser::errors::UnexpectedEnd::UnexpectedEnd (
    size_t position,
    std::string expected,
    std::optional< char32_t > needs_closing )
```

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.
<i>needs_closing</i>	a character describing the type of bracket that needs to be closed, if any.

## 6.28.3 Member Function Documentation

### 6.28.3.1 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.

### 6.28.3.2 needs\_closing()

```
std::optional< char32_t > wr22::regex_parser::parser::errors::UnexpectedEnd::needs_closing ( )
const
```

Get the bracket that needs to be closed.

### 6.28.3.3 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

## 6.29 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"

### 6.29.1 Detailed Description

A regex part specifying any single character ( . ).

### 6.29.2 Constructor & Destructor Documentation

#### 6.29.2.1 Wildcard()

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

### 6.29.3 Member Function Documentation

### 6.29.3.1 operator==( )

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

## 6.29.4 Member Data Documentation

### 6.29.4.1 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 <optional>
#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.*
- class `wr22::regex_parser::parser::errors::TooStronglyNested`  
*The error signalling that the regular expression has too many levels of nesting to be parsed.*

#### 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 <optional>
9 #include <stdexcept>
10 #include <string>
11
12 namespace wr22::regex_parser::parser::errors {
13
14     struct ParseError : public std::runtime_error {
15         using std::runtime_error::runtime_error;
16     };
17
18     class UnexpectedEnd : public ParseError {
19     public:
20         UnexpectedEnd(size_t position, std::string expected, std::optional<char32_t> needs_closing);
21
22         size_t position() const;
23         const std::string& expected() const;
24         std::optional<char32_t> needs_closing() const;
25
26     private:
27         size_t m_position;
28         std::string m_expected;
29         std::optional<char32_t> m_needs_closing;
30     };
31
32     class ExpectedEnd : public ParseError {
33     public:
34         ExpectedEnd(size_t position, char32_t char_got);
35
36         size_t position() const;
37         char32_t char_got() const;
38
39     private:
40         size_t m_position;
41         char32_t m_char_got;
42     };
43
44     class UnexpectedChar : public ParseError {
45     public:
46         UnexpectedChar(
47             size_t position,
48             char32_t char_got,
49             std::string expected,
50             std::optional<char32_t> needs_closing);
51
52         size_t position() const;
53         char32_t char_got() const;
54         const std::string& expected() const;
55         std::optional<char32_t> needs_closing() const;
56
57     private:
58         size_t m_position;
59         char32_t m_char_got;
60         std::string m_expected;
61         std::optional<char32_t> m_needs_closing;
62     };
63
64     class InvalidRange : public ParseError {
65     public:
66         InvalidRange(span::Span span, char32_t first, char32_t last);
67
68         span::Span span() const;
69         char32_t first() const;
70         char32_t last() const;
71
72     private:
73         span::Span m_span;
74         char32_t m_first;
75         char32_t m_last;
76     };
77
78     class TooStronglyNested : public ParseError {
79     public:
80         TooStronglyNested();
81     };
82 } // 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 &regex)  
*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/utils/adts.hpp>
#include <iosfwd>
#include <string>
#include <nlohmann/json.hpp>
```

### Classes

- struct [wr22::regex\\_parser::regex::capture::None](#)  
*Denotes a 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/utils/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
36         std::string name;
37         NamedCaptureFlavor flavor;
38         bool operator==(const Name& rhs) const = default;
39         static constexpr const char* code_name = "name";
40     };
41     void to_json(nlohmann::json& j, const Name& capture);
42
43     using Adt = utils::Adt<None, Index, Name>;
44 } // namespace capture
45
46 //
```



```

61 class Capture : public capture::Adt {
62 public:
63     using capture::Adt::Adt;
64 };
65
66 std::ostream& operator<<(std::ostream& out, const Capture& capture);
67 void to_json(nlohmann::json& j, const Capture& capture);
68
69 } // 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
11     struct CharacterClassData {
12         std::vector<SpannedCharacterRange> ranges;
13
14         bool inverted;
15
16         bool operator==(const CharacterClassData& rhs) const = default;
17     };
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 (nllohmann::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 // nllohmann
11 #include <nllohmann/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/utils/adt.hpp>
#include <wr22/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/utils/adt.hpp>
9 #include <wr22/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);
```

```

63     bool operator==(const Alternatives& rhs) const = default;
64     static constexpr const char* code_name = "alternatives";
65
66     std::vector<SpannedPart> alternatives;
67 };
68 void to_json(nlohmann::json& j, const Alternatives& part);
69
70 struct Sequence {
71     explicit Sequence(std::vector<SpannedPart> items);
72     bool operator==(const Sequence& rhs) const = default;
73     static constexpr const char* code_name = "sequence";
74
75     std::vector<SpannedPart> items;
76 };
77 void to_json(nlohmann::json& j, const Sequence& part);
78
79 struct Group {
80     explicit Group(Capture capture, SpannedPart inner);
81     bool operator==(const Group& rhs) const = default;
82     static constexpr const char* code_name = "group";
83
84     Capture capture;
85     utils::Box<SpannedPart> inner;
86 };
87 void to_json(nlohmann::json& j, const Group& part);
88
89 struct Optional {
90     explicit Optional(SpannedPart inner);
91     bool operator==(const Optional& rhs) const = default;
92     static constexpr const char* code_name = "optional";
93
94     utils::Box<SpannedPart> inner;
95 };
96 void to_json(nlohmann::json& j, const Optional& part);
97
98 struct Plus {
99     explicit Plus(SpannedPart inner);
100    bool operator==(const Plus& rhs) const = default;
101    static constexpr const char* code_name = "plus";
102
103    utils::Box<SpannedPart> inner;
104 };
105 void to_json(nlohmann::json& j, const Plus& part);
106
107 struct Star {
108     explicit Star(SpannedPart inner);
109    bool operator==(const Star& rhs) const = default;
110    static constexpr const char* code_name = "star";
111
112    utils::Box<SpannedPart> inner;
113 };
114 void to_json(nlohmann::json& j, const Star& part);
115
116 struct Wildcard {
117     explicit Wildcard() = default;
118    bool operator==(const Wildcard& rhs) const = default;
119    static constexpr const char* code_name = "wildcard";
120 };
121 void to_json(nlohmann::json& j, const Wildcard& part);
122
123 struct CharacterClass {
124     explicit CharacterClass(CharacterClassData data);
125    bool operator==(const CharacterClass& rhs) const = default;
126    static constexpr const char* code_name = "character_class";
127
128    CharacterClassData data;
129 };
130 void to_json(nlohmann::json& j, const CharacterClass& part);
131
132 using Adt = utils::
133     Adt<Empty, Literal, Alternatives, Sequence, Group, Optional, Plus, Star, Wildcard,
134     CharacterClass>;
135 } // namespace part
136
137 class Part : public part::Adt {
138 public:
139     using part::Adt::Adt;
140 };
141 void to_json(nlohmann::json& j, const Part& part);
142
143 class SpannedPart {
144 public:
145     explicit SpannedPart(Part part, span::Span span);
146
147     bool operator==(const SpannedPart& other) const = default;
148     bool operator!=(const SpannedPart& other) const = default;
149
150

```

```

210     const Part& part() const;
211     Part& part();
212
213     span::Span span() const;
214
215 private:
216     Part m_part;
217     span::Span m_span;
218 };
219 void to_json(nlohmann::json& j, const SpannedPart& part);
220
221 std::ostream& operator<<(std::ostream& out, const SpannedPart& part);
222
223 } // 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
20 bool operator==(const SpannedCharacterRange& rhs) const = default;
21 };
22
23 std::ostream& operator<<(std::ostream& out, const SpannedCharacterRange& range);
24 void to_json(nlohmann::json& j, const SpannedCharacterRange& range);
25
26 } // 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 }

```



```

19     size_t begin;
20     size_t end;
21 };
22
23
24 class Span {
25 public:
26     static Span make_empty(size_t position);
27
28     static Span make_single_position(size_t position);
29
30     static Span make_from_positions(size_t begin, size_t end);
31
32     static Span make_with_length(size_t begin, size_t length);
33
34     Span extend_right(size_t num_positions) const;
35
36     size_t length() const;
37
38     size_t begin() const;
39
40     size_t end() const;
41
42     bool operator==(const Span& other) const = default;
43     bool operator!=(const Span& other) const = default;
44
45 private:
46     explicit Span(size_t begin, size_t end);
47
48     size_t m_begin;
49     size_t m_end;
50 };
51
52 std::ostream& operator<<(std::ostream& out, Span span);
53 void to_json(nlohmann::json& j, Span span);
54
55 } // namespace wr22::regex_parser::span

```

## 7.19 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\(nlohmann::json &j, const Name &capture\)](#)

## 7.20 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](#)

## 7.21 src/parser/regex.cpp File Reference

```
#include <wr22/regex_parser/parser/errors.hpp>
#include <wr22/regex_parser/parser/regex.hpp>
#include <wr22/regex_parser/regex/part.hpp>
#include <wr22/regex_parser/span/span.hpp>
#include <wr22/unicode/conversion.hpp>
#include <wr22/utils/nonconcurrent_semaphore.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 &regex)  
*Parse a regular expression into its AST.*

## 7.22 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)`

## 7.23 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)`

## 7.24 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)`

## 7.25 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)`

## 7.26 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)`



# Index

## Adt

- [wr22::regex\\_parser::regex::capture](#), 14
- [wr22::regex\\_parser::regex::part](#), 16

## Alternatives

- [wr22::regex\\_parser::regex::part::Alternatives](#), 20

## alternatives

- [wr22::regex\\_parser::regex::part::Alternatives](#), 20

## Angles

- [wr22::regex\\_parser::regex](#), 12

## AnglesWithP

- [wr22::regex\\_parser::regex](#), 12

## Apostrophes

- [wr22::regex\\_parser::regex](#), 12

## begin

- [wr22::regex\\_parser::span::InvalidSpan](#), 36
- [wr22::regex\\_parser::span::Span](#), 52

## capture

- [wr22::regex\\_parser::regex::part::Group](#), 30

## char\_got

- [wr22::regex\\_parser::parser::errors::ExpectedEnd](#), 28
- [wr22::regex\\_parser::parser::errors::UnexpectedChar](#), 61

## character

- [wr22::regex\\_parser::regex::part::Literal](#), 37

## CharacterClass

- [wr22::regex\\_parser::regex::part::CharacterClass](#), 22

## code\_name

- [wr22::regex\\_parser::regex::capture::Index](#), 32
- [wr22::regex\\_parser::regex::capture::Name](#), 39
- [wr22::regex\\_parser::regex::capture::None](#), 40
- [wr22::regex\\_parser::regex::part::Alternatives](#), 20
- [wr22::regex\\_parser::regex::part::CharacterClass](#), 22
- [wr22::regex\\_parser::regex::part::Empty](#), 27
- [wr22::regex\\_parser::regex::part::Group](#), 30
- [wr22::regex\\_parser::regex::part::Literal](#), 37
- [wr22::regex\\_parser::regex::part::Optional](#), 41
- [wr22::regex\\_parser::regex::part::Plus](#), 50
- [wr22::regex\\_parser::regex::part::Sequence](#), 51
- [wr22::regex\\_parser::regex::part::Star](#), 59
- [wr22::regex\\_parser::regex::part::Wildcard](#), 65

## contains

- [wr22::regex\\_parser::regex::CharacterRange](#), 24

## data

- [wr22::regex\\_parser::regex::part::CharacterClass](#), 22

## Empty

- [wr22::regex\\_parser::regex::part::Empty](#), 26

## end

- [wr22::regex\\_parser::span::InvalidSpan](#), 36
- [wr22::regex\\_parser::span::Span](#), 53

## expect\_end

- [wr22::regex\\_parser::parser::Parser< Iter, Sentinel >](#), 44

## expected

- [wr22::regex\\_parser::parser::errors::UnexpectedChar](#), 61
- [wr22::regex\\_parser::parser::errors::UnexpectedEnd](#), 63

## ExpectedEnd

- [wr22::regex\\_parser::parser::errors::ExpectedEnd](#), 28

## extend\_right

- [wr22::regex\\_parser::span::Span](#), 53

## first

- [wr22::regex\\_parser::parser::errors::InvalidRange](#), 34
- [wr22::regex\\_parser::regex::CharacterRange](#), 25
- [wr22::regex\\_parser::regex::InvalidCharacterRange](#), 33

## flavor

- [wr22::regex\\_parser::regex::capture::Name](#), 39

## from\_endpoints

- [wr22::regex\\_parser::regex::CharacterRange](#), 25

## from\_single\_character

- [wr22::regex\\_parser::regex::CharacterRange](#), 25

## Group

- [wr22::regex\\_parser::regex::part::Group](#), 30

- [include/wr22/regex\\_parser/parser/errors.hpp](#), 67, 68

- [include/wr22/regex\\_parser/parser/regex.hpp](#), 69

- [include/wr22/regex\\_parser/regex/capture.hpp](#), 69, 70

- [include/wr22/regex\\_parser/regex/character\\_class\\_data.hpp](#), 71

- [include/wr22/regex\\_parser/regex/character\\_range.hpp](#), 71, 72

- [include/wr22/regex\\_parser/regex/named\\_capture\\_flavor.hpp](#), 73

- [include/wr22/regex\\_parser/regex/part.hpp](#), 74, 75

- [include/wr22/regex\\_parser/regex/spanned\\_character\\_range.hpp](#), 77

- include/wr22/regex\_parser/span/span.hpp, 78
- Index
  - wr22::regex\_parser::regex::capture::Index, 31
- inner
  - wr22::regex\_parser::regex::part::Group, 30
  - wr22::regex\_parser::regex::part::Optional, 42
  - wr22::regex\_parser::regex::part::Plus, 50
  - wr22::regex\_parser::regex::part::Star, 59
- InvalidCharacterRange
  - wr22::regex\_parser::regex::InvalidCharacterRange, 32
- InvalidRange
  - wr22::regex\_parser::parser::errors::InvalidRange, 34
- InvalidSpan
  - wr22::regex\_parser::span::InvalidSpan, 36
- inverted
  - wr22::regex\_parser::regex::CharacterClassData, 23
- is\_single\_character
  - wr22::regex\_parser::regex::CharacterRange, 25
- items
  - wr22::regex\_parser::regex::part::Sequence, 51
- last
  - wr22::regex\_parser::parser::errors::InvalidRange, 34
  - wr22::regex\_parser::regex::CharacterRange, 25
  - wr22::regex\_parser::regex::InvalidCharacterRange, 33
- length
  - wr22::regex\_parser::span::Span, 53
- Literal
  - wr22::regex\_parser::regex::part::Literal, 37
- make\_empty
  - wr22::regex\_parser::span::Span, 53
- make\_from\_positions
  - wr22::regex\_parser::span::Span, 53
- make\_single\_position
  - wr22::regex\_parser::span::Span, 54
- make\_with\_length
  - wr22::regex\_parser::span::Span, 54
- Name
  - wr22::regex\_parser::regex::capture::Name, 38
- name
  - wr22::regex\_parser::regex::capture::Name, 39
- NamedCaptureFlavor
  - wr22::regex\_parser::regex, 12
- needs\_closing
  - wr22::regex\_parser::parser::errors::UnexpectedChar, 61
  - wr22::regex\_parser::parser::errors::UnexpectedEnd, 63
- None
  - wr22::regex\_parser::regex::capture::None, 40
- operator!=
  - wr22::regex\_parser::regex::SpannedPart, 57
  - wr22::regex\_parser::span::Span, 54
- operator<<
  - wr22::regex\_parser::regex, 12, 13
  - wr22::regex\_parser::span, 18
- operator==
  - wr22::regex\_parser::regex::capture::Index, 31
  - wr22::regex\_parser::regex::capture::Name, 39
  - wr22::regex\_parser::regex::capture::None, 40
  - wr22::regex\_parser::regex::CharacterClassData, 23
  - wr22::regex\_parser::regex::CharacterRange, 26
  - wr22::regex\_parser::regex::part::Alternatives, 20
  - wr22::regex\_parser::regex::part::CharacterClass, 22
  - wr22::regex\_parser::regex::part::Empty, 27
  - wr22::regex\_parser::regex::part::Group, 30
  - wr22::regex\_parser::regex::part::Literal, 37
  - wr22::regex\_parser::regex::part::Optional, 41
  - wr22::regex\_parser::regex::part::Plus, 49
  - wr22::regex\_parser::regex::part::Sequence, 51
  - wr22::regex\_parser::regex::part::Star, 58
  - wr22::regex\_parser::regex::part::Wildcard, 64
  - wr22::regex\_parser::regex::SpannedCharacterRange, 55
  - wr22::regex\_parser::regex::SpannedPart, 57
  - wr22::regex\_parser::span::Span, 55
- Optional
  - wr22::regex\_parser::regex::part::Optional, 41
- parse\_alternatives
  - wr22::regex\_parser::parser::Parser< Iter, Sentinel >, 44
- parse\_atom
  - wr22::regex\_parser::parser::Parser< Iter, Sentinel >, 44
- parse\_char\_class
  - wr22::regex\_parser::parser::Parser< Iter, Sentinel >, 45
- parse\_char\_literal
  - wr22::regex\_parser::parser::Parser< Iter, Sentinel >, 45
- parse\_group
  - wr22::regex\_parser::parser::Parser< Iter, Sentinel >, 45
- parse\_group\_name
  - wr22::regex\_parser::parser::Parser< Iter, Sentinel >, 46
- parse\_regex
  - wr22::regex\_parser::parser, 10
  - wr22::regex\_parser::parser::Parser< Iter, Sentinel >, 46
- parse\_sequence
  - wr22::regex\_parser::parser::Parser< Iter, Sentinel >, 46
- parse\_sequence\_or\_empty
  - wr22::regex\_parser::parser::Parser< Iter, Sentinel >, 47
- parse\_wildcard



- wr22::regex\_parser::parser::Parser< Iter, Sentinel >, [47](#)
- Parser
  - wr22::regex\_parser::parser, [10](#)
  - wr22::regex\_parser::parser::Parser< Iter, Sentinel >, [43](#)
- part
  - wr22::regex\_parser::regex::SpannedPart, [57](#)
- Plus
  - wr22::regex\_parser::regex::part::Plus, [49](#)
- position
  - wr22::regex\_parser::parser::errors::ExpectedEnd, [28](#)
  - wr22::regex\_parser::parser::errors::UnexpectedChar, [62](#)
  - wr22::regex\_parser::parser::errors::UnexpectedEnd, [63](#)
- range
  - wr22::regex\_parser::regex::SpannedCharacterRange, [56](#)
- ranges
  - wr22::regex\_parser::regex::CharacterClassData, [23](#)
- Sequence
  - wr22::regex\_parser::regex::part::Sequence, [51](#)
- span
  - wr22::regex\_parser::parser::errors::InvalidRange, [35](#)
  - wr22::regex\_parser::regex::SpannedCharacterRange, [56](#)
  - wr22::regex\_parser::regex::SpannedPart, [57](#)
- SpannedPart
  - wr22::regex\_parser::regex::SpannedPart, [56](#)
- src/parser/capture.cpp, [79](#)
- src/parser/errors.cpp, [80](#)
- src/parser/regex.cpp, [80](#)
- src/regex/character\_range.cpp, [81](#)
- src/regex/named\_capture\_flavor.cpp, [81](#)
- src/regex/part.cpp, [81](#)
- src/regex/spanned\_character\_range.cpp, [82](#)
- src/span/span.cpp, [83](#)
- Star
  - wr22::regex\_parser::regex::part::Star, [58](#)
- to\_json
  - wr22::regex\_parser::regex, [13](#), [14](#)
  - wr22::regex\_parser::regex::capture, [15](#)
  - wr22::regex\_parser::regex::part, [16–18](#)
  - wr22::regex\_parser::span, [18](#)
- TooStronglyNested
  - wr22::regex\_parser::parser::errors::TooStronglyNested, [60](#)
- UnexpectedChar
  - wr22::regex\_parser::parser::errors::UnexpectedChar, [61](#)
- UnexpectedEnd
  - wr22::regex\_parser::parser::errors::UnexpectedEnd, [63](#)
- Wildcard
  - wr22::regex\_parser::regex::part::Wildcard, [64](#)
- 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, [27](#)
  - char\_got, [28](#)
  - ExpectedEnd, [28](#)
  - position, [28](#)
- wr22::regex\_parser::parser::errors::InvalidRange, [33](#)
  - first, [34](#)
  - InvalidRange, [34](#)
  - last, [34](#)
  - span, [35](#)
- wr22::regex\_parser::parser::errors::ParseError, [42](#)
- wr22::regex\_parser::parser::errors::TooStronglyNested, [59](#)
  - TooStronglyNested, [60](#)
- wr22::regex\_parser::parser::errors::UnexpectedChar, [60](#)
  - char\_got, [61](#)
  - expected, [61](#)
  - needs\_closing, [61](#)
  - position, [62](#)
  - UnexpectedChar, [61](#)
- wr22::regex\_parser::parser::errors::UnexpectedEnd, [62](#)
  - expected, [63](#)
  - needs\_closing, [63](#)
  - position, [63](#)
  - UnexpectedEnd, [63](#)
- wr22::regex\_parser::parser::Parser< Iter, Sentinel >, [42](#)
  - expect\_end, [44](#)
  - parse\_alternatives, [44](#)
  - parse\_atom, [44](#)
  - parse\_char\_class, [45](#)
  - parse\_char\_literal, [45](#)
  - parse\_group, [45](#)
  - parse\_group\_name, [46](#)
  - parse\_regex, [46](#)
  - parse\_sequence, [46](#)
  - parse\_sequence\_or\_empty, [47](#)
  - parse\_wildcard, [47](#)
  - Parser, [43](#)
- wr22::regex\_parser::regex, [11](#)
  - Angles, [12](#)
  - AnglesWithP, [12](#)
  - Apostrophes, [12](#)
  - NamedCaptureFlavor, [12](#)
  - operator<<, [12](#), [13](#)
  - to\_json, [13](#), [14](#)
- wr22::regex\_parser::regex::Capture, [21](#)
- wr22::regex\_parser::regex::capture, [14](#)

- Adt, 14
- to\_json, 15
- wr22::regex\_parser::regex::capture::Index, 31
  - code\_name, 32
  - Index, 31
  - operator==, 31
- wr22::regex\_parser::regex::capture::Name, 38
  - code\_name, 39
  - flavor, 39
  - Name, 38
  - name, 39
  - operator==, 39
- wr22::regex\_parser::regex::capture::None, 39
  - code\_name, 40
  - None, 40
  - operator==, 40
- wr22::regex\_parser::regex::CharacterClassData, 23
  - inverted, 23
  - operator==, 23
  - ranges, 23
- wr22::regex\_parser::regex::CharacterRange, 24
  - contains, 24
  - first, 25
  - from\_endpoints, 25
  - from\_single\_character, 25
  - is\_single\_character, 25
  - last, 25
  - operator==, 26
- wr22::regex\_parser::regex::InvalidCharacterRange, 32
  - first, 33
  - InvalidCharacterRange, 32
  - last, 33
- wr22::regex\_parser::regex::Part, 48
- wr22::regex\_parser::regex::part, 15
  - Adt, 16
  - to\_json, 16–18
- wr22::regex\_parser::regex::part::Alternatives, 19
  - Alternatives, 20
  - alternatives, 20
  - code\_name, 20
  - operator==, 20
- wr22::regex\_parser::regex::part::CharacterClass, 21
  - CharacterClass, 22
  - code\_name, 22
  - data, 22
  - operator==, 22
- wr22::regex\_parser::regex::part::Empty, 26
  - code\_name, 27
  - Empty, 26
  - operator==, 27
- wr22::regex\_parser::regex::part::Group, 29
  - capture, 30
  - code\_name, 30
  - Group, 30
  - inner, 30
  - operator==, 30
- wr22::regex\_parser::regex::part::Literal, 36
  - character, 37
  - code\_name, 37
  - Literal, 37
  - operator==, 37
- wr22::regex\_parser::regex::part::Optional, 40
  - code\_name, 41
  - inner, 42
  - operator==, 41
  - Optional, 41
- wr22::regex\_parser::regex::part::Plus, 49
  - code\_name, 50
  - inner, 50
  - operator==, 49
  - Plus, 49
- wr22::regex\_parser::regex::part::Sequence, 50
  - code\_name, 51
  - items, 51
  - operator==, 51
  - Sequence, 51
- wr22::regex\_parser::regex::part::Star, 58
  - code\_name, 59
  - inner, 59
  - operator==, 58
  - Star, 58
- wr22::regex\_parser::regex::part::Wildcard, 64
  - code\_name, 65
  - operator==, 64
  - Wildcard, 64
- wr22::regex\_parser::regex::SpannedCharacterRange, 55
  - operator==, 55
  - range, 56
  - span, 56
- wr22::regex\_parser::regex::SpannedPart, 56
  - operator!=, 57
  - operator==, 57
  - part, 57
  - span, 57
  - SpannedPart, 56
- wr22::regex\_parser::span, 18
  - operator<<, 18
  - to\_json, 18
- wr22::regex\_parser::span::InvalidSpan, 35
  - begin, 36
  - end, 36
  - InvalidSpan, 36
- wr22::regex\_parser::span::Span, 52
  - begin, 52
  - end, 53
  - extend\_right, 53
  - length, 53
  - make\_empty, 53
  - make\_from\_positions, 53
  - make\_single\_position, 54
  - make\_with\_length, 54
  - operator!=, 54
  - operator==, 55