

## 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	11
5.5.1.1 NamedCaptureFlavor	11
5.5.2 Function Documentation	12
5.5.2.1 operator<<() [1/3]	12
5.5.2.2 operator<<() [2/3]	12
5.5.2.3 operator<<() [3/3]	12
5.5.2.4 to_json() [1/4]	12
5.5.2.5 to_json() [2/4]	13
5.5.2.6 to_json() [3/4]	13
5.5.2.7 to_json() [4/4]	13
5.6 wr22::regex_parser::regex::capture Namespace Reference	13
5.6.1 Typedef Documentation	13
5.6.1.1 Adt	14
5.6.2 Function Documentation	14
5.6.2.1 to_json() [1/3]	14
5.6.2.2 to_json() [2/3]	14
5.6.2.3 to_json() [3/3]	14
5.7 wr22::regex_parser::regex::part Namespace Reference	14
5.7.1 Detailed Description	15
5.7.2 Typedef Documentation	15
5.7.2.1 Adt	15
5.7.3 Function Documentation	15
5.7.3.1 to_json() [1/9]	15

5.7.3.2 to_json() [2/9]	16
5.7.3.3 to_json() [3/9]	16
5.7.3.4 to_json() [4/9]	16
5.7.3.5 to_json() [5/9]	16
5.7.3.6 to_json() [6/9]	16
5.7.3.7 to_json() [7/9]	16
5.7.3.8 to_json() [8/9]	17
5.7.3.9 to_json() [9/9]	17
5.8 wr22::regex_parser::span Namespace Reference	17
5.8.1 Function Documentation	17
5.8.1.1 operator<<()	17
5.8.1.2 to_json()	17
5.9 wr22::regex_parser::utils Namespace Reference	18
5.9.1 Function Documentation	18
5.9.1.1 Box() [1/2]	18
5.9.1.2 Box() [2/2]	19
5.9.1.3 operator!=(()) [1/2]	19
5.9.1.4 operator!=(()) [2/2]	19
5.9.1.5 operator==(()) [1/2]	19
5.9.1.6 operator==(()) [2/2]	19
5.10 wr22::regex_parser::utils::detail Namespace Reference	20
5.11 wr22::regex_parser::utils::detail::adt Namespace Reference	20
<b>6 Class Documentation</b>	<b>21</b>
6.1 wr22::regex_parser::utils::Adt< Variants > Class Template Reference	21
6.1.1 Detailed Description	22
6.1.2 Member Typedef Documentation	22
6.1.2.1 VariantType	22
6.1.3 Constructor & Destructor Documentation	22
6.1.3.1 Adt()	22
6.1.4 Member Function Documentation	23
6.1.4.1 as_variant() [1/2]	23
6.1.4.2 as_variant() [2/2]	23
6.1.4.3 visit() [1/2]	23
6.1.4.4 visit() [2/2]	23
6.1.5 Member Data Documentation	24
6.1.5.1 m_variant	24
6.2 wr22::regex_parser::regex::part::Alternatives Struct Reference	24
6.2.1 Detailed Description	24
6.2.2 Constructor & Destructor Documentation	24
6.2.2.1 Alternatives()	25
6.2.3 Member Function Documentation	25

6.2.3.1 operator==( )	25
6.2.4 Member Data Documentation	25
6.2.4.1 alternatives	25
6.2.4.2 code_name	25
6.3 wr22::regex_parser::utils::Box< T > Class Template Reference	25
6.3.1 Detailed Description	26
6.3.2 Constructor & Destructor Documentation	26
6.3.2.1 Box() [1/3]	26
6.3.2.2 Box() [2/3]	27
6.3.2.3 Box() [3/3]	27
6.3.3 Member Function Documentation	27
6.3.3.1 construct_in_place()	27
6.3.3.2 operator*( ) [1/2]	28
6.3.3.3 operator*( ) [2/2]	28
6.4 wr22::regex_parser::utils::BoxIsEmpty Struct Reference	28
6.4.1 Member Function Documentation	28
6.4.1.1 what()	29
6.5 wr22::regex_parser::regex::Capture Class Reference	29
6.5.1 Detailed Description	29
6.6 wr22::regex_parser::regex::part::Empty Struct Reference	29
6.6.1 Detailed Description	30
6.6.2 Constructor & Destructor Documentation	30
6.6.2.1 Empty()	30
6.6.3 Member Function Documentation	30
6.6.3.1 operator==( )	30
6.6.4 Member Data Documentation	30
6.6.4.1 code_name	30
6.7 wr22::regex_parser::parser::errors::ExpectedEnd Class Reference	31
6.7.1 Detailed Description	31
6.7.2 Constructor & Destructor Documentation	31
6.7.2.1 ExpectedEnd()	31
6.7.3 Member Function Documentation	32
6.7.3.1 char_got()	32
6.7.3.2 position()	32
6.8 wr22::regex_parser::regex::part::Group Struct Reference	32
6.8.1 Detailed Description	33
6.8.2 Constructor & Destructor Documentation	33
6.8.2.1 Group()	33
6.8.3 Member Function Documentation	33
6.8.3.1 operator==( )	33
6.8.4 Member Data Documentation	33
6.8.4.1 capture	33

6.8.4.2 <code>code_name</code>	34
6.8.4.3 <code>inner</code>	34
6.9 <code>wr22::regex_parser::regex::capture::Index</code> Struct Reference	34
6.9.1 Detailed Description	34
6.9.2 Constructor & Destructor Documentation	34
6.9.2.1 <code>Index()</code>	34
6.9.3 Member Function Documentation	35
6.9.3.1 <code>operator==()</code>	35
6.9.4 Member Data Documentation	35
6.9.4.1 <code>code_name</code>	35
6.10 <code>wr22::regex_parser::span::InvalidSpan</code> Struct Reference	35
6.10.1 Detailed Description	36
6.10.2 Constructor & Destructor Documentation	36
6.10.2.1 <code>InvalidSpan()</code>	36
6.10.3 Member Data Documentation	36
6.10.3.1 <code>begin</code>	36
6.10.3.2 <code>end</code>	36
6.11 <code>wr22::regex_parser::regex::part::Literal</code> Struct Reference	36
6.11.1 Detailed Description	37
6.11.2 Constructor & Destructor Documentation	37
6.11.2.1 <code>Literal()</code>	37
6.11.3 Member Function Documentation	37
6.11.3.1 <code>operator==()</code>	37
6.11.4 Member Data Documentation	37
6.11.4.1 <code>character</code>	37
6.11.4.2 <code>code_name</code>	38
6.12 <code>wr22::regex_parser::utils::detail::adt::MultiCallable&lt; Fs &gt;</code> Struct Template Reference	38
6.12.1 Constructor & Destructor Documentation	38
6.12.1.1 <code>MultiCallable()</code>	38
6.13 <code>wr22::regex_parser::regex::capture::Name</code> Struct Reference	38
6.13.1 Detailed Description	39
6.13.2 Constructor & Destructor Documentation	39
6.13.2.1 <code>Name()</code>	39
6.13.3 Member Function Documentation	39
6.13.3.1 <code>operator==()</code>	39
6.13.4 Member Data Documentation	39
6.13.4.1 <code>code_name</code>	40
6.13.4.2 <code>flavor</code>	40
6.13.4.3 <code>name</code>	40
6.14 <code>wr22::regex_parser::regex::capture::None</code> Struct Reference	40
6.14.1 Detailed Description	40
6.14.2 Constructor & Destructor Documentation	40

6.14.2.1 None()	41
6.14.3 Member Function Documentation	41
6.14.3.1 operator==( )	41
6.14.4 Member Data Documentation	41
6.14.4.1 code_name	41
6.15 wr22::regex_parser::regex::part::Optional Struct Reference	41
6.15.1 Detailed Description	42
6.15.2 Constructor & Destructor Documentation	42
6.15.2.1 Optional()	42
6.15.3 Member Function Documentation	42
6.15.3.1 operator==( )	42
6.15.4 Member Data Documentation	42
6.15.4.1 code_name	42
6.15.4.2 inner	43
6.16 wr22::regex_parser::parser::errors::ParseError Struct Reference	43
6.16.1 Detailed Description	43
6.17 wr22::regex_parser::parser::Parser< Iter, Sentinel > Class Template Reference	43
6.17.1 Detailed Description	44
6.17.2 Constructor & Destructor Documentation	44
6.17.2.1 Parser()	44
6.17.3 Member Function Documentation	45
6.17.3.1 expect_end()	45
6.17.3.2 parse_alternatives()	45
6.17.3.3 parse_atom()	45
6.17.3.4 parse_char_literal()	46
6.17.3.5 parse_group()	46
6.17.3.6 parse_group_name()	46
6.17.3.7 parse_regex()	47
6.17.3.8 parse_sequence()	47
6.17.3.9 parse_sequence_or_empty()	47
6.17.3.10 parse_wildcard()	48
6.18 wr22::regex_parser::regex::Part Class Reference	48
6.18.1 Detailed Description	49
6.19 wr22::regex_parser::regex::part::Plus Struct Reference	49
6.19.1 Detailed Description	50
6.19.2 Constructor & Destructor Documentation	50
6.19.2.1 Plus()	50
6.19.3 Member Function Documentation	50
6.19.3.1 operator==( )	50
6.19.4 Member Data Documentation	50
6.19.4.1 code_name	50
6.19.4.2 inner	50

6.20 wr22::regex_parser::regex::part::Sequence Struct Reference	51
6.20.1 Detailed Description	51
6.20.2 Constructor & Destructor Documentation	51
6.20.2.1 Sequence()	51
6.20.3 Member Function Documentation	51
6.20.3.1 operator==( )	51
6.20.4 Member Data Documentation	52
6.20.4.1 code_name	52
6.20.4.2 items	52
6.21 wr22::regex_parser::span::Span Class Reference	52
6.21.1 Detailed Description	53
6.21.2 Member Function Documentation	53
6.21.2.1 begin()	53
6.21.2.2 end()	53
6.21.2.3 length()	53
6.21.2.4 make_empty()	53
6.21.2.5 make_from_positions()	53
6.21.2.6 make_single_position()	54
6.21.2.7 make_with_length()	54
6.21.2.8 operator!=( )	54
6.21.2.9 operator==( )	55
6.22 wr22::regex_parser::regex::SpannedPart Class Reference	55
6.22.1 Detailed Description	55
6.22.2 Constructor & Destructor Documentation	55
6.22.2.1 SpannedPart()	55
6.22.3 Member Function Documentation	56
6.22.3.1 operator!=( )	56
6.22.3.2 operator==( )	56
6.22.3.3 part() [1/2]	56
6.22.3.4 part() [2/2]	56
6.22.3.5 span()	56
6.23 wr22::regex_parser::regex::part::Star Struct Reference	57
6.23.1 Detailed Description	57
6.23.2 Constructor & Destructor Documentation	57
6.23.2.1 Star()	57
6.23.3 Member Function Documentation	57
6.23.3.1 operator==( )	57
6.23.4 Member Data Documentation	58
6.23.4.1 code_name	58
6.23.4.2 inner	58
6.24 wr22::regex_parser::parser::errors::UnexpectedChar Class Reference	58
6.24.1 Detailed Description	59



6.24.2 Constructor & Destructor Documentation	59
6.24.2.1 UnexpectedChar()	59
6.24.3 Member Function Documentation	59
6.24.3.1 char_got()	59
6.24.3.2 expected()	59
6.24.3.3 position()	60
6.25 wr22::regex_parser::parser::errors::UnexpectedEnd Class Reference	60
6.25.1 Detailed Description	60
6.25.2 Constructor & Destructor Documentation	60
6.25.2.1 UnexpectedEnd()	60
6.25.3 Member Function Documentation	61
6.25.3.1 expected()	61
6.25.3.2 position()	61
6.26 wr22::regex_parser::regex::part::Wildcard Struct Reference	61
6.26.1 Detailed Description	62
6.26.2 Constructor & Destructor Documentation	62
6.26.2.1 Wildcard()	62
6.26.3 Member Function Documentation	62
6.26.3.1 operator==(())	62
6.26.4 Member Data Documentation	62
6.26.4.1 code_name	62
<b>7 File Documentation</b>	<b>63</b>
7.1 include/wr22/regex_parser/parser/errors.hpp File Reference	63
7.2 errors.hpp	64
7.3 include/wr22/regex_parser/parser/regex.hpp File Reference	64
7.4 regex.hpp	65
7.5 include/wr22/regex_parser/regex/capture.hpp File Reference	65
7.6 capture.hpp	66
7.7 include/wr22/regex_parser/regex/named_capture_flavor.hpp File Reference	66
7.8 named_capture_flavor.hpp	67
7.9 include/wr22/regex_parser/regex/part.hpp File Reference	67
7.10 part.hpp	69
7.11 include/wr22/regex_parser/span/span.hpp File Reference	70
7.12 span.hpp	71
7.13 include/wr22/regex_parser/utils/adt.hpp File Reference	71
7.14 adt.hpp	72
7.15 include/wr22/regex_parser/utils/box.hpp File Reference	73
7.16 box.hpp	74
7.17 src/parser/capture.cpp File Reference	74
7.18 src/parser/errors.cpp File Reference	75
7.19 src/parser/regex.cpp File Reference	75

<a href="#">7.20 src/regex/named_capture_flavor.cpp File Reference</a>	76
<a href="#">7.21 src/regex/part.cpp File Reference</a>	76
<a href="#">7.22 src/span/span.cpp File Reference</a>	77
<a href="#">7.23 src/utils/box.cpp File Reference</a>	77
<b><a href="#">Index</a></b>	<b>79</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> . . . . .	13
<a href="#">wr22::regex_parser::regex::part</a> The namespace with the variants of <a href="#">Part</a> . . . . .	14
<a href="#">wr22::regex_parser::span</a> . . . . .	17
<a href="#">wr22::regex_parser::utils</a> . . . . .	18
<a href="#">wr22::regex_parser::utils::detail</a> . . . . .	20
<a href="#">wr22::regex_parser::utils::detail::adt</a> . . . . .	20



## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

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

wr22::regex_parser::utils::Adt< Variants > . . . . .	21
wr22::regex_parser::regex::Capture . . . . .	29
wr22::regex_parser::regex::Part . . . . .	48
wr22::regex_parser::regex::part::Alternatives . . . . .	24
wr22::regex_parser::utils::Box< T > . . . . .	25
wr22::regex_parser::utils::Box< wr22::regex_parser::regex::SpannedPart > . . . . .	25
wr22::regex_parser::regex::part::Empty . . . . .	29
std::exception	
wr22::regex_parser::utils::BoxIsEmpty . . . . .	28
wr22::regex_parser::regex::part::Group . . . . .	32
wr22::regex_parser::regex::capture::Index . . . . .	34
wr22::regex_parser::regex::part::Literal . . . . .	36
wr22::regex_parser::regex::capture::Name . . . . .	38
wr22::regex_parser::regex::capture::None . . . . .	40
wr22::regex_parser::regex::part::Optional . . . . .	41
wr22::regex_parser::parser::Parser< Iter, Sentinel > . . . . .	43
wr22::regex_parser::regex::part::Plus . . . . .	49
std::runtime_error	
wr22::regex_parser::parser::errors::ParseError . . . . .	43
wr22::regex_parser::parser::errors::ExpectedEnd . . . . .	31
wr22::regex_parser::parser::errors::UnexpectedChar . . . . .	58
wr22::regex_parser::parser::errors::UnexpectedEnd . . . . .	60
wr22::regex_parser::span::InvalidSpan . . . . .	35
wr22::regex_parser::regex::part::Sequence . . . . .	51
wr22::regex_parser::span::Span . . . . .	52
wr22::regex_parser::regex::SpannedPart . . . . .	55
wr22::regex_parser::regex::part::Star . . . . .	57
wr22::regex_parser::regex::part::Wildcard . . . . .	61
wr22::regex_parser::utils::detail::adt::Fs	
wr22::regex_parser::utils::detail::adt::MultiCallable< Fs > . . . . .	38



## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">wr22::regex_parser::utils::Adt&lt; Variants &gt;</a>	21
A helper class that simplifies creation of algebraic data types	
<a href="#">wr22::regex_parser::regex::part::Alternatives</a>	24
A regex part with the list of alternatives to be matched	
<a href="#">wr22::regex_parser::utils::Box&lt; T &gt;</a>	25
A copyable and equality-comparable wrapper around <code>std::unique_ptr</code>	
<a href="#">wr22::regex_parser::utils::BoxIsEmpty</a>	28
<a href="#">wr22::regex_parser::regex::Capture</a>	29
Group capture behavior	
<a href="#">wr22::regex_parser::regex::part::Empty</a>	29
An empty regex part	
<a href="#">wr22::regex_parser::parser::errors::ExpectedEnd</a>	31
The error when the parser expected the input to end, but it did not	
<a href="#">wr22::regex_parser::regex::part::Group</a>	32
A regex part that represents a group in parentheses	
<a href="#">wr22::regex_parser::regex::capture::Index</a>	34
Denotes a group captured by index	
<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::utils::detail::adt::MultiCallable&lt; Fs &gt;</a>	38
<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>	40
Denotes an non-capturing group	
<a href="#">wr22::regex_parser::regex::part::Optional</a>	41
A regex part specifying an optional quantifier <code>((expression)?)</code>	
<a href="#">wr22::regex_parser::parser::errors::ParseError</a>	43
The base class for parse errors	
<a href="#">wr22::regex_parser::parser::Parser&lt; Iter, Sentinel &gt;</a>	43
A regex parser	
<a href="#">wr22::regex_parser::regex::Part</a>	48
A part of a regular expression and its AST node type	
<a href="#">wr22::regex_parser::regex::part::Plus</a>	49
A regex part specifying an "at least one" quantifier <code>((expression)+)</code>	

<a href="#">wr22::regex_parser::regex::part::Sequence</a>	
A regex part with the list of items to be matched one after another . . . . .	51
<a href="#">wr22::regex_parser::span::Span</a>	
Character position range in the input string . . . . .	52
<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)	55
<a href="#">wr22::regex_parser::regex::part::Star</a>	
A regex part specifying an "at least zero" quantifier (( <a href="#">expression</a> )* . . . . .	57
<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 . . . . .	58
<a href="#">wr22::regex_parser::parser::errors::UnexpectedEnd</a>	
The error when the parser hit the end of the input earlier than it expected . . . . .	60
<a href="#">wr22::regex_parser::regex::part::Wildcard</a>	
A regex part specifying any single character (.) . . . . .	61



## Chapter 4

# File Index

### 4.1 File List

Here is a list of all files with brief descriptions:

include/wr22/regex_parser/parser/ <a href="#">errors.hpp</a> . . . . .	63
include/wr22/regex_parser/parser/ <a href="#">regex.hpp</a> . . . . .	64
include/wr22/regex_parser/regex/ <a href="#">capture.hpp</a> . . . . .	65
include/wr22/regex_parser/regex/ <a href="#">named_capture_flavor.hpp</a> . . . . .	66
include/wr22/regex_parser/regex/ <a href="#">part.hpp</a> . . . . .	67
include/wr22/regex_parser/span/ <a href="#">span.hpp</a> . . . . .	70
include/wr22/regex_parser/utils/ <a href="#">adt.hpp</a> . . . . .	71
include/wr22/regex_parser/utils/ <a href="#">box.hpp</a> . . . . .	73
src/parser/ <a href="#">capture.cpp</a> . . . . .	74
src/parser/ <a href="#">errors.cpp</a> . . . . .	75
src/parser/ <a href="#">regex.cpp</a> . . . . .	75
src/regex/ <a href="#">named_capture_flavor.cpp</a> . . . . .	76
src/regex/ <a href="#">part.cpp</a> . . . . .	76
src/span/ <a href="#">span.cpp</a> . . . . .	77
src/utils/ <a href="#">box.cpp</a> . . . . .	77



## Chapter 5

# Namespace Documentation

### 5.1 wr22 Namespace Reference

#### Namespaces

- namespace [regex\\_parser](#)

### 5.2 wr22::regex\_parser Namespace Reference

#### Namespaces

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

### 5.3 wr22::regex\_parser::parser Namespace Reference

#### Namespaces

- namespace [errors](#)

#### Classes

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

#### Functions

- template<typename Iter , typename Sentinel >  
[Parser](#) (Iter begin, Sentinel end) -> [Parser](#)< Iter, Sentinel >  
*The type deduction guideline for [Parser](#).*
- [regex::SpannedPart parse\\_regex](#) (const std::u32string\_view &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

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

#### 5.3.1.2 `Parser()`

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

The type deduction guideline for `Parser`.

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

### Classes

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

## 5.5 wr22::regex\_parser::regex Namespace Reference

### Namespaces

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

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

### Classes

- class [Capture](#)

*Group capture behavior.*

- class [Part](#)

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

- 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, 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)`

### 5.5.1 Enumeration Type Documentation

#### 5.5.1.1 NamedCaptureFlavor

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

The flavor (dialect) of a named group capture.

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

## Enumerator

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

## 5.5.2 Function Documentation

### 5.5.2.1 operator<<() [1/3]

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

### 5.5.2.2 operator<<() [2/3]

```
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`.

### 5.5.2.3 operator<<() [3/3]

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

### 5.5.2.4 to\_json() [1/4]

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

#### 5.5.2.5 to\_json() [2/4]

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

#### 5.5.2.6 to\_json() [3/4]

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

#### 5.5.2.7 to\_json() [4/4]

```
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 [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](#) >

## 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)

### 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>
```

### 5.7.3 Function Documentation

#### 5.7.3.1 to\_json() [1/9]

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

#### 5.7.3.2 to\_json() [2/9]

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

#### 5.7.3.3 to\_json() [3/9]

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

#### 5.7.3.4 to\_json() [4/9]

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

#### 5.7.3.5 to\_json() [5/9]

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

#### 5.7.3.6 to\_json() [6/9]

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

#### 5.7.3.7 to\_json() [7/9]

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

### 5.7.3.8 to\_json() [8/9]

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

### 5.7.3.9 to\_json() [9/9]

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

## 5.8 wr22::regex\_parser::span Namespace Reference

### Classes

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

### Functions

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

### 5.8.1 Function Documentation

#### 5.8.1.1 operator<<()

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

#### 5.8.1.2 to\_json()

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

## 5.9 wr22::regex\_parser::utils Namespace Reference

### Namespaces

- namespace [detail](#)

### Classes

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

### Functions

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

### 5.9.1 Function Documentation

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

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

Type deduction guideline for [Box](#) (std::unique\_ptr adoption).

### 5.9.1.2 Box() [2/2]

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

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

### 5.9.1.3 operator!=() [1/2]

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

Compare two compatible ADTs for non-equality.

### 5.9.1.4 operator!=() [2/2]

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

### 5.9.1.5 operator==() [1/2]

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

Compare two compatible ADTs for equality.

### 5.9.1.6 operator==() [2/2]

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

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

### Namespaces

- namespace [adt](#)

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

### Classes

- struct [MultiCallable](#)

## Chapter 6

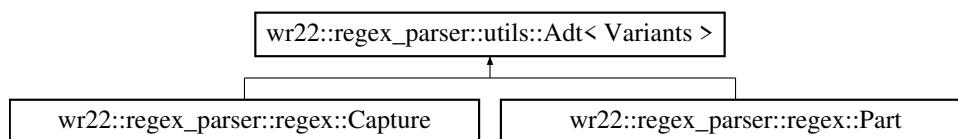
# Class Documentation

### 6.1 wr22::regex\_parser::utils::Adt< Variants > Class Template Reference

A helper class that simplifies creation of algebraic data types.

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

Inheritance diagram for wr22::regex\_parser::utils::Adt< Variants >:



#### Public Types

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

#### Public Member Functions

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

## Protected Attributes

- [VariantType m\\_variant](#)

### 6.1.1 Detailed Description

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

A helper class that simplifies creation of algebraic data types.

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

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

### 6.1.2 Member Typedef Documentation

#### 6.1.2.1 VariantType

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

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

### 6.1.3 Constructor & Destructor Documentation

#### 6.1.3.1 Adt()

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

Constructor for each of the variants.

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

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



## 6.1.4 Member Function Documentation

### 6.1.4.1 as\_variant() [1/2]

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

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

### 6.1.4.2 as\_variant() [2/2]

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

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

### 6.1.4.3 visit() [1/2]

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

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

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

### 6.1.4.4 visit() [2/2]

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

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

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

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

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

## 6.1.5 Member Data Documentation

### 6.1.5.1 m\_variant

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

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

- include/wr22/regex\_parser/utils/[adt.hpp](#)

## 6.2 wr22::regex\_parser::regex::part::Alternatives Struct Reference

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

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

### Public Member Functions

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

### Public Attributes

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

### Static Public Attributes

- static constexpr const char \* [code\\_name](#) = "alternatives"

### 6.2.1 Detailed Description

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

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

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

### 6.2.2 Constructor & Destructor Documentation

### 6.2.2.1 Alternatives()

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

## 6.2.3 Member Function Documentation

### 6.2.3.1 operator==( )

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

## 6.2.4 Member Data Documentation

### 6.2.4.1 alternatives

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

The list of the alternatives.

### 6.2.4.2 code\_name

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

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

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

## 6.3 wr22::regex\_parser::utils::Box< T > Class Template Reference

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

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

## Public Member Functions

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

## Static Public Member Functions

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

### 6.3.1 Detailed Description

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

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

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

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

### 6.3.2 Constructor & Destructor Documentation

#### 6.3.2.1 `Box()` [1/3]

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

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

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

### 6.3.2.2 Box() [2/3]

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

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

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

```
Box(std::make_unique<T>(args...))
```

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

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

### 6.3.2.3 Box() [3/3]

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

Copy constructor.

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

#### Parameters

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

#### Exceptions

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

## 6.3.3 Member Function Documentation

### 6.3.3.1 construct\_in\_place()

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

Construct a value on the heap in place.

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

### 6.3.3.2 operator\*() [1/2]

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

Dereferencing operator: obtain a reference to the stored value.

#### Exceptions

<a href="#">BoxIsEmpty</a>	if this <a href="#">Box</a> does not contain a value at the moment.
----------------------------	---

### 6.3.3.3 operator\*() [2/2]

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

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

#### Exceptions

<a href="#">BoxIsEmpty</a>	if this <a href="#">Box</a> does not contain a value at the moment.
----------------------------	---

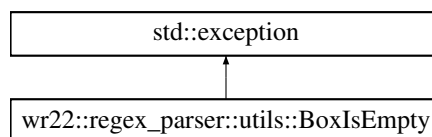
The documentation for this class was generated from the following file:

- include/wr22/regex\_parser/utils/[box.hpp](#)

## 6.4 wr22::regex\_parser::utils::BoxIsEmpty Struct Reference

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

Inheritance diagram for wr22::regex\_parser::utils::BoxIsEmpty:



### Public Member Functions

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

### 6.4.1 Member Function Documentation

#### 6.4.1.1 what()

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

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

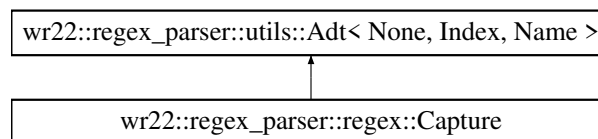
- include/wr22/regex\_parser/utils/[box.hpp](#)
- src/utils/[box.cpp](#)

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

Group capture behavior.

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

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



### Additional Inherited Members

#### 6.5.1 Detailed Description

Group capture behavior.

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

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

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

## 6.6 wr22::regex\_parser::regex::part::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](#)

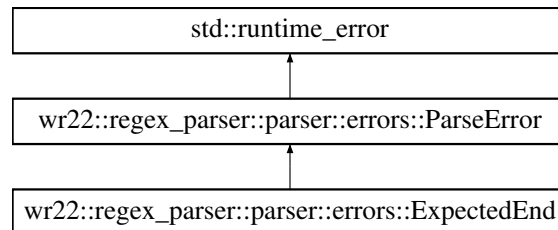


## 6.7 wr22::regex\_parser::parser::errors::ExpectedEnd Class Reference

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:

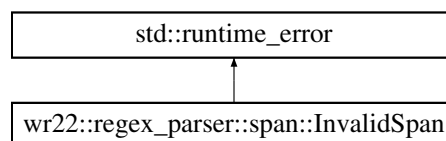
- include/wr22/regex\_parser/regex/capture.hpp

## 6.10 wr22::regex\_parser::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

### 6.10.1 Detailed Description

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

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

### 6.10.2 Constructor & Destructor Documentation

#### 6.10.2.1 InvalidSpan()

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

### 6.10.3 Member Data Documentation

#### 6.10.3.1 begin

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

#### 6.10.3.2 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](#)

## 6.11 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"

### 6.11.1 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.

### 6.11.2 Constructor & Destructor Documentation

#### 6.11.2.1 Literal()

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

### 6.11.3 Member Function Documentation

#### 6.11.3.1 operator==( )

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

### 6.11.4 Member Data Documentation

#### 6.11.4.1 character

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

#### 6.11.4.2 code\_name

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

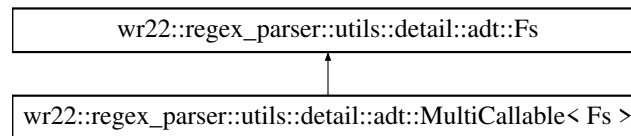
The documentation for this struct was generated from the following files:

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

## 6.12 wr22::regex\_parser::utils::detail::adt::MultiCallable< Fs > Struct Template Reference

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

Inheritance diagram for wr22::regex\_parser::utils::detail::adt::MultiCallable< Fs >:



### Public Member Functions

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

### 6.12.1 Constructor & Destructor Documentation

#### 6.12.1.1 MultiCallable()

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

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

- [include/wr22/regex\\_parser/utils/adt.hpp](#)

## 6.13 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"

### 6.13.1 Detailed Description

Denotes a group captured by name.

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

### 6.13.2 Constructor & Destructor Documentation

#### 6.13.2.1 Name()

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

### 6.13.3 Member Function Documentation

#### 6.13.3.1 operator==( )

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

### 6.13.4 Member Data Documentation

#### 6.13.4.1 code\_name

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

#### 6.13.4.2 flavor

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

#### 6.13.4.3 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

## 6.14 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"

#### 6.14.1 Detailed Description

Denotes an non-capturing group.

#### 6.14.2 Constructor & Destructor Documentation

### 6.14.2.1 None()

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

## 6.14.3 Member Function Documentation

### 6.14.3.1 operator==(

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

## 6.14.4 Member Data Documentation

### 6.14.4.1 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

## 6.15 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"

### 6.15.1 Detailed Description

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

### 6.15.2 Constructor & Destructor Documentation

#### 6.15.2.1 Optional()

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

Convenience constructor.

### 6.15.3 Member Function Documentation

#### 6.15.3.1 operator==( )

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

### 6.15.4 Member Data Documentation

#### 6.15.4.1 code\_name

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

#### 6.15.4.2 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:

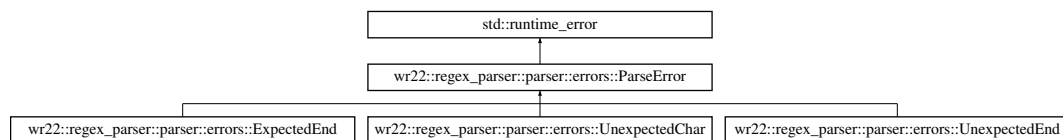
- include/wr22/regex\_parser/regex/[part.hpp](#)
- src/regex/[part.cpp](#)

## 6.16 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:



### 6.16.1 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](#)

## 6.17 wr22::regex\_parser::parser::Parser<Iter, Sentinel> Class Template Reference

A regex parser.

## Public Member Functions

- [Parser](#) (Iter begin, Sentinel end)  
*Constructor.*
- void [expect\\_end](#) ()  
*Ensure that the parser has consumed all of the input.*
- [regex::SpannedPart](#) [parse\\_regex](#) ()  
*Parse a regex consuming part of the remaining input.*
- [regex::SpannedPart](#) [parse\\_alternatives](#) ()  
*Intermediate rule: parse a pipe-separated list of alternatives (e.g.*
- [regex::SpannedPart](#) [parse\\_sequence](#) ()  
*Intermediate rule: parse a sequence of atoms (e.g.*
- [regex::SpannedPart](#) [parse\\_sequence\\_or\\_empty](#) ()  
*Intermediate rule: parse a possibly empty sequence of atoms.*
- [regex::SpannedPart](#) [parse\\_atom](#) ()  
*Intermediate rule: parse an atom.*
- [regex::SpannedPart](#) [parse\\_wildcard](#) ()  
*Intermediate rule: parse a wildcard (.*
- [regex::SpannedPart](#) [parse\\_char\\_literal](#) ()  
*Intermediate rule: parse a character literal.*
- [regex::SpannedPart](#) [parse\\_group](#) ()  
*Intermediate rule: parse a parenthesized group (any capture variant).*
- std::pair< std::string, [Span](#) > [parse\\_group\\_name](#) ()  
*Intermediate rule: parse a group name.*

### 6.17.1 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.

### 6.17.2 Constructor & Destructor Documentation

#### 6.17.2.1 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.

## 6.17.3 Member Function Documentation

### 6.17.3.1 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

<a href="#"><code>errors::ExpectedEnd</code></a>	if this is not the case.
--	--------------------------

### 6.17.3.2 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

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

### 6.17.3.3 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.
---------------------------------	--------------------------------

**6.17.3.4 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.
------------------------------------	--

**6.17.3.5 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`).

**6.17.3.6 parse\_group\_name()**

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

Intermediate rule: parse a group name.

**Returns**

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



### 6.17.3.7 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

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

### 6.17.3.8 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.
---------------------------------	--------------------------------

### 6.17.3.9 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.
---------------------------------	--------------------------------

**6.17.3.10 parse\_wildcard()**

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

Intermediate rule: parse a wildcard ( .

).

**Returns**

the wildcard AST node.

**Exceptions**

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

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

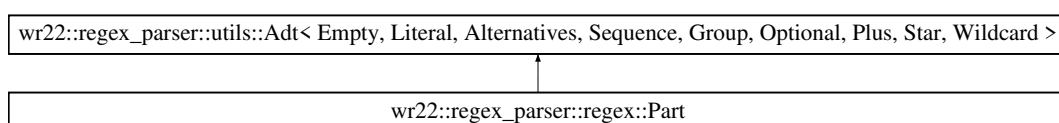
- [src/parser/regex.cpp](#)

**6.18 wr22::regex\_parser::regex::Part Class Reference**

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

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

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



## Additional Inherited Members

### 6.18.1 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 [parts::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`

## 6.19 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"

### 6.19.1 Detailed Description

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

### 6.19.2 Constructor & Destructor Documentation

#### 6.19.2.1 Plus()

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

Convenience constructor.

### 6.19.3 Member Function Documentation

#### 6.19.3.1 operator==( )

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

### 6.19.4 Member Data Documentation

#### 6.19.4.1 code\_name

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

#### 6.19.4.2 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

## 6.20 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"

#### 6.20.1 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`.

#### 6.20.2 Constructor & Destructor Documentation

##### 6.20.2.1 Sequence()

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

#### 6.20.3 Member Function Documentation

##### 6.20.3.1 operator==( )

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

## 6.20.4 Member Data Documentation

### 6.20.4.1 code\_name

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

### 6.20.4.2 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

## 6.21 wr22::regex\_parser::span::Span Class Reference

Character position range in the input string.

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

### Public Member Functions

- 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.*

### 6.21.1 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.

### 6.21.2 Member Function Documentation

#### 6.21.2.1 `begin()`

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

Get the `begin` position of the span.

#### 6.21.2.2 `end()`

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

Get the `end` position of the span.

#### 6.21.2.3 `length()`

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

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

#### 6.21.2.4 `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`.

#### 6.21.2.5 `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> .
--------------------	----------------------------------

**6.21.2.6 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.
--------------------	---

**6.21.2.7 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.
--------------------	---

**6.21.2.8 operator"!="()**

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



### 6.21.2.9 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](#)

## 6.22 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 [SpannedPart](#)s 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.*

### 6.22.1 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 [SpannedPart](#)s themselves).

### 6.22.2 Constructor & Destructor Documentation

#### 6.22.2.1 SpannedPart()

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

## 6.22.3 Member Function Documentation

### 6.22.3.1 operator"!=()

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

### 6.22.3.2 operator==(

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

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

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

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

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

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

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

### 6.22.3.5 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](#)

## 6.23 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"

#### 6.23.1 Detailed Description

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

#### 6.23.2 Constructor & Destructor Documentation

##### 6.23.2.1 Star()

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

Convenience constructor.

#### 6.23.3 Member Function Documentation

##### 6.23.3.1 operator==()

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

## 6.23.4 Member Data Documentation

### 6.23.4.1 code\_name

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

### 6.23.4.2 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:

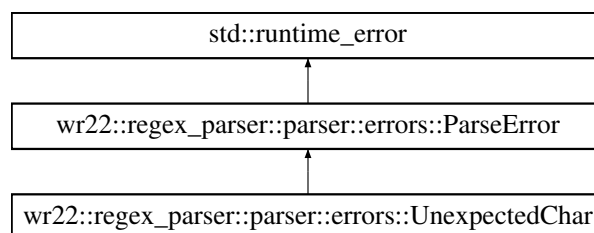
- include/wr22/regex\_parser/regex/[part.hpp](#)
- src/regex/[part.cpp](#)

## 6.24 wr22::regex\_parser::parser::errors::UnexpectedChar Class Reference

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

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

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



## Public Member Functions

- [UnexpectedChar](#) (size\_t [position](#), char32\_t [char\\_got](#), std::string [expected](#))  
*Constructor.*
- size\_t [position](#) () const  
*Get the input position. See the constructor docs for a more detailed description.*
- char32\_t [char\\_got](#) () const  
*Get the character the parser has received.*
- const std::string & [expected](#) () const  
*Get the description of expected characters.*

### 6.24.1 Detailed Description

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

### 6.24.2 Constructor & Destructor Documentation

#### 6.24.2.1 UnexpectedChar()

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

Constructor.

##### Parameters

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

### 6.24.3 Member Function Documentation

#### 6.24.3.1 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.

#### 6.24.3.2 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.

### 6.24.3.3 position()

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

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

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

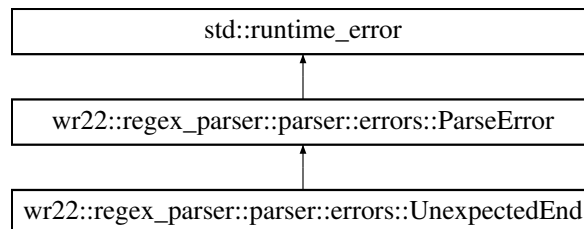
- include/wr22/regex\_parser/parser/errors.hpp
- src/parser/errors.cpp

## 6.25 wr22::regex\_parser::parser::errors::UnexpectedEnd Class Reference

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

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

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



### Public Member Functions

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

### 6.25.1 Detailed Description

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

### 6.25.2 Constructor & Destructor Documentation

#### 6.25.2.1 UnexpectedEnd()

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

Constructor.

## Parameters

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

## 6.25.3 Member Function Documentation

### 6.25.3.1 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.

### 6.25.3.2 position()

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

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

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

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

## 6.26 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"

### 6.26.1 Detailed Description

A regex part specifying any single character ( . ).

### 6.26.2 Constructor & Destructor Documentation

#### 6.26.2.1 Wildcard()

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

### 6.26.3 Member Function Documentation

#### 6.26.3.1 operator==( )

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

### 6.26.4 Member Data Documentation

#### 6.26.4.1 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 <exception>
#include <stdexcept>
#include <string>
```

#### Classes

- struct [wr22::regex\\_parser::parser::errors::ParseError](#)  
*The base class for parse errors.*
- class [wr22::regex\\_parser::parser::errors::UnexpectedEnd](#)  
*The error when the parser hit the end of the input earlier than it expected.*
- class [wr22::regex\\_parser::parser::errors::ExpectedEnd](#)  
*The error when the parser expected the input to end, but it did not.*
- class [wr22::regex\\_parser::parser::errors::UnexpectedChar](#)  
*The error when the parser got a character it didn't expect at the current position.*

#### 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 // STL
4 #include <exception>
5 #include <stdexcept>
6 #include <string>
7
8 namespace wr22::regex_parser::parser::errors {
9
10 struct ParseError : public std::runtime_error {
11     using std::runtime_error::runtime_error;
12 };
13
14 class UnexpectedEnd : public ParseError {
15 public:
16     UnexpectedEnd(size_t position, std::string expected);
17
18     size_t position() const;
19     const std::string& expected() const;
20
21 private:
22     size_t m_position;
23     std::string m_expected;
24 };
25
26 class ExpectedEnd : public ParseError {
27 public:
28     ExpectedEnd(size_t position, char32_t char_got);
29
30     size_t position() const;
31     char32_t char_got() const;
32
33 private:
34     size_t m_position;
35     char32_t m_char_got;
36 };
37
38 class UnexpectedChar : public ParseError {
39 public:
40     UnexpectedChar(size_t position, char32_t char_got, std::string expected);
41
42     size_t position() const;
43     char32_t char_got() const;
44     const std::string& expected() const;
45
46 private:
47     size_t m_position;
48     char32_t m_char_got;
49     std::string m_expected;
50 };
51
52 } // 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/regex_parser/utils/adt.hpp>
#include <iosfwd>
#include <string>
#include <nlohmann/json.hpp>
```

### Classes

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

### Namespaces

- namespace [wr22](#)
- namespace [wr22::regex\\_parser](#)
- namespace [wr22::regex\\_parser::regex](#)
- namespace [wr22::regex\\_parser::regex::capture](#)

### Typedefs

- using [wr22::regex\\_parser::regex::capture::Adt](#) = [utils::Adt](#)< None, Index, Name >

### Functions

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

## 7.6 capture.hpp

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

```

1 #pragma once
2
3 // wr22
4 #include <wr22/regex_parser/regex/named_capture_flavor.hpp>
5 #include <wr22/regex_parser/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 //
47 class Capture : public capture::Adt {
48 public:
49     using capture::Adt::Adt;
50 };
51
52 std::ostream& operator<<(std::ostream& out, const Capture& capture);
53 void to_json(nlohmann::json& j, const Capture& capture);
54
55 } // namespace wr22::regex_parser::regex

```

## 7.7 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)

## 7.8 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
```

## 7.9 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/span/span.hpp>
#include <wr22/regex_parser/utils/adt.hpp>
#include <wr22/regex_parser/utils/box.hpp>
#include <iosfwd>
#include <memory>
#include <vector>
#include <nlohmann/json.hpp>
```

## Classes

- struct [wr22::regex\\_parser::regex::part::Empty](#)  
*An empty regex part.*
- struct [wr22::regex\\_parser::regex::part::Literal](#)  
*An 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  $(.)$ .*
- 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 [SpannedPart](#)s 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 >

## 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::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`.*

## 7.10 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/span/span.hpp>
7 #include <wr22/regex_parser/utils/adt.hpp>
8 #include <wr22/regex_parser/utils/box.hpp>
9
10 // stl
11 #include <iosfwd>
12 #include <memory>
13 #include <vector>
14
15 // nlohmann
16 #include <nlohmann/json.hpp>
17
18 namespace wr22::regex_parser::regex {
19
20 // Forward declarations.
21 class Part;
22 class SpannedPart;
23
24 namespace part {
25     struct Empty {
26         explicit Empty() = default;
27         bool operator==(const Empty& rhs) const = default;
28         static constexpr const char* code_name = "empty";
29     };
30     void to_json(nlohmann::json& j, const Empty& part);
31
32     struct Literal {
33         explicit Literal(char32_t character);
34         bool operator==(const Literal& rhs) const = default;
35         static constexpr const char* code_name = "literal";
36
37         char32_t character;
38     };
39     void to_json(nlohmann::json& j, const Literal& part);
40
41     struct Alternatives {
42         explicit Alternatives(std::vector<SpannedPart> alternatives);
43         bool operator==(const Alternatives& rhs) const = default;
44         static constexpr const char* code_name = "alternatives";
45
46         std::vector<SpannedPart> alternatives;
47     };
48     void to_json(nlohmann::json& j, const Alternatives& part);
49
50     struct Sequence {
51         explicit Sequence(std::vector<SpannedPart> items);
52         bool operator==(const Sequence& rhs) const = default;
53         static constexpr const char* code_name = "sequence";
54
55         std::vector<SpannedPart> items;
56     };
57     void to_json(nlohmann::json& j, const Sequence& part);
58
59     struct Group {
60         explicit Group(Capture capture, SpannedPart inner);
61         bool operator==(const Group& rhs) const = default;
62         static constexpr const char* code_name = "group";
63
64         Capture capture;
65         utils::Box<SpannedPart> inner;
66     };
67     void to_json(nlohmann::json& j, const Group& part);
68
69     struct Optional {
70         explicit Optional(SpannedPart inner);
71         bool operator==(const Optional& rhs) const = default;
72         static constexpr const char* code_name = "optional";
73
74         utils::Box<SpannedPart> inner;
75     };
76     void to_json(nlohmann::json& j, const Optional& part);
77
78     struct Plus {
79         explicit Plus(SpannedPart inner);
80         bool operator==(const Plus& rhs) const = default;
81         static constexpr const char* code_name = "plus";
82     };
83 }

```

```

128         utils::Box<SpannedPart> inner;
129     };
130     void to_json(nlohmann::json& j, const Plus& part);
131
132     struct Star {
133         explicit Star(SpannedPart inner);
134         bool operator==(const Star& rhs) const = default;
135         static constexpr const char* code_name = "star";
136     };
137     void to_json(nlohmann::json& j, const Star& part);
138
139     struct Wildcard {
140         explicit Wildcard() = default;
141         bool operator==(const Wildcard& rhs) const = default;
142         static constexpr const char* code_name = "wildcard";
143     };
144     void to_json(nlohmann::json& j, const Wildcard& part);
145
146     using Adt = utils::
147         Adt<Empty, Literal, Alternatives, Sequence, Group, Optional, Plus, Star, Wildcard>;
148 } // namespace part
149
150 class Part : public part::Adt {
151 public:
152     using part::Adt::Adt;
153 };
154 void to_json(nlohmann::json& j, const Part& part);
155
156 class SpannedPart {
157 public:
158     explicit SpannedPart(Part part, span::Span span);
159
160     bool operator==(const SpannedPart& other) const = default;
161     bool operator!=(const SpannedPart& other) const = default;
162
163     const Part& part() const;
164     Part& part();
165
166     span::Span span() const;
167
168 private:
169     Part m_part;
170     span::Span m_span;
171 };
172 void to_json(nlohmann::json& j, const SpannedPart& part);
173
174 std::ostream& operator<<(std::ostream& out, const SpannedPart& part);
175
176 } // namespace wr22::regex_parser::regex

```

## 7.11 include/wr22/regex\_parser/span/span.hpp File Reference

```

#include <cstddef>
#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`)

## 7.12 span.hpp

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

```
1 #pragma once
2
3 // stl
4 #include <cstddef>
5 #include <stdexcept>
6 #include <ostream>
7
8 // nlohmann
9 #include <nlohmann/json.hpp>
10
11 namespace wr22::regex_parser::span {
12
13     struct InvalidSpan : public std::runtime_error {
14         InvalidSpan(size_t begin, size_t end);
15
16         size_t begin;
17         size_t end;
18     };
19
20     class Span {
21     public:
22         static Span make_empty(size_t position);
23
24         static Span make_single_position(size_t position);
25
26         static Span make_from_positions(size_t begin, size_t end);
27
28         static Span make_with_length(size_t begin, size_t length);
29
30         size_t length() const;
31
32         size_t begin() const;
33
34         size_t end() const;
35
36         bool operator==(const Span& other) const = default;
37         bool operator!=(const Span& other) const = default;
38
39     private:
40         explicit Span(size_t begin, size_t end);
41
42         size_t m_begin;
43         size_t m_end;
44     };
45
46     std::ostream& operator<<(std::ostream& out, Span span);
47     void to_json(nlohmann::json& j, Span span);
48 } // namespace wr22::regex_parser::span
```

## 7.13 include/wr22/regex\_parser/utils/adt.hpp File Reference

```
#include <utility>
#include <variant>
```

## Classes

- struct `wr22::regex_parser::utils::detail::adt::MultiCallable< Fs >`
- class `wr22::regex_parser::utils::Adt< Variants >`  
*A helper class that simplifies creation of algebraic data types.*

## Namespaces

- namespace [wr22](#)
- namespace [wr22::regex\\_parser](#)
- namespace [wr22::regex\\_parser::utils](#)
- namespace [wr22::regex\\_parser::utils::detail](#)
- namespace [wr22::regex\\_parser::utils::detail::adt](#)

## Functions

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

## 7.14 adt.hpp

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

```

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

```

```

106 };
107
108 template <typename... Variants>
109 bool operator==(const Adt<Variants...>& lhs, const Adt<Variants...>& rhs) {
110     return lhs.as_variant() == rhs.as_variant();
111 }
112
113 template <typename... Variants>
114 bool operator!=(const Adt<Variants...>& lhs, const Adt<Variants...>& rhs) {
115     return !(lhs == rhs);
116 }
117
118 // namespace wr22::regex_parser::utils
119
120 }

```

## 7.15 include/wr22/regex\_parser/utils/box.hpp File Reference

```

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

```

### Classes

- struct [wr22::regex\\_parser::utils::BoxIsEmpty](#)
- class [wr22::regex\\_parser::utils::Box< T >](#)  
*A copyable and equality-comparable wrapper around `std::unique_ptr`.*

### Namespaces

- namespace [wr22](#)
- namespace [wr22::regex\\_parser](#)
- namespace [wr22::regex\\_parser::utils](#)

### Functions

- template<typename T >  
[wr22::regex\\_parser::utils::Box](#) (T &&value) -> Box< T >  
*Type deduction guideline for [Box](#) (value initialization).*
- template<typename T >  
[wr22::regex\\_parser::utils::Box](#) (std::unique\_ptr< T > ptr) -> Box< T >  
*Type deduction guideline for [Box](#) (std::unique\_ptr adoption).*
- template<typename T, typename U >  
bool [wr22::regex\\_parser::utils::operator==](#) (const Box< T > &lhs, const Box< U > &rhs)
- template<typename T, typename U >  
bool [wr22::regex\\_parser::utils::operator!=](#) (const Box< T > &lhs, const Box< U > &rhs)

## 7.16 box.hpp

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

```

1 #pragma once
2
3 // stl
4 #include <exception>
5 #include <memory>
6 #include <utility>
7
8 namespace wr22::regex_parser::utils {
9
10 struct BoxIsEmpty : public std::exception {
11     const char* what() const noexcept override;
12 };
13
14 template <typename T>
15 class Box {
16 public:
17     explicit Box(T&& value) : m_ptr(std::make_unique<T>(std::forward<T>(value))) {}
18
19     explicit Box(std::unique_ptr<T> ptr) : m_ptr(std::move(ptr)) {}
20
21     template <typename Dummy = T>
22     Box(const Box& other) : m_ptr(std::make_unique<T>(*other)) {}
23
24     template <typename... Args>
25     static Box<T> construct_in_place(Args&&... args) {
26         return Box(std::make_unique<T>(std::forward<Args>(args)...));
27     }
28
29     const T& operator*() const {
30         if (m_ptr == nullptr) {
31             throw BoxIsEmpty{};
32         }
33         return *m_ptr;
34     }
35
36     T& operator*() {
37         if (m_ptr == nullptr) {
38             throw BoxIsEmpty{};
39         }
40         return *m_ptr;
41     }
42
43 private:
44     std::unique_ptr<T> m_ptr;
45 };
46
47 template <typename T>
48 Box(T&& value) -> Box<T>;
49
50 template <typename T>
51 Box(std::unique_ptr<T> ptr) -> Box<T>;
52
53 template <typename T, typename U>
54 bool operator==(const Box<T>& lhs, const Box<U>& rhs) {
55     return *lhs == *rhs;
56 }
57
58 template <typename T, typename U>
59 bool operator!=(const Box<T>& lhs, const Box<U>& rhs) {
60     return !(lhs == rhs);
61 }
62
63 } // namespace wr22::regex_parser::utils

```

## 7.17 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 <boost/locale/utf.hpp>
#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`)

## 7.18 src/parser/errors.cpp File Reference

```
#include <wr22/regex_parser/parser/errors.hpp>
#include <fmt/core.h>
#include <boost/locale/encoding_utf.hpp>
```

## Namespaces

- namespace [wr22](#)
- namespace [wr22::regex\\_parser](#)
- namespace [wr22::regex\\_parser::parser](#)
- namespace [wr22::regex\\_parser::parser::errors](#)

## 7.19 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 <optional>
#include <stdexcept>
#include <string>
#include <vector>
#include <boost/locale/encoding_utf.hpp>
#include <boost/locale/utf.hpp>
```

## 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.*

## 7.20 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)

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

```
#include "wr22/regex_parser/span/span.hpp"
#include <boost/locale/encoding_utf.hpp>
#include <wr22/regex_parser/regex/part.hpp>
#include <iterator>
#include <ostream>
#include <boost/locale/utf.hpp>
#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::to\_json (nlohmann::json &j, const Part &part)`
- `void wr22::regex\_parser::regex::to\_json (nlohmann::json &j, const SpannedPart &part)`

## 7.22 src/span/span.cpp File Reference

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

## Namespaces

- namespace [wr22](#)
- namespace [wr22::regex\\_parser](#)
- namespace [wr22::regex\\_parser::span](#)

## Functions

- `std::ostream & wr22::regex\_parser::span::operator<< (std::ostream &out, Span span)`
- `void wr22::regex\_parser::span::to\_json (nlohmann::json &j, Span span)`

## 7.23 src/utls/box.cpp File Reference

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

## Namespaces

- namespace [wr22](#)
- namespace [wr22::regex\\_parser](#)
- namespace [wr22::regex\\_parser::utls](#)





# Index

Adt  
    wr22::regex\_parser::regex::capture, [13](#)  
    wr22::regex\_parser::regex::part, [15](#)  
    wr22::regex\_parser::utils::Adt< Variants >, [22](#)

Alternatives  
    wr22::regex\_parser::regex::part::Alternatives, [24](#)

alternatives  
    wr22::regex\_parser::regex::part::Alternatives, [25](#)

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

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

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

as\_variant  
    wr22::regex\_parser::utils::Adt< Variants >, [23](#)

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

Box  
    wr22::regex\_parser::utils, [18](#)  
    wr22::regex\_parser::utils::Box< T >, [26](#), [27](#)

capture  
    wr22::regex\_parser::regex::part::Group, [33](#)

char\_got  
    wr22::regex\_parser::parser::errors::ExpectedEnd, [32](#)  
    wr22::regex\_parser::parser::errors::UnexpectedChar, [59](#)

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

code\_name  
    wr22::regex\_parser::regex::capture::Index, [35](#)  
    wr22::regex\_parser::regex::capture::Name, [39](#)  
    wr22::regex\_parser::regex::capture::None, [41](#)  
    wr22::regex\_parser::regex::part::Alternatives, [25](#)  
    wr22::regex\_parser::regex::part::Empty, [30](#)  
    wr22::regex\_parser::regex::part::Group, [33](#)  
    wr22::regex\_parser::regex::part::Literal, [37](#)  
    wr22::regex\_parser::regex::part::Optional, [42](#)  
    wr22::regex\_parser::regex::part::Plus, [50](#)  
    wr22::regex\_parser::regex::part::Sequence, [52](#)  
    wr22::regex\_parser::regex::part::Star, [58](#)  
    wr22::regex\_parser::regex::part::Wildcard, [62](#)

construct\_in\_place  
    wr22::regex\_parser::utils::Box< T >, [27](#)

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

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

expect\_end  
    wr22::regex\_parser::parser::Parser< Iter, Sentinel >, [45](#)

expected  
    wr22::regex\_parser::parser::errors::UnexpectedChar, [59](#)  
    wr22::regex\_parser::parser::errors::UnexpectedEnd, [61](#)

ExpectedEnd  
    wr22::regex\_parser::parser::errors::ExpectedEnd, [31](#)

flavor  
    wr22::regex\_parser::regex::capture::Name, [40](#)

Group  
    wr22::regex\_parser::regex::part::Group, [33](#)

include/wr22/regex\_parser/parser/errors.hpp, [63](#), [64](#)  
include/wr22/regex\_parser/parser/regex.hpp, [64](#), [65](#)  
include/wr22/regex\_parser/regex/capture.hpp, [65](#), [66](#)  
include/wr22/regex\_parser/regex/named\_capture\_flavor.hpp, [66](#), [67](#)  
include/wr22/regex\_parser/regex/part.hpp, [67](#), [69](#)  
include/wr22/regex\_parser/span/span.hpp, [70](#), [71](#)  
include/wr22/regex\_parser/utils/adt.hpp, [71](#), [72](#)  
include/wr22/regex\_parser/utils/box.hpp, [73](#), [74](#)

Index  
    wr22::regex\_parser::regex::capture::Index, [34](#)

inner  
    wr22::regex\_parser::regex::part::Group, [34](#)  
    wr22::regex\_parser::regex::part::Optional, [42](#)  
    wr22::regex\_parser::regex::part::Plus, [50](#)  
    wr22::regex\_parser::regex::part::Star, [58](#)

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

items  
    wr22::regex\_parser::regex::part::Sequence, [52](#)

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

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

m\_variant  
    wr22::regex\_parser::utils::Adt< Variants >, [24](#)

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
- MultiCallable
  - wr22::regex\_parser::utils::detail::adt::MultiCallable< Fs >, 38
- Name
  - wr22::regex\_parser::regex::capture::Name, 39
- name
  - wr22::regex\_parser::regex::capture::Name, 40
- NamedCaptureFlavor
  - wr22::regex\_parser::regex, 11
- None
  - wr22::regex\_parser::regex::capture::None, 40
- operator!=
  - wr22::regex\_parser::regex::SpannedPart, 56
  - wr22::regex\_parser::span::Span, 54
  - wr22::regex\_parser::utils, 19
- operator<<
  - wr22::regex\_parser::regex, 12
  - wr22::regex\_parser::span, 17
- operator\*
  - wr22::regex\_parser::utils::Box< T >, 27, 28
- operator==
  - wr22::regex\_parser::regex::capture::Index, 35
  - wr22::regex\_parser::regex::capture::Name, 39
  - wr22::regex\_parser::regex::capture::None, 41
  - wr22::regex\_parser::regex::part::Alternatives, 25
  - wr22::regex\_parser::regex::part::Empty, 30
  - wr22::regex\_parser::regex::part::Group, 33
  - wr22::regex\_parser::regex::part::Literal, 37
  - wr22::regex\_parser::regex::part::Optional, 42
  - wr22::regex\_parser::regex::part::Plus, 50
  - wr22::regex\_parser::regex::part::Sequence, 51
  - wr22::regex\_parser::regex::part::Star, 57
  - wr22::regex\_parser::regex::part::Wildcard, 62
  - wr22::regex\_parser::regex::SpannedPart, 56
  - wr22::regex\_parser::span::Span, 54
  - wr22::regex\_parser::utils, 19
- Optional
  - wr22::regex\_parser::regex::part::Optional, 42
- parse\_alternatives
  - wr22::regex\_parser::parser::Parser< Iter, Sentinel >, 45
- parse\_atom
  - wr22::regex\_parser::parser::Parser< Iter, Sentinel >, 45
- parse\_char\_literal
  - wr22::regex\_parser::parser::Parser< Iter, Sentinel >, 46
- parse\_group
  - wr22::regex\_parser::parser::Parser< Iter, Sentinel >, 46
- 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 >, 47
- parse\_sequence\_or\_empty
  - wr22::regex\_parser::parser::Parser< Iter, Sentinel >, 47
- parse\_wildcard
  - wr22::regex\_parser::parser::Parser< Iter, Sentinel >, 48
- Parser
  - wr22::regex\_parser::parser, 10
  - wr22::regex\_parser::parser::Parser< Iter, Sentinel >, 44
- part
  - wr22::regex\_parser::regex::SpannedPart, 56
- Plus
  - wr22::regex\_parser::regex::part::Plus, 50
- position
  - wr22::regex\_parser::parser::errors::ExpectedEnd, 32
  - wr22::regex\_parser::parser::errors::UnexpectedChar, 59
  - wr22::regex\_parser::parser::errors::UnexpectedEnd, 61
- Sequence
  - wr22::regex\_parser::regex::part::Sequence, 51
- span
  - wr22::regex\_parser::regex::SpannedPart, 56
- SpannedPart
  - wr22::regex\_parser::regex::SpannedPart, 55
- src/parser/capture.cpp, 74
- src/parser/errors.cpp, 75
- src/parser/regex.cpp, 75
- src/regex/named\_capture\_flavor.cpp, 76
- src/regex/part.cpp, 76
- src/span/span.cpp, 77
- src/utils/box.cpp, 77
- Star
  - wr22::regex\_parser::regex::part::Star, 57
- to\_json
  - wr22::regex\_parser::regex, 12, 13
  - wr22::regex\_parser::regex::capture, 14
  - wr22::regex\_parser::regex::part, 15–17
  - wr22::regex\_parser::span, 17
- UnexpectedChar
  - wr22::regex\_parser::parser::errors::UnexpectedChar, 59

- UnexpectedEnd
  - wr22::regex\_parser::parser::errors::UnexpectedEnd, 60
- VariantType
  - wr22::regex\_parser::utils::Adt< Variants >, 22
- visit
  - wr22::regex\_parser::utils::Adt< Variants >, 23
- what
  - wr22::regex\_parser::utils::BoxIsEmpty, 28
- Wildcard
  - wr22::regex\_parser::regex::part::Wildcard, 62
- 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, 31
  - char\_got, 32
  - ExpectedEnd, 31
  - position, 32
- wr22::regex\_parser::parser::errors::ParseError, 43
- wr22::regex\_parser::parser::errors::UnexpectedChar, 58
  - char\_got, 59
  - expected, 59
  - position, 59
  - UnexpectedChar, 59
- wr22::regex\_parser::parser::errors::UnexpectedEnd, 60
  - expected, 61
  - position, 61
  - UnexpectedEnd, 60
- wr22::regex\_parser::parser::Parser< Iter, Sentinel >, 43
  - expect\_end, 45
  - parse\_alternatives, 45
  - parse\_atom, 45
  - parse\_char\_literal, 46
  - parse\_group, 46
  - parse\_group\_name, 46
  - parse\_regex, 46
  - parse\_sequence, 47
  - parse\_sequence\_or\_empty, 47
  - parse\_wildcard, 48
  - Parser, 44
- wr22::regex\_parser::regex, 11
  - Angles, 12
  - AnglesWithP, 12
  - Apostrophes, 12
  - NamedCaptureFlavor, 11
  - operator<<, 12
  - to\_json, 12, 13
- wr22::regex\_parser::regex::Capture, 29
- wr22::regex\_parser::regex::capture, 13
  - Adt, 13
  - to\_json, 14
- wr22::regex\_parser::regex::capture::Index, 34
  - code\_name, 35
  - Index, 34
  - operator==, 35
- wr22::regex\_parser::regex::capture::Name, 38
  - code\_name, 39
  - flavor, 40
  - Name, 39
  - name, 40
  - operator==, 39
- wr22::regex\_parser::regex::capture::None, 40
  - code\_name, 41
  - None, 40
  - operator==, 41
- wr22::regex\_parser::regex::Part, 48
- wr22::regex\_parser::regex::part, 14
  - Adt, 15
  - to\_json, 15–17
- wr22::regex\_parser::regex::part::Alternatives, 24
  - Alternatives, 24
  - alternatives, 25
  - code\_name, 25
  - operator==, 25
- wr22::regex\_parser::regex::part::Empty, 29
  - code\_name, 30
  - Empty, 30
  - operator==, 30
- wr22::regex\_parser::regex::part::Group, 32
  - capture, 33
  - code\_name, 33
  - Group, 33
  - inner, 34
  - operator==, 33
- wr22::regex\_parser::regex::part::Literal, 36
  - character, 37
  - code\_name, 37
  - Literal, 37
  - operator==, 37
- wr22::regex\_parser::regex::part::Optional, 41
  - code\_name, 42
  - inner, 42
  - operator==, 42
  - Optional, 42
- wr22::regex\_parser::regex::part::Plus, 49
  - code\_name, 50
  - inner, 50
  - operator==, 50
  - Plus, 50
- wr22::regex\_parser::regex::part::Sequence, 51
  - code\_name, 52
  - items, 52
  - operator==, 51
  - Sequence, 51
- wr22::regex\_parser::regex::part::Star, 57
  - code\_name, 58
  - inner, 58
  - operator==, 57
  - Star, 57
- wr22::regex\_parser::regex::part::Wildcard, 61

- code\_name, 62
- operator==, 62
- Wildcard, 62
- wr22::regex\_parser::regex::SpannedPart, 55
  - operator!=, 56
  - operator==, 56
  - part, 56
  - span, 56
  - SpannedPart, 55
- wr22::regex\_parser::span, 17
  - operator<<, 17
  - to\_json, 17
- wr22::regex\_parser::span::InvalidSpan, 35
  - begin, 36
  - end, 36
  - InvalidSpan, 36
- wr22::regex\_parser::span::Span, 52
  - begin, 53
  - end, 53
  - length, 53
  - make\_empty, 53
  - make\_from\_positions, 53
  - make\_single\_position, 54
  - make\_with\_length, 54
  - operator!=, 54
  - operator==, 54
- wr22::regex\_parser::utils, 18
  - Box, 18
  - operator!=, 19
  - operator==, 19
- wr22::regex\_parser::utils::Adt< Variants >, 21
  - Adt, 22
  - as\_variant, 23
  - m\_variant, 24
  - VariantType, 22
  - visit, 23
- wr22::regex\_parser::utils::Box< T >, 25
  - Box, 26, 27
  - construct\_in\_place, 27
  - operator\*, 27, 28
- wr22::regex\_parser::utils::BoxIsEmpty, 28
  - what, 28
- wr22::regex\_parser::utils::detail, 20
- wr22::regex\_parser::utils::detail::adt, 20
- wr22::regex\_parser::utils::detail::adt::MultiCallable< Fs
  - >, 38
  - MultiCallable, 38