# Writing Regexps 2021-22 / Regex Parser

Generated by Doxygen 1.9.3

**1 Namespace Index** 1

1.1 Namespace List . . . 1

**2 Hierarchical Index** 3

2.1 Class Hierarchy . . . 3

**3 Class Index** 5

3.1 Class List . . . 5

**4 File Index** 7

4.1 File List . . . 7

**5 Namespace Documentation** 9

5.1 wr22 Namespace Reference . . . 9

5.2 wr22::regex_parser Namespace Reference . . . 9

5.3 wr22::regex_parser::parser Namespace Reference . . . 9

5.3.1 Function Documentation . . . 10

5.3.1.1 parse_regex() . . . 10

5.3.1.2 Parser() . . . 10

5.4 wr22::regex_parser::parser::errors Namespace Reference . . . 10

5.5 wr22::regex_parser::regex Namespace Reference . . . 11

5.5.1 Enumeration Type Documentation . . . 11

5.5.1.1 NamedCaptureFlavor . . . 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

5.6 wr22::regex_parser::regex::capture Namespace Reference . . . 12

5.6.1 Typedef Documentation . . . 12

5.6.1.1 Adt . . . 13

5.7 wr22::regex_parser::regex::part Namespace Reference . . . 13

5.7.1 Detailed Description . . . 13

5.7.2 Typedef Documentation . . . 13

5.7.2.1 Adt . . . 13

5.8 wr22::regex_parser::utils Namespace Reference . . . 14

5.8.1 Typedef Documentation . . . 14

5.8.1.1 utf_traits . . . 14

5.8.2 Function Documentation . . . 15

5.8.2.1 Box() [1/2] . . . 15

5.8.2.2 Box() [2/2] . . . 15

5.8.2.3 operator"!=() [1/2] . . . 15

5.8.2.4 operator"!=() [2/2] . . . 15

5.8.2.5 operator==() [1/2] . . . 15

5.8.2.6 operator==() [2/2] . . . 16

5.9 wr22::regex_parser::utils::detail Namespace Reference . . . . . 16
5.10 wr22::regex_parser::utils::detail::adt Namespace Reference . . . . . 16

**6 Class Documentation** **17**

6.1 wr22::regex_parser::utils::Adt< Variants > Class Template Reference . . . . . 17
6.1.1 Detailed Description . . . . . 18
6.1.2 Member Typedef Documentation . . . . . 18
6.1.2.1 VariantType . . . . . 18
6.1.3 Constructor & Destructor Documentation . . . . . 18
6.1.3.1 Adt() . . . . . 18
6.1.4 Member Function Documentation . . . . . 19
6.1.4.1 as_variant() [1/2] . . . . . 19
6.1.4.2 as_variant() [2/2] . . . . . 19
6.1.4.3 visit() [1/2] . . . . . 19
6.1.4.4 visit() [2/2] . . . . . 19
6.1.5 Member Data Documentation . . . . . 20
6.1.5.1 m_variant . . . . . 20
6.2 wr22::regex_parser::regex::part::Alternatives Struct Reference . . . . . 20
6.2.1 Detailed Description . . . . . 20
6.2.2 Constructor & Destructor Documentation . . . . . 20
6.2.2.1 Alternatives() . . . . . 21
6.2.3 Member Function Documentation . . . . . 21
6.2.3.1 operator==() . . . . . 21
6.2.4 Member Data Documentation . . . . . 21
6.2.4.1 alternatives . . . . . 21
6.3 wr22::regex_parser::utils::Box< T > Class Template Reference . . . . . 21
6.3.1 Detailed Description . . . . . 22
6.3.2 Constructor & Destructor Documentation . . . . . 22
6.3.2.1 Box() [1/3] . . . . . 22
6.3.2.2 Box() [2/3] . . . . . 22
6.3.2.3 Box() [3/3] . . . . . 23
6.3.3 Member Function Documentation . . . . . 24
6.3.3.1 construct_in_place() . . . . . 24
6.3.3.2 operator*() [1/2] . . . . . 24
6.3.3.3 operator*() [2/2] . . . . . 24
6.4 wr22::regex_parser::utils::BoxIsEmpty Struct Reference . . . . . 25
6.4.1 Member Function Documentation . . . . . 25
6.4.1.1 what() . . . . . 25
6.5 wr22::regex_parser::regex::Capture Class Reference . . . . . 25
6.5.1 Detailed Description . . . . . 26
6.6 wr22::regex_parser::regex::part::Empty Struct Reference . . . . . 26
6.6.1 Detailed Description . . . . . 26

6.6.2 Constructor & Destructor Documentation . . . 26
6.6.2.1 Empty() . . . 26
6.6.3 Member Function Documentation . . . 26
6.6.3.1 operator==() . . . 27
6.7 wr22::regex_parser::parser::errors::ExpectedEnd Class Reference . . . 27
6.7.1 Detailed Description . . . 27
6.7.2 Constructor & Destructor Documentation . . . 27
6.7.2.1 ExpectedEnd() . . . 27
6.7.3 Member Function Documentation . . . 28
6.7.3.1 char_got() . . . 28
6.7.3.2 position() . . . 28
6.8 wr22::regex_parser::regex::part::Group Struct Reference . . . 28
6.8.1 Detailed Description . . . 29
6.8.2 Constructor & Destructor Documentation . . . 29
6.8.2.1 Group() . . . 29
6.8.3 Member Function Documentation . . . 29
6.8.3.1 operator==() . . . 29
6.8.4 Member Data Documentation . . . 29
6.8.4.1 capture . . . 29
6.8.4.2 inner . . . 30
6.9 wr22::regex_parser::regex::capture::Index Struct Reference . . . 30
6.9.1 Detailed Description . . . 30
6.9.2 Constructor & Destructor Documentation . . . 30
6.9.2.1 Index() . . . 30
6.9.3 Member Function Documentation . . . 30
6.9.3.1 operator==() . . . 30
6.10 wr22::regex_parser::utils::UnicodeStringView::InvalidUtf8 Struct Reference . . . 31
6.10.1 Detailed Description . . . 31
6.10.2 Member Function Documentation . . . 31
6.10.2.1 what() . . . 31
6.11 wr22::regex_parser::regex::part::Literal Struct Reference . . . 31
6.11.1 Detailed Description . . . 32
6.11.2 Constructor & Destructor Documentation . . . 32
6.11.2.1 Literal() . . . 32
6.11.3 Member Function Documentation . . . 32
6.11.3.1 operator==() . . . 32
6.11.4 Member Data Documentation . . . 32
6.11.4.1 character . . . 32
6.12 wr22::regex_parser::utils::detail::adt::MultiCallable< Fs > Struct Template Reference . . . 33
6.12.1 Constructor & Destructor Documentation . . . 33
6.12.1.1 MultiCallable() . . . 33
6.13 wr22::regex_parser::regex::capture::Name Struct Reference . . . 33

6.13.1 Detailed Description . . . 34
6.13.2 Constructor & Destructor Documentation . . . 34
6.13.2.1 Name() . . . 34
6.13.3 Member Function Documentation . . . 34
6.13.3.1 operator==() . . . 34
6.13.4 Member Data Documentation . . . 34
6.13.4.1 flavor . . . 34
6.13.4.2 name . . . 34
6.14 wr22::regex_parser::regex::capture::None Struct Reference . . . 35
6.14.1 Detailed Description . . . 35
6.14.2 Constructor & Destructor Documentation . . . 35
6.14.2.1 None() . . . 35
6.14.3 Member Function Documentation . . . 35
6.14.3.1 operator==() . . . 35
6.15 wr22::regex_parser::regex::part::Optional Struct Reference . . . 35
6.15.1 Detailed Description . . . 36
6.15.2 Constructor & Destructor Documentation . . . 36
6.15.2.1 Optional() . . . 36
6.15.3 Member Function Documentation . . . 36
6.15.3.1 operator==() . . . 36
6.15.4 Member Data Documentation . . . 36
6.15.4.1 inner . . . 37
6.16 wr22::regex_parser::parser::errors::ParseError Struct Reference . . . 37
6.16.1 Detailed Description . . . 37
6.17 wr22::regex_parser::parser::Parser< Iter, Sentinel > Class Template Reference . . . 37
6.17.1 Detailed Description . . . 38
6.17.2 Constructor & Destructor Documentation . . . 38
6.17.2.1 Parser() . . . 38
6.17.3 Member Function Documentation . . . 39
6.17.3.1 expect_end() . . . 39
6.17.3.2 parse_alternatives() . . . 39
6.17.3.3 parse_atom() . . . 39
6.17.3.4 parse_char_literal() . . . 40
6.17.3.5 parse_group() . . . 40
6.17.3.6 parse_group_name() . . . 40
6.17.3.7 parse_regex() . . . 41
6.17.3.8 parse_sequence() . . . 41
6.17.3.9 parse_sequence_or_empty() . . . 41
6.18 wr22::regex_parser::regex::Part Class Reference . . . 42
6.18.1 Detailed Description . . . 42
6.19 wr22::regex_parser::regex::part::Plus Struct Reference . . . 43
6.19.1 Detailed Description . . . 43

6.19.2 Constructor & Destructor Documentation . . . 43
6.19.2.1 Plus() . . . 43
6.19.3 Member Function Documentation . . . 43
6.19.3.1 operator==() . . . 43
6.19.4 Member Data Documentation . . . 43
6.19.4.1 inner . . . 44
6.20 wr22::regex_parser::regex::part::Sequence Struct Reference . . . 44
6.20.1 Detailed Description . . . 44
6.20.2 Constructor & Destructor Documentation . . . 44
6.20.2.1 Sequence() . . . 44
6.20.3 Member Function Documentation . . . 45
6.20.3.1 operator==() . . . 45
6.20.4 Member Data Documentation . . . 45
6.20.4.1 items . . . 45
6.21 wr22::regex_parser::regex::part::Star Struct Reference . . . 45
6.21.1 Detailed Description . . . 45
6.21.2 Constructor & Destructor Documentation . . . 46
6.21.2.1 Star() . . . 46
6.21.3 Member Function Documentation . . . 46
6.21.3.1 operator==() . . . 46
6.21.4 Member Data Documentation . . . 46
6.21.4.1 inner . . . 46
6.22 wr22::regex_parser::parser::errors::UnexpectedChar Class Reference . . . 46
6.22.1 Detailed Description . . . 47
6.22.2 Constructor & Destructor Documentation . . . 47
6.22.2.1 UnexpectedChar() . . . 47
6.22.3 Member Function Documentation . . . 47
6.22.3.1 char_got() . . . 47
6.22.3.2 expected() . . . 48
6.22.3.3 position() . . . 48
6.23 wr22::regex_parser::parser::errors::UnexpectedEnd Class Reference . . . 48
6.23.1 Detailed Description . . . 48
6.23.2 Constructor & Destructor Documentation . . . 49
6.23.2.1 UnexpectedEnd() . . . 49
6.23.3 Member Function Documentation . . . 49
6.23.3.1 expected() . . . 49
6.23.3.2 position() . . . 49
6.24 wr22::regex_parser::utils::UnicodeStringView Class Reference . . . 49
6.24.1 Detailed Description . . . 50
6.24.2 Constructor & Destructor Documentation . . . 50
6.24.2.1 UnicodeStringView() . . . 50
6.24.3 Member Function Documentation . . . 51

6.24.3.1 begin() . . . 51
6.24.3.2 end() . . . 51
6.24.3.3 raw() . . . 51
6.24.4 Friends And Related Function Documentation . . . 51
6.24.4.1 UnicodeStringViewIterator . . . 51
6.25 wr22::regex_parser::utils::UnicodeStringViewIterator Class Reference . . . 51
6.25.1 Detailed Description . . . 52
6.25.2 Member Typedef Documentation . . . 52
6.25.2.1 category_type . . . 52
6.25.2.2 value_type . . . 52
6.25.3 Constructor & Destructor Documentation . . . 53
6.25.3.1 UnicodeStringViewIterator() . . . 53
6.25.4 Member Function Documentation . . . 53
6.25.4.1 operator"!=() . . . 53
6.25.4.2 operator∗() . . . 53
6.25.4.3 operator++() . . . 53
6.25.4.4 operator==() . . . 54
6.25.5 Friends And Related Function Documentation . . . 54
6.25.5.1 UnicodeStringView . . . 54

**7 File Documentation** **55**

7.1 include/wr22/regex_parser/parser/errors.hpp File Reference . . . 55
7.2 errors.hpp . . . 56
7.3 include/wr22/regex_parser/parser/regex.hpp File Reference . . . 56
7.4 regex.hpp . . . 57
7.5 include/wr22/regex_parser/regex/capture.hpp File Reference . . . 57
7.6 capture.hpp . . . 58
7.7 include/wr22/regex_parser/regex/named_capture_flavor.hpp File Reference . . . 58
7.8 named_capture_flavor.hpp . . . 59
7.9 include/wr22/regex_parser/regex/part.hpp File Reference . . . 59
7.10 part.hpp . . . 60
7.11 include/wr22/regex_parser/utils/adt.hpp File Reference . . . 61
7.12 adt.hpp . . . 62
7.13 include/wr22/regex_parser/utils/box.hpp File Reference . . . 63
7.14 box.hpp . . . 63
7.15 include/wr22/regex_parser/utils/utf8_string_view.hpp File Reference . . . 64
7.16 utf8_string_view.hpp . . . 65
7.17 src/parser/capture.cpp File Reference . . . 65
7.18 src/parser/errors.cpp File Reference . . . 66
7.19 src/parser/regex.cpp File Reference . . . 66
7.20 src/regex/named_capture_flavor.cpp File Reference . . . 67
7.21 src/regex/part.cpp File Reference . . . 67

7.22 src/utils/box.cpp File Reference . . . . . 68
7.23 src/utils/utf8_string_view.cpp File Reference . . . . . 68

**Index** **69**

# Chapter 1

# Namespace Index

## 1.1 Namespace List

Here is a list of all namespaces with brief descriptions:


wr22 . . . . . 9
wr22::regex_parser . . . . . 9
wr22::regex_parser::parser . . . . . 9
wr22::regex_parser::parser::errors . . . . . 10
wr22::regex_parser::regex . . . . . 11
wr22::regex_parser::regex::capture . . . . . 12
wr22::regex_parser::regex::part
The namespace with the variants of `Part` . . . . . 13
wr22::regex_parser::utils . . . . . 14
wr22::regex_parser::utils::detail . . . . . 16
wr22::regex_parser::utils::detail::adt . . . . . 16

# Chapter 2

# Hierarchical Index

## 2.1 Class Hierarchy

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


wr22::regex_parser::utils::Adt< Variants > . . . 17
  wr22::regex_parser::regex::Capture . . . 25
  wr22::regex_parser::regex::Part . . . 42
wr22::regex_parser::regex::part::Alternatives . . . 20
wr22::regex_parser::utils::Box< T > . . . 21
wr22::regex_parser::utils::Box< wr22::regex_parser::regex::Part > . . . 21
wr22::regex_parser::regex::part::Empty . . . 26
std::exception
  wr22::regex_parser::utils::BoxIsEmpty . . . 25
  wr22::regex_parser::utils::UnicodeStringView::InvalidUtf8 . . . 31
wr22::regex_parser::regex::part::Group . . . 28
wr22::regex_parser::regex::capture::Index . . . 30
wr22::regex_parser::regex::part::Literal . . . 31
wr22::regex_parser::regex::capture::Name . . . 33
wr22::regex_parser::regex::capture::None . . . 35
wr22::regex_parser::regex::part::Optional . . . 35
wr22::regex_parser::parser::Parser< Iter, Sentinel > . . . 37
wr22::regex_parser::regex::part::Plus . . . 43
std::runtime_error
  wr22::regex_parser::parser::errors::ParseError . . . 37
    wr22::regex_parser::parser::errors::ExpectedEnd . . . 27
    wr22::regex_parser::parser::errors::UnexpectedChar . . . 46
    wr22::regex_parser::parser::errors::UnexpectedEnd . . . 48
wr22::regex_parser::regex::part::Sequence . . . 44
wr22::regex_parser::regex::part::Star . . . 45
wr22::regex_parser::utils::UnicodeStringView . . . 49
wr22::regex_parser::utils::UnicodeStringViewIterator . . . 51
wr22::regex_parser::utils::detail::adt::Fs
  wr22::regex_parser::utils::detail::adt::MultiCallable< Fs > . . . 33

# Chapter 3

# Class Index

## 3.1 Class List

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


wr22::regex_parser::utils::Adt< Variants >
A helper class that simplifies creation of algebraic data types . . . 17
wr22::regex_parser::regex::part::Alternatives
A regex part with the list of alternatives to be matched . . . 20
wr22::regex_parser::utils::Box< T >
A copyable and equality-comparable wrapper around `std::unique_ptr` . . . 21
wr22::regex_parser::utils::BoxIsEmpty . . . 25
wr22::regex_parser::regex::Capture
Group capture behavior . . . 25
wr22::regex_parser::regex::part::Empty
An empty regex part . . . 26
wr22::regex_parser::parser::errors::ExpectedEnd
The error when the parser expected the input to end, but it did not . . . 27
wr22::regex_parser::regex::part::Group
A regex part that represents a group in parentheses . . . 28
wr22::regex_parser::regex::capture::Index
Denotes a group captured by index . . . 30
wr22::regex_parser::utils::UnicodeStringView::InvalidUtf8
An error thrown when the string's encoding is not valid UTF-8 . . . 31
wr22::regex_parser::regex::part::Literal
An regex part that matches a single character literally . . . 31
wr22::regex_parser::utils::detail::adt::MultiCallable< Fs > . . . 33
wr22::regex_parser::regex::capture::Name
Denotes a group captured by name . . . 33
wr22::regex_parser::regex::capture::None
Denotes an non-capturing group . . . 35
wr22::regex_parser::regex::part::Optional
A regex part specifying an optional quantifier (`(expression)?`) . . . 35
wr22::regex_parser::parser::errors::ParseError
The base class for parse errors . . . 37
wr22::regex_parser::parser::Parser< Iter, Sentinel >
A regex parser . . . 37
wr22::regex_parser::regex::Part
A part of a regular expression and its AST node type . . . 42
wr22::regex_parser::regex::part::Plus
A regex part specifying an "at least one" quantifier (`(expression)+`) . . . 43

wr22::regex_parser::regex::part::Sequence
A regex part with the list of items to be matched one after another . . . . . 44
wr22::regex_parser::regex::part::Star
A regex part specifying an "at least zero" quantifier ((expression)*) . . . . . 45
wr22::regex_parser::parser::errors::UnexpectedChar
The error when the parser got a character it didn't expect at the current position . . . . . 46
wr22::regex_parser::parser::errors::UnexpectedEnd
The error when the parser hit the end of the input earlier than it expected . . . . . 48
wr22::regex_parser::utils::UnicodeStringView
A wrapper around std::string_view that enables UTF-8 codepoint iteration . . . . . 49
wr22::regex_parser::utils::UnicodeStringViewIterator
Iterator over code points for UnicodeStringView . . . . . 51

# Chapter 4

# File Index

## 4.1 File List

Here is a list of all files with brief descriptions:

include/wr22/regex_parser/parser/errors.hpp . . . 55
include/wr22/regex_parser/parser/regex.hpp . . . 56
include/wr22/regex_parser/regex/capture.hpp . . . 57
include/wr22/regex_parser/regex/named_capture_flavor.hpp . . . 58
include/wr22/regex_parser/regex/part.hpp . . . 59
include/wr22/regex_parser/utils/adt.hpp . . . 61
include/wr22/regex_parser/utils/box.hpp . . . 63
include/wr22/regex_parser/utils/utf8_string_view.hpp . . . 64
src/parser/capture.cpp . . . 65
src/parser/errors.cpp . . . 66
src/parser/regex.cpp . . . 66
src/regex/named_capture_flavor.cpp . . . 67
src/regex/part.cpp . . . 67
src/utils/box.cpp . . . 68
src/utils/utf8_string_view.cpp . . . 68

# 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 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::Part parse_regex (const utils::UnicodeStringView &regex)

  *Parse a regular expression into its AST.*

### 5.3.1 Function Documentation

#### 5.3.1.1 parse_regex()

```
regex::Part wr22::regex_parser::parser::parse_regex (
            const utils::UnicodeStringView & regex )
```

Parse a regular expression into its AST.

The regular expression is a string view in the UTF-8 encoding (the `utils::UnicodeStringView` wrapper is used for convenience and type safety; see its docs for additional information). It is parsed and its object representation (see the docs for `regex::Part`) 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**

| | |
|---|---|
| *errors::ParseError* | 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.*

### 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)
- std::ostream & operator<< (std::ostream &out, NamedCaptureFlavor flavor)
- std::ostream & operator<< (std::ostream &out, const Part &part)

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

### 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 Part & part )
```

Convert a `Part` 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 )
```

## 5.6 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 >

### 5.6.1 Typedef Documentation

#### 5.6.1.1 Adt

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

## 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)*`).*

### Typedefs

- using Adt = utils::Adt< Empty, Literal, Alternatives, Sequence, Group, Optional, Plus, Star >

### 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>
```

## 5.8 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
- class UnicodeStringView

  *A wrapper around `std::string_view` that enables UTF-8 codepoint iteration.*
- class UnicodeStringViewIterator

  *Iterator over code points for `UnicodeStringView`.*

### Typedefs

- using utf_traits = boost::locale::utf::utf_traits< char >

### 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)

### 5.8.1 Typedef Documentation

#### 5.8.1.1 utf_traits

```
using wr22::regex_parser::utils::utf_traits = typedef boost::locale::utf::utf_traits<char>
```

## 5.8.2 Function Documentation

### 5.8.2.1 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).

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

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

Type deduction guideline for Box (value initialization).

### 5.8.2.3 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.

### 5.8.2.4 operator"!=() [2/2]

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

### 5.8.2.5 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.

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

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

## 5.9 wr22::regex_parser::utils::detail Namespace Reference

### Namespaces

- namespace adt

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

### Classes

- struct MultiCallable

# Chapter 6

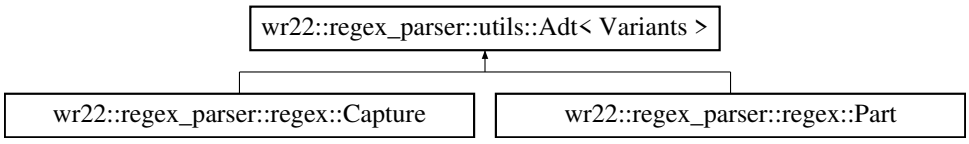
# Class Documentation

## 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< Part > alternatives)
- bool operator== (const Alternatives &rhs) const =default

### Public Attributes

- std::vector< Part > alternatives

  *The list of the 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 `Part`s.

### 6.2.2 Constructor & Destructor Documentation

#### 6.2.2.1 Alternatives()

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

The list of the alternatives.

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

  *Derefencing operator: obtain a const reference to the stored value.*
- T & operator* ()

  *Derefencing 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 bahavior.

### 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**

| | |
|---|---|
| *`other`* | the Box from which to copy. |

**Exceptions**

| | |
|---|---|
| *BoxIsEmpty* | if `other` 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]
```

Derefencing operator: obtain a reference to the stored value.

**Exceptions**

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

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

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

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

**Exceptions**

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

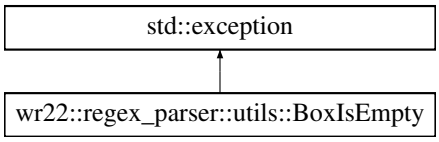
The documentation for this class was generated from the following file:

- include/wr22/regex_parser/utils/box.hpp

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

`#include <box.hpp>`

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




### Public Member Functions

- const char * what () const noexcept override

### 6.4.1 Member Function Documentation

#### 6.4.1.1 what()

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

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

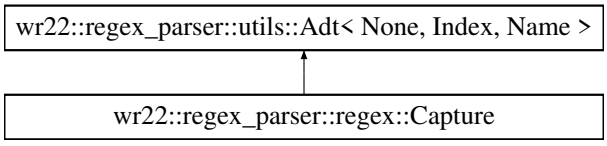
- include/wr22/regex_parser/utils/box.hpp
- src/utils/box.cpp

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

Group capture behavior.

`#include <capture.hpp>`

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



### Additional Inherited Members

### 6.5.1 Detailed Description

Group capture behavior.

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

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

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

## 6.6 wr22::regex_parser::regex::part::Empty Struct Reference

An empty regex part.

`#include <part.hpp>`

### Public Member Functions

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

### 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]
```

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

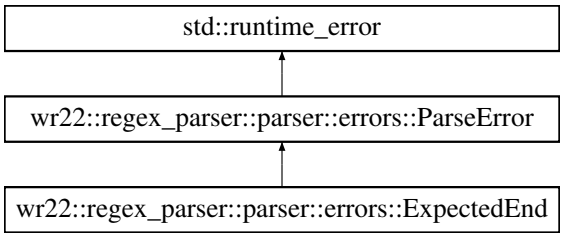
- include/wr22/regex_parser/regex/part.hpp

## 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**

| | |
|---|---|
| *position* | the 0-based position in the input when the parser has encountered the end of input. |
| *char_got* | 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, Part inner)

  *Convenience constructor.*
- bool operator== (const Group &rhs) const =default

### Public Attributes

- Capture capture

  *Capture behavior.*
- utils::Box< Part > inner

  *The smart pointer to the group contents.*

### 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 `Part`. 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,
            Part 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 inner

```
utils::Box<Part> 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

### 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]
```

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

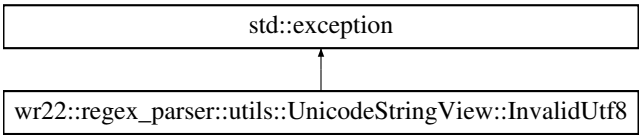
- include/wr22/regex_parser/regex/capture.hpp

## 6.10 wr22::regex_parser::utils::UnicodeStringView::InvalidUtf8 Struct Reference

An error thrown when the string's encoding is not valid UTF-8.

`#include <utf8_string_view.hpp>`

Inheritance diagram for wr22::regex_parser::utils::UnicodeStringView::InvalidUtf8:




### Public Member Functions

- const char * what () const noexcept override

### 6.10.1 Detailed Description

An error thrown when the string's encoding is not valid UTF-8.

### 6.10.2 Member Function Documentation

#### 6.10.2.1 what()

`const char * wr22::regex_parser::utils::UnicodeStringView::InvalidUtf8::what ( ) const [override], [noexcept]`

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

- include/wr22/regex_parser/utils/utf8_string_view.hpp
- src/utils/utf8_string_view.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

## 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
```

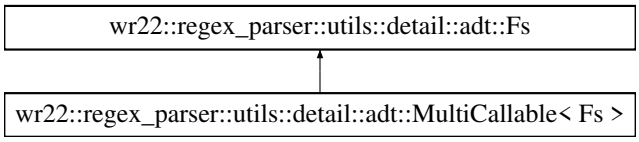
The documentation for this struct was generated from the following files:

- 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

### 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 flavor

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

#### 6.13.4.2 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

### 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]
```

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 (Part inner)

  *Convenience constructor.*
- bool operator== (const Optional &rhs) const =default

## Public Attributes

- utils::Box< Part > inner

  *The smart pointer to the subexpression under the quantifier.*

### 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 (
            Part 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 inner

```
utils::Box<Part> 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:

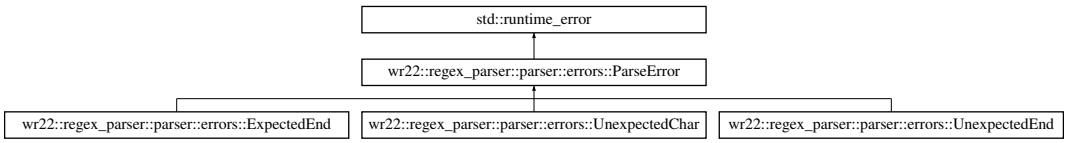
- include/wr22/regex_parser/regex/part.hpp
- src/regex/part.cpp

## 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::Part parse_regex ()

  *Parse a regex consuming part of the remaining input.*
- regex::Part parse_alternatives ()

  *Intermediate rule: parse a pipe-separated list of alternatives (e.g.*
- regex::Part parse_sequence ()

  *Intermediate rule: parse a sequence of atoms (e.g.*
- regex::Part parse_sequence_or_empty ()

  *Intermediate rule: parse a possibly empty sequence of atoms.*
- regex::Part parse_atom ()

  *Intermediate rule: parse an atom.*
- regex::Part parse_char_literal ()

  *Intermediate rule: parse a character literal.*
- regex::Part parse_group ()

  *Intermediate rule: parse a parenthesized group (any capture variant).*
- std::string 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 is 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**

| | |
|---|---|
| *errors::ExpectedEnd* | if this is not the case. |

### 6.17.3.2 parse_alternatives()

```
template<typename Iter , typename Sentinel >
regex::Part 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**

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

### 6.17.3.3 parse_atom()

```
template<typename Iter , typename Sentinel >
regex::Part 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::Part` depending on the atom kind).

**Exceptions**

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

#### 6.17.3.4 parse_char_literal()

```
template<typename Iter , typename Sentinel >
regex::Part 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**

| | |
|---|---|
| *errors::UnexpectedEnd* | if all characters from the input have already been consumed. |

#### 6.17.3.5 parse_group()

```
template<typename Iter , typename Sentinel >
regex::Part 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::string 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::Part 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::Part` depending on the input).

**Exceptions**

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

#### 6.17.3.8 parse_sequence()

```
template<typename Iter , typename Sentinel >
regex::Part 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**

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

#### 6.17.3.9 parse_sequence_or_empty()

```
template<typename Iter , typename Sentinel >
regex::Part 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**

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

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

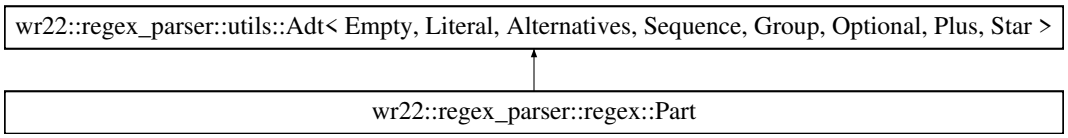
- src/parser/regex.cpp

## 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 `Part`s 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::Literal`s.

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.

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 (Part inner)

  *Convenience constructor.*
- bool operator== (const Plus &rhs) const =default

## Public Attributes

- utils::Box< Part > inner

  *The smart pointer to the subexpression under the quantifier.*

## 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 (
            Part 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 inner

```
utils::Box<Part> 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< Part > items)
- bool operator== (const Sequence &rhs) const =default

### Public Attributes

- std::vector< Part > items

  *The list of the subexpressions.*

### 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< Part > 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 items

```
std::vector<Part> 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::regex::part::Star Struct Reference

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

`#include <part.hpp>`

### Public Member Functions

- Star (Part inner)

  *Convenience constructor.*
- bool operator== (const Star &rhs) const =default

### Public Attributes

- utils::Box< Part > inner

  *The smart pointer to the subexpression under the quantifier.*

### 6.21.1 Detailed Description

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

### 6.21.2 Constructor & Destructor Documentation

#### 6.21.2.1 Star()

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

Convenience constructor.

### 6.21.3 Member Function Documentation

#### 6.21.3.1 operator==()

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

### 6.21.4 Member Data Documentation

#### 6.21.4.1 inner

```
utils::Box<Part> 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:

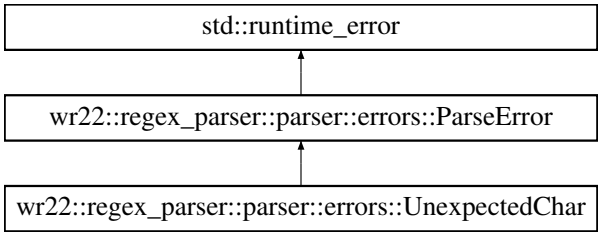
- include/wr22/regex_parser/regex/part.hpp
- src/regex/part.cpp

## 6.22 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.*

### 6.22.1 Detailed Description

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

### 6.22.2 Constructor & Destructor Documentation

#### 6.22.2.1 UnexpectedChar()

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

Constructor.

**Parameters**

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

### 6.22.3 Member Function Documentation

#### 6.22.3.1 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.

#### 6.22.3.2 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.

#### 6.22.3.3 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:

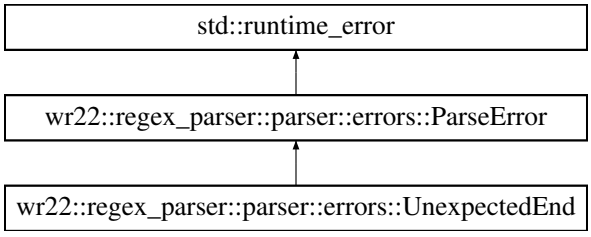
- include/wr22/regex_parser/parser/errors.hpp
- src/parser/errors.cpp

## 6.23 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.*

### 6.23.1 Detailed Description

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

### 6.23.2 Constructor & Destructor Documentation

#### 6.23.2.1 UnexpectedEnd()

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

Constructor.

**Parameters**

| | |
|---|---|
| *position* | the 0-based position in the input when the parser has encountered the end of input. |
| *expected* | a textual description of a class of characters expected instead. |

### 6.23.3 Member Function Documentation

#### 6.23.3.1 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.

#### 6.23.3.2 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

## 6.24 wr22::regex_parser::utils::UnicodeStringView Class Reference

A wrapper around `std::string_view` that enables UTF-8 codepoint iteration.

`#include <utf8_string_view.hpp>`

### Classes

- struct InvalidUtf8

  *An error thrown when the string's encoding is not valid UTF-8.*

### Public Member Functions

- UnicodeStringView (const std::string_view &raw)

  *Constructor.*
- const std::string_view & raw () const

  *Access the wrapped `std::string_view`.*
- UnicodeStringViewIterator begin () const

  *Create an iterator to the beginning of the string.*
- UnicodeStringViewIterator end () const

  *Create an iterator past the end of the string.*

### Friends

- class UnicodeStringViewIterator

## 6.24.1 Detailed Description

A wrapper around `std::string_view` that enables UTF-8 codepoint iteration.

An `UnicodeStringView` holds a "raw" `std::string_view` and assumes it is UTF-8 encoded. It provides the `begin()` & `end()` methods, which return iterators that work with Unicode codepoints instead of raw bytes. This makes this type useful in contexts where one needs to iterate over Unicode code points (like characters, but technically a different thing) in an `std::string` or an `std::string_view`.

## 6.24.2 Constructor & Destructor Documentation

### 6.24.2.1 UnicodeStringView()

```
wr22::regex_parser::utils::UnicodeStringView::UnicodeStringView (
            const std::string_view & raw )  [explicit]
```

Constructor.

Wrap an existing `std::string_view`.

The ownership of the data pointed to by `raw` is not taken, and the string contents are not copied. Hence, just like with an ordinary `std::string_view`, if the pointed data is invalidated, the `UnicodeStringView` referring to it is invalidated as well.

This constructor does not check that `raw` is a correct UTF-8 encoded string.

SAFETY: If the string contents change during the `UnicodeStringView`'s lifetime or the string is destroyed, the behavior is undefined.

### 6.24.3 Member Function Documentation

#### 6.24.3.1 begin()

```
UnicodeStringViewIterator wr22::regex_parser::utils::UnicodeStringView::begin ( ) const
```

Create an iterator to the beginning of the string.

SAFETY: The returned iterator must not outlive this object.

#### 6.24.3.2 end()

```
UnicodeStringViewIterator wr22::regex_parser::utils::UnicodeStringView::end ( ) const
```

Create an iterator past the end of the string.

SAFETY: The returned iterator must not outlive this object.

#### 6.24.3.3 raw()

```
const std::string_view & wr22::regex_parser::utils::UnicodeStringView::raw ( ) const
```

Access the wrapped `std::string_view`.

### 6.24.4 Friends And Related Function Documentation

#### 6.24.4.1 UnicodeStringViewIterator

```
friend class UnicodeStringViewIterator  [friend]
```

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

- include/wr22/regex_parser/utils/utf8_string_view.hpp
- src/utils/utf8_string_view.cpp

## 6.25 wr22::regex_parser::utils::UnicodeStringViewIterator Class Reference

Iterator over code points for `UnicodeStringView`.

```
#include <utf8_string_view.hpp>
```

## Public Types

- using value_type = char32_t

  *The type of value this iterator yields. Part of the STL iterator interface.*
- using category_type = std::forward_iterator_tag

  *This iterator's category. Part of the STL iterator interface.*

## Public Member Functions

- UnicodeStringViewIterator ()=delete

  *The default constructor is meaningless and is thus deleted.*
- UnicodeStringViewIterator & operator++ ()

  *Iterator interface: pre-increment.*
- char32_t operator* ()

  *Iterator interface: dereferencing.*
- bool operator== (const UnicodeStringViewIterator &rhs) const
- bool operator!= (const UnicodeStringViewIterator &rhs) const

## Friends

- class UnicodeStringView

### 6.25.1 Detailed Description

Iterator over code points for `UnicodeStringView`.

SAFETY: No object of this class must outlive the `UnicodeStringView` that has created it. If this is violated, the behavior is undefined.

### 6.25.2 Member Typedef Documentation

#### 6.25.2.1 category_type

```
using wr22::regex_parser::utils::UnicodeStringViewIterator::category_type = std::forward_↩
iterator_tag
```

This iterator's category. Part of the STL iterator interface.

#### 6.25.2.2 value_type

```
using wr22::regex_parser::utils::UnicodeStringViewIterator::value_type = char32_t
```

The type of value this iterator yields. Part of the STL iterator interface.

## 6.25.3 Constructor & Destructor Documentation

### 6.25.3.1 UnicodeStringViewIterator()

```
wr22::regex_parser::utils::UnicodeStringViewIterator::UnicodeStringViewIterator ( )  [delete]
```

The default constructor is meaningless and is thus deleted.

## 6.25.4 Member Function Documentation

### 6.25.4.1 operator"!=()

```
bool wr22::regex_parser::utils::UnicodeStringViewIterator::operator!= (
            const UnicodeStringViewIterator & rhs ) const
```

### 6.25.4.2 operator∗()

```
char32_t wr22::regex_parser::utils::UnicodeStringViewIterator::operator* ( )
```

Iterator interface: dereferencing.

**Exceptions**

| | |
|---|---|
| *UnicodeStringView::InvalidUtf8* | if the codepoint decodes into an invalid or incomplete value. |

### 6.25.4.3 operator++()

```
UnicodeStringViewIterator & wr22::regex_parser::utils::UnicodeStringViewIterator::operator++ (
)
```

Iterator interface: pre-increment.

**Exceptions**

| | |
|---|---|
| *UnicodeStringView::InvalidUtf8* | if the codepoint decodes into an invalid or incomplete value. |

#### 6.25.4.4 operator==()

```
bool wr22::regex_parser::utils::UnicodeStringViewIterator::operator== (
            const UnicodeStringViewIterator & rhs ) const
```

### 6.25.5 Friends And Related Function Documentation

#### 6.25.5.1 UnicodeStringView

```
friend class UnicodeStringView  [friend]
```

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

- include/wr22/regex_parser/utils/utf8_string_view.hpp
- src/utils/utf8_string_view.cpp

# 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
15 struct ParseError : public std::runtime_error {
16     using std::runtime_error::runtime_error;
17 };
18
20 class UnexpectedEnd : public ParseError {
21 public:
27     UnexpectedEnd(size_t position, std::string expected);
28
30     size_t position() const;
33     const std::string& expected() const;
34
35 private:
36     size_t m_position;
37     std::string m_expected;
38 };
39
41 class ExpectedEnd : public ParseError {
42 public:
48     ExpectedEnd(size_t position, char32_t char_got);
49
51     size_t position() const;
54     char32_t char_got() const;
55
56 private:
57     size_t m_position;
58     char32_t m_char_got;
59 };
60
62 class UnexpectedChar : public ParseError {
63 public:
70     UnexpectedChar(size_t position, char32_t char_got, std::string expected);
71
73     size_t position() const;
76     char32_t char_got() const;
79     const std::string& expected() const;
80
81 private:
82     size_t m_position;
83     char32_t m_char_got;
84     std::string m_expected;
85 };
86
87 }  // 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 <wr22/regex_parser/utils/utf8_string_view.hpp>
#include <string_view>
```

### Namespaces

- namespace wr22
- namespace wr22::regex_parser
- namespace wr22::regex_parser::parser

### Functions

- regex::Part wr22::regex_parser::parser::parse_regex (const utils::UnicodeStringView &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 #include <wr22/regex_parser/utils/utf8_string_view.hpp>
6
7 // stl
8 #include <string_view>
9
10 namespace wr22::regex_parser::parser {
11
26 regex::Part parse_regex(const utils::UnicodeStringView& regex);
27
28 }  // 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>
```

### 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

- std::ostream & wr22::regex_parser::regex::operator<< (std::ostream &out, 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 namespace wr22::regex_parser::regex {
12
13 class Capture;
14
15 namespace capture {
17     struct None {
18         explicit None() = default;
19         bool operator==(const None& rhs) const = default;
20     };
21
23     struct Index {
24         explicit Index() = default;
25         bool operator==(const Index& rhs) const = default;
26     };
27
32     struct Name {
33         explicit Name(std::string name, NamedCaptureFlavor flavor);
34
35         std::string name;
36         NamedCaptureFlavor flavor;
37         bool operator==(const Name& rhs) const = default;
38     };
39
40     using Adt = utils::Adt<None, Index, Name>;
41 }  // namespace capture
42
48 //
52 class Capture : public capture::Adt {
53 public:
54     using capture::Adt::Adt;
55 };
56
57 std::ostream& operator«(std::ostream& out, const Capture& capture);
58
59 }  // namespace wr22::regex_parser::regex
```

## 7.7 include/wr22/regex_parser/regex/named_capture_flavor.hpp File Reference

```
#include <iosfwd>
```

### 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)

## 7.8 named_capture_flavor.hpp

Go to the documentation of this file.

```
1 #pragma once
2
3 // stl
4 #include <iosfwd>
5
6 namespace wr22::regex_parser::regex {
7
12 enum class NamedCaptureFlavor
13 {
16     Apostrophes,
19     Angles,
21     AnglesWithP,
22 };
23
24 std::ostream& operator«(std::ostream& out, NamedCaptureFlavor flavor);
25
26 }  // namespace wr22::regex_parser::regex
```

## 7.9 include/wr22/regex_parser/regex/part.hpp File Reference

```
#include <wr22/regex_parser/regex/capture.hpp>
#include <wr22/regex_parser/utils/adt.hpp>
#include <wr22/regex_parser/utils/box.hpp>
#include <iosfwd>
#include <memory>
#include <vector>
```

### 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)*`).*
- class wr22::regex_parser::regex::Part

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

## 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 >

## Functions

- std::ostream & wr22::regex_parser::regex::operator<< (std::ostream &out, const Part &part)

  *Convert a `Part` 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 <wr22/regex_parser/regex/capture.hpp>
5 #include <wr22/regex_parser/utils/adt.hpp>
6 #include <wr22/regex_parser/utils/box.hpp>
7
8 // stl
9 #include <iosfwd>
10 #include <memory>
11 #include <vector>
12
13 namespace wr22::regex_parser::regex {
14
15 // Forward declaration of 'Part'.
16 class Part;
17
21 namespace part {
26     struct Empty {
27         explicit Empty() = default;
28         bool operator==(const Empty& rhs) const = default;
29     };
30
35     struct Literal {
36         explicit Literal(char32_t character);
37         bool operator==(const Literal& rhs) const = default;
38
39         char32_t character;
40     };
41
50     struct Alternatives {
51         explicit Alternatives(std::vector<Part> alternatives);
```

```
52          bool operator==(const Alternatives& rhs) const = default;
53
55          std::vector<Part> alternatives;
56      };
57
63      struct Sequence {
64          explicit Sequence(std::vector<Part> items);
65          bool operator==(const Sequence& rhs) const = default;
66
68          std::vector<Part> items;
69      };
70
81      struct Group {
83          explicit Group(Capture capture, Part inner);
84          bool operator==(const Group& rhs) const = default;
85
87          Capture capture;
89          utils::Box<Part> inner;
90      };
91
93      struct Optional {
95          explicit Optional(Part inner);
96          bool operator==(const Optional& rhs) const = default;
97
99          utils::Box<Part> inner;
100     };
101
103     struct Plus {
105         explicit Plus(Part inner);
106         bool operator==(const Plus& rhs) const = default;
107
109         utils::Box<Part> inner;
110     };
111
113     struct Star {
115         explicit Star(Part inner);
116         bool operator==(const Star& rhs) const = default;
117
119         utils::Box<Part> inner;
120     };
121
122     using Adt = utils::Adt<Empty, Literal, Alternatives, Sequence, Group, Optional, Plus, Star>;
123 }  // namespace part
124
148 class Part : public part::Adt {
149 public:
150     using part::Adt::Adt;
151 };
152
154 std::ostream& operator<<(std::ostream& out, const Part& part);
155
156 }  // namespace wr22::regex_parser::regex
```

## 7.11 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.*

## 7.12 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
28 template <typename... Variants>
29 class Adt {
30 public:
32     using VariantType = std::variant<Variants...>;
33
43     template <typename V>
44     Adt(V variant) : m_variant(std::move(variant)) {}
45
74     template <typename... Fs>
75     decltype(auto) visit(Fs&&... visitors) const {
76         return std::visit(
77             detail::adt::MultiCallable<Fs...>(std::forward<Fs>(visitors)...),
78             m_variant);
79     }
80
87     template <typename... Fs>
88     decltype(auto) visit(Fs&&... visitors) {
89         return std::visit(
90             detail::adt::MultiCallable<Fs...>(std::forward<Fs>(visitors)...),
91             m_variant);
92     }
93
95     const VariantType& as_variant() const {
96         return m_variant;
97     }
98
100     VariantType& as_variant() {
101         return m_variant;
102     }
103
104 protected:
105     VariantType m_variant;
106 };
107
109 template <typename... Variants>
110 bool operator==(const Adt<Variants...>& lhs, const Adt<Variants...>& rhs) {
111     return lhs.as_variant() == rhs.as_variant();
112 }
113
115 template <typename... Variants>
116 bool operator!=(const Adt<Variants...>& lhs, const Adt<Variants...>& rhs) {
117     return !(lhs == rhs);
118 }
119
120 }  // namespace wr22::regex_parser::utils
```

# 7.13 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)

# 7.14 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
25 template <typename T>
26 class Box {
27 public:
32     explicit Box(T&& value) : m_ptr(std::make_unique<T>(std::forward<T>(value))) {}
33
49     explicit Box(std::unique_ptr<T> ptr) : m_ptr(std::move(ptr)) {}
50
57     template <typename Dummy = T>
58     Box(const Box& other) : m_ptr(std::make_unique<T>(*other)) {}
```

```
59
63      template <typename... Args>
64      static Box<T> construct_in_place(Args&&... args) {
65          return Box(std::make_unique<T>(std::forward<Args>(args)...));
66      }
67
71      const T& operator*() const {
72          if (m_ptr == nullptr) {
73              throw BoxIsEmpty{};
74          }
75          return *m_ptr;
76      }
77
81      T& operator*() {
82          if (m_ptr == nullptr) {
83              throw BoxIsEmpty{};
84          }
85          return *m_ptr;
86      }
87
88  private:
89      std::unique_ptr<T> m_ptr;
90  };
91
93  template <typename T>
94  Box(T&& value) -> Box<T>;
95
97  template <typename T>
98  Box(std::unique_ptr<T> ptr) -> Box<T>;
99
100 template <typename T, typename U>
101 bool operator==(const Box<T>& lhs, const Box<U>& rhs) {
102     return *lhs == *rhs;
103 }
104
105 template <typename T, typename U>
106 bool operator!=(const Box<T>& lhs, const Box<U>& rhs) {
107     return !(lhs == rhs);
108 }
109
110 }  // namespace wr22::regex_parser::utils
```

## 7.15 include/wr22/regex_parser/utils/utf8_string_view.hpp File Reference

```
#include <compare>
#include <cstddef>
#include <exception>
#include <iterator>
#include <optional>
#include <string_view>
```

### Classes

- class wr22::regex_parser::utils::UnicodeStringView

  *A wrapper around `std::string_view` that enables UTF-8 codepoint iteration.*
- struct wr22::regex_parser::utils::UnicodeStringView::InvalidUtf8

  *An error thrown when the string's encoding is not valid UTF-8.*
- class wr22::regex_parser::utils::UnicodeStringViewIterator

  *Iterator over code points for `UnicodeStringView`.*

### Namespaces

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

## 7.16 utf8_string_view.hpp

Go to the documentation of this file.

```
1 #pragma once
2
3 // STL
4 #include <compare>
5 #include <cstddef>
6 #include <exception>
7 #include <iterator>
8 #include <optional>
9 #include <string_view>
10
11 namespace wr22::regex_parser::utils {
12
13 class UnicodeStringViewIterator;
14
22 class UnicodeStringView {
23     friend class UnicodeStringViewIterator;
24
25 public:
27     struct InvalidUtf8 : std::exception {
28         const char* what() const noexcept override;
29     };
30
42     explicit UnicodeStringView(const std::string_view& raw);
43
45     const std::string_view& raw() const;
46
51     UnicodeStringViewIterator begin() const;
52
57     UnicodeStringViewIterator end() const;
58
59 private:
60     std::string_view m_raw;
61 };
62
68 class UnicodeStringViewIterator {
69     friend class UnicodeStringView;
70
71 public:
73     using value_type = char32_t;
74
76     using category_type = std::forward_iterator_tag;
77
79     UnicodeStringViewIterator() = delete;
80
85     UnicodeStringViewIterator& operator++();
86
91     char32_t operator*();
92
93     bool operator==(const UnicodeStringViewIterator& rhs) const;
94     bool operator!=(const UnicodeStringViewIterator& rhs) const;
95
96 private:
97     using UnderlyingIterator = std::string_view::const_iterator;
98
99     explicit UnicodeStringViewIterator(
100         const UnderlyingIterator& iter,
101         const UnderlyingIterator& end);
102     std::optional<char32_t> decode_current_codepoint();
103     void update_current_codepoint_size();
104
105     UnderlyingIterator m_iter;
106     UnderlyingIterator m_end;
107
108     // A memoization of the result of decoding of the last code point.
109     // This is done purely for optimization to avoid having to decode a code point
110     // twice: for dereferencing the iterator and for incrementing it. The value
111     // of 0 indicates an absence of a memoized value (0 is not a valid byte size of
112     // a codepoint).
113     size_t m_last_codepoint_size = 0;
114 };
115
116 }  // 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

### Functions

- std::ostream & wr22::regex_parser::regex::operator<< (std::ostream &out, const Capture &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 <boost/locale/utf.hpp>
#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>
```

### 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::Part wr22::regex_parser::parser::parse_regex (const utils::UnicodeStringView &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)

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

```
#include <wr22/regex_parser/regex/part.hpp>
#include <iterator>
#include <ostream>
#include <boost/locale/utf.hpp>
```

### Namespaces

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

### Functions

- std::ostream & wr22::regex_parser::regex::operator<< (std::ostream &out, const Part &part)

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

## 7.22 src/utils/box.cpp File Reference

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

### Namespaces

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

## 7.23 src/utils/utf8_string_view.cpp File Reference

```
#include <wr22/regex_parser/utils/utf8_string_view.hpp>
#include <boost/locale/utf.hpp>
#include <cassert>
```

### Namespaces

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

### Typedefs

- using wr22::regex_parser::utils::utf_traits = boost::locale::utf::utf_traits< char >

# Index


Adt
wr22::regex_parser::regex::capture, 12
wr22::regex_parser::regex::part, 13
wr22::regex_parser::utils::Adt< Variants >, 18
Alternatives
wr22::regex_parser::regex::part::Alternatives, 20
alternatives
wr22::regex_parser::regex::part::Alternatives, 21
Angles
wr22::regex_parser::regex, 11
AnglesWithP
wr22::regex_parser::regex, 11
Apostrophes
wr22::regex_parser::regex, 11
as_variant
wr22::regex_parser::utils::Adt< Variants >, 19

begin
wr22::regex_parser::utils::UnicodeStringView, 51
Box
wr22::regex_parser::utils, 15
wr22::regex_parser::utils::Box< T >, 22

capture
wr22::regex_parser::regex::part::Group, 29
category_type
wr22::regex_parser::utils::UnicodeStringViewIterator, 52
char_got
wr22::regex_parser::parser::errors::ExpectedEnd, 28
wr22::regex_parser::parser::errors::UnexpectedChar, 47
character
wr22::regex_parser::regex::part::Literal, 32
construct_in_place
wr22::regex_parser::utils::Box< T >, 24

Empty
wr22::regex_parser::regex::part::Empty, 26
end
wr22::regex_parser::utils::UnicodeStringView, 51
expect_end
wr22::regex_parser::parser::Parser< Iter, Sentinel >, 39
expected
wr22::regex_parser::parser::errors::UnexpectedChar, 47
wr22::regex_parser::parser::errors::UnexpectedEnd, 49

ExpectedEnd
wr22::regex_parser::parser::errors::ExpectedEnd, 27

flavor
wr22::regex_parser::regex::capture::Name, 34

Group
wr22::regex_parser::regex::part::Group, 29

include/wr22/regex_parser/parser/errors.hpp, 55, 56
include/wr22/regex_parser/parser/regex.hpp, 56, 57
include/wr22/regex_parser/regex/capture.hpp, 57, 58
include/wr22/regex_parser/regex/named_capture_flavor.hpp, 58, 59
include/wr22/regex_parser/regex/part.hpp, 59, 60
include/wr22/regex_parser/utils/adt.hpp, 61, 62
include/wr22/regex_parser/utils/box.hpp, 63
include/wr22/regex_parser/utils/utf8_string_view.hpp, 64, 65
Index
wr22::regex_parser::regex::capture::Index, 30
inner
wr22::regex_parser::regex::part::Group, 29
wr22::regex_parser::regex::part::Optional, 36
wr22::regex_parser::regex::part::Plus, 43
wr22::regex_parser::regex::part::Star, 46
items
wr22::regex_parser::regex::part::Sequence, 45

Literal
wr22::regex_parser::regex::part::Literal, 32

m_variant
wr22::regex_parser::utils::Adt< Variants >, 20
MultiCallable
wr22::regex_parser::utils::detail::adt::MultiCallable< Fs >, 33

Name
wr22::regex_parser::regex::capture::Name, 34
name
wr22::regex_parser::regex::capture::Name, 34
NamedCaptureFlavor
wr22::regex_parser::regex, 11
None
wr22::regex_parser::regex::capture::None, 35

operator!=
wr22::regex_parser::utils, 15

wr22::regex_parser::utils::UnicodeStringViewIterator, 53
operator<<
wr22::regex_parser::regex, 12
operator*
wr22::regex_parser::utils::Box< T >, 24
wr22::regex_parser::utils::UnicodeStringViewIterator, 53
operator++
wr22::regex_parser::utils::UnicodeStringViewIterator, 53
operator==
wr22::regex_parser::regex::capture::Index, 30
wr22::regex_parser::regex::capture::Name, 34
wr22::regex_parser::regex::capture::None, 35
wr22::regex_parser::regex::part::Alternatives, 21
wr22::regex_parser::regex::part::Empty, 26
wr22::regex_parser::regex::part::Group, 29
wr22::regex_parser::regex::part::Literal, 32
wr22::regex_parser::regex::part::Optional, 36
wr22::regex_parser::regex::part::Plus, 43
wr22::regex_parser::regex::part::Sequence, 45
wr22::regex_parser::regex::part::Star, 46
wr22::regex_parser::utils, 15
wr22::regex_parser::utils::UnicodeStringViewIterator, 53
Optional
wr22::regex_parser::regex::part::Optional, 36

parse_alternatives
wr22::regex_parser::parser::Parser< Iter, Sentinel >, 39
parse_atom
wr22::regex_parser::parser::Parser< Iter, Sentinel >, 39
parse_char_literal
wr22::regex_parser::parser::Parser< Iter, Sentinel >, 40
parse_group
wr22::regex_parser::parser::Parser< Iter, Sentinel >, 40
parse_group_name
wr22::regex_parser::parser::Parser< Iter, Sentinel >, 40
parse_regex
wr22::regex_parser::parser, 10
wr22::regex_parser::parser::Parser< Iter, Sentinel >, 40
parse_sequence
wr22::regex_parser::parser::Parser< Iter, Sentinel >, 41
parse_sequence_or_empty
wr22::regex_parser::parser::Parser< Iter, Sentinel >, 41
Parser
wr22::regex_parser::parser, 10
wr22::regex_parser::parser::Parser< Iter, Sentinel >, 38
Plus
wr22::regex_parser::regex::part::Plus, 43
position
wr22::regex_parser::parser::errors::ExpectedEnd, 28
wr22::regex_parser::parser::errors::UnexpectedChar, 48
wr22::regex_parser::parser::errors::UnexpectedEnd, 49

raw
wr22::regex_parser::utils::UnicodeStringView, 51

Sequence
wr22::regex_parser::regex::part::Sequence, 44
src/parser/capture.cpp, 65
src/parser/errors.cpp, 66
src/parser/regex.cpp, 66
src/regex/named_capture_flavor.cpp, 67
src/regex/part.cpp, 67
src/utils/box.cpp, 68
src/utils/utf8_string_view.cpp, 68
Star
wr22::regex_parser::regex::part::Star, 46

UnexpectedChar
wr22::regex_parser::parser::errors::UnexpectedChar, 47
UnexpectedEnd
wr22::regex_parser::parser::errors::UnexpectedEnd, 49
UnicodeStringView
wr22::regex_parser::utils::UnicodeStringView, 50
wr22::regex_parser::utils::UnicodeStringViewIterator, 54
UnicodeStringViewIterator
wr22::regex_parser::utils::UnicodeStringView, 51
wr22::regex_parser::utils::UnicodeStringViewIterator, 53
utf_traits
wr22::regex_parser::utils, 14

value_type
wr22::regex_parser::utils::UnicodeStringViewIterator, 52
VariantType
wr22::regex_parser::utils::Adt< Variants >, 18
visit
wr22::regex_parser::utils::Adt< Variants >, 19

what
wr22::regex_parser::utils::BoxIsEmpty, 25
wr22::regex_parser::utils::UnicodeStringView::InvalidUtf8, 31
wr22, 9
wr22::regex_parser, 9
wr22::regex_parser::parser, 9
parse_regex, 10
Parser, 10
wr22::regex_parser::parser::errors, 10

wr22::regex_parser::parser::errors::ExpectedEnd, 27
char_got, 28
ExpectedEnd, 27
position, 28
wr22::regex_parser::parser::errors::ParseError, 37
wr22::regex_parser::parser::errors::UnexpectedChar, 46
char_got, 47
expected, 47
position, 48
UnexpectedChar, 47
wr22::regex_parser::parser::errors::UnexpectedEnd, 48
expected, 49
position, 49
UnexpectedEnd, 49
wr22::regex_parser::parser::Parser< Iter, Sentinel >, 37
expect_end, 39
parse_alternatives, 39
parse_atom, 39
parse_char_literal, 40
parse_group, 40
parse_group_name, 40
parse_regex, 40
parse_sequence, 41
parse_sequence_or_empty, 41
Parser, 38
wr22::regex_parser::regex, 11
Angles, 11
AnglesWithP, 11
Apostrophes, 11
NamedCaptureFlavor, 11
operator<<, 12
wr22::regex_parser::regex::Capture, 25
wr22::regex_parser::regex::capture, 12
Adt, 12
wr22::regex_parser::regex::capture::Index, 30
Index, 30
operator==, 30
wr22::regex_parser::regex::capture::Name, 33
flavor, 34
Name, 34
name, 34
operator==, 34
wr22::regex_parser::regex::capture::None, 35
None, 35
operator==, 35
wr22::regex_parser::regex::Part, 42
wr22::regex_parser::regex::part, 13
Adt, 13
wr22::regex_parser::regex::part::Alternatives, 20
Alternatives, 20
alternatives, 21
operator==, 21
wr22::regex_parser::regex::part::Empty, 26
Empty, 26
operator==, 26
wr22::regex_parser::regex::part::Group, 28
capture, 29
Group, 29
inner, 29
operator==, 29
wr22::regex_parser::regex::part::Literal, 31
character, 32
Literal, 32
operator==, 32
wr22::regex_parser::regex::part::Optional, 35
inner, 36
operator==, 36
Optional, 36
wr22::regex_parser::regex::part::Plus, 43
inner, 43
operator==, 43
Plus, 43
wr22::regex_parser::regex::part::Sequence, 44
items, 45
operator==, 45
Sequence, 44
wr22::regex_parser::regex::part::Star, 45
inner, 46
operator==, 46
Star, 46
wr22::regex_parser::utils, 14
Box, 15
operator!=, 15
operator==, 15
utf_traits, 14
wr22::regex_parser::utils::Adt< Variants >, 17
Adt, 18
as_variant, 19
m_variant, 20
VariantType, 18
visit, 19
wr22::regex_parser::utils::Box< T >, 21
Box, 22
construct_in_place, 24
operator*, 24
wr22::regex_parser::utils::BoxIsEmpty, 25
what, 25
wr22::regex_parser::utils::detail, 16
wr22::regex_parser::utils::detail::adt, 16
wr22::regex_parser::utils::detail::adt::MultiCallable< Fs >, 33
MultiCallable, 33
wr22::regex_parser::utils::UnicodeStringView, 49
begin, 51
end, 51
raw, 51
UnicodeStringView, 50
UnicodeStringViewIterator, 51
wr22::regex_parser::utils::UnicodeStringView::InvalidUtf8, 31
what, 31
wr22::regex_parser::utils::UnicodeStringViewIterator, 51
category_type, 52
operator!=, 53

operator*, 53
operator++, 53
operator==, 53
UnicodeStringView, 54
UnicodeStringViewIterator, 53
value_type, 52