

Writing Regexprs 2021-22 / Regex Parser

Generated by Doxygen 1.9.2

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	11
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	12
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	13
5.8.1 Typedef Documentation	14
5.8.1.1 utf_traits	14
5.8.2 Function Documentation	14
5.8.2.1 operator!=(())	14
5.8.2.2 operator==(())	15
5.9 wr22::regex_parser::utils::detail Namespace Reference	15
5.10 wr22::regex_parser::utils::detail::adt Namespace Reference	15
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 Member Function Documentation	20
6.2.2.1 operator==()	20
6.2.3 Member Data Documentation	21
6.2.3.1 alternatives	21
6.3 wr22::regex_parser::regex::Capture Class Reference	21
6.4 wr22::regex_parser::regex::part::Empty Struct Reference	21
6.4.1 Detailed Description	22
6.4.2 Member Function Documentation	22
6.4.2.1 operator==()	22
6.5 wr22::regex_parser::parser::errors::ExpectedEnd Class Reference	22
6.5.1 Detailed Description	22
6.5.2 Constructor & Destructor Documentation	23
6.5.2.1 ExpectedEnd()	23
6.5.3 Member Function Documentation	23
6.5.3.1 char_got()	23
6.5.3.2 position()	23
6.6 wr22::regex_parser::regex::part::Group Struct Reference	23
6.6.1 Detailed Description	24
6.6.2 Member Function Documentation	24
6.6.2.1 operator==()	24
6.6.3 Member Data Documentation	24
6.6.3.1 capture	24
6.6.3.2 inner	25
6.7 wr22::regex_parser::regex::capture::Index Struct Reference	25
6.7.1 Member Function Documentation	25
6.7.1.1 operator==()	25
6.8 wr22::regex_parser::utils::UnicodeStringView::InvalidUtf8 Struct Reference	25

6.8.1 Detailed Description	26
6.8.2 Member Function Documentation	26
6.8.2.1 what()	26
6.9 wr22::regex_parser::regex::part::Literal Struct Reference	26
6.9.1 Detailed Description	26
6.9.2 Member Function Documentation	27
6.9.2.1 operator==()	27
6.9.3 Member Data Documentation	27
6.9.3.1 character	27
6.10 wr22::regex_parser::utils::detail::adt::MultiCallable< Fs > Struct Template Reference	27
6.10.1 Constructor & Destructor Documentation	27
6.10.1.1 MultiCallable()	28
6.11 wr22::regex_parser::regex::capture::Name Struct Reference	28
6.11.1 Member Function Documentation	28
6.11.1.1 operator==()	28
6.11.2 Member Data Documentation	28
6.11.2.1 name	28
6.12 wr22::regex_parser::regex::capture::None Struct Reference	29
6.12.1 Member Function Documentation	29
6.12.1.1 operator==()	29
6.13 wr22::regex_parser::parser::errors::ParseError Struct Reference	29
6.13.1 Detailed Description	29
6.14 wr22::regex_parser::parser::Parser< Iter, Sentinel > Class Template Reference	30
6.14.1 Detailed Description	30
6.14.2 Constructor & Destructor Documentation	30
6.14.2.1 Parser()	30
6.14.3 Member Function Documentation	31
6.14.3.1 expect_end()	31
6.14.3.2 parse_alternatives()	31
6.14.3.3 parse_atom()	31
6.14.3.4 parse_char_literal()	32
6.14.3.5 parse_group()	32
6.14.3.6 parse_regex()	33
6.14.3.7 parse_sequence()	33
6.15 wr22::regex_parser::regex::Part Class Reference	33
6.15.1 Detailed Description	34
6.16 wr22::regex_parser::regex::part::Sequence Struct Reference	34
6.16.1 Detailed Description	35
6.16.2 Member Function Documentation	35
6.16.2.1 operator==()	35
6.16.3 Member Data Documentation	35
6.16.3.1 items	35

6.17 wr22::regex_parser::parser::errors::UnexpectedEnd Class Reference	35
6.17.1 Detailed Description	36
6.17.2 Constructor & Destructor Documentation	36
6.17.2.1 UnexpectedEnd()	36
6.17.3 Member Function Documentation	36
6.17.3.1 expected()	36
6.17.3.2 position()	37
6.18 wr22::regex_parser::utils::UnicodeStringView Class Reference	37
6.18.1 Detailed Description	37
6.18.2 Constructor & Destructor Documentation	38
6.18.2.1 UnicodeStringView()	38
6.18.3 Member Function Documentation	38
6.18.3.1 begin()	38
6.18.3.2 end()	38
6.18.3.3 raw()	38
6.18.4 Friends And Related Function Documentation	39
6.18.4.1 UnicodeStringViewIterator	39
6.19 wr22::regex_parser::utils::UnicodeStringViewIterator Class Reference	39
6.19.1 Detailed Description	40
6.19.2 Member Typedef Documentation	40
6.19.2.1 category_type	40
6.19.2.2 value_type	40
6.19.3 Constructor & Destructor Documentation	40
6.19.3.1 UnicodeStringViewIterator()	40
6.19.4 Member Function Documentation	40
6.19.4.1 operator!=(())	40
6.19.4.2 operator*()	40
6.19.4.3 operator++()	41
6.19.4.4 operator==(())	41
6.19.5 Friends And Related Function Documentation	41
6.19.5.1 UnicodeStringView	41
7 File Documentation	43
7.1 include/wr22/regex_parser/parser/errors.hpp File Reference	43
7.2 errors.hpp	43
7.3 include/wr22/regex_parser/parser/regex.hpp File Reference	44
7.4 regex.hpp	44
7.5 include/wr22/regex_parser/regex/capture.hpp File Reference	45
7.6 capture.hpp	45
7.7 include/wr22/regex_parser/regex/named_capture_flavor.hpp File Reference	46
7.8 named_capture_flavor.hpp	46
7.9 include/wr22/regex_parser/regex/part.hpp File Reference	47

7.10 part.hpp	48
7.11 include/wr22/regex_parser/utils/adt.hpp File Reference	48
7.12 adt.hpp	49
7.13 include/wr22/regex_parser/utils/utf8_string_view.hpp File Reference	50
7.14 utf8_string_view.hpp	50
7.15 src/parser/capture.cpp File Reference	51
7.16 src/parser/errors.cpp File Reference	52
7.17 src/parser/regex.cpp File Reference	52
7.18 src/regex/named_capture_flavor.cpp File Reference	53
7.19 src/regex/part.cpp File Reference	53
7.20 src/utils/utf8_string_view.cpp File Reference	53
Index	55

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	13
wr22::regex_parser::utils::detail	15
wr22::regex_parser::utils::detail::adt	15

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	21
wr22::regex_parser::regex::Part	33
wr22::regex_parser::regex::part::Alternatives	20
wr22::regex_parser::regex::part::Empty	21
std::exception	
wr22::regex_parser::utils::UnicodeStringView::InvalidUtf8	25
wr22::regex_parser::regex::part::Group	23
wr22::regex_parser::regex::capture::Index	25
wr22::regex_parser::regex::part::Literal	26
wr22::regex_parser::regex::capture::Name	28
wr22::regex_parser::regex::capture::None	29
wr22::regex_parser::parser::Parser< Iter, Sentinel >	30
std::runtime_error	
wr22::regex_parser::parser::errors::ParseError	29
wr22::regex_parser::parser::errors::ExpectedEnd	22
wr22::regex_parser::parser::errors::UnexpectedEnd	35
wr22::regex_parser::regex::part::Sequence	34
wr22::regex_parser::utils::UnicodeStringView	37
wr22::regex_parser::utils::UnicodeStringViewIterator	39
wr22::regex_parser::utils::detail::adt::Fs	
wr22::regex_parser::utils::detail::adt::MultiCallable< Fs >	27

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 >	17
A helper class that simplifies creation of algebraic data types	
wr22::regex_parser::regex::part::Alternatives	20
A regex part with the list of alternatives to be matched	
wr22::regex_parser::regex::Capture	21
wr22::regex_parser::regex::part::Empty	21
An empty regex part	
wr22::regex_parser::parser::errors::ExpectedEnd	22
The error when the parser expected the input to end, but it did not	
wr22::regex_parser::regex::part::Group	23
A regex part that represents a group in parentheses	
wr22::regex_parser::regex::capture::Index	25
wr22::regex_parser::utils::UnicodeStringView::InvalidUtf8	25
An error thrown when the string's encoding is not valid UTF-8	
wr22::regex_parser::regex::part::Literal	26
An regex part that matches a single character literally	
wr22::regex_parser::utils::detail::adt::MultiCallable< Fs >	27
wr22::regex_parser::regex::capture::Name	28
wr22::regex_parser::regex::capture::None	29
wr22::regex_parser::parser::errors::ParseError	29
The base class for parse errors	
wr22::regex_parser::parser::Parser< Iter, Sentinel >	30
A regex parser	
wr22::regex_parser::regex::Part	33
A part of a regular expression and its AST node type	
wr22::regex_parser::regex::part::Sequence	34
A regex part with the list of items to be matched one after another	
wr22::regex_parser::parser::errors::UnexpectedEnd	35
The error when the parser hit the end of the input earlier than it expected	
wr22::regex_parser::utils::UnicodeStringView	37
A wrapper around <code>std::string_view</code> that enables UTF-8 codepoint iteration	
wr22::regex_parser::utils::UnicodeStringViewIterator	39
Iterator over code points for <code>UnicodeStringView</code>	

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	43
include/wr22/regex_parser/parser/ regex.hpp	44
include/wr22/regex_parser/regex/ capture.hpp	45
include/wr22/regex_parser/regex/ named_capture_flavor.hpp	46
include/wr22/regex_parser/regex/ part.hpp	47
include/wr22/regex_parser/utls/ adt.hpp	48
include/wr22/regex_parser/utls/ utf8_string_view.hpp	50
src/parser/ capture.cpp	51
src/parser/ errors.cpp	52
src/parser/ regex.cpp	52
src/regex/ named_capture_flavor.cpp	53
src/regex/ part.cpp	53
src/utls/ utf8_string_view.cpp	53

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

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

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

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

5.3.1.2 `Parser()`

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

The type deduction guideline for `Parser`.

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

Classes

- class `ExpectedEnd`
The error when the parser expected the input to end, but it did not.
- struct `ParseError`
The base class for parse errors.
- class `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](#)
- class [Part](#)

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

Enumerations

- enum class [NamedCaptureFlavor](#) { [Apostrophes](#) , [Angles](#) , [AnglesWithP](#) }

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]
```

Enumerator

Apostrophes	
Angles	
AnglesWithP	

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](#)
- struct [Name](#)
- struct [None](#)

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](#)
A regex part that matches a single character literally.
- struct [Sequence](#)
A regex part with the list of items to be matched one after another.

Typedefs

- using [Adt](#) = [utils::Adt](#)< [Empty](#), [Literal](#), [Alternatives](#), [Sequence](#), [Group](#) >

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>
```

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 [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.

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 operator"!=()

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

Compare two compatible ADTs for non-equality.

5.8.2.2 operator==()

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

Compare two compatible ADTs for equality.

5.9 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

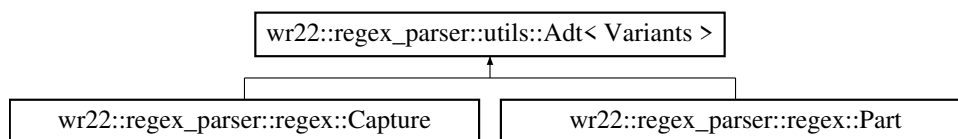
Class Documentation

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

- 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 [Parts](#).

6.2.2 Member Function Documentation

6.2.2.1 operator==()

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

6.2.3 Member Data Documentation

6.2.3.1 alternatives

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

The list of the alternatives.

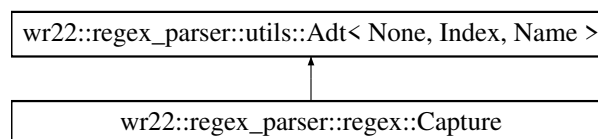
The documentation for this struct was generated from the following file:

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

6.3 wr22::regex_parser::regex::Capture Class Reference

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

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



Additional Inherited Members

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

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

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

An empty regex part.

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

Public Member Functions

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

6.4.1 Detailed Description

An empty regex part.

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

6.4.2 Member Function Documentation

6.4.2.1 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:

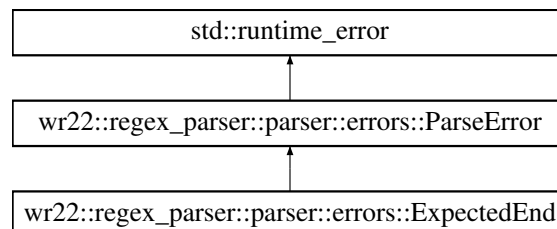
- include/wr22/regex_parser/regex/part.hpp

6.5 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.

6.5.1 Detailed Description

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

6.5.2 Constructor & Destructor Documentation

6.5.2.1 ExpectedEnd()

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

Constructor.

Parameters

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

6.5.3 Member Function Documentation

6.5.3.1 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.

6.5.3.2 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](#)

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

A regex part that represents a group in parentheses.

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

Public Member Functions

- bool `operator==` (const `Group` &rhs) const

Public Attributes

- `Capture` `capture`
`Capture` behavior.
- `std::unique_ptr< Part > inner`
The (smart) pointer to the group contents.

6.6.1 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.

6.6.2 Member Function Documentation

6.6.2.1 `operator==()`

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

6.6.3 Member Data Documentation

6.6.3.1 `capture`

`Capture` wr22::regex_parser::regex::part::Group::capture

`Capture` behavior.

6.6.3.2 inner

```
std::unique_ptr<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

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

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

Public Member Functions

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

6.7.1 Member Function Documentation

6.7.1.1 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:

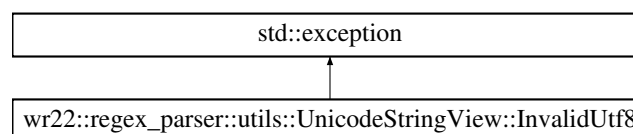
- include/wr22/regex_parser/regex/capture.hpp

6.8 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

6.8.1 Detailed Description

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

6.8.2 Member Function Documentation

6.8.2.1 `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`

6.9 `wr22::regex_parser::regex::part::Literal` Struct Reference

An regex part that matches a single character literally.

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

Public Member Functions

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

Public Attributes

- `char32_t character`

6.9.1 Detailed Description

An regex part that matches a single character literally.

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

6.9.2 Member Function Documentation

6.9.2.1 operator==()

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

6.9.3 Member Data Documentation

6.9.3.1 character

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

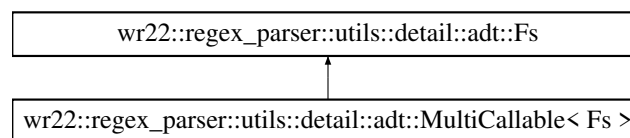
The documentation for this struct was generated from the following file:

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

6.10 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)

6.10.1 Constructor & Destructor Documentation

6.10.1.1 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](#)

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

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

Public Member Functions

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

Public Attributes

- std::string [name](#)

6.11.1 Member Function Documentation

6.11.1.1 operator==()

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

6.11.2 Member Data Documentation

6.11.2.1 name

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

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

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

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

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

Public Member Functions

- bool `operator==` (const `None` &rhs) const =default

6.12.1 Member Function Documentation

6.12.1.1 `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:

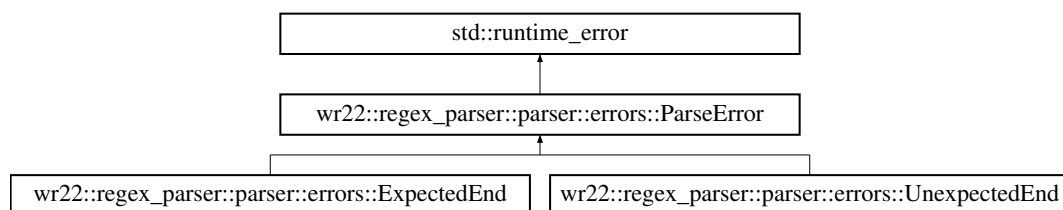
- include/wr22/regex_parser/regex/capture.hpp

6.13 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:



6.13.1 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

6.14 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_atom](#) ()
Intermediate rule: parse an atom.
- [regex::Part parse_char_literal](#) ()
Intermediate rule: parse a character literal.
- [regex::Part parse_group](#) ()
[WIP] Intermediate rule: parse a parenthesized group (any capture variant).

6.14.1 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.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 Parser()

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

Constructor.

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

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

6.14.3 Member Function Documentation

6.14.3.1 expect_end()

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

Ensure that the parser has consumed all of the input.

Does nothing if all input has been consumed.

Exceptions

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

6.14.3.2 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

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

6.14.3.3 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

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

6.14.3.4 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

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

6.14.3.5 parse_group()

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

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

Not currently implemented.

Returns

the parsed group (`regex::part::Group`).

6.14.3.6 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

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

6.14.3.7 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

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

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

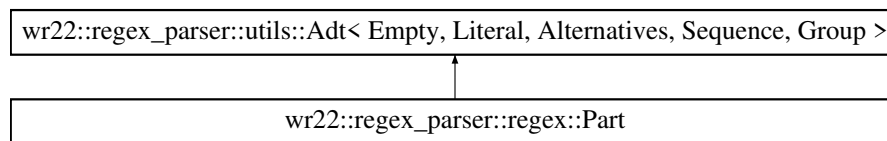
- `src/parser/regex.cpp`

6.15 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

6.15.1 Detailed Description

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

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

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

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

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

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

6.16 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

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

Public Attributes

- `std::vector< Part > items`
The list of the subexpressions.

6.16.1 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`.

6.16.2 Member Function Documentation

6.16.2.1 operator==()

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

6.16.3 Member Data Documentation

6.16.3.1 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 file:

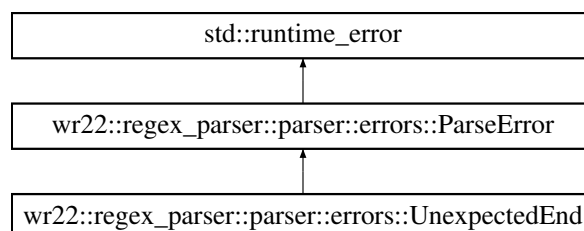
- `include/wr22/regex_parser/regex/part.hpp`

6.17 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.

6.17.1 Detailed Description

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

6.17.2 Constructor & Destructor Documentation

6.17.2.1 UnexpectedEnd()

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

Constructor.

Parameters

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

6.17.3 Member Function Documentation

6.17.3.1 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.

6.17.3.2 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

6.18 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](#)

6.18.1 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`.

6.18.2 Constructor & Destructor Documentation

6.18.2.1 `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.

6.18.3 Member Function Documentation

6.18.3.1 `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.

6.18.3.2 `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.

6.18.3.3 `raw()`

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

Access the wrapped `std::string_view`.

6.18.4 Friends And Related Function Documentation

6.18.4.1 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

6.19 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](#)

6.19.1 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.

6.19.2 Member Typedef Documentation

6.19.2.1 category_type

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

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

6.19.2.2 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.

6.19.3 Constructor & Destructor Documentation

6.19.3.1 UnicodeStringViewIterator()

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

The default constructor is meaningless and is thus deleted.

6.19.4 Member Function Documentation

6.19.4.1 operator"!="()

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

6.19.4.2 operator*()

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

Iterator interface: dereferencing.

Exceptions

<code>UnicodeStringView::InvalidUtf8</code>	if the codepoint decodes into an invalid or incomplete value.
---	---

6.19.4.3 operator++()

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

Iterator interface: pre-increment.

Exceptions

<code>UnicodeStringView::InvalidUtf8</code>	if the codepoint decodes into an invalid or incomplete value.
---	---

6.19.4.4 operator==()

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

6.19.5 Friends And Related Function Documentation**6.19.5.1 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.

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:
22     UnexpectedEnd(size_t position, std::string expected);
23
24     size_t position() const;
25     const std::string& expected() const;
26
27 private:
28     size_t m_position;
29     std::string m_expected;
30 };
31
32 class ExpectedEnd : public ParseError {
33 public:
34     ExpectedEnd(size_t position, char32_t char_got);
35
36     size_t position() const;
37     char32_t char_got() const;
38
39 private:
40     size_t m_position;
41     char32_t m_char_got;
42 };
43
44 } // 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 ®ex)
Parse a regular expression into its AST.

7.4 regex.hpp

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

```

1 #pragma once
2
3 // wr22
4 #include <wr22/regex_parser/regex/part.hpp>
5 #include <wr22/regex_parser/utils/utf8_string_view.hpp>
6
7 // stl
8 #include <string_view>
9
10 namespace wr22::regex_parser::parser {
11
12     regex::Part parse_regex(const utils::UnicodeStringView& regex);
13
14 } // namespace wr22::regex_parser::parser

```

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

```
#include <wr22/regex_parser/utils/adt.hpp>
#include <iosfwd>
#include <string>
```

Classes

- struct [wr22::regex_parser::regex::capture::None](#)
- struct [wr22::regex_parser::regex::capture::Index](#)
- struct [wr22::regex_parser::regex::capture::Name](#)
- class [wr22::regex_parser::regex::Capture](#)

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/utils/adt.hpp>
5
6 // stl
7 #include <iosfwd>
8 #include <string>
9
10 namespace wr22::regex_parser::regex {
11
12     class Capture;
13
14     namespace capture {
15         struct None {
16             bool operator==(const None& rhs) const = default;
17         };
18
19         struct Index {
20             bool operator==(const Index& rhs) const = default;
21         };
22
23         struct Name {
24             std::string name;
25             // TODO: flavor.
26             bool operator==(const Name& rhs) const = default;
```

```

27     };
28
29     using Adt = utils::Adt<None, Index, Name>;
30 } // namespace capture
31
32 class Capture : public capture::Adt {
33 public:
34     using capture::Adt::Adt;
35 };
36
37 std::ostream& operator<<(std::ostream& out, const Capture& capture);
38
39 } // 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](#) }

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
8     enum class NamedCaptureFlavor
9     {
10         Apostrophes,
11         Angles,
12         AnglesWithP,
13     };
14
15 std::ostream& operator<<(std::ostream& out, NamedCaptureFlavor flavor);
16
17 } // 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 <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.
- 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](#) >

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
7 // stl
8 #include <iosfwd>
9 #include <memory>
10 #include <vector>
11
12 namespace wr22::regex_parser::regex {
13
14 // Forward declaration of 'Part'.
15 class Part;
16
17 namespace part {
18     struct Empty {
19         bool operator==(const Empty& rhs) const = default;
20     };
21
22     struct Literal {
23         char32_t character;
24         bool operator==(const Literal& rhs) const = default;
25     };
26
27     struct Alternatives {
28         std::vector<Part> alternatives;
29         bool operator==(const Alternatives& rhs) const = default;
30     };
31
32     struct Sequence {
33         std::vector<Part> items;
34         bool operator==(const Sequence& rhs) const = default;
35     };
36
37     struct Group {
38         Capture capture;
39         std::unique_ptr<Part> inner;
40         bool operator==(const Group& rhs) const;
41     };
42
43     using Adt = utils::Adt<Empty, Literal, Alternatives, Sequence, Group>;
44 } // namespace part
45
46 class Part : public part::Adt {
47 public:
48     using part::Adt::Adt;
49 };
50
51 std::ostream& operator<<(std::ostream& out, const Part& part);
52
53 } // 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         };
16     } // namespace detail::adt
17
18     template <typename... Variants>
19     class Adt {
20     public:
21         using VariantType = std::variant<Variants...>;
22
23         template <typename V>
24         Adt(V variant) : m_variant(std::move(variant)) {}
25
26         template <typename... Fs>
27         decltype(auto) visit(Fs&&... visitors) const {
28             return std::visit(
29                 detail::adt::MultiCallable<Fs...>(std::forward<Fs>(visitors)...),
30                 m_variant);
31         }
32
33         template <typename... Fs>
34         decltype(auto) visit(Fs&&... visitors) {
35             return std::visit(
36                 detail::adt::MultiCallable<Fs...>(std::forward<Fs>(visitors)...),
37                 m_variant);
38         }
39
40         const VariantType& as_variant() const {
41             return m_variant;
42         }
43
44         VariantType& as_variant() {
45             return m_variant;
46         }
47     protected:
48         VariantType m_variant;
49     };

```

```

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

```

7.13 include/wr22/regex_parser/utils/utf8_string_view.hpp File Reference

```

#include <compare>
#include <cstdint>
#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](#)

7.14 utf8_string_view.hpp

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

```

1 #pragma once
2
3 // STL
4 #include <compare>
5 #include <cstdint>
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
15     class UnicodeStringView {
16     friend class UnicodeStringViewIterator;
17

```

```

24
25 public:
26     struct InvalidUtf8 : std::exception {
27         const char* what() const noexcept override;
28     };
29
30
42     explicit UnicodeStringView(const std::string_view& raw);
43
44     const std::string_view& raw() const;
45
46     UnicodeStringViewIterator begin() const;
47
48     UnicodeStringViewIterator end() const;
49
50 private:
51     std::string_view m_raw;
52 };
53
54 class UnicodeStringViewIterator {
55     friend class UnicodeStringView;
56
57 public:
58     using value_type = char32_t;
59
60     using category_type = std::forward_iterator_tag;
61
62     UnicodeStringViewIterator() = delete;
63
64     UnicodeStringViewIterator& operator++();
65
66     char32_t operator*();
67
68     bool operator==(const UnicodeStringViewIterator& rhs) const;
69     bool operator!=(const UnicodeStringViewIterator& rhs) const;
70
71 private:
72     using UnderlyingIterator = std::string_view::const_iterator;
73
74     explicit UnicodeStringViewIterator(
75         const UnderlyingIterator& iter,
76         const UnderlyingIterator& end);
77     std::optional<char32_t> decode_current_codepoint();
78     void update_current_codepoint_size();
79
80     UnderlyingIterator m_iter;
81     UnderlyingIterator m_end;
82
83     // A memoization of the result of decoding of the last code point.
84     // This is done purely for optimization to avoid having to decode a code point
85     // twice: for dereferencing the iterator and for incrementing it. The value
86     // of 0 indicates an absence of a memoized value (0 is not a valid byte size of
87     // a codepoint).
88     size_t m_last_codepoint_size = 0;
89 };
90
91 } // namespace wr22::regex_parser::utils

```

7.15 src/parser/capture.cpp File Reference

```

#include <wr22/regex_parser/regex/capture.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 Capture &capture`)

7.16 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`

7.17 src/parser/regex.cpp File Reference

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

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 ®ex`)
Parse a regular expression into its AST.

7.18 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)`

7.19 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`.

7.20 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, [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, [38](#)

capture

- wr22::regex_parser::regex::part::Group, [24](#)

category_type

- wr22::regex_parser::utils::UnicodeStringViewIterator, [40](#)

char_got

- wr22::regex_parser::parser::errors::ExpectedEnd, [23](#)

character

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

end

- wr22::regex_parser::utils::UnicodeStringView, [38](#)

expect_end

- wr22::regex_parser::parser::Parser< Iter, Sentinel >, [31](#)

expected

- wr22::regex_parser::parser::errors::UnexpectedEnd, [36](#)

ExpectedEnd

- wr22::regex_parser::parser::errors::ExpectedEnd, [23](#)

include/wr22/regex_parser/parser/errors.hpp, [43](#)

include/wr22/regex_parser/parser/regex.hpp, [44](#)

include/wr22/regex_parser/regex/capture.hpp, [45](#)

include/wr22/regex_parser/regex/named_capture_flavor.hpp, [46](#)

include/wr22/regex_parser/regex/part.hpp, [47](#), [48](#)

include/wr22/regex_parser/utils/adt.hpp, [48](#), [49](#)

include/wr22/regex_parser/utils/utf8_string_view.hpp, [50](#)

inner

- wr22::regex_parser::regex::part::Group, [24](#)

items

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

m_variant

- wr22::regex_parser::utils::Adt< Variants >, [20](#)

MultiCallable

- wr22::regex_parser::utils::detail::adt::MultiCallable< Fs >, [27](#)

name

- wr22::regex_parser::regex::capture::Name, [28](#)

NamedCaptureFlavor

- wr22::regex_parser::regex, [11](#)

operator!=

- wr22::regex_parser::utils, [14](#)

- wr22::regex_parser::utils::UnicodeStringViewIterator, [40](#)

operator<<

- wr22::regex_parser::regex, [11](#), [12](#)

operator*

- wr22::regex_parser::utils::UnicodeStringViewIterator, [40](#)

operator++

- wr22::regex_parser::utils::UnicodeStringViewIterator, [41](#)

operator==

- wr22::regex_parser::regex::capture::Index, [25](#)

- wr22::regex_parser::regex::capture::Name, [28](#)

- wr22::regex_parser::regex::capture::None, [29](#)

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

- wr22::regex_parser::regex::part::Empty, [22](#)

- wr22::regex_parser::regex::part::Group, [24](#)

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

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

- wr22::regex_parser::utils, [14](#)

- wr22::regex_parser::utils::UnicodeStringViewIterator, [41](#)

parse_alternatives

- wr22::regex_parser::parser::Parser< Iter, Sentinel >, [31](#)

parse_atom

- wr22::regex_parser::parser::Parser< Iter, Sentinel >, [31](#)

parse_char_literal

- wr22::regex_parser::parser::Parser< Iter, Sentinel >, [32](#)

- parse_group
 - wr22::regex_parser::parser::Parser< Iter, Sentinel >, 32
- parse_regex
 - wr22::regex_parser::parser, 10
 - wr22::regex_parser::parser::Parser< Iter, Sentinel >, 32
- parse_sequence
 - wr22::regex_parser::parser::Parser< Iter, Sentinel >, 33
- Parser
 - wr22::regex_parser::parser, 10
 - wr22::regex_parser::parser::Parser< Iter, Sentinel >, 30
- position
 - wr22::regex_parser::parser::errors::ExpectedEnd, 23
 - wr22::regex_parser::parser::errors::UnexpectedEnd, 36
- raw
 - wr22::regex_parser::utils::UnicodeStringView, 38
- src/parser/capture.cpp, 51
- src/parser/errors.cpp, 52
- src/parser/regex.cpp, 52
- src/regex/named_capture_flavor.cpp, 53
- src/regex/part.cpp, 53
- src/utils/utf8_string_view.cpp, 53
- UnexpectedEnd
 - wr22::regex_parser::parser::errors::UnexpectedEnd, 36
- UnicodeStringView
 - wr22::regex_parser::utils::UnicodeStringView, 38
 - wr22::regex_parser::utils::UnicodeStringViewIterator, 41
- UnicodeStringViewIterator
 - wr22::regex_parser::utils::UnicodeStringView, 39
 - wr22::regex_parser::utils::UnicodeStringViewIterator, 40
- utf_traits
 - wr22::regex_parser::utils, 14
- value_type
 - wr22::regex_parser::utils::UnicodeStringViewIterator, 40
- VariantType
 - wr22::regex_parser::utils::Adt< Variants >, 18
- visit
 - wr22::regex_parser::utils::Adt< Variants >, 19
- what
 - wr22::regex_parser::utils::UnicodeStringView::InvalidUtf8, 26
- 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, 22
 - char_got, 23
 - ExpectedEnd, 23
 - position, 23
 - wr22::regex_parser::parser::errors::ParseError, 29
 - wr22::regex_parser::parser::errors::UnexpectedEnd, 35
 - expected, 36
 - position, 36
 - UnexpectedEnd, 36
 - wr22::regex_parser::parser::Parser< Iter, Sentinel >, 30
 - expect_end, 31
 - parse_alternatives, 31
 - parse_atom, 31
 - parse_char_literal, 32
 - parse_group, 32
 - parse_regex, 32
 - parse_sequence, 33
 - Parser, 30
 - wr22::regex_parser::regex, 11
 - Angles, 11
 - AnglesWithP, 11
 - Apostrophes, 11
 - NamedCaptureFlavor, 11
 - operator<<, 11, 12
 - wr22::regex_parser::regex::Capture, 21
 - wr22::regex_parser::regex::capture, 12
 - Adt, 12
 - wr22::regex_parser::regex::capture::Index, 25
 - operator==, 25
 - wr22::regex_parser::regex::capture::Name, 28
 - name, 28
 - operator==, 28
 - wr22::regex_parser::regex::capture::None, 29
 - operator==, 29
 - wr22::regex_parser::regex::Part, 33
 - wr22::regex_parser::regex::part, 13
 - Adt, 13
 - wr22::regex_parser::regex::part::Alternatives, 20
 - alternatives, 21
 - operator==, 20
 - wr22::regex_parser::regex::part::Empty, 21
 - operator==, 22
 - wr22::regex_parser::regex::part::Group, 23
 - capture, 24
 - inner, 24
 - operator==, 24
 - wr22::regex_parser::regex::part::Literal, 26
 - character, 27
 - operator==, 27
 - wr22::regex_parser::regex::part::Sequence, 34
 - items, 35
 - operator==, 35
 - wr22::regex_parser::utils, 13
 - operator!=, 14
 - operator==, 14

- 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::detail, [15](#)
- wr22::regex_parser::utils::detail::adt, [15](#)
- wr22::regex_parser::utils::detail::adt::MultiCallable< Fs
>, [27](#)
 - MultiCallable, [27](#)
- wr22::regex_parser::utils::UnicodeStringView, [37](#)
 - begin, [38](#)
 - end, [38](#)
 - raw, [38](#)
 - UnicodeStringView, [38](#)
 - UnicodeStringViewIterator, [39](#)
- wr22::regex_parser::utils::UnicodeStringView::InvalidUtf8,
[25](#)
 - what, [26](#)
- wr22::regex_parser::utils::UnicodeStringViewIterator, [39](#)
 - category_type, [40](#)
 - operator!=, [40](#)
 - operator*, [40](#)
 - operator++, [41](#)
 - operator==, [41](#)
 - UnicodeStringView, [41](#)
 - UnicodeStringViewIterator, [40](#)
 - value_type, [40](#)