"...one of the most highly regarded and expertly designed C++ library projects in the world."
— Herb Sutter and Andrei Alexandrescu, C++ Coding Standards

POSIX Extended Regular Expression Syntax

Synopsis

The POSIX-Extended regular expression syntax is supported by the POSIX C regular expression API's, and variations are used by the utilities `egrep` and `awk`. You can construct POSIX extended regular expressions in Boost.Regex by passing the flag `extended` to the regex constructor, for example:

```
// e1 is a case sensitive POSIX-Extended expression:
boost::regex e1(my_expression, boost::regex::extended);
// e2 a case insensitive POSIX-Extended expression:
boost::regex e2(my_expression, boost::regex::extended|boost::regex::icase);
```

POSIX Extended Syntax

In POSIX-Extended regular expressions, all characters match themselves except for the following special characters:

```
.[{}()\*+?|^$
```

Wildcard:

The single character '.' when used outside of a character set will match any single character except:

- The NULL character when the flag `match_no_dot_null` is passed to the matching algorithms.
- The newline character when the flag `match_not_dot_newline` is passed to the matching algorithms.

Anchors:

A '^' character shall match the start of a line when used as the first character of an expression, or the first character of a sub-expression.

A '$' character shall match the end of a line when used as the last character of an expression, or the last character of a sub-expression.

Marked sub-expressions:

A section beginning (and ending) acts as a marked sub-expression. Whatever matched the sub-expression is split out in a separate field by the matching algorithms. Marked sub-expressions can also repeated, or referred to by a back-reference.

Repeats:

Any atom (a single character, a marked sub-expression, or a character class) can be repeated with the *, +, ?, and {} operators.

The * operator will match the preceding atom *zero or more times*, for example the expression `a*b` will match any of the following:

```
b
ab
aaaaaaaab
```

The + operator will match the preceding atom *one or more times*, for example the expression a+b will match any of the following:

```
ab
aaaaaaaab
```

But will not match:

```
b
```

The ? operator will match the preceding atom *zero or one times*, for example the expression ca?b will match any of the following:

```
cb
cab
```

But will not match:

```
caab
```

An atom can also be repeated with a bounded repeat:

a{n} Matches 'a' repeated *exactly n times*.

a{n,} Matches 'a' repeated *n or more times*.

a{n, m} Matches 'a' repeated *between n and m times inclusive*.

For example:

```
^a{2,3}$
```

Will match either of:

```
aa
aaa
```

But neither of:

```
a
aaaa
```

It is an error to use a repeat operator, if the preceding construct can not be repeated, for example:

```
a(*)
```

Will raise an error, as there is nothing for the * operator to be applied to.

Back references:

An escape character followed by a digit *n*, where *n* is in the range 1-9, matches the same string that was matched by sub-expression *n*. For example the expression:

```
^(a*)[^a]*\1$
```

Will match the string:

```
aaabbaaa
```

But not the string:

```
aaabba
```

Caution

The POSIX standard does not support back-references for "extended" regular expressions, this is a compatible extension to that standard.

Alternation

The | operator will match either of its arguments, so for example: `abc|def` will match either "abc" or "def".

Parenthesis can be used to group alternations, for example: `ab(d|ef)` will match either of "abd" or "abef".

Character sets:

A character set is a bracket-expression starting with [and ending with], it defines a set of characters, and matches any single character that is a member of that set.

A bracket expression may contain any combination of the following:

Single characters:

For example `[abc]`, will match any of the characters 'a', 'b', or 'c'.

Character ranges:

For example `[a-c]` will match any single character in the range 'a' to 'c'. By default, for POSIX-Extended regular expressions, a character *x* is within the range *y* to *z*, if it collates within that range; this results in locale specific behavior . This behavior can be turned off by unsetting the `collate` option flag - in which case whether a character appears within a range is determined by comparing the code points of the characters only.

Negation:

If the bracket-expression begins with the ^ character, then it matches the complement of the characters it contains, for example `[^a-c]` matches any character that is not in the range `a-c`.

Character classes:

An expression of the form `[[:name:]]` matches the named character class "name", for example `[[:lower:]]` matches any lower case character. See character class names.

Collating Elements:

An expression of the form `[[.col.]` matches the collating element *col*. A collating element is any single character, or any sequence of characters that collates as a single unit. Collating elements may also be used as the end point of a range, for example: `[[.ae.]-c]` matches the character sequence "ae", plus any single character in the range "ae"-c, assuming that "ae" is treated as a single collating element in the current locale.

Collating elements may be used in place of escapes (which are not normally allowed inside character sets), for example `[[.^.]abc]` would match either one of the characters 'abc^'.

As an extension, a collating element may also be specified via its symbolic name, for example:

```
[[.NUL.]]
```

matches a NUL character.

Equivalence classes:

An expression of the form `[[=col=]]`, matches any character or collating element whose primary sort key is the same as that for collating element *col*, as with collating elements the name *col* may be a symbolic name. A primary sort key is one that ignores case, accentation, or locale-specific tailorings; so for example `[[=a=]]` matches any of the characters: a, À, Á, Â, Ã, Ä, Å, A, à, á, â, ã, ä and å. Unfortunately implementation of this is reliant on the platform's collation and localisation support; this feature can not be relied upon to work portably across all platforms, or even all locales on one platform.

Combinations:

All of the above can be combined in one character set declaration, for example: `[[:digit:]a-c[.NUL.]]`.

Escapes

The POSIX standard defines no escape sequences for POSIX-Extended regular expressions, except that:

- Any special character preceded by an escape shall match itself.
- The effect of any ordinary character being preceded by an escape is undefined.
- An escape inside a character class declaration shall match itself: in other words the escape character is not "special" inside a character class declaration; so `[\^]` will match either a literal '\' or a '^'.

However, that's rather restrictive, so the following standard-compatible extensions are also supported by Boost.Regex:

Escapes matching a specific character

The following escape sequences are all synonyms for single characters:

Escape	Character
\a	'\a'
\e	0x1B
\f	\f
\n	\n
\r	\r
\t	\t
\v	\v
\b	\b (but only inside a character class declaration).
\cX	An ASCII escape sequence - the character whose code point is X % 32
\xdd	A hexadecimal escape sequence - matches the single character whose code point is 0xdd.
\x{dddd}	A hexadecimal escape sequence - matches the single character whose code point is 0xdddd.
\0ddd	An octal escape sequence - matches the single character whose code point is 0ddd.

Escape	Character
\N{Name}	Matches the single character which has the symbolic name *Name*. For example `\\N{newline}` matches the single character \n.

"Single character" character classes:

Any escaped character *x*, if *x* is the name of a character class shall match any character that is a member of that class, and any escaped character *X*, if *x* is the name of a character class, shall match any character not in that class.

The following are supported by default:

Escape sequence	Equivalent to
`\d`	`[[:digit:]]`
`\l`	`[[:lower:]]`
`\s`	`[[:space:]]`
`\u`	`[[:upper:]]`
`\w`	`[[:word:]]`
`\D`	`[^[:digit:]]`
`\L`	`[^[:lower:]]`
`\S`	`[^[:space:]]`
`\U`	`[^[:upper:]]`
`\W`	`[^[:word:]]`

Character Properties

The character property names in the following table are all equivalent to the names used in character classes.

Form	Description	Equivalent character set form
`\pX`	Matches any character that has the property X.	`[[:X:]]`
`\p{Name}`	Matches any character that has the property Name.	`[[:Name:]]`
`\PX`	Matches any character that does not have the property X.	`[^[:X:]]`
`\P{Name}`	Matches any character that does not have the property Name.	`[^[:Name:]]`

For example `\pd` matches any "digit" character, as does `\p{digit}`.

Word Boundaries

The following escape sequences match the boundaries of words:

Escape	Meaning
`\<`	Matches the start of a word.
`\>`	Matches the end of a word.
`\b`	Matches a word boundary (the start or end of a word).

Escape	Meaning
\B	Matches only when not at a word boundary.

Buffer boundaries

The following match only at buffer boundaries: a "buffer" in this context is the whole of the input text that is being matched against (note that ^ and $ may match embedded newlines within the text).

Escape	Meaning
\`	Matches at the start of a buffer only.
\'	Matches at the end of a buffer only.
\A	Matches at the start of a buffer only (the same as \`).
\z	Matches at the end of a buffer only (the same as \').
\Z	Matches an optional sequence of newlines at the end of a buffer: equivalent to the regular expression \n*\z

Continuation Escape

The sequence \G matches only at the end of the last match found, or at the start of the text being matched if no previous match was found. This escape useful if you're iterating over the matches contained within a text, and you want each subsequence match to start where the last one ended.

Quoting escape

The escape sequence \Q begins a "quoted sequence": all the subsequent characters are treated as literals, until either the end of the regular expression or \E is found. For example the expression: \Q*+\Ea+ would match either of:

```
\*+a
\*+aaa
```

Unicode escapes

Escape	Meaning
\C	Matches a single code point: in Boost regex this has exactly the same effect as a "." operator.
\X	Matches a combining character sequence: that is any non-combining character followed by a sequence of zero or more combining characters.

Any other escape

Any other escape sequence matches the character that is escaped, for example \@ matches a literal '@'.

Operator precedence

The order of precedence for of operators is as follows:

1. Collation-related bracket symbols [==] [::] [..]
2. Escaped characters \
3. Character set (bracket expression) []
4. Grouping ()
5. Single-character-ERE duplication * + ? {m,n}
6. Concatenation
7. Anchoring ^$

8. Alternation |

What Gets Matched

When there is more that one way to match a regular expression, the "best" possible match is obtained using the leftmost-longest rule.

Variations

Egrep

When an expression is compiled with the flag `egrep` set, then the expression is treated as a newline separated list of POSIX-Extended expressions, a match is found if any of the expressions in the list match, for example:

```
boost::regex e("abc\ndef", boost::regex::egrep);
```

will match either of the POSIX-Basic expressions "abc" or "def".

As its name suggests, this behavior is consistent with the Unix utility `egrep`, and with grep when used with the -E option.

awk

In addition to the POSIX-Extended features the escape character is special inside a character class declaration.

In addition, some escape sequences that are not defined as part of POSIX-Extended specification are required to be supported - however Boost.Regex supports these by default anyway.

Options

There are a variety of flags that may be combined with the `extended` and `egrep` options when constructing the regular expression, in particular note that the `newline_alt` option alters the syntax, while the `collate`, `nosubs` and `icase` options modify how the case and locale sensitivity are to be applied.

References

IEEE Std 1003.1-2001, Portable Operating System Interface (POSIX), Base Definitions and Headers, Section 9, Regular Expressions.

IEEE Std 1003.1-2001, Portable Operating System Interface (POSIX), Shells and Utilities, Section 4, Utilities, egrep.

IEEE Std 1003.1-2001, Portable Operating System Interface (POSIX), Shells and Utilities, Section 4, Utilities, awk.