Language Elements

The text/parsing framework contains a regular expression engine that accepts an extensive set of regular expression elements, enabling you to efficiently search for text patterns. This topic details the set of characters, operators, and constructs that you can use to define regular expressions.

Whitespace characters are ignored in all regular expression patterns. Use the `\s` character class to indicate whitespace matching.

This topic contains these sections:

Character Escapes

Most of the important regular expression language operators are unescaped single characters. The escape character `\` (a single backslash) signals to the regular expression parser that the character following the backslash is not an operator. For example, the parser treats an asterisk (`*`) as a repeating quantifier and a backslash followed by an asterisk (`\*`) as the Unicode character `\u002A`.

Escaped Character	Description
(Ordinary characters)	Characters other than `. $ ^ { [ ( | ) * + ? \` match themselves.
\a	Matches a bell (alarm) `\u0007`.
\t	Matches a tab `\u0009`.
\r	Matches a carriage return `\u000D`.
\v	Matches a vertical tab `\u000B`.
\f	Matches a form feed `\u000C`.
\n	Matches a new line `\u000A`.
\e	Matches an escape `\u001B`.
\040	Matches an ASCII character as octal (exactly three digits). The character `\040` represents a space.
\x20	Matches an ASCII character using hexadecimal representation (exactly two digits).
\u0020	Matches a Unicode character using hexadecimal representation (exactly four digits).
\	When followed by a character that is not recognized as an escaped character, matches that character. For example, `\*` is the same as `\x2A`.

Character Classes

The following table summarizes character matching syntax.

Character Class	Description
.	Matches any character except \n. When within a character class, the `.` will be treated as a period character.
[aeiou]	Matches any single character included in the specified set of characters.
[^aeiou]	Matches any single character not in the specified set of characters.
[0-9a-fA-F]	Use of a hyphen (`-`) allows specification of contiguous character ranges.
\p{name}	Matches any character in the Unicode general category specified by name (for example, `Ll`, `Nd`, `Z`).
\P{name}	Matches any character not in Unicode general category specified in name.
\w	Matches any word character, which includes letters, digits, and underscores.
\W	Matches any non-word character.
\s	Matches any whitespace character.
\S	Matches any non-whitespace character.
\d	Matches any decimal digit.
\D	Matches any non-digit.
[.\w\s]	Escaped built-in character classes such as `\w` and `\s` may be used in a character class. This example matches any period, word or whitespace character.

Supported Unicode General Categories

Unicode defines the general categories and descriptions listed in the following table. These categories can be used with the `\p` and `\P` character classes described above.

Category	Description
Lu	Letter, Uppercase
Ll	Letter, Lowercase
Lt	Letter, Titlecase
Lm	Letter, Modifier
Lo	Letter, Other
Mn	Mark, Nonspacing
Mc	Mark, Spacing Combining
Me	Mark, Enclosing
Nd	Number, Decimal Digit
Nl	Number, Letter

No	Number, Other
Pc	Punctuation, Connector
Pd	Punctuation, Dash
Ps	Punctuation, Open
Pe	Punctuation, Close
Pi	Punctuation, Initial quote
Pf	Punctuation, Final quote
Po	Punctuation, Other
Sm	Symbol, Math
Sc	Symbol, Currency
Sk	Symbol, Modifier
So	Symbol, Other
Zs	Separator, Space
Zl	Separator, Line
Zp	Separator, Paragraph
Cc	Other, Control
Cf	Other, Format
Cs	Other, Surrogate
Co	Other, Private Use
Cn	Other, Not Assigned

Additional special categories are supported that represent a set of Unicode character categories, as shown in the following table:

Category	Description
C	(All control characters) `Cc`, `Cf`, `Cs`, `Co`, and `Cn`.
L	(All letters) `Lu`, `Ll`, `Lt`, `Lm`, and `Lo`.
M	(All diacritic marks) `Mm`, `Mc`, and `Me`.
N	(All numbers) `Nd`, `Nl`, and `No`.
P	(All punctuation) `Pc`, `Pd`, `Ps`, `Pe`, `Pi`, `Pf` and `Po`.
S	(All symbols) `Sm`, `Sc`, `Sk`, and `So`.
Z	(All separators) `Zs`, `Zl`, and `Zp`.

Quantifiers

Quantifiers add optional quantity data to a regular expression. A quantifier expression applies to the character, group, or character class that immediately precedes it.

The following table describes the metacharacters that affect matching quantity.

Quantifier	Description
*	Specifies zero or more matches; for example, \w* or (abc)*. Same as {0,}.
+	Specifies one or more matches; for example, \w+ or (abc)+. Same as {1,}.
?	Specifies zero or one matches; for example, \w? or (abc)?. Same as {0,1}.
{n}	Specifies exactly n matches; for example, (pizza){2}.
{n,}	Specifies at least n matches; for example, (abc){2,}.
{n,m}	Specifies at least n, but no more than m, matches.

Atomic Zero-Width Assertions

The metacharacters described in the following table do not cause the engine to advance through the string or consume characters. They simply cause a match to succeed or fail depending on the current position in the string. For instance, ^ specifies that the current position is at the beginning of a line or string. Thus, the regular expression ^#region returns only those occurrences of the character string #region that occur at the beginning of a line.

Assertion	Description
^	Specifies that the match must occur at the beginning of the document or the beginning of the line.
$	Specifies that the match must occur at the end of the string, before \n at the end of the string, or at the end of the line.
\A	Specifies that the match must occur at the beginning of the document.
\z	Specifies that the match must occur at the end of the document.
\b	Specifies that the match must occur on a boundary between \w (alphanumeric) and \W (nonalphanumeric) characters.
\B	Specifies that the match must not occur on a \b boundary.

Grouping Constructs

Grouping constructs allow you to capture groups of subexpressions and to increase the efficiency of regular expressions with noncapturing lookahead and lookbehind modifiers.

( )	Captures the matched substring if used in a find/replace operation. In lexing, provides non-captured grouping. Find/replace captures using ( ) are numbered automatically based on the order of the opening parenthesis, starting from one. The first capture, capture element number zero, is the text matched by the whole regular expression pattern.
(?=)	Zero-width positive lookahead assertion. Continues match only if the subexpression matches at this position on the right. For example, _(?=\w) matches an underscore followed by a word character, without matching the word character.
(?!)	Zero-width negative lookahead assertion. Continues match only if the subexpression does not match at this position on the right. For example, \b(?!un)\w+\b matches words that do not begin with un.
(?	Zero-width positive lookbehind assertion. Continues match only if the subexpression

<=)	matches at this position on the left. For example, `(?<=19)99` matches instances of `99` that follow `19`.
(?<!)	Zero-width negative lookbehind assertion. Continues match only if the subexpression does not match at this position on the left.

Substitutions

Substitutions are allowed only within find/replace replacement patterns.

Character escapes and substitutions are the only special constructs recognized in a replacement pattern. For example, the replacement pattern `a*$1b` inserts the string `a*` followed by the substring matched by the first capturing group, if any, followed by the string `b`. The `*` character is not recognized as a metacharacter within a replacement pattern. Similarly, `$` patterns are not recognized within regular expression matching patterns. Within regular expressions, `$` designates the end of the string.

The following table shows how to define named and numbered replacement patterns.

Construct	Description
$1	Substitutes the last substring matched by group number 1 (decimal). The second group is number 2 (`$2`), and so on.
$0	Substitutes a copy of the entire match itself.
$&	Substitutes a copy of the entire match itself.
$$	Substitutes a single `$` literal.

Other Constructs

The following table lists other regular expression constructs.

Construct	Description
" "	Encapsulates a fixed string of characters.
{ }	Provides a call to a lexical macro. The use of a `WordMacro` (which is similar to `\w`) would appear as `{WordMacro}`.
(?#)	Inline comment inserted within a regular expression. The comment terminates at the first closing parenthesis character.
|	Provides an alternation construct that matches any one of the terms separated by the `|` (vertical bar) character. For example, `cat|dog|tiger`. The leftmost successful match wins.