



capevo
business process evolution

www.capevo.com

Regular expressions

Versionshistorik

Version	Dato	Ændringer / Beskrivelse
4.1	17-07-2008	Versionshistorik tilføjet
4.3	11-02-2009	Manual opdateret

Regular Expressions in Capevo XForm

In Capevo XForm Regular Expressions are used to define client side syntax validation of user input under the administrative UI's tab strip "Field validation".

Regular Expressions is a term used to refer to a pattern-matching technology for processing text that has existed in the UNIX world for years and has now been incorporated into the .NET Base Class Library used by the Capevo XForm application. A Regular Expression itself is an string that represents a pattern, encoded using the regular expression language and syntax.

Although there is no standards body governing the regular expression language, Perl 5, by virtue of it's popularity, has set the standard for regular expression syntax. The .NET Framework Regular Expressions library is designed to be compatible with Perl 5 regular expressions, as well as including additional features not found elsewhere.

Syntax

The following table, from the .NET documentation, contains the complete list of metacharacters and their behavior in the context of regular expressions:

Character	Description
\	Marks the next character as a special character, a literal, a backreference, or an octal escape. For example, 'n' matches the character "n". '\n' matches a newline character. The sequence '\\ matches "\" and "\\(" matches "(".
^	Matches the position at the beginning of the input string. If the RegExp object's Multiline property is set, ^ also matches the position following '\n' or '\r'.
\$	Matches the position at the end of the input string. If the RegExp

object's Multiline property is set, \$ also matches the position preceding '\n' or '\r'.

- * Matches the preceding character or subexpression zero or more times. For example, zo* matches "z" and "zoo". * is equivalent to {0,}.
- + Matches the preceding character or subexpression one or more times. For example, 'zo+' matches "zo" and "zoo", but not "z". + is equivalent to {1,}.
- ? Matches the preceding character or subexpression zero or one time. For example, "do(es)?" matches the "do" in "do" or "does". ? is equivalent to {0,1}
- {n} n is a nonnegative integer. Matches exactly n times. For example, 'o{2}' does not match the 'o' in "Bob," but matches the two o's in "food".
- {n,} n is a nonnegative integer. Matches at least n times. For example, 'o{2,}' does not match the "o" in "Bob" and matches all the o's in "fooooo". 'o{1,}' is equivalent to 'o+'. 'o{0,}' is equivalent to 'o*'.
- {n,m} M and n are nonnegative integers, where n <= m. Matches at least n and at most m times. For example, "o{1,3}" matches the first three o's in "fooooo". 'o{0,1}' is equivalent to 'o?'. Note that you cannot put a space between the comma and the numbers.
- ? When this character immediately follows any of the other quantifiers (*, +, ?, {n}, {n,}, {n,m}), the matching pattern is non-greedy. A non-greedy pattern matches as little of the searched string as possible, whereas the default greedy pattern matches as much of the searched string as possible. For example, in the string "oooo", 'o+?' matches a single "o", while 'o+' matches all 'o's.
- .
- (pattern) A subexpression that matches pattern and captures the match. The captured match can be retrieved from the resulting Matches collection using the \$0...\$9 properties. To match parentheses characters (), use '\(' or '\)'.
- (?:pattern) A subexpression that matches pattern but does not capture the match,

that is, it is a non-capturing match that is not stored for possible later use. This is useful for combining parts of a pattern with the "or" character (|). For example, 'industr(?:y|ies)' is a more economical expression than 'industry|industries'.

- (?=pattern) A subexpression that performs a positive lookahead search, which matches the string at any point where a string matching pattern begins. This is a non-capturing match, that is, the match is not captured for possible later use. For example 'Windows (?!=95|98|NT|2000)' matches "Windows" in "Windows 2000" but not "Windows" in "Windows 3.1". Lookaheads do not consume characters, that is, after a match occurs, the search for the next match begins immediately following the last match, not after the characters that comprised the lookahead.
- (?!pattern) A subexpression that performs a negative lookahead search, which matches the search string at any point where a string not matching pattern begins. This is a non-capturing match, that is, the match is not captured for possible later use. For example 'Windows (?!95|98|NT|2000)' matches "Windows" in "Windows 3.1" but does not match "Windows" in "Windows 2000". Lookaheads do not consume characters, that is, after a match occurs, the search for the next match begins immediately following the last match, not after the characters that comprised the lookahead.
- x|y Matches either x or y. For example, 'z|food' matches "z" or "food". '(z|f)ood' matches "zood" or "food".
- [xyz] A character set. Matches any one of the enclosed characters. For example, '[abc]' matches the 'a' in "plain".
- [^xyz] A negative character set. Matches any character not enclosed. For example, '[^abc]' matches the 'p' in "plain".
- [a-z] A range of characters. Matches any character in the specified range. For example, '[a-z]' matches any lowercase alphabetic character in the range 'a' through 'z'.
- [^a-z] A negative range characters. Matches any character not in the specified range. For example, '[^a-z]' matches any character not in the range 'a' through 'z'.
- \b Matches a word boundary, that is, the position between a word and a space. For example, 'er\b' matches the 'er' in "never" but not the 'er' in "verb".

\B	Matches a nonword boundary. 'er\b' matches the 'er' in "verb" but not the 'er' in "never".
\cx	Matches the control character indicated by x. For example, \cM matches a Control-M or carriage return character. The value of x must be in the range of A-Z or a-z. If not, c is assumed to be a literal 'c' character.
\d	Matches a digit character. Equivalent to [0-9].
\D	Matches a nondigit character. Equivalent to [^0-9].
\f	Matches a form-feed character. Equivalent to \x0c and \cL.
\n	Matches a newline character. Equivalent to \x0a and \cJ.
\r	Matches a carriage return character. Equivalent to \x0d and \cM.
\s	Matches any white space character including space, tab, form-feed, and so on. Equivalent to [\f\n\r\t\v].
\S	Matches any non-white space character. Equivalent to [^\f\n\r\t\v].
\t	Matches a tab character. Equivalent to \x09 and \cI.
\v	Matches a vertical tab character. Equivalent to \x0b and \cK.
\w	Matches any word character including underscore. Equivalent to '[A-Za-z0-9_]'.
\W	Matches any nonword character. Equivalent to '[^A-Za-z0-9_]'.
\xn	Matches n, where n is a hexadecimal escape value. Hexadecimal escape values must be exactly two digits long. For example, '\x41' matches "A". '\x041' is equivalent to '\x04' & "1". Allows ASCII codes to be used in regular expressions.
\num	Matches num, where num is a positive integer. A reference back to captured matches. For example, '(.)\1' matches two consecutive identical characters.
\n	Identifies either an octal escape value or a backreference. If \n is preceded by at least n captured subexpressions, n is a backreference.

Otherwise, `n` is an octal escape value if `n` is an octal digit (0-7).

- | | |
|-------------------|--|
| <code>\nm</code> | Identifies either an octal escape value or a backreference. If <code>\nm</code> is preceded by at least <code>nm</code> captured subexpressions, <code>nm</code> is a backreference. If <code>\nm</code> is preceded by at least <code>n</code> captures, <code>n</code> is a backreference followed by literal <code>m</code> . If neither of the preceding conditions exists, <code>\nm</code> matches octal escape value <code>nm</code> when <code>n</code> and <code>m</code> are octal digits (0-7). |
| <code>\nml</code> | Matches octal escape value <code>nml</code> when <code>n</code> is an octal digit (0-3) and <code>m</code> and <code>l</code> are octal digits (0-7). |
| <code>\un</code> | Matches <code>n</code> , where <code>n</code> is a Unicode character expressed as four hexadecimal digits. For example, <code>\u00A9</code> matches the copyright symbol (©). |