POSIX Bracket Expressions

POSIX bracket expressions are a special kind of character classes. POSIX bracket expressions match one character out of a set of characters, just like regular character classes. They use the same syntax with square brackets. A hyphen creates a range, and a caret at the start negates the bracket expression.

One key syntactic difference is that the backslash is NOT a metacharacter in a POSIX bracket expression. So in POSIX, the regular expression `[\d]` matches a `\` or a `d`. To match a `]`, put it as the first character after the opening `[` or the negating `^`. To match a `-`, put it right before the closing `]`. To match a `^`, put it before the final literal `-` or the closing `]`. Put together, `[]\d^-]` matches `]`, `\`, `d`, `^` or `-`.

The main purpose of bracket expressions is that they adapt to the user's or application's locale. A locale is a collection of rules and settings that describe language and cultural conventions, like sort order, date format, etc. The POSIX standard defines these locales.

Generally, only POSIX-compliant regular expression engines have proper and full support for POSIX bracket expressions. Some non-POSIX regex engines support POSIX character classes, but usually don't support collating sequences and character equivalents. Regular expression engines that support Unicode use Unicode properties and scripts to provide functionality similar to POSIX bracket expressions. In Unicode regex engines, shorthand character classes like `\w` normally match all relevant Unicode characters, alleviating the need to use locales.

Character Classes

Don't confuse the POSIX term "character class" with what is normally called a regular expression character class. `[x-z0-9]` is an example of what this tutorial calls a "character class" and what POSIX calls a "bracket expression". `[:digit:]` is a POSIX character class, used inside a bracket expression like `[x-z[:digit:]]`. The POSIX character class names must be written all lowercase.

When used on ASCII strings, these two regular expressions find exactly the same matches: a single character that is either `x`, `y`, `z`, or a digit. When used on strings with non-ASCII characters, the `[:digit:]` class may include digits in other scripts, depending on the locale.

The POSIX standard defines 12 character classes. The table below lists all 12, plus the `[:ascii:]` and `[:word:]` classes that some regex flavors also support. The table also shows equivalent character classes that you can use in ASCII and Unicode regular expressions if the POSIX classes are unavailable. The ASCII equivalents correspond exactly what is defined in the POSIX standard. The Unicode equivalents correspond to what most Unicode regex engines match. The POSIX standard does not define a Unicode locale. Some classes also have Perl-style shorthand equivalents.

Java does not support POSIX bracket expressions, but does support POSIX character classes using the `\p` operator. Though the `\p` syntax is borrowed from the syntax for Unicode properties, the POSIX classes in Java only match ASCII characters as indicated below. The class names are case sensitive. Unlike the POSIX syntax which can only be used inside a bracket expression, Java's `\p` can be used inside and outside bracket expressions.

In Java 8 and prior, it does not matter whether you use the `Is` prefix with the `\p` syntax or not. So in Java 8, `\p{Alnum}` and `\p{IsAlnum}` are identical. In Java 9 and later there is a difference. Without the `Is` prefix, the behavior is exactly the same as in previous versions of Java. The syntax with the `Is` prefix now matches Unicode characters too. For `\p{IsPunct}` this also means that it no longer matches the ASCII characters that are in the Symbol Unicode category.

The JGsoft flavor supports both the POSIX and Java syntax. Originally it matched Unicode characters using either syntax. As of JGsoft V2, it matches only ASCII characters when using the POSIX syntax, and Unicode characters when using the Java syntax.

POSIX	Description	ASCII	Unicode	Shorthand	Java
`[:alnum:]`	Alphanumeric characters	`[a-zA-Z0-9]`	`[\p{L}\p{Nl}\p{Nd}]`		`\p{Alnum}`
`[:alpha:]`	Alphabetic characters	`[a-zA-Z]`	`\p{L}\p{Nl}`		`\p{Alpha}`
`[:ascii:]`	ASCII characters	`[\x00-\x7F]`	`\p{InBasicLatin}`		`\p{ASCII}`
`[:blank:]`	Space and tab	`[ \t]`	`[\p{Zs}\t]`	`\h`	`\p{Blank}`
`[:cntrl:]`	Control characters	`[\x00-\x1F\x7F]`	`\p{Cc}`		`\p{Cntrl}`
`[:digit:]`	Digits	`[0-9]`	`\p{Nd}`	`\d`	`\p{Digit}`
`[:graph:]`	Visible characters (anything except spaces and control characters)	`[\x21-\x7E]`	`[^\p{Z}\p{C}]`		`\p{Graph}`
`[:lower:]`	Lowercase letters	`[a-z]`	`\p{Ll}`	`\l`	`\p{Lower}`
`[:print:]`	Visible characters and spaces (anything except control characters)	`[\x20-\x7E]`	`\P{C}`		`\p{Print}`
`[:punct:]`	Punctuation (and symbols).	`[!"\#$%&'()*+,\-./:;<=>?@\[\\\]^_‘{|}~]`	`\p{P}`		`\p{Punct}`
`[:space:]`	All whitespace characters, including line breaks	`[ \t\r\n\v\f]`	`[\p{Z}\t\r\n\v\f]`	`\s`	`\p{Space}`
`[:upper:]`	Uppercase letters	`[A-Z]`	`\p{Lu}`	`\u`	`\p{Upper}`

[:word:]	Word characters (letters, numbers and underscores)	[A-Za-z0-9_]	[\p{L}\p{Nl} \p{Nd}\p{Pc}]	\w	\p{IsWord}
[:xdigit:]	Hexadecimal digits	[A-Fa-f0-9]	[A-Fa-f0-9]		\p{XDigit}
POSIX	**Description**	**ASCII**	**Unicode**	**Shorthand**	**Java**

Collating Sequences

A POSIX locale can have collating sequences to describe how certain characters or groups of characters should be ordered. In Czech, for example, `ch` as in `chemie` ("chemistry" in Czech) is a digraph. This means it should be treated as if it were one character. It is ordered between `h` and `i` in the Czech alphabet. You can use the collating sequence element `[.ch.]` inside a bracket expression to match `ch` when the Czech locale (cs-CZ) is active. The regex `[[.ch.]]emie` matches `chemie`. Notice the double square brackets. One pair for the bracket expression, and one pair for the collating sequence.

Other than POSIX-compliant engines part of a POSIX-compliant system, none of the regex flavors discussed in this tutorial support collating sequences.

Note that a fully POSIX-compliant regex engine treats `ch` as a single character when the locale is set to Czech. This means that `[^x]emie` also matches `chemie`. `[^x]` matches a single character that is not an `x`, which includes `ch` in the Czech POSIX locale.

In any other regular expression engine, or in a POSIX engine using a locale that does not treat `ch` as a digraph, `[^x]emie` matches the misspelled word `cemie` but not `chemie`, as `[^x]` cannot match the two characters `ch`.

Finally, note that not all regex engines claiming to implement POSIX regular expressions actually have full support for collating sequences. Sometimes, these engines use the regular expression syntax defined by POSIX, but don't have full locale support. You may want to try the above matches to see if the engine you're using does. Tcl's regexp command, for example, supports the syntax for collating sequences. But Tcl only supports the Unicode locale, which does not define any collating sequences. The result is that in Tcl, a collating sequence specifying a single character matches just that character. All other collating sequences result in an error.

Character Equivalents

A POSIX locale can define character equivalents that indicate that certain characters should be considered as identical for sorting. In French, for example, accents are ignored when ordering words. `élève` comes before `être` which comes before `événement`. `é` and `ê` are all the same as `e`, but `l` comes before `t` which comes before `v`. With the locale set to French, a POSIX-compliant regular expression engine matches `e`, `é`, `è` and `ê` when you use the collating sequence `[=e=]` in the bracket expression `[[=e=]]`.

If a character does not have any equivalents, the character equivalence token simply reverts to the character itself. `[[=x=][=z=]]`, for example, is the same as `[xz]` in the French locale.

Like collating sequences, POSIX character equivalents are not available in any regex engine discussed in this tutorial, other than those following the POSIX standard. And those that do may not have the necessary POSIX locale support. Here too Tcl's regexp command supports the syntax for character equivalents. But the Unicode locale, the only one Tcl supports, does not define any character equivalents. This effectively means that `[[=e=]]` and `[e]` are exactly the same in Tcl, and only match `e`, for any character you may try instead of "e".

Page URL: https://www.regular-expressions.info/posixbrackets.html
Page last updated: 15 August 2021
Site last updated: 03 January 2022