Word Boundaries

The metacharacter `\b` is an anchor like the caret and the dollar sign. It matches at a position that is called a “word boundary”. This match is zero-length.

There are three different positions that qualify as word boundaries:

- Before the first character in the string, if the first character is a word character.
- After the last character in the string, if the last character is a word character.
- Between two characters in the string, where one is a word character and the other is not a word character.

Simply put: `\b` allows you to perform a “whole words only” search using a regular expression in the form of `\bword\b`. A “word character” is a character that can be used to form words. All characters that are not “word characters” are “non-word characters”.

Exactly which characters are word characters depends on the regex flavor you’re working with. In most flavors, characters that are matched by the short-hand character class `\w` are the characters that are treated as word characters by word boundaries. Java is an exception. Java supports Unicode for `\b` but not for `\w`.

Most flavors, except the ones discussed below, have only one metacharacter that matches both before a word and after a word. This is because any position between characters can never be both at the start and at the end of a word. Using only one operator makes things easier for you.

Since digits are considered to be word characters, `\b4\b` can be used to match a 4 that is not part of a larger number. This regex does not match `44 sheets of a4`. So saying “`\b` matches before and after an alphanumeric sequence” is more exact than saying “before and after a word”.

`\B` is the negated version of `\b`. `\B` matches at every position where `\b` does not. Effectively, `\B` matches at any position between two word characters as well as at any position between two non-word characters.

Looking Inside The Regex Engine

Let’s see what happens when we apply the regex `\bis\b` to the string `This island is beautiful`. The engine starts with the first token `\b` at the first character `T`. Since this token is zero-length, the position before the character is inspected. `\b` matches here, because the T is a word character and the character before it is the void before the start of the string. The engine continues with the next token: the literal `i`. The engine does not advance to the next character in the string, because the previous regex token was zero-length. `i` does not match `T`, so the engine retries the first token at the next character position.

`\b` cannot match at the position between the `T` and the `h`. It cannot match between the `h` and the `i` either, and neither between the `i` and the `s`.

The next character in the string is a space. `\b` matches here because the space is not a word character, and the preceding character is. Again, the engine continues with the `i` which does not match with the space.

Advancing a character and restarting with the first regex token, `\b` matches between the space and the second `i` in the string. Continuing, the regex engine finds that `i` matches `i` and `s` matches `s`. Now, the engine tries to match the second `\b` at the position before the `l`. This fails because this position is between two word characters. The engine reverts to the start of the regex and advances one character to the `s` in `island`. Again, the `\b` fails to match and continues to do so until the second space is reached. It matches there, but matching the `i` fails.

But `\b` matches at the position before the third `i` in the string. The engine continues, and finds that `i` matches `i` and `s` matches `s`. The last token in the regex, `\b`, also matches at the position before the third space in the string because the space is not a word character, and the character before it is.

The engine has successfully matched the word `is` in our string, skipping the two earlier occurrences of the characters i and s. If we had used the regular expression `is`, it would have matched the `is` in `This`.

Tcl Word Boundaries

Word boundaries, as described above, are supported by most regular expression flavors. Notable exceptions are the POSIX and XML Schema flavors, which don't support word boundaries at all. Tcl uses a different syntax.

In Tcl, `\b` matches a backspace character, just like `\x08` in most regex flavors (including Tcl's). `\B` matches a single backslash character in Tcl, just like `\\` in all other regex flavors (and Tcl too).

Tcl uses the letter "y" instead of the letter "b" to match word boundaries. `\y` matches at any word boundary position, while `\Y` matches at any position that is not a word boundary. These Tcl regex tokens match exactly the same as `\b` and `\B` in Perl-style regex flavors. They don't discriminate between the start and the end of a word.

Tcl has two more word boundary tokens that do discriminate between the start and end of a word. `\m` matches only at the start of a word. That is, it matches at any position that has a non-word character to the left of it, and a word character to the right of it. It also matches at the start of the string if the first character in the string is a word character. `\M` matches only at the end of a word. It matches at any position that has a word character to the left of it, and a non-word character to the right of it. It also matches at the end of the string if the last character in the string is a word character.

The only regex engine that supports Tcl-style word boundaries (besides Tcl itself) is the JGsoft engine. In PowerGREP and EditPad Pro, `\b` and `\B` are Perl-style word boundaries, while `\y`, `\Y`, `\m` and `\M` are Tcl-style word boundaries.

In most situations, the lack of `\m` and `\M` tokens is not a problem. `\yword\y` finds "whole words only" occurrences of "word" just like `\mword\M` would. `\Mword\m` could never match anywhere, since `\M` never matches at a position followed by a word character, and `\m` never at a position preceded by one. If your regular expression needs to match characters before or after `\y`, you can easily specify in the regex whether these characters should be word characters or non-word characters. If you want to match any word, `\y\w+\y` gives the same result as `\m.+\M`. Using `\w` instead of the dot automatically restricts the first `\y` to the start of a word, and the second `\y` to the end of a word. Note that `\y.+\y` would not work. This regex matches each word, and also each sequence of non-word characters between the words in your subject string. That said, if your flavor supports `\m` and `\M`, the regex engine could apply `\m\w+\M` slightly faster than `\y\w+\y`, depending on its internal optimizations.

If your regex flavor supports lookahead and lookbehind, you can use `(?<!\w)(?=\w)` to emulate Tcl's `\m` and `(?<=\w)(?!\w)` to emulate `\M`. Though quite a bit more verbose, these lookaround constructs match exactly the same as Tcl's word boundaries.

If your flavor has lookahead but not lookbehind, and also has Perl-style word boundaries, you can use `\b(?=\w)` to emulate Tcl's `\m` and `\b(?!\w)` to emulate `\M`. `\b` matches at the start or end of a word, and the lookahead checks if the next character is part of a word or not. If it is we're at the start of a word. Otherwise, we're at the end of a word.

GNU Word Boundaries

The GNU extensions to POSIX regular expressions add support for the `\b` and `\B` word boundaries, as described above. GNU also uses its own syntax for start-of-word and end-of-word boundaries. `\<` matches at the start of a

word, like Tcl's `\m`. `\>` matches at the end of a word, like Tcl's `\M`.

Boost also treats `\<` and `\>` as word boundaries when using the ECMAScript, extended, egrep, or awk grammar.

POSIX Word Boundaries

The POSIX standard defines `[[:<:]]` as a start-of-word boundary, and `[[:>:]]` as an end-of-word boundary. Though the syntax is borrowed from POSIX bracket expressions, these tokens are word boundaries that have nothing to do with and cannot be used inside character classes. Tcl and GNU also support POSIX word boundaries. PCRE supports POSIX word boundaries starting with version 8.34. Boost supports them in all its grammars.

Make a Donation

Page URL: https://www.regular-expressions.info/wordboundaries.html
Page last updated: 22 November 2019
Site last updated: 08 April 2020