Rizone.soft

Examples

In this topic we will discuss using the Notepad3 regular expression feature efficiently. If you are a regular expression expert, you are welcome to skip to the regular expression reference document (Oniguruma). Or read the Oniguruma Introduction for your technical jargon fix.

Fundamental Regular Expression Examples

1. Plaintext

`Sample` matches `sample`, as well as the fourth word in `this is a sample line of text`.

2. Anchors

- ^ matches the beginning of a line. For example, `^the` will match the first `the` in `the earth revolves around the sun`, but not the second one.
- $ matches the end of a line (EOL). For example, `business$` will match the second `business` in `business is business`, but not the first one.
- `\b` matches either boundary of a word. That is, where on its left there is an alphabetic character, a decimal digit or an underscore and on its right there is no such character, or vice versa. For example, `\bbackward\b` will match the integral word `backward` but not the part in `backwards` in `as an adverb 'backward' is identical to 'backwards'`.

3. Wildcards

- `.` matches a single character that is not an end of line. Hence `a.` will match the first two characters in `apple` and the third to fourth ones in `orange`.
- `\d` matches a decimal digit. That is, from `0` to `9`.
- `\D` matches the complement of `\d`. That is, anything that is not a decimal digit.
- `\s` matches a whitespace character. Usually this matches a space, a Tab or an EOL.
- `\S` matches the complement of `\S`. That is, anything that is not a whitespace character.
- `\w` matches a character that is an alphabetic character, a decimal digit, or an underscore. This matches characters that form identifiers in a number of programming languages including C and Java.
- `\W` matches the complement of `\w`. That is, anything that is not an alphabetic character, a decimal digit, or an underscore.
- `[...]` where `...` is a series of characters (the first one shall not be ^) matches any of (but

not none of) the characters within the brackets. For example, `[aeiou]` will match any vowel characters in `the quick brown fox jumps over the lazy dog.` A hyphen can be used to designate a closed interval in which all characters will match. For example, `[a-d0-4]` matches any of `abcd01234`.
- `[^...]` matches the complement of `[...]`. That is, anything other than the series of characters specified.

4. Disjunction

`|` is a binary operator specifying that exactly one of its side shall match. For example, `yes|no` matches either `yes` or `no`. This operator has a lower priority than character junctions, ergo, the example above is not meant to match `yeso` or `yeno`.

5. Quantifiers

- `{N}` means to match the operand preceding it repeatedly, up to exactly N times. For example, `ap{2}le` says `p` is to be matches twice so it will match `apple`, and `b.{2}a` will match the first four characters in *banana*.
- `{N,}` means to match the operand preceding it repeatedly, at least N times. The match is determinated to be the longest string that contains at least N consecutive copies of the operand qualified. For example, `b.{2,}a` will match the entire word *banana*, despite the sub-string match *bana*.
- `{N,}?` is similar to `{N,}?`, but matches the shortest string. For example, `b.{2,}?a` will match the sub-string *bana* in *banana*.
- `{N,M}` means to match the operand preceding it repeatedly, at least N times but at most M times. The match is determinated to be the longest string of such quantity. For example, `b.{4,6}a` will match the first eight characters in *bananananana*.
- `{N,M}?` is similar to `{N,M}?`, but matches the shortest string. For example, `b.{4,6}?a` will match the first six characters in *bananananana*.
- `?` is the same as `{0,1}`. It effectively means 'this operand is optional'.
- `+` is the same as `{1,}`. It effectively means 'this operand is required, but there can be more than one, and I want as many as possible'.
- `+?` is the same as `{1,}?`. It effectively means 'this operand is required, but there can be more than one, and I want as few as possible'.
- `*` is the same as `{0,}`. It effectively means 'this operand is optional, but there can be more than one, and I want as many as possible'.
- `*?` is the same as `{0,}?`. It effectively means 'this operand is optional, but there can be more than one, and I want as few as possible'.

6. Escape Sequences

- `\t` matches a Tab.
- `\r` matches a carriage return (CR).
- `\n` matches a line feed (LF).

- `\\` matches a plain forwardslash.

7. Groups

`(...)` where `...` is an arbitrary sub-expression makes that sub-expression an integral operand. For example, `ap{2}le` will match `apple` because the `{2}` quantifier applies to the single character `p` preceding it, and `ba(na){2}` will match `banana` because this time it is the group consisting of `na` that is to be matched. When a group matches some characters, the group is said to capture those characters.

When replacing by regular expression, the special placeholder `\N` where N is a positive integer designates the characters captured by the Nth group. For example, replacing `a(.)(z?)(a)` with `\3\2\1` in `banana` results in `banna`. Because the sub-string `ana` is a match (try removing parentheses to find it out), and the first group captures `n`, the second group captures no characters (because `z` is not a match and has to be matched zero times), the third group captures a literal `a`, then the sub-string is replaced with `\3\2\1` which is `an`.

Groups are numbered in the order that their opening brackets appear. Hence, as long as `((a)(b))?c` matches `abc`, the first group is made up of the outermost brackets and captures `ab`, the second group captures `a` and the third group captures `b`.

`(?:...)` is an uncapturing group. It is not assigned a number and has no effect on subscripting of capturing groups. It is otherwise identical to a capturing group. For example, as long as `(?:(a)(b))?c` matches `abc`, the first group captures `a` and the second group captures `b`. There is not a third group.

Compound Regular Expression Examples

Word boundaries are omitted for simplification. If you would like to use these patterns in public projects, you probably need to qualify them with proper word boundaries.

1. To match either boolean constant, that is, `true` or `false`: `(true|false)`
2. To match any fixed-width integer type in, `that is`, `int8_t`, `int16_t`, `int32_t`, `int64_t`, as well as their unsigned counterparts: `u?int(8|16|32|64)_t`
3. To match an identifier or keyword in C or Java, that is, anything begins with an ASCII letter or an underscore, optionally followed by a series of others, each of which will match `\w`: `[A-Za-z_]\w*`
4. To match a preprocesser directive in C, that is, a line initiated by a `#` optionally preceded by whitespaces and terminated by an unescaped EOL: `^[ \t]*#(.*?[^\\])?$`
5. To match a string literal in C or Java, that is, a series of unescaped or escaped characters wrapped by a pair of double qoutes: `"([^\\]|(\\.))*?"`
6. To match a DOS path: `[A-Za-z]:(\\[^\\/:*?"<>|]*)*`

Advanced Regular Expression Examples

1. Assertions

- `(?=...)` is called a positive lookahead assertion. It matches zero characters by requiring the string following it to match the pattern specified. For example, `p(?=p)` will match the second `p` in `pineapple`, but none of the others.
- `(?!...)` is called a negative lookahead assertion. It matches zero characters by forbidding the string following it to match the pattern specified. For example, `p(?!p)` will match the first and third `p` in `pineapple`, but not the second.
- `(?<=...)` is called a positive lookbehind assertion. It matches zero characters by requiring the string preceding it to match the pattern specified. For example, `(?<=a)p` will match the second `p` in `pineapple`, but none of the others.
- `(?<!...)` is called a negative lookbehind assertion. It matches zero characters by forbidding the string preceding it to match the pattern specified. For example, `(?<!a)p` will match the first and third `p` in `pineapple`, but not the second.

2. Backreferences

`\N` where `N` is a positive integer is called a backreference. A backreference requires a part of the string be identical to another part of itself (which is by then a literal string rather than a regular expression) without knowing what the part is first. For example, `(.{2})\1` matches two arbitrary characters first then another copy of them, hence will match `aabb`, `abab` or even `$*$*`, but will not match `abac`.

Conclusion

LH_Mouse, prepared theses regular expression examples for us. Thanks! 🙂

Our Software

Rizonesoft Office

Notepad3

Firemin

Complete Internet Repair

BIOS Beep Codes Viewer

DVD Drive Repair

Stuck Pixel Repair

Carbon CD

USB Repair