VS Code RegExp

Reference 1

There are many flavors of RegExp- VS Code uses Rust Rust RegExp specifically Here

Microsoft VS RegExp
exe64=x64\(.*?)(

Substitutions in Reg Exp

File

GOOD- Quick Reference
PDF Version

> **Note- The link- `Microsoft VS RegExp` suggests an older easier form of RegExp thats worth checking out if this isnt working**

- Wildcards
  - `.` - 1 match a single char (any, except new line `\n`)
  - `*` - 0+ - zero or more occurances.
  - `+` - 1+ -one or more occurances. Requires its occurance.
  - `?` - match 0 or more, but as few characters as possible (?).
    - `.+` = one or more of any char **As above- this is a (non-null) Wildcard**
    - `.*` = zero or more of any char. **This is a wildcard functionally**
    - `*?`,`+?`, `??`, `{#...}?` Lazy modifiers of the above.
  - {#} - specify the number of times something *preceeding* should occur. Goes after. `[0-9]{3} = 142 or 999`
    - {#,#} - specify a range of occurances. e.g. [0-9]{2,4} match 10 - 9999
    - {#,} - match at least # times
- Anchors
  - `^` - anchor to front
  - `\r?$` -anchor to rear. Old character was just `$`
- Parens and brackets
  - [...] - give characters to match.
    - [], [c] - match a single char
    - [a-f], [0-9] - match a range
    - [pre], [.)-] - match a single character in a set. [abc] = a, b, c not d, matches a in aa.
  - (...) - specify a string to search for
    - (...) -> `\1` define a pattern then Reference it later (in the same pattern) with \1 (\2, \3) etc. `({3}[x])z\1 = xxxzxxx`
    - (...) -> `$1` - in replacements, VS 2011+ replaced \1 with $1.

- Logic
  - `?!` = NOT , `real(?!ity)` matched real and really but not reality.
  - [^...] - NOT in set - as above for brackets, but invalidates `be[^abc]` does not match bear but would match bee.
  - | = OR - Match either the sequence before or after. `([0-9]|[a-f]|[^g-z])` - to find hex `#([0-9]|[a-f]){6}` find hex colors
- Types
  - `(\b\S+\s?){1,}` USEFUL- find words - start at word boundary -repeat to go to non-space- find 0 or 1 space- then all of it {1,}- one or more words.
  - `\p{...}` match a type
  - `\b` -specifies a word boundary- usually a space `\bin` matches inside but not bin.
  - `\r?\n` - line break carriage return. This is system independent - 0 or more \r (returns), then \n. Or at least Unix/Win
  - `(?([^\r\n])\s)` Whitespace
  - `\d` digit
  - \u#### -unicode (specific char)
  - `((\".+?\")|('.+?'))` = quoted string
  - `\b0[xX]([0-9a-fA-F])\b` matches hex- Matches "0xc67f" but not "0xc67fc67f".
  - `\b[0-9]*\.*[0-9]+\b` int and floats.
- Characters
  - \ -escape character - e.g. `\.` is a literal `.` not anychar.
  - \b - word boundary
  - \t tab character (\u0009)
  - \r carriage return \u000D
  - \v vert tab
  - \f form feed
  - `\n` new line (\u000A)
  - \e escape \u001b
  - \### | \## matches an **ASCII Character** in Octal | Hex
  - `\cC` matches CtrlC
  - `\octal` 2-3 digit octal character code
  - `\x` hex 2-digit hex character code
  - `\u` hex 4-digit hex character code
  - \b Backspace \u0008 NOTE-This is redundant with word boundry, not sure how it'd work.
  - \e Escape \u001B
- Classes- generally UPPER CASE is NOT a category.
  - \S -any non-White-Space
  - \s - any whitespace character
  - \D - Matches a nondigit char - equiv to `\P{Nd}`
  - \d - a decimal digit (e.g number 0-9)
    - \d{1,5}- Match from one to five decimal digits.

- \w Alphanumeric ('Word character')
- \W non word character
- \p{**} - is in a category
- \P{**} is NOT in a category
- Categories (more specific classes) (`\p{...}`)
    - Ll -Letter, Lowercase
    - Lu -Letter, Uppercase
    - Lt -Letter, Titlecase
    - Lo -Letter, Other
    - Lm -Letter, Modifier
    - Mn -Mark, Nonspacing
    - Nd -Number, Decimal Digit
    - Pc -Punctuation, Connector. This category includes ten characters, the most commonly used of which is the LOWLINE character (_), u+005F.
    - L -Any letter
    - Mn - Mark, Nonspacing
    - Mc - Mark, Spacing Combining
    - Me - Mark, Enclosing
    - M - All diacritic marks. This includes the Mn, Mc, and Me categories.
    - Nd - Number, Decimal Digit
    - Nl - Number, Letter
    - No - Number, Other
    - N - All numbers. This includes the Nd, Nl, and No categories.
    - Pc - Punctuation, Connector
    - Pd - Punctuation, Dash
    - Ps - Punctuation, Open
    - Pe - Punctuation, Close
    - Pi - Punctuation, Initial quote (may behave like Ps or Pe depending on usage)
    - Pf - Punctuation, Final quote (may behave like Ps or Pe depending on usage)
    - Po - Punctuation, Other
    - P - All punctuation characters. This includes the Pc, Pd, Ps, Pe, Pi, Pf, and Po categories.
    - Sm - Symbol, Math
    - Sc - Symbol, Currency
    - Sk - Symbol, Modifier
    - So - Symbol, Other
    - S - All symbols. This includes the Sm, Sc, Sk, and So categories.
    - Zs - Separator, Space
    - Zl - Separator, Line
    - Zp - Separator, Paragraph
    - Z - All separator characters. This includes the Zs, Zl, and Zp categories.
    - Cc - Other, Control
    - Cf - Other, Format
    - Cs - Other, Surrogate

        - Co - Other, Private Use
        - Cn - Other, Not Assigned (no characters have this property)
        - C - All control characters. This includes the Cc, Cf, Cs, Co, and Cn categories.

- Anchors
    - ^ match must occur at the beginning of the string;
        - multiline mode, it must occur at the beginning of the line.
    - $ the match must occur at the end of the string or before \n at the end of the string;
        - in multiline mode, it must occur at the end of the line or before \n at the end of the line.
    - \A The match must occur at the beginning of the string only (no multiline support).
    - \Z The match must occur at the end of the string, or before \n at the end of the string.
    - \z The match must occur at the end of the string only.
    - \G The match must start at the position where the previous match ended.
    - \b The match must occur on a **word boundary.**
    - \B The match must not occur on a word boundary.
      ^()(\S+?)([ ]{2,}\t*[ ]{2,})(\b.*)$
      ^()(\S+?)(([ ]{3,}|)\t([ ]{3,}|))(\b.*)$

Group Referencing (Reusing a previously used group)

- (exp) Indexed group
- (?<name>exp) - Named group
- (?<name1-name2>exp) - Balancing group
- (?:exp) - Noncapturing group
- (?=exp) - Zero-width positive lookahead
- (?!exp) - Zero-width negative lookahead
- (?<=exp) - Zero-width positive lookbehind
- (?<!exp) - Zero-width negative lookbehind
- (?>exp) - Non-backtracking (greedy)

Use To substitute

- $n - Substring matched by group number n
- ${ - name} Substring matched by group
- $$ - Literal $ character
- $& - Copy of whole match
- $` ( `DollarSign + Backtick` - Text before the match
- $' - Text after the match
- $+ - Last captured group
- $ - _ Entire input string

Conditionals

- a |b - Either a or b
- (?(exp) yes | no) -> yes if exp is matched and =no if exp isn't matched
- (?(name) yes | no) -> yes if name is matched and = no if name isn't matched

Comments

- `(?# comment)` Add inline comment
- # Add x-mode comment
-

backreferencing

- `\n` - Indexed group
- `\k<name>` - Named group
- Options
- `(?imnsximnsx)` Set or disable the specified options
- `(?imnsximnsx:exp)` Set or disable the specified options within the expression
    - `i` - Case-insensitive
    - `m` - Multiline mode
    - `n` - Explicit (named)
    - `s` - Single-line mode
    - `x` - Ignore white space

Greedy Lazy Matches
0 or more times | * Greedy *? Lazy
1 or more times | + Greedy +? Lacy
? | ?? Lazy 0 or 1 time |
{n} {n}? Exactly n times
{n,} {n,}? At least n times
{n,m} {n,m}? From n to m times

Greedy	Lazy	Matches
*	*?	0 or more times
+	+?	1 or more times
?	??	0 or 1 time
{n}	{n}?	Exactly n times
{n,}	{n,}?	At least n times
{n,m}	{n,m}?	From n to m times

Note- {n,m} and similar are recorded weird because they disappear from table when entered verbatim, it must be the format of the table code.

The regular expression pattern \b(?<n2>\d{2}-)?``(?(n2)``\d{7}|\d{3}-\d{2}-\d{4})\b is interpreted as shown in the following table.

Pattern Description

- \b Start at a word boundary.

- `(?<n2>\d{2}-)?` Match zero or one occurrence of two digits followed by a hyphen. Name this capturing group n2.
- `(?(n2)` Test whether n2 was matched in the input string.
- `)\d{7}` If n2 was matched, match seven decimal digits.
- `|\d{3}-\d{2}-\d{4}` If n2 was not matched, match three decimal digits, a hyphen, two decimal digits, another hyphen, and four decimal digits.
- `\b` Match a word boundary.

Examples

Verifying Email:

```
public static void Main()
  {
     Stopwatch sw;
     string[] addresses = { "AAAAAAAAAA@contoso.com",
                            "AAAAAAAAAAaaaaaaaaaa!@contoso.com" };
     // The following regular expression should not actually be used to
     // validate an email address.
     string pattern = @"^[0-9A-Z]([-.\w]*[0-9A-Z])*$";
     string input;

     foreach (var address in addresses) {
        string mailBox = address.Substring(0, address.IndexOf("@"));
        int index = 0;
        for (int ctr = mailBox.Length - 1; ctr >= 0; ctr--) {
           index++;

           input = mailBox.Substring(ctr, index);
           sw = Stopwatch.StartNew();
           Match m = Regex.Match(input, pattern, RegexOptions.IgnoreCase);
           sw.Stop();
           if (m.Success)
              Console.WriteLine("{0,2}. Matched '{1,25}' in {2}",
                                index, m.Value, sw.Elapsed);
           else
              Console.WriteLine("{0,2}. Failed  '{1,25}' in {2}",
                                index, input, sw.Elapsed);
        }
        Console.WriteLine();
     }
  }
```

Example with open and close paren- might need this someday.

https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference

Better Source

(?< name1 - name2 > subexpression)

Defines a balancing group definition. For more information, see the "Balancing Group Definition" section in Grouping Constructs.

```
(((?'Open'\()[^\(\)]*)+((?'Close-Open'\))[^\(\)]*)+)*(?(Open)(?!))$
```

"((1-3)*(3-1))*" *in* "3+2^*((1-3)*(3-1))"

Parsing a string - Serial No Example in C#

My Examples

([1]*?) Standard

Finding a table

```
(^\|)(.*?)(\|)(.*?)(\|$)
```

For parsing tables:

Intention is to find ^| ... | ... |$

```
`(^[\[\_.\:\+\*-\w]*)([\t]{2})(.*?$)` -
```

convert input from R into (- xxxx - xxxx)

Convert var: text -

```
(^[.$\w\s\(\;\)]*)([:])
```

Bold betweem the parentheses (xxx) -> (xxx)

```
([(])(.*?)([)])
**$1$2$3**
```

A series of entries like this:

It would not continue the match onto the following line for some reason.
encoding:
The name of an encoding, default "native.enc". See connections.
encoding:
The name of an encoding, default "native.enc". See connections.

```
(^[a-z.]+?)(:)([\n^.*])
```

and replace with -

```
    `$1`
```

Or also, though I am not sure this wasnt something else:

```
  (^[\w]*?)([\n])(^.*?$)
```

replace with

```
    $1 $3
```

Matches patterns with a single word, and then the next line- like this

> **Quote**
>
> **timeout**
> timeout in seconds, ignored if 0. This is a limit for the elapsed time running command in a separate process. Fractions of seconds are ignored.
>
> **show.output.on.console**
> logical (not NA), indicates whether to capture the output of the command and show it on the R console (not used by Rterm, which shows the output in the terminal unless wait is false).
>
> **invisible**
> logical (not NA), indicates whether a command window should be visible on the screen.**

Which returns text like this:

- timeout - timeout in seconds, ignored if 0. This is a limit for the elapsed time running command in a separate process. Fractions of seconds are ignored.
- show.output.on.console - logical (not NA), indicates whether to capture the output of the command and show it on the R console (not used by Rterm, which shows the output in the terminal unless wait is false).
- minimized - logical (not NA), indicates whether a command window should be displayed initially as a minimized window.

Use:

```
(^[\b\w., ]+?)([\n]|[\t\n])(^.+?$)
(^[\b\w., ]+?)([\n])(^.+?$)
```

and replace with

```
        - _$1_ - $3
        - `$1` - $3
        - **$1** - $3
```

Replace

Used for grabbing flags and commenting them

-d --diff Compare two files with each other.
-a --add Add folder(s) to the last active window.
-g --goto file:line[:character] Open a file at the path on the specified
line and character position.
-n --new-window Force to open a new window.
-r --reuse-window Force to open a file or folder in an

Use:

```
(-{1,2}[\w\S]+)(( \<.{3,30}\>)|)
```

replace with

```
$1$3
```

Which returns text like this:

`-d --diff <file> <file>` Compare two files with each other.
`-a --add <folder>` Add folder(s) to the last active window.
`-g --goto <file:line[:character]>` Open a file at the path on the specified line and character position.
`-n --new-window` Force to open a new window.
`-r --reuse-window` Force to open a file or folder in an

```
(`)([ ]{3,})(\S) -> $1 | $3
```

To match... TEMPLATE

Input from

example

Which returns text like this:

result

Use:

```
<code>
```

replace with

```
<code>
```

`Replace`

`Source`

example
example

example
example

Use:

```
<code>
<code>
<code>
```

replace with

```
<code>
```

Which returns text like this:

- example
- example
- example
- example

Replace

Source

example
example
example
example

Use:

```
<code>
<code>
<code>
```

replace with

```
<code>
```

Which returns text like this:

- example
- example
- example
- example

Replace

Source

example
example
example
example

Use:

```
<code>
<code>
<code>
```

replace with

```
<code>
```

Which returns text like this:

- example
- example

- example
- example

Replace

Source

example
example
example
example

Use:

```
<code>
<code>
<code>
```

replace with

```
<code>
```

Which returns text like this:

- example
- example
- example
- example

Replace 1 or two # with ># (##, >##) to remove comments in code blocks

```
(\#)(\#|)
(\#)(\#)([]|[ ])
```

replace with : version 2 removes the space after.

Replace literal '*tabular*' data into an unordered list with code indicators around the first col. This is for items that are generally:

```
item    definition
item2   definition2
item3   definition3
```

yields:

```
- `item` - definition
- `item2` - definition2
- `item3` - definition3
```

Rex Exp - more general and dynamic moving down this list. Right now the last one performs the best.

```
1. (\S*?)(\t)(.*) -most specific
2. ^()(\S*?)(\t)(.*)$ -add empty group to make replacements match.
3. ^([\s]*?)([\S]*?)(\t| )(.*)$
4. ^([\s\t ]*?)(\b[\S]+?)(\t|[ ]*)(\b.*)$
5. (\S*?)(\t|[ ]*)(\b\S.*)$
6. ^()(\S+?)([ ]*\t+[ ]*)(\b.*)$
7. ^()(\S+?)(([ ]{3,}|)\t*([ ]{3,}|))(\b.*)$
```

Replace 1 or two # with ># (##, >##) to remove comments in code blocks

```
    (\#)(\#|)
    (\#)(\#)([]|[ ])
```

replace with : version 2 removes the space after.

```
    >$1$2
```

1. a-zA-Z [1] ↩