MVC Documentation

Metacharacter is used to match a specific type of character

\d = any digit from 0 to 0  
\w = any character from a to z, A to Z and 0 to 9  
\s any whitespace character (space, tab, etc.)

^ = the start of the string

$ = the end of the string

Regular expression:  
/^abc/ - string: abc = match = yes (because it starts with abc and not ends with abc)  
/^abc/ - string: 123abc = match = no (because it ends with abc ant not starts with abc)

/abc$/ - string: 123abc = match = yes (because it ends with abc and not starts with abc)

/abc$/ - string: abc123 = match = no (because it doesn’t end with abc but starts with abc)

Repetition:  
\* = repeated zero or more times  
+= repeated one or more times  
regular expression:

/a\*bc/ - string = abc = match = yes (because the preceding letter (a) is repeated 0 or more times)  
/a\*bc/ - string = bc = match = yes (because the preceding letter(a) is repeated zero times)  
  
/a+bc/ - string = bc = match = no (because the + needs to specify 1 or more a’s)  
/a+bc/ - string = abc = match = yes (because the letter (a) occurs 1 time )  
/a+bc – string = aaaabc = match yes(because the letter (a) occurs more than once)

The dot .  
. = match any single character: letter, number, whitespace etc.  
/ab.de/ - string = abcde = match = yes (because there is a character in the string (c))

/ab.de/ - string = ab4de = match = yes (because there is a letter in the string (4))  
/ab.de/ - string = abde = match = no (because the isn’t a character between b and d)

|  |  |
| --- | --- |
| Expression | Meaning |
| [abc] | A single character of :a, b or c |
| [^abc] | Any single character except a,b, or c |
| [a-z] | Any single character in the range a-z |
| [a-zA-Z] | Any single character in the range a-z or A-Z |
| ^ | Start of line |
| $ | End of line |
| \A | Start of string |
| \z | End of string |
| . | Any single character |
| \s | Any whitespace character |
| \S | Any non-whitespace character |
| \d | Any digit |
| \D | Any non-digit |
| \w | Any word character (letter, number, underscore) |
| \W | Any non-word character |
| \b | Any word boundary |
| (…) | Capture everything enclosed |
| (a|b) | A or b |
| ? | Zero or one |
| \* | Zero or more |
| + | One or more |
| a{3} | Exactly 3 of a |
| a{3,} | 3 or more of a |
| a{3,6} | Between 3 and 6 of a |
| i | case insensitive |
| m | treat als multiline string |
| s | dot matches newline |
| x | ignore whitespace in regex |
| A | matches only at the start of string |
| D | matches only at the end of string |
| U | non-greedy matching by default |

Combining regular expressions  
/abc.\*/ - string = abcdef = match = yes (because the dot represents ‘def’ and it occurs zero or more times )   
/abc.\*/ - string = abc = match = yes (because abc occurs more than zero or one times)

Escaping  
\ = match metacharacters by escaping them / what comes after \ means that its looking for it  
/abc./ - string = abcd = match = yes (because the d replaces the dot)  
/abc\./ - string = abcd = match = no (because there is no dot specified in the string)  
/abc\. / - string = abc. = match = yes (because it’s looking for the dot and it’s in the string)

i = means to make it CASE INSENSITIVE / so it doesn’t care if its case sensitive or not  
 /abc/ - string = Abc = match = no (because the lowercase a is not the same as the uppercase A)  
/abc/i - string = Abc = match = yes (because it doesn’t care if its upper case or lower case)

[] = match **one** of any of the characters in brackets .   
example: [abc] will match a or b or c and nothing else  
/a[123]b/ - string = a2b = match = yes (because 2 is in the regular expression)  
/a[123]b/ - string = a4b = match = no (because 4 is not in regular expression)  
/a[123]+b/ - string = a321322b = match = yes (because the characters repeat more than once)

/a[1-5]b/ - string = a2b = match = yes (because the 2 in in the range of string matchthe regular expression 1-5)   
/a[1-5]b/ - string = a6b = match = no (because the 6 in the string is outside the range of 1-5 in the regular expression)

/[a-z0-9 ] + / - string = hello there = match = yes (because the string is between a-z and because of the + , repeated characters are permitted)

[^ ] = every character EXCEPT for the characters specified (inside the brackets) also works on ranges.  
/a[^123]b/ - string = a2b = match = no (because the string contains a number but the regular expression doesn’t allow the numbers from 1 to 3)

/a[^123]b/ - string = a4b = match = yes(because the string (4) is not in the specified character set in the regular expression)

/[^ a-z]+/ - string = hello = match = no (because the string (hello) is in between of specified character set a to z in the regular expression)  
/[^ a-z]+/ - string = HELLO = match = yes (because the regular expression only specifies lower case letters), if there was an I after the regular expression, it won’t work ; because it makes it doesn’t care about the case sensitivity.

Extracting parts

Example of using preg\_match:

preg\_match (“/[a-z]+/”, “abcd”) => 1 (match)  
preg\_match (“/[a-z]+/”, “1234”) => 0 (no match)

preg\_match (“/[a-z]+/”, “abcd”, $matches)  
$matches => [“abcd”]  
$matches will contain the results of the match, so in this case, It will store the “abc” in the $matches variable.

Capture groups in regular expressions

() = capture the regular expression inside the parentheses   
preg\_match($reg\_exp, $string, $matches);  
$reg\_exp = /a[123]+b/;  
$string = a222b;  
$matches = [0 => “a222b”]

To capture only a specified outcome:

$reg\_exp /a([123]+)b/   
$string = a222b;  
$matches = [1 => “222”]

$reg\_exp = /([a-zA-z]+) (\d+)/ ;  
$string = Jan 1992;  
$matches = [  
0 => “Jan 1992”,  
1 => “Jan”,  
2 => “1992”  
]  
You can also give names to capture groups

(?<name>regex) = give the capture group a name  
  
So with capture groups the key becomes the month and the year. (in this case)

$reg\_exp = /(?<month>[a-zA-z]+) (?<year>\d+)/ ;  
$string = Jan 1992;  
$matches = [  
0 => “Jan 1992”,  
month => “Jan”,  
year => “1992”

str\_replace()

str\_replace(‘blue’, ‘red’, “Roses are blue”).   
Original: Roses are red.

strops()

$pos = strops(‘Dave likes chocolate’, ‘choc’);

If($pos === false){  
$found\_choc = false;  
} (in this case true)

Preg\_replace

$result = preg\_replace($reg\_exp, $replacement, $string)  
Searches $string for matches to $reg\_exp and replaces them with $replacement

|  |  |  |  |
| --- | --- | --- | --- |
| $reg\_exp | $replacement | $string | result |
| /abc/ | def | abc | def |
| /\d+/ | -- | abc123def | abc--def |
| /\s+/ | , | A b c d | A,b,c,d |

|  |  |  |  |
| --- | --- | --- | --- |
| $reg\_exp | $replacement | original $string | result modified |
| /ab(c)/ | \1de | abc | cde |
| /(\w+) and (\w+)/ | \1 or \2 | Jack and Jill | Jack or Jill |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Backreferences to capture groups

Refer to the text in a capture group using \1, \2 and so on:

$result = preg\_replace($reg\_exp, $replacement, $string)

Turning the route into a regular expression  
 {controller}/{action}

preg\_replace(‘/\//’, ‘\\/’, $route); \ = escape string

translates to:

{controller}\/ {action}

{controller}/{action}

$route = ‘/^’ . $route . ‘$/’;

/^(?P<controller>[a-z-]+)\/(?P<action>[a-z-]+)$/