Regular Expression

About

A list of useful regular expression examples.Regular Expressions are used throughout FreeSWITCH. See Dialplan_XML for complete examples of the regular expressions used in conditional statements in the dialplan.

Capturing Values

Matching items in parentheses are captured in variables sequentially named $1, $2, etc. $0 contains the entire pattern.

Examples

Any toll free Number

Matches any toll free number

```
^(\+?1)?(8(00|44|55|66|77|88)[2-9]\d{6})$
```

Example: **8005551212** or **18005551212** or **+18005551212** will match and in each case $2 is populated with "8005551212"

Explanation:

- ^ - indicates the beginning of a regular expression (required);
- (- indicates the beginning of a regular expression block - blocks are important to define inner expressions so they can be referenced by the variables $1, $2, $3, etc;
- \+1|1? - indicates optional digits '+1' or '1' (the ? sign defines the literal as optional);
-) - closes the block;
- 8 - matches the literal '8';
- (- another block starting;
- 00|55|66|77|88 - matches 00 or 55 or 66 or 77 or 88 (you guessed right, the | sign is the OR operator);
-) - closes the inner block;
- [2-9] - matches one digit in the range from 2 to 9 (2, 3, 4, 5, 6, 7, 8 and 9), and as you also guessed the [] pair of brackets encloses a range; other range examples: [0-9] matches 0 to 9; [a-z] matches a, b, c, ...z);
- \d - matches any valid digit (same as [0-9]);
- {6} - defines the number of occurrences for the previous expression, i.e. exactly 6 digits in the range 0-9. This can also contain a variable number of occurrences, for example to match a sequence of 6 to 9 digits: {6,9}; or to match at least 6 with no maximum: {6,};
-) - closes the other block;
- $ - indicates the end of the regular expression (required);

Any string of exactly seven digits

Matches any string of exactly seven digits

```
^(\d{7})$
Example: 5551212 will match with $1 having "5551212"
         95551212 will not match (string of digits too long)
         555121A will not match (no alpha chars allowed)
```

Any string of exactly 10 digits

Matches any string of exactly 10 digits

```
^(\d{10})$
Example: 8005551212 will match with $1 having "8005551212"
         18005551212 will not match (string of digits too long)
```

Any string of exactly 11 digits

Matches any string of exactly 11 digits

```
^(\d{11})$
Example: 18005551212 will match with $1 having "18005551212"
         8005551212 will not match (string of digits too short)
```

A string with prefix of 9 and an additional 11 digits

Matches a string with a prefix of 9 and an additional 11 digits.

```
^9(\d{11})$
Example: 918005551212 will match with $1 having "18005551212"
         8005551212 will not match (string of digits too short)
```

Any string of exactly 3 or 4 digits

Matches any string of exactly 3 or 4 digits

```
(^\d{4}$|^\d{3}$)
Example: 1001 will match with $1 having "1001"
         102 will match with $1 having "102"
         *9999 will not match
         *98 will not match
```

Any string from 3 to 5 digits

Matches any string from 3 to 5 digits

```
(^\d{3,5}$)
Example: 10010 will match with $1 having "10010"
         102 will match with $1 having "102"
         *9999 will not match
         *98 will not match
```

Variable number of digits

Matches a variable number of digits after a prefix of '9'

```
^9(\d+)$
Example: 977 will match ($1 = 77)
         877 will not match
         966736 will match ($1 = 66736)
         9118299983 will match ($1 = 118299983)
         9 will not match (+ means one or more)
```

International country codes

```
  <extension name="international" continue="true">
    <!--
            ^\+(1|  # NANPA
       2[1-689]\d|  # 21X,22X,23X,24X,25X,26X,28X,29X
            2[07]|  # 20, 27
          3[0-469]|  # 30,31,32,33,34,36,39
          3[578]\d|  # 35X,37X,38X
         4[0-13-9]|  # 40,41,43,44,45,46,47,48,49
              42\d|  # 42X
           5[09]\d|  # 50X, 59X
            5[1-8]|  # 51,52,53,54,55,56,57,58
            6[0-6]|  # 60,61,62,63,64,65,66
          6[7-9]\d|  # 67X,68X,69X
                 7|
       8[035789]\d|  # 80X,83X,85X,87X,88X,89X
           8[1246]|  # 81,82,84,86
           9[0-58]|  # 90,91,92,93,94,95,98
          9[679]\d   # 96X,97X,99X
                )(\d+)
    -->
    <condition field="destination_number" expression="^\+(1|2[1-689]\d|2[07]|3[0-469]|3[578]\d|4[0-13-9]|42\d|5
[09]\d|5[1-8]|6[0-6]|6[7-9]\d|7|8[035789]\d|8[1246]|9[0-58]|9[679]\d)(\d+)">
     <action application="set" data="country_code=$1"/>
     <action application="set" data="national_number=$2"/>
    </condition>
  </extension>
```

NANPA +1NxxNxxXXXX E.164 Dialstring

```
^(\+1|1)?([2-9]\d\d[2-9]\d{6})$
Example: +13172222222 matches
         13172222222 still matches because +1 or 1 are optional
         3172222222 still matches because +1 or 1 are optional
         3171222222 does not match and is not a valid NANPA number.
```

LNP (Local Number Portability)

```
^(?:1)?([2-9]\d{2}[2-9]\d{6})(?:;npdi=(?:yes|no))?(?:;rn=([2-9]\d{2}[2-9]\d{6}))?(?:;npdi=(?:yes|no))?
```

Example:

```
6045555555;npdi=yes;rn=7785550001
6045555555;rn=7785550001;npdi=yes
16045555555;npdi=yes;rn=7785550001
6045555555;rn=7785550001
Result:
$1=6045555555
$2=7785550001
```

Example:

```
6045555555
16045555555
6045555555;npdi=yes
6045555555;npdi=no
Result:
$1=6045555555
$2=empty
```

Clustering vs Capturing

Sometimes, you need to use parentheses to group a set of choices, but you are not interested in saving what is matched. Place a question mark and colon after the opening parenthesis to accomplish this. The example below matches a leading "+" or leading "00", but the matched information is not saved. The second set of parentheses does save information that is matched.

```
^(?:\+|00)(\d+)$
Example: +13171222222 matches and captures 13171222222 in $1
         0013171222222 matches and captures 13171222222 in $1
```

The following example matches a number and saves the information in two pieces as leading characters ($1) and the telephone number ($2).

```
^(\+|00)(\d+)$
Example: +13171222222 matches and captures + in $1 and 13171222222 in $2
         0013171222222 matches and captures 00 in $1 and 13171222222 in $2
```

Parentheses can be nested. When trying to determine which variable holds the matched information, count the opening parentheses from left to right. The first opening parenthesis will store information in $1, the second opening parenthesis will store information in $2 and so on.

```
^((\+|00)(\d+))$
Example: +13171222222 matches and captures +13171222222 in $1, + in $2 and 13171222222 in $3
         0013171222222 matches and captures 0013171222222 in $1, 00 in $2 and 13171222222 in $3
```

In the following example, the first set of parentheses does not store any information.

```
^(?:(\+|00)(\d+))$
Example: +13171222222 matches and captures + in $1 and 13171222222 in $2
         0013171222222 matches and captures 00 in $1 and 13171222222 in $2
```

Greediness

```
^rn=(.*);
Example: rn=1234567890;npdi; matches and captures 1234567890;npdi in $1
```

If you do .*? it will be un-greedy (or use \d to only match the numbers if you know it will be a number)

```
^rn=(.*?);
Example: rn=1234567890;npdi; matches and captures 1234567890 in $1
```

Case Insensitive

To make a match case insensitive, prepend (?i) to your match string.

```
Example: (?i)restricted matches both Restricted and restricted
```

Further Reading

Regular Expressions are serious business. They are quite useful in computing in general. Here are some other resources for learning more:
http://regexlib.com/Default.aspx
http://www.regular-expressions.info/
http://www.weitz.de/regex-coach/
http://regexlib.com/CheatSheet.aspx
http://www.zytrax.com/tech/web/regex.htm
http://oreilly.com/catalog/9781565922570/
http://perldoc.perl.org/perlreref.html

Test Your Regex

Use fs_cli

See: mod_commands#regex

Online Tools

- http://erik.eae.net/playground/regexp/regexp.html
- http://rubular.com
- http://scriptular.com/

Simple Perl Script

You can also use the following perl script to test your regular expression, or make some adjustment of regex based on what you need.

```
#!/usr/bin/perl
$x=$ARGV[0];
if($x=~/^(\+1|1)?([2-9]\d\d[2-9]\d{6})$/)
{
       print "Input: $x\n";
       print "\$1 Output: ".$1."\n";
       print "\$2 Output: ".$2."\n";
}
```

Result:

```
# perl x.pl 12135551212
Input: 12135551212
$1 Output: 1
$2 Output: 2135551212
```

```
# perl x.pl +12135551212
Input: +12135551212
$1 Output: +1
$2 Output: 2135551212
```

```
# perl x.pl 2135551212
Input: 2135551212
$1 Output:
$2 Output: 2135551212
```

See Also

regex API command performs pattern matching on the fs_cli

Regular Expression Cheat Sheet: http://overapi.com/regex/