Your Question

Article: 00115
Question:
What is Net Report's Regular Expression Technical Formalism?

Net Report Answer

Introduction

This document gives a general introduction to Regular Expressions before explaining Net Report's Regular Expression formalism.

Table of Contents

General Introduction to Regular Expressions

Regular expressions are a way to search for substrings ("matches") in strings. This is done by searching with "patterns" through the string. For example, regular expressions describe a match as a string which (in a simple case) consists of the character types to match and quantifiers for how many times you want to have the character type matched. Regular expressions enable the following operations to be performed on a string with regular expressions:

- Test for a pattern: search through a string and check whether a pattern matches a substring, returning true or false.
- Extract a substring: search for a substring and return that substring.
- Replace a substring: search for a substring that matches a pattern and replace it by another string.
- Backreferences: store ("capture") a part of the matches substring for later reuse. This is done by placing the substring in parentheses (...).

Brackets, Ranges and Negation

Bracket expressions introduce our first metacharacters, in this case the square brackets which allow us to define list of things to test for. These lists can be grouped into what are known as Character Classes typically comprising well known groups such as all numbers etc..

Metacharacter	Meaning
[ ]	Match anything inside the brackets for one character position once and only once, for example, [12] means match the target to either 1 or 2 while [0123456789] means match to any character in the range 0 to 9.
-	The - (endash) inside brackets is the 'range separator' and allows you to define a range, in the example above [0123456789] could be rewritten as [0-9]. For example, you can define more than one range inside a list e.g. [0-9A-C] means check for 0 to 9 and A to C (but not a to c). Note: to test for - inside brackets (as a literal) it must come first or last, that is, [-0-9] will test for - and 0 to 9.
^	The ^ (accent circumflex or caret) inside brackets negates the expression, for example, [^Ff] means anything except upper or lowercase F and [^a-z] means everything except lower case a to z. Note: spaces, or in this case the lack of them, between ranges are very important.

Positioning (or Anchors)

It is possible to control where in a target string the matches are valid. The following is a list of metacharacters that affect the position of the search:

Metacharacter	Meaning
^	The ^ (accent circumflex or caret) outside brackets means look only at the beginning of the target string.
$	The $ (dollar sign) means look only at the end of the target string, for example, fox$ (can be written as $fox) will find a match in 'silver fox' but not in 'the fox jumped over the moon'.
.	The . (period) means any character(s) in this position, for example, ton. will find tons and tonneau but not wanton because it has no following character.

Quantifiers (Iteration 'metacharacters')

The following is a set of quantifiers (iteration metacharacters) that can control the number of times a character or string is found in our searches.

Metacharacter	Meaning
?	The ? (question mark) matches the preceding character 0 or 1 times only, for example, colou?r will find both color and colour.
*	The * (asterisk) matches the preceding character 0 or more times, for example, tre* will find tree and tread and trough.
+	The + (plus) matches the previous character 1 or more times, for example, tre+ will find tree and tread but not trough.
{n}	Matches the preceding character n times exactly, for example, to find a local phone number we could use [0-9]{3}-[0-9]{4} which would find any number of the form 123-4567. Note: the - endash in this case, because it is outside the brackets, is a literal. The value is enclosed in braces.
{n,m}	Matches the preceding character at least n times but not more than m times, for example, 'ba{2,3}b' will find 'baab' and 'baaab' but NOT 'bab' or 'baaaab'. Values are enclosed in braces..

More 'metacharacters'

The following is a set of additional metacharacters that provide added power to searches:

Metacharacter	Meaning
()	The ((opening parenthesis) and) (closing parenthesis) may be used to group (or bind) parts of a search expression together
|	The | (pipe or vertical bar) is called *alternation* and means find the left hand OR right values, for example, gr(a|e)y will find 'gray' or 'grey'.

Net Report Expressions

The Net Report Filter Engine uses the following Net Report-specific Expressions, which we will refer to here as “Net Report Expressions”:

- Named Capture
- Backreference
- Regular Expression Predefinition
- Capture Renaming
- Stored Capture
- Multiple Capture on Quantifiers
- Named Capture with Reference

Named Capture (Group)

This function enables you to define a name associated to the value of the substring corresponding to the pattern captured in the regular expression. For example in the .Net (Framework).

Grouping Construct	Description
(?<name>exp)	Captures the matched substring into a group name or number name. The string used for name must not contain any punctuation and it cannot begin with a number. You can use single quotes instead of angle brackets; for example, (?'name').

Backreference

The treatment of a backreference replaces the reference by a regular expression corresponding to the value of the capture which it refers to.

Note: the term “Target String” refers to the string which we will be searching in which we want to find our match or search pattern

Exemple :

Regular Expression: <(?<tag>[^>]+)>(?<value>[^<]+)</\k<tag>>
Target String: Our email address
Capture Values:
tag = Our email address
value = sales@netreport.fr

Regular Expression Predefinition

This replaces a name by its pattern in the regular expression. This makes regular expressions easier to read and maintain.

(?:<name>) or (?:'name')

This specifies that the definition of the pattern referred to by the name 'name' must be replaced in the regular expression.

A form with capture of the syntax with capture is as follows

(?<name>) or (?'name')

This function is transitive.

For example:

Definitions:
IP: [0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}
Date: [0-9]{4}/[0-9]{2}/[0-9]{2} [0-9]{2}:[0-9]{2}:[0-9]{2}
Regular Expression: (?<src>(?:<ip>)) (?<server down time>(?:<date>)) the server is down
Target String: 2003/12/22 15:02:56 192.168.0.3 the server is down
Capture Values:

src	=	toto
server down time	=	2003/12/22 15:02:56

Example:

Definitions:
Ip: [0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}
port: [0-9]+
IpPort: (?<ip>):(?<port>)
Regular Expression: (?<IpPort>) (?<server down time>(?:<date>)) the server is down
Translated Regular Expression: (?:<ip>)(?:<port>) (?<server down time>(?:<date>)) the server is down

Capture Renaming

Enables you to rename captures that are included in predefinitions. Predefinitions can present problems concerning named captures which are included in the predefinition; it is therefore practical to be able to rename them.

The previous example can be rewritten as follows:

Definitions:
IP: [0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}
Date: [0-9]{4}/[0-9]{2}/[0-9]{2} [0-9]{2}:[0-9]{2}:[0-9]{2}
Regular Expression: (?<ip:src>) (?<date:server down time>) the server is down
Target String: 2003/12/22 15:02:56 192.168.0.3 the server is down
Capture Values:
src = toto
server down time = 2003/12/22 15:02:56

For a predefinition with capture the renaming applies to the capture. For a predefinition without capture the renaming applies to the captures it includes.

Syntax:

For a predefinition with capture:
(?<predefinition name:new name>)
For a predefinition without capture:
(?:<predefinition name:new name sub capture #1**[:**new name sub capture #n**]+**>)

The last redefinition prevails.

You can redefine at the first level all the captures included by the predefinition either directly or by transitivity. The connection between the captures and their names is established according to the order in which they appear. That is the nth name renames the nth capture.

It is not necessary to specify all the names, however, to rename the nth capture you must rename the n-1 previous captures.

Example:

Definitions:

```
Date: [0-9]{4}/[0-9]{2}/[0-9]{2} [0-9]{2}:[0-9]{2}:[0-9]{2}
Port: [0-9]+
IpPort: (?<Ip>) (?<Port>)
Regular Expression: (?<Ip:src>) (?<Port:src_port>) the server is down
Regular Expression: (?:<IpPort:src:src_port>) the server is down
```

Stored Capture

Stores the data captured in a named variable which will be reused following that.

Syntax:

```
?<$name>
```

If the « name » variable already exists then its value is replaced.

If the capture is multi-value then the different values of the capture are also stored.

The variable can be referenced, to do so its name must be preceded by the $ character (please see the section concerning Capture with Named Reference).

Multiple Capture on Quantifiers

Enables you to capture in a collection a pattern which corresponds to multiple values in the source string.

Example:

```
Regular Expression: (?<head>recipients<) *(?:(?<addresses:&Fields>[^@
]+@[^ ]+)[ ;]*)+(?<footer>>)
Target String: recipients<luc@dataset.fr    ggo@dataset.fr ; pierre@dataset.fr>
Capture Values:
Head                =       toto
Addresses(0)    =       luc@dataset.fr
Addresses(1)    =       ggo@dataset.fr
Addresses(2)    =       pierre@dataset.fr
footer              =       >
```

Named Capture with Reference

A Named Capture with Reference enables you to associate a capture name to a value according to the value of the previous capture.

Syntax:

(?<&name>exp)

<name> is either the name of a capture obtained from searching the target string or a stored capture (global variable) (please see the section concerning the stored captures).

In the case of the value of a capture this creates a new capture whose name is the last value obtained for the capture named ‘name’. If the resulting capture already had a value, then the value is added to it and the capture becomes a multi-value capture.

Example:

Definitions:

attr: [a-z]+

value: [0-9]+

Regular Expression: (?:(?<attr>)=(?:(?<value:&attr>),?)+ ?)+

Target String: string="toto a=5,6,7 b=8,9,10 fffff"

attr	a	b
a	5	8
b	6	9
	7	10

In the case of a global variable this creates a capture whose name corresponds to the item in rank n in the global variable (if the variable is single value then n = 1).

N being the rank of the captured value for the capture.

Example:

Definitions:

attr: [a-z]+

value: [0-9]+

Regular Expression: (?<&$Fields>[^ ""]+|(?:<VBString>))

Target String: "192.168.14 20/02/99 "a short message"

$Fields	Ip	Date	msg
Ip	192.168.14	20/02/99	"a short message"
Date			
Msg			