Author: Marco Petris Version: 1.1 Date: 17.01.10

Old query syntax	New query syntax	Remarks
bicycle	"bicycle"	
\where	"where"	But note: the quotation mark has to be escaped: "and then he said: \"hello\""
b.*	reg="\bb.*?\b"	\b is the word boundary marker and ? indicates a reluctant quantification otherwise we would match phrases rather than words because the .* would consume even the word boundaries
be.*	reg="\bbe.*?\b"	
ta[!aeiou]*	reg="\bta[^aeiou]*\b"	^ inverts the character class [aeiou]
t[d-j]*.* t[d-j]+.*	reg="\bt[d-j]*.*?\b" reg="\bt[d-j]+.*?\b"	reg="\bt[d-j]+.*?\b" makes more sense here since reg="\bt[d-j]*.*?\b" equals reg="\bt.*?\b" because [d-j]* includes empty sequences and the following .*? matches everything
.*ed	reg="\S*ed\b"	\S means „not a whitespace character"
@cars	@cars	
"the sky is blue"	"the sky is blue"	
freq = 5	freq = 5	
freq < 100	freq < 100	
freq >= 50	freq >= 50	
freq = 5-10	freq = 5-10	
simil friend 70%	simil = "friend" 70%	
tag = town	tag = "town"	
bob, builder	"bob", "builder"	
.*ed, freq > 20	reg="\S*ed\b", freq > 20	
bob & builder	"bob" & "builder" 10	10 is the span size for the collocation, if omitted the default value is 5
@cars - @germancars	@cars - @germancars	
te.* - test	reg="\bte.*?\b" - "test"	
an;u.*	"an";reg="\bu.*?\b"	
an;u.*;request	"an";reg="\bu.*?\b";"request"	
te.* - freq >= 100	reg="\bte.*?\b" - freq >= 100	
te.* where freq < 100	reg="\bte.*?\b" where freq < 100	
windows;crashed where freq < 100	"windows";"crashed" where freq < 100	
a.* where simil andy 50%	reg="\ba.*?\b" where simil= "andy" 50%	
markus where tag = villain	"markus" where tag = "villain"	
markus where tag = villain, tag = heir	"markus" where tag = "villain", tag = "heir"	
markus where tag = villain | tag = foe	"markus" where tag = "villain" | tag = "foe"	

The most important regular-expression constructs

For a full description see the documentation of Java regular expressions.

Construct	Matches
Characters	
x	The character *x*
`\\`	The backslash character
`\"`	The character "
Character classes	
`[abc]`	`a`, `b`, or `c` (simple class)
`[^abc]`	Any character except `a`, `b`, or `c` (negation)
`[a-zA-Z]`	`a` through `z` or `A` through `Z`, inclusive (range)
`[a-d[m-p]]`	`a` through `d`, or `m` through `p`: `[a-dm-p]` (union)
`[a-z&&[def]]`	`d`, `e`, or `f` (intersection)
`[a-z&&[^bc]]`	`a` through `z`, except for `b` and `c`: `[ad-z]` (subtraction)
`[a-z&&[^m-p]]`	`a` through `z`, and not `m` through `p`: `[a-lq-z]`(subtraction)
Predefined character classes	
`.`	Any character (may or may not match line terminators)
`\d`	A digit: `[0-9]`
`\D`	A non-digit: `[^0-9]`
`\s`	A whitespace character: `[ \t\n\x0B\f\r]`
`\S`	A non-whitespace character: `[^\s]`
`\w`	A word character: `[a-zA-Z_0-9]`
`\W`	A non-word character: `[^\w]`
POSIX character classes (US-ASCII only)	
`\p{Lower}`	A lower-case alphabetic character: `[a-z]`
`\p{Upper}`	An upper-case alphabetic character: `[A-Z]`
`\p{ASCII}`	All ASCII: `[\x00-\x7F]`
`\p{Alpha}`	An alphabetic character: `[\p{Lower}\p{Upper}]`
`\p{Digit}`	A decimal digit: `[0-9]`
`\p{Alnum}`	An alphanumeric character: `[\p{Alpha}\p{Digit}]`
`\p{Punct}`	Punctuation: One of `` !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~ ``
`\p{Graph}`	A visible character: `[\p{Alnum}\p{Punct}]`
`\p{Print}`	A printable character: `[\p{Graph}\x20]`
`\p{Blank}`	A space or a tab: `[ \t]`
`\p{Cntrl}`	A control character: `[\x00-\x1F\x7F]`
`\p{XDigit}`	A hexadecimal digit: `[0-9a-fA-F]`
`\p{Space}`	A whitespace character: `[ \t\n\x0B\f\r]`

Boundary matchers

`^`	The beginning of a line
`$`	The end of a line
`\b`	A word boundary
`\B`	A non-word boundary
`\A`	The beginning of the input
`\G`	The end of the previous match
`\Z`	The end of the input but for the final terminator, if any
`\z`	The end of the input

Greedy quantifiers

X`?`	*X*, once or not at all
X`*`	*X*, zero or more times
X`+`	*X*, one or more times
X`{`*n*`}`	*X*, exactly *n* times
X`{`*n*`,}`	*X*, at least *n* times
X`{`*n*`,`*m*`}`	*X*, at least *n* but not more than *m* times

Reluctant quantifiers

X`??`	*X*, once or not at all
X`*?`	*X*, zero or more times
X`+?`	*X*, one or more times
X`{`*n*`}?`	*X*, exactly *n* times
X`{`*n*`,}?`	*X*, at least *n* times
X`{`*n*`,`*m*`}?`	*X*, at least *n* but not more than *m* times

Possessive quantifiers

X`?+`	*X*, once or not at all
X`*+`	*X*, zero or more times
X`++`	*X*, one or more times
X`{`*n*`}+`	*X*, exactly *n* times
X`{`*n*`,}+`	*X*, at least *n* times
X`{`*n*`,`*m*`}+`	*X*, at least *n* but not more than *m* times

Logical operators

XY	*X* followed by *Y*
X`|`*Y*	Either *X* or *Y*
`(`*X*`)`	X

Quotation

`\`	Nothing, but quotes the following character
`\Q`	Nothing, but quotes all characters until `\E`
`\E`	Nothing, but ends quoting started by `\Q`