Reguläre Expressions für Text-Stammdaten

Grundprinzip

Für Strings werden ab der Version 13.2.8 werden normale reguläre Expressions genutzt. Damit sind grundsätzlich Zeichenauswahl-Filter (**[egh]**) , vordefinierte Zeichenklassen (\d - Zahl), Quantoren und auch die Behandlung von Sonderzeichen möglich für komplexere mehrzeilige Ausdrucks-Vergleiche.

Es ist lediglich zu beachten, dass zunächst immer die <Filterfunktion> getrennt durch das Tilde-Zeichen ~ und der regulären Expression definiert sein muss. Die Groß-und Kleinschreibung wird ignoriert.

Nutzbare Stammdaten

Folgende Stammdaten-Felder lassen sich in Filtern auswerten und nutzen.

Wert	Rückgabe-Beispiel	Semantik und Besonderheiten
WKN		<WKN>
ISIN oder Symbol		<ISIN>
Videotextname		
Markt	_DAX_,_MDAX_	analog Segments nur mit Kurznamen der Märkte und Verwendung von _
Notizen		Mehrzeiliger Text, der über den Chart oder über Doppelklick auf einen Titel gepflegt werden kann
Kurzname		
BasiswertISIN		

Wert	Rückgabe-Beispiel	Semantik und Besonderheiten
Variables	Yahoo-Symobl[isin]:LEO,	Zugeordnete und vorhandene Internet-Variablen z.B. Yahoo-Symbole <Variablenname>:<Wert>,
Segments	DAX,MDAX,Deutschland,	Zugeordnete <Marktsegment>,
Watchlists	Märkte,meineWatchlist,*Depot,	Zugeordnete <Watchlist>,

Verwendung von reguläre Expressions

Weitere Details zur Verwendung finden Sie hier:

https://www.regular-expressions.info/reference.html oder

https://de.wikipedia.org/wiki/Regul%C3%A4rer_Ausdruck

Regular Expression Basic Syntax Reference

Characters		
Character	**Description**	**Example**
Any character except [\^$.|?*+()	All characters except the listed special characters match a single instance of themselves. { and } are literal characters, unless they're part of a valid regular expression token (e.g. the {n} quantifier).	a matches a
\ (backslash) followed by any of [\^$.|?*+(){}	A backslash escapes special characters to suppress their special meaning.	\+ matches +
\Q...\E	Matches the characters between \Q and \E literally, suppressing the meaning of special characters.	\Q+-*/\E matches+-*/

\xFF where FF are 2 hexadecimal digits	Matches the character with the specified ASCII/ANSI value, which depends on the code page used. Can be used in character classes.	\xA9 matches © when using the Latin-1 code page.
\n, \r and \t	Match an LF character, CR character and a tab character respectively. Can be used in character classes.	\r\n matches a DOS/Windows CRLF line break.
\a, \e, \f and\v	Match a bell character (\x07), escape character (\x1B), form feed (\x0C) and vertical tab (\x0B) respectively. Can be used in character classes.	
\cA through \cZ	Match an ASCII character Control+A through Control+Z, equivalent to\x01 through \x1A. Can be used in character classes.	\cM\cJ matches a DOS/Windows CRLF line break.
Character Classes or Character Sets [abc]		
Character	**Description**	**Example**
[(opening square bracket)	Starts a character class. A character class matches a single character out of all the possibilities offered by the character class. Inside a character class, different rules apply. The rules in this section are only valid inside character classes. The rules outside this section are not valid in character classes, except for a few character escapes that are indicated with "can be used inside character classes".	
Any character except ^-]\ add that character to the possible matches for the character class.	All characters except the listed special characters.	[abc] matches a, b orc
\ (backslash) followed by any of ^-]\	A backslash escapes special characters to suppress their special meaning.	[\^\]] matches ^ or]
- (hyphen) except immediately after the opening [	Specifies a range of characters. (Specifies a hyphen if placed immediately after the opening [)	[a-zA-Z0-9] matches any letter or digit
^ (caret) immediately after the opening [	Negates the character class, causing it to match a single character*not* listed in the character class. (Specifies a caret if placed anywhere except after the opening [)	[^a-d] matches x (any character except a, b, c or d)
\d, \w and \s	Shorthand character classes matching digits, word characters (letters, digits, and underscores), and whitespace (spaces, tabs, and line breaks). Can be used inside and outside character classes.	[\d\s] matches a character that is a digit or whitespace

\D, \W and \S	Negated versions of the above. Should be used only outside character classes. (Can be used inside, but that is confusing.)	\D matches a character that is not a digit
[\b]	Inside a character class, \b is a backspace character.	[\b\t] matches a backspace or tab character
Dot		
Character	**Description**	**Example**
. (dot)	Matches any single character except line break characters \r and \n. Most regex flavors have an option to make the dot match line break characters too.	. matches x or (almost) any other character
Anchors		
Character	**Description**	**Example**
^ (caret)	Matches at the start of the string the regex pattern is applied to. Matches a position rather than a character. Most regex flavors have an option to make the caret match after line breaks (i.e. at the start of a line in a file) as well.	^. matches a inabc\ndef. Also matches d in "multi-line" mode.
$ (dollar)	Matches at the end of the string the regex pattern is applied to. Matches a position rather than a character. Most regex flavors have an option to make the dollar match before line breaks (i.e. at the end of a line in a file) as well. Also matches before the very last line break if the string ends with a line break.	.$ matches f inabc\ndef. Also matches c in "multi-line" mode.
\A	Matches at the start of the string the regex pattern is applied to. Matches a position rather than a character. Never matches after line breaks.	\A. matches a in abc
\Z	Matches at the end of the string the regex pattern is applied to. Matches a position rather than a character. Never matches before line breaks, except for the very last line break if the string ends with a line break.	.\Z matches f inabc\ndef
\z	Matches at the end of the string the regex pattern is applied to. Matches a position rather than a character. Never matches before line breaks.	.\z matches f inabc\ndef
Word Boundaries		
Character	**Description**	**Example**

\b	Matches at the position between a word character (anything matched by \w) and a non-word character (anything matched by [^\w] or \W) as well as at the start and/or end of the string if the first and/or last characters in the string are word characters.	.\b matches c in abc
\B	Matches at the position between two word characters (i.e the position between \w\w) as well as at the position between two non-word characters (i.e. \W\W).	\B.\B matches b in abc
Alternation		
Character	**Description**	**Example**
| (pipe)	Causes the regex engine to match either the part on the left side, or the part on the right side. Can be strung together into a series of options.	abc|def|xyz matchesabc, def or xyz
| (pipe)	The pipe has the lowest precedence of all operators. Use grouping to alternate only part of the regular expression.	abc(def|xyz)matches abcdef orabcxyz
Quantifiers		
Character	**Description**	**Example**
? (question mark)	Makes the preceding item optional. Greedy, so the optional item is included in the match if possible.	abc? matches ab orabc
??	Makes the preceding item optional. Lazy, so the optional item is excluded in the match if possible. This construct is often excluded from documentation because of its limited use.	abc?? matches ab orabc
* (star)	Repeats the previous item zero or more times. Greedy, so as many items as possible will be matched before trying permutations with less matches of the preceding item, up to the point where the preceding item is not matched at all.	".*" matches"def" "ghi" inabc "def" "ghi" jkl
*? (lazy star)	Repeats the previous item zero or more times. Lazy, so the engine first attempts to skip the previous item, before trying permutations with ever increasing matches of the preceding item.	".*?" matches "def"inabc "def" "ghi" jkl

+ (plus)	Repeats the previous item once or more. Greedy, so as many items as possible will be matched before trying permutations with less matches of the preceding item, up to the point where the preceding item is matched only once.	".+" matches"def" "ghi" inabc "def" "ghi" jkl
+? (lazy plus)	Repeats the previous item once or more. Lazy, so the engine first matches the previous item only once, before trying permutations with ever increasing matches of the preceding item.	".+?" matches "def"inabc "def" "ghi" jkl
{n} where n is an integer >= 1	Repeats the previous item exactly n times.	a{3} matches aaa
{n,m} where n >= 0 and m >= n	Repeats the previous item between n and m times. Greedy, so repeating m times is tried before reducing the repetition to n times.	a{2,4} matches aaaa,aaa or aa
{n,m}? where n >= 0 and m >= n	Repeats the previous item between n and m times. Lazy, so repeating n times is tried before increasing the repetition to m times.	a{2,4}? matches aa,aaa or aaaa
{n,} where n >= 0	Repeats the previous item at least n times. Greedy, so as many items as possible will be matched before trying permutations with less matches of the preceding item, up to the point where the preceding item is matched only n times.	a{2,} matches aaaaain aaaaa
{n,}? where n >= 0	Repeats the previous item n or more times. Lazy, so the engine first matches the previous item n times, before trying permutations with ever increasing matches of the preceding item.	a{2,}? matches aa inaaaaa

Regular Expression Advanced Syntax Reference

Grouping and Backreferences		
Syntax	**Description**	**Example**

(regex)	Round brackets group the regex between them. They capture the text matched by the regex inside them that can be reused in a backreference, and they allow you to apply regex operators to the entire grouped regex.	(abc){3}matchesabcabcabc. First group matches abc.
(?:regex)	Non-capturing parentheses group the regex so you can apply regex operators, but do not capture anything and do not create backreferences.	(?:abc){3}matchesabcabcabc. No groups.
\1 through \9	Substituted with the text matched between the 1st through 9th pair of capturing parentheses. Some regex flavors allow more than 9 backreferences.	(abc|def)=\1matchesabc=abc ordef=def, but not abc=def ordef=abc.
Modifiers		
Syntax	**Description**	**Example**
(?i)	Turn on case insensitivity for the remainder of the regular expression. (Older regex flavors may turn it on for the entire regex.)	te(?i)stmatches teSTbut not TEST.
(?-i)	Turn off case insensitivity for the remainder of the regular expression.	(?i)te(?-i)stmatches TEstbut not TEST.
(?s)	Turn on "dot matches newline" for the remainder of the regular expression. (Older regex flavors may turn it on for the entire regex.)	
(?-s)	Turn off "dot matches newline" for the remainder of the regular expression.	
(?m)	Caret and dollar match after and before newlines for the remainder of the regular expression. (Older regex flavors may apply this to the entire regex.)	
(?-m)	Caret and dollar only match at the start and end of the string for the remainder of the regular expression.	
(?x)	Turn on free-spacing mode to ignore whitespace between regex tokens, and allow # comments.	
(?-x)	Turn off free-spacing mode.	

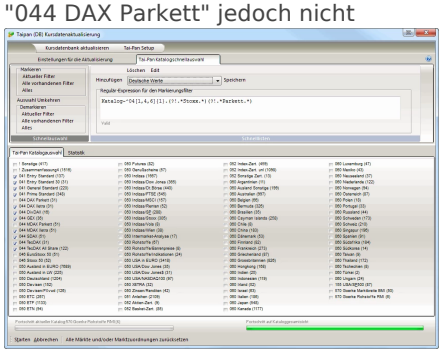
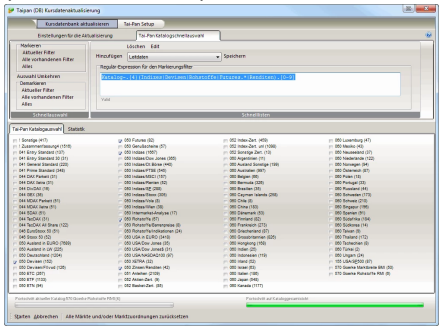
(?i-sm)	Turns on the option "i" and turns off "s" and "m" for the remainder of the regular expression. (Older regex flavors may apply this to the entire regex.)	
(?i-sm:regex)	Matches the regex inside the span with the option "i" turned on and "m" and "s" turned off.	(?i:te)stmatches TEstbut not TEST.
Atomic Grouping and Possessive Quantifiers		
Syntax	**Description**	**Example**
(?>regex)	Atomic groups prevent the regex engine from backtracking back into the group (forcing the group to discard part of its match) after a match has been found for the group. Backtracking can occur inside the group before it has matched completely, and the engine can backtrack past the entire group, discarding its match entirely. Eliminating needless backtracking provides a speed increase. Atomic grouping is often indispensable when nesting quantifiers to prevent a catastrophic amount of backtracking as the engine needlessly tries pointless permutations of the nested quantifiers.	x(?>\w+)x is more efficient than x\w+x if the second x cannot be matched.
?+, *+, ++ and{m,n}+	Possessive quantifiers are a limited yet syntactically cleaner alternative to atomic grouping. Only available in a few regex flavors. They behave as normal greedy quantifiers, except that they will not give up part of their match for backtracking.	x++ is identical to (?>x+)
Lookaround		
Syntax	**Description**	**Example**
(?=regex)	Zero-width positive lookahead. Matches at a position where the pattern inside the lookahead can be matched. Matches only the position. It does not consume any characters or expand the match. In a pattern likeone(?=two)three, both two and three have to match at the position where the match of one ends.	t(?=s)matches the second t instreets.

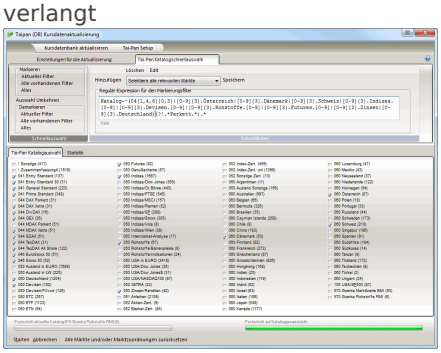
(?!regex)	Zero-width negative lookahead. Identical to positive lookahead, except that the overall match will only succeed if the regex inside the lookahead fails to match.	t(?!s)matches the firstt in streets.
(?<=regex)	Zero-width positive lookbehind. Matches at a position if the pattern inside the lookahead can be matched ending at that position (i.e. to the left of that position). Depending on the regex flavor you're using, you may not be able to use quantifiers and/or alternation inside lookbehind.	(?<=s)tmatches the firstt in streets.
(?<!regex)	Zero-width negative lookbehind. Matches at a position if the pattern inside the lookahead cannot be matched ending at that position.	(?<!s)tmatches the second t instreets.
Continuing from The Previous Match		
Syntax	**Description**	**Example**
\G	Matches at the position where the previous match ended, or the position where the current match attempt started (depending on the tool or regex flavor). Matches at the start of the string during the first match attempt.	\G[a-z] first matches a, then matches b and then fails to match in ab_cd.
Conditionals		
Syntax	**Description**	**Example**
(?(?=regex)then|else)	If the lookahead succeeds, the "then" part must match for the overall regex to match. If the lookahead fails, the "else" part must match for the overall regex to match. Not just positive lookahead, but all four lookarounds can be used. Note that the lookahead is zero-width, so the "then" and "else" parts need to match and consume the part of the text matched by the lookahead as well.	(?(?<=a)b|c)matches the second b and the first c inbabxcac
(?(1)then|else)	If the first capturing group took part in the match attempt thus far, the "then" part must match for the overall regex to match. If the first capturing group did not take part in the match, the "else" part must match for the overall regex to match.	(a)?(?(1)b|c)matches ab, the first c and the second c inbabxcac
Comments		
Syntax	**Description**	**Example**

(?#comment)	Everything between (?# and) is ignored by the regex engine.	a(?#foobar)bmatches ab

Beispiele

Das nachfolgend aufgeführte Feld "Katalog" als Feldname ist nur in der Katalog-Filter-Funktion nutzbar. Die Nutzung von Expressions lässt sich hiermit aber perfekt zeigen.

Ziel	Beispiel	Erklärung
Automatische Markierung im Tai-Pan-Katalog aller Deutschland-Kataloge	Katalog~^04[1,4,6]{1}.(?!.*Stoxx.*)(?!.*Parkett.*)	Maskiert alle Märkte startend mit 04, gefolgt von 1, 4 oder 6. Dann mit einem beliebigen Zeichen und ausschließlich *STOXX* und *Parkett*, womit "044 DAX Xetra" markiert wird, "044 DAX Parkett" jedoch nicht 
Automatische Markierung im Tai-Pan-Katalog aller Leitdaten	Katalog~[0-9]{3}.(Indizes|Devisen|Rohstoffe|Futures.*|Renditen).[0-9]	Es werden alle Kataloge markiert die mit 3 Zahlen beginnen, einem beliebigen Zeichen fortgesetzt werden und dann mit Indizes oder Devisen fortgesetzt werden. Am Ende muss nochmals mindestens eine Zahl folgen nach einem beliebigen Zeichen (hier Leerzeichen). 

Ziel	Beispiel	Erklärung
Automatische Markierung aller relevanter Märkte	Katalog~^(04[1,4,6]{0,3}|[0-9]{3}.Österreich|[0-9]{3}.Dänemark|[0-9]{3}.Schweiz|[0-9]{3}.Indizes.[0-9]|[0-9]{3}.Devisen.[0-9]|[0-9]{3}.Rohstoffe.[0-9]|[0-9]{3}.Futures.[0-9]|[0-9]{3}.Zinsen|[0-9]{3}.Deutschland)(?!.*Parkett.*).*	Start immer mit 3 Ziffern und Ausschluß der Parkett-Kataloge für die 041,044,046-Kataloge. In den anderen sind .* Joker gesetzt und teilweise wird wie bei Indizes nach einem beliebigen Zeichen eine Ziffer verlangt 

Revision #1
Created 20 June 2022 12:36:31 by Jens Werschmoeller
Updated 20 June 2022 12:37:23 by Jens Werschmoeller