The Regular Expression Editor Manual

Jesper K. Pedersen

Contents

Abstract

This Handbook describes the Regular Expression Editor widget

Chapter 1

Introduction

The regular expression editor is an editor for editing regular expression in a graphical style (in contrast to the ASCII syntax). Traditionally regular expressions have been typed in the ASCII syntax, which for example looks like `^.*kde\b`. The major drawbacks of this style are:

- It is hard to understand for non-programmers.
- It requires that you *escape* certain symbols (to match a star for example, you need to type `\*`).
- It requires that you remember rules for *precedence* (What does `x|y*` match? a single `x` or a number of `y`, *OR* a number of `x` and `y`'s mixed?)

The regular expression editor, on the other hand, lets you *draw* your regular expression in an unambiguous way. The editor solves at least item two and three above. It might not make regular expressions available for the non-programmers, though only tests by users can tell that. So, if are you a non programmer, who has gained the power of regular expression from this editor, then please let me know.

Chapter 2

What is a Regular Expression

A regular expression is a way to specify *conditions* to be fulfilled for a situation in mind. Normally when you search in a text editor you specify the text to search for *literally*, using a regular expression, on the other hand, you tell what a given match would look like. Examples of this include *I'm searching for the word KDE, but only at the beginning of the line*, or *I'm searching for the word the, but it must stand on its own*, or *I'm searching for files starting with the word test, followed by a number of digits, for example test12, test107 and test007*

You build regular expressions from smaller regular expressions, just like you build large Lego toys from smaller subparts. As in the Lego world, there are a number of basic building blocks. In the following I will describe each of these basic building blocks using a number of examples.

Example 2.1 Searching for normal text.

If you just want to search for a given text, then regular expression is definitely not a good choice. The reason for this is that regular expressions assign special meaning to some characters. This includes the following characters: `.*|$`. Thus if you want to search for the text `kde.` (i.e. the characters `kde` followed by a period), then you would need to specify this as `kde\.`[1] Writing `\.` rather than just `.` is called *escaping*.

Example 2.2 Matching URLs

When you select something looking like a URL in KDE, then the program **klipper** will offer to start **konqueror** with the selected URL.
Klipper does this by matching the selection against several different regular expressions, when one of the regular expressions matches, the accommodating command will be offered.
The regular expression for URLs says (among other things), that the selection must start with the text `http://` . This is described using regular expressions by prefixing the text `http://` with a hat (the `^` character).
The above is an example of matching positions using regular expressions. Similar, the position *end-of-line* can be matched using the character `$` (i.e. a dollar sign).

Example 2.3 Searching for the word the, but not there, breathe or another

Two extra position types can be matches in the above way, namely *the position at a word boundary*, and *the position at a non-word boundary*. The positions are specified using the text `\b` (for word-boundary) and `\B` (for non-word boundary)
Thus, searching for the word `the` can be done using the regular expression `\bthe\b`. This specifies that we are searching for `the` with no letters on each side of it (i.e. with a word boundary on each side)
The four position matching regular expressions are inserted in the regular expression editor using four different positions tool

Example 2.4 Searching for either this or that

Imagine that you want to run through your document searching for either the word `this` or the word `that`. With a normal search method you could do this in two sweeps, the first time around, you would search for `this`, and the second time around you would search for `that`.
Using regular expression searches you would search for both in the same sweep. You do this by searching for `this|that`, i.e. separating the two words with a vertical bar.[2]
In the regular expression editor you do not write the vertical bar yourself, but instead select the alternative tool, and insert the smaller regular expressions above each other.

Example 2.5 Matching anything

Regular expressions are often compared to wildcard matching in the shell - that is the capability to specify a number of files using the asterisk. You will most likely recognize wildcard matching from the following examples:

- `ls *.txt` - here `*.txt` is the shell wildcard matching every file ending with the `.txt` extension.
- `cat test??.res` - matching every file starting with `test` followed by two arbitrary characters, and finally followed by the test `.res`

In the shell the asterisk matches any character any number of times. In other words, the asterisk matches *anything*. This is written like `.*` with regular expression syntax. The dot matches any single character, i.e. just *one* character, and the asterisk, says that the regular expression prior to it should be matched any number of times. Together this says any single character any number of times.
This may seem overly complicated, but when you get the larger picture you will see the power. Let me show you another basic regular expression: `a`. The letter `a` on its own is a regular expression that matches a single letter, namely the letter `a`. If we combine this with the asterisk, i.e. `a*`, then we have a regular expression matching any number of a's.
We can combine several regular expression after each other, for example `ba(na)*`. [3] Imagine you had typed this regular expression into the search field in a text editor, then you would have found the following words (among others): `ba`, `bana`, `banana`, `bananananananana`
Given the information above, it hopefully isn't hard for you to write the shell wildcard `test??.res` as a regular expression. Answer: `test..\.res`. The dot on its own is any character. To match a single dot you must write `\.`[4]. In other word, the regular expression `\.` matches a dot, while a dot on its own matches any character.
In the regular expression editor, a repeated regular expression is created using the repeat tool

Example 2.6 Replacing & with & in a HTML document

In HTML the special character `&` must be written as `&` - this is similar to escaping in regular expressions.
Imagine that you have written an HTML document in a normal editor (e.g. XEmacs or Kate), and you totally forgot about this rule. What you would do when realized your mistake was to replace every occurrences of `&` with `&`.
This can easily be done using normal search and replace, there is, however, one glitch. Imagine that you did remember this rule - *just a bit* - and did it right in some places. Replacing unconditionally would result in `&` being replaced with `&amp;`
What you really want to say is that `&` should only be replaced if it is *not* followed by the letters `amp;`. You can do this using regular expressions using *positive lookahead*.
The regular expression, which only matches an ampersand if it is not followed by the letters `amp;` looks as follows: `&(?!amp;)`. This is, of course, easier to read using the regular expression editor, where you would use the lookahead tools.

Chapter 3

Using the Regular Expression Editor

This chapter will tell you about how the regular expression editor works.

3.1 The organization of the screen

The most important part of the editor is of course the editing area, this is the area where you draw your regular expression. This area is the larger gray one in the middle.

Above the editing area you have two Toolbars, the first one contains the editing actions - much like drawing tools in a drawing program. The second Toolbar contains the *What's This?* button, and buttons for undo and redo.

Below the editing area you find the regular expression currently build, in the so called ascii syntax. The ascii syntax is updated while you edit the regular expression in the graphical editor. If you rather want to update the ascii syntax then please do, the graphical editor is updated on the fly to reflect your changes.

Finally to the left of the editor area you will find a number of pre-built regular expressions. They serve two purposes: (1) When you load the editor with a regular expression then this regular expression is made *nicer* or more comprehensive by replacing common regular expressions. In

the screen dump above, you can see how the ascii syntax ".*" have been replaced with a box saying "anything". (2) When you insert regular expression you may find building blocks for your own regular expression from the set of pre build regular expressions. See the section on user defined regular expressions to learn how to save your own regular expressions.

3.2 Editing Tools

The text in this section expects that you have read the chapter on what a regular expression is, or have previous knowledge on this subject.

All the editing tools are located in the toolbar above editing area. Each of them will be described in the following.

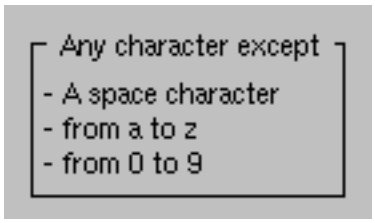
3.2.1 Selection Tool

The selection tool is used to mark elements for cut-and-paste and drag-and-drop. This is very similar to a selection tool in any drawing program.

3.2.2 Text Tool

Using this tool you will insert normal text to match. The text is matched literally, i.e. you do not have to worry about escaping of special characters. In the example above the following regular expression will be build: `abc\*\\\)`

3.2.3 Character Tool



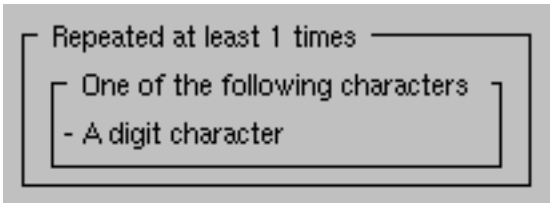
Using this tool you insert character ranges. Examples includes what in ASCII text says `[0-9]`, `[^a-zA-Z,_]`. When inserting an item with this tool a dialog will appear, in which you specify the character ranges.

See description of repeated regular expressions.

3.2.4 Any Character Tool

This is the regular expression ″dot″ (.). It matches any single character.

3.2.5 Repeat Tool



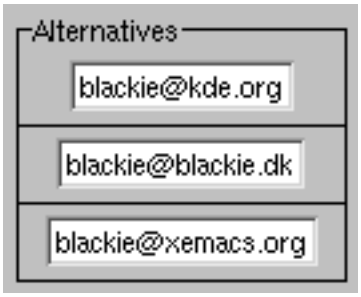
This is the repeated elements. This includes what in ASCII syntax is represented using an asterix (*), a plus (+), a question mark (?), and ranges ({3,5}). When you insert an item using this tool, a dialog will appear asking for the number of times to repeat.

You specify what to repeat by drawing the repeated content inside the box which this tool inserts.

Repeated elements can both be built from the outside in and the inside out. That is you can first draw what to be repeated, select it and use the repeat tool to repeat it. Alternatively you can first insert the repeat element, and draw what is to be repeated inside it.

See description on the repeated regular expressions.

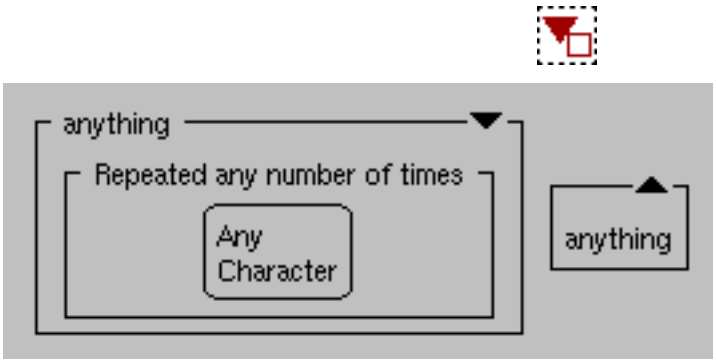
3.2.6 Alternative Tool



This is the alternative regular expression (|). You specify the alternatives by drawing each alternative on top of each other inside the box that this tool inserts.

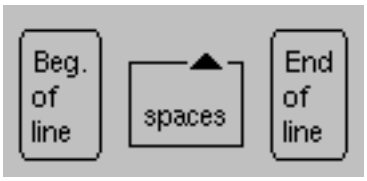
See description on alternative regular expressions

3.2.7 Compound Tool



The compound tool does not represent any regular expressions. It is used to group other sub parts together in a box, which easily can be collapsed to only its title. This can be seen in the right part of the screen dump above.

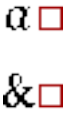
3.2.8 Line Start/End Tools



The line start and line end tools matches the start of the line, and the end of the line respectively. The regular expression in the screen dump above thus matches lines only matches spaces.

See description of position regular expressions.

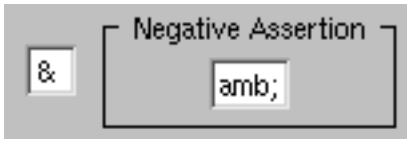
3.2.9 Word (Non)Boundary Tools



The boundary tools matches a word boundary respectively a non-word boundary. The regular expression in the screen dump thus matches any words starting with `the`. The word `the` itself is, however, not matched.

See description of boundary regular expressions.

3.2.10 Positive/Negative Lookahead Tools



The look ahead tools either specify a positive or negative regular expression to match. The match is, however, not part of the total match.

Note: You are only allowed to place lookaheads at the end of the regular expressions. The Regular Expression Editor widget does not enforce this.

See description of look ahead regular expressions.

3.3 User Defined Regular Expressions

Located at the left of the editing area is a list box containing user defined regular expressions. Some regular expressions are pre-installed with your KDE installation, while others you can save yourself.

These regular expression serves two purposes (see detailed description), namely (1) to offer you a set of building block and (2) to make common regular expressions prettier.

You can save your own regular expressions by right clicking the mouse button in the editing area, and choosing `Save Regular Expression`.

If the regular expression you save is within a compound container then the regular expression will take part in making subsequent regular expressions prettier.

User defined regular expressions can be deleted or renamed by pressing the right mouse button on top of the regular expression in question in the list box.

Chapter 4

Reporting bugs and Suggesting Features

Bug reports and feature requests should be submitted through the KDE Bug Tracking System. **Before** you report a bug or suggest a feature, please check that it hasn't already been reported/suggested.

Chapter 5

Frequently Asked Questions

5.1 Does the regular expression editor support back references?

No currently this is not supported. It is planned for the next version.

5.2 Does the regular expression editor support showing matches?

No, hopefully this will be available in the next version.

5.3 I'm the author of a KDE program, how can I use this widget in my application?

See The documentation for the class KRegExpEditorInterface.

5.4 I cannot find the *Edit Regular expression* button in for example konqueror on another KDE3 installation, why?

The regular expression widget is located in the package KDE-utils. If you do not have this package installed, then the *edit regular expressions* buttons will not appear in the programs.

Chapter 6

Credits and License