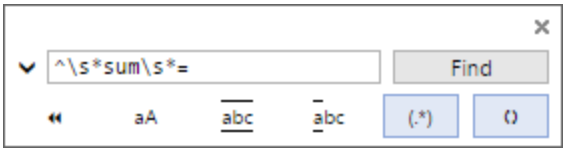
Regular expressions in the Code Editors' Quick Search dialog

Blog by Frank Doodeman

There's a handy feature in the Quick Search dialog of the IDE's Code Editors: you can use regular expressions in your Search term.



This blog gives a few simple but useful examples.

Where is my variable assigned a value?

Suppose you have a variable or field named SUM, and you quickly want to find the places where some value is assigned to it by means of an assignment statement. In a ProcScript assignment statement, the variable being assigned to is always at the beginning of the line, and it is followed by a single equal sign followed by whatever value is being assigned to it. Therefore, use this regular expression to find all assignment statements that address variable SUM:

^\s*sum\s*=

Let's briefly go through the parts of this regular expression to see how it works:

- It starts with a caret sign: ^. This matches the start of a line.
- Then there's \s*. The \s matches any blank character, such as space or tab, and the * specifies that it matches a sequence of zero or more of them.
- This is followed by the literal text sum, which matches the string "sum". By the way, sum matches without taking case into consideration – behavior controlled by the match case button, which is off by default.
- Then there's another \s*, matching a series of zero or more blank characters.
- It ends with a literal =, which matches an equal sign.

Where is my variable used?

To find where your variable SUM is used, use this regular expression as your Search term:

\<sum\>

This specifies the literal text sum enclosed in \< and \> characters. The use of \< and \>, which match the start and end of a word respectively, ensures that the expression won't find strings like **SUM**MARY, CHECK**SUM** or CON**SUM**PTION.

Remove trailing blanks from all lines

You can't see trailing blanks, but they may still annoy you. With this simple regular expression, you can find them and replace them with nothing:

\s+$

The \s we saw earlier: it matches any blank character. The + that follows it makes it match one or more of them. The $ stands for the end of the line. Use this regular expression as the Search term in the Quick Search dialog and empty the Replace term field in that dialog (make sure you set the dialog to Replace mode by pressing CTRL+H, or by clicking the down-arrow symbol on the left). Click the All button and voilà: no more trailing blanks in any of your lines.

Consistent use of the Struct arrow operator

You may have a coding convention for Struct member references that specifies that you use the -> operator without surrounding blanks, like this: vStruct->mbr. But suppose you stumble upon a piece of code from a former colleague who liked to have space characters around the arrow operator, like this: vStruct -> mbr. You can easily correct that using the Quick Search dialog and regular expressions. Your Search term should specify that you want to find the arrow operator enclosed in blanks:

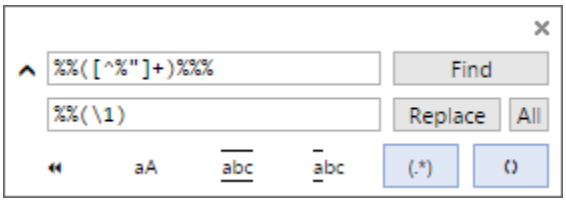
\s+->\s+

Then by simply using -> as your Replace term you can replace all these space-consuming arrow operators by compact ones.

Use the new style of string substitution

In the old days, ProcScript string substitution was done using expressions such as "%%FIELD.ENT%%%", where the initial two percent signs indicate the start of the string substitution and the trailing three percent signs indicate its end. In Uniface 10 (and in recent Uniface 9 versions) you can do this in a better way: "%%(FIELD.ENT)". The double percent sign indicates the start of the substitution; the expression inside the parentheses is expanded and used as the result of the substitution. Not only is this easier on the eye, but it is also more flexible, as you can specify any expression between those parentheses.

Of course, if you encounter pieces of code that use the old-style string substitutions, you will want to replace them with new-style ones. Regular expressions come to the rescue:



The Search term starts with the double percent sign that indicates the start of a string substitution. The [^%"]+ means a series of one or more characters that are not percent signs or double quote characters. This part of the Search term is inside parentheses, because that makes it possible to refer to it in the Replace term – or more accurately, refer to whatever it matches. The regular expression ends with the three percent signs that indicate the end of an old-style string substitution.

The Replace term also starts with a double percent sign, since that is also the start of a new-style string substitution. Next comes an opening parenthesis. In the context of a Replace term, this is a literal character; it has no special meaning. The \1 indicates whatever is matched by the part inside the parentheses in the Search term. Then the Replace term ends, with the literal closing parenthesis.

Now, a single click on the All button replaces all old-style string substitutions with new-style substitutions.

Use special comment markers

In some of our Uniface projects, we use a special comment style – a semicolon followed by a dash – to write comments in our ProcScript code. We do this to distinguish real comments from code that was temporarily commented out by using plain semicolons.

If you encounter a piece of code that you know has real comments, but the programmer did not use the above convention, you can use the Quick Search box and regular expressions to correct this. Specify this as a Search term, to find comments that are not special style:

^([^";]*);([^\-])

This regular expression consists of four parts:

- A caret sign: ^. This matches the start of a line.
- The first expression inside parentheses: ([^";]*). This matches any sequence of characters that are neither double quotes nor semicolons. It is enclosed in parentheses to make it possible to refer to it in the Replace term. The double quote is specified here because we don't want to match lines that have semicolons inside hard-coded strings.
- A plain semicolon character: ;. This indicates the start of a ProcScript comment.
- The second expression inside parentheses: ([^\-]). This matches a single character that is not a dash. The dash needs to be escaped with a backslash character because, in a regular expression, a dash has special meaning if it is inside square brackets.

Then use this as the Replace term:

\1;-\2

This specifies the first matched subexpression – the part of the line that precedes the semicolon; then a semicolon with a dash – the comment indicator that we want to use; and finally, the second matched subexpression, which is the first character that followed the semicolon.

Alert readers might realize that there may be lines with old-style comment that are not caught by the regular expression mentioned here. Which ones? That is left as an exercise for you: please post your answers in the comments section.

References

The IDE Code Editors are based on the Scintilla text editor, and therefore use Scintilla's specific regular expression capabilities (in posix mode). On some points, this might slightly deviate from the regular expression syntax that you are used to. For more information, visit https://www.scintilla.org/SciTERegEx.html