REGEX TOY: TESTING REGULAR EXPRESSIONS IN ABAP

Summary

Regular expressions are a powerful tool for processing text-based information effectively and efficiently. The Regex Toy is a small, interactive tool aimed at ABAP developers who want to test their regular expressions quickly. It improves upon similar tools available on the Web by reflecting specifically the semantics of regular expressions in ABAP.

Created on: 4 May 2006

Author Bio

Ralph Benzinger joined SAP in 2003. He is currently a developer in the ABAP language core development group at Walldorf, where he is working on the ABAP compiler and run-time environment. Ralph received a Ph.D. in Computer Science from Cornell University in the United States. Prior to joining SAP, he worked as management consultant in the Business Technology Office of McKinsey & Company.

Table of Contents

Testing Regular Expressions in ABAP

Since the introduction of regular expressions (or regexes, for short) into ABAP with NetWeaver Release 2004s, their adoption is steadily picking up as more and more developers discover their power and usefulness in processing text-based data. Information validation, extraction, and transformation often become a simple matter of a carefully crafted regex that would otherwise require many lines of ABAP code. But the many benefits of regular expressions are not my concern here.

Instead, I would like to dwell upon the difficulty in writing correct expressions that all users of regexes can testify to. The complexity of some of the more advanced features such as submatching, combined with the relative illegibility of regexes, has more than once lead to unexpected results that kept me scratching my head. In order to track down why my regex does or doesn't match a given text when it really should or shouldn't I either meditate over the regex pattern until inspiration hits me, or I fiddle around with both pattern and text by trial-and-error until I figure out what went wrong. Generally, I choose the latter approach and even wrote my own tool to assist me.

There are, in fact, many such regex matching tools freely available on the Web, the most popular ones being The Regex Coach, Visual Regexp, and The Regulator (currently MIA). While all of these tools offer great functionality, they also uniformly suffer from one major defect – they do not adequately reflect the way regexes work in ABAP.

There are basically two different "schools" when it comes to regular expressions: Perl and POSIX. Both flavors differ significantly in how matches are computed: whereas POSIX returns the *leftmost-longest match*, Perl computes what I call the *leftmost-first match*. For example, the statement

```
    FIND REGEX '(a+|[ab]+)b'
         IN 'xxxaaabaabbaxxx'
         MATCH OFFSET off MATCH LENGTH len.
```

will find the underlined sequence

xxx**aaabaabb**axxx

using POSIX-style matching, but would find the underlined sequence

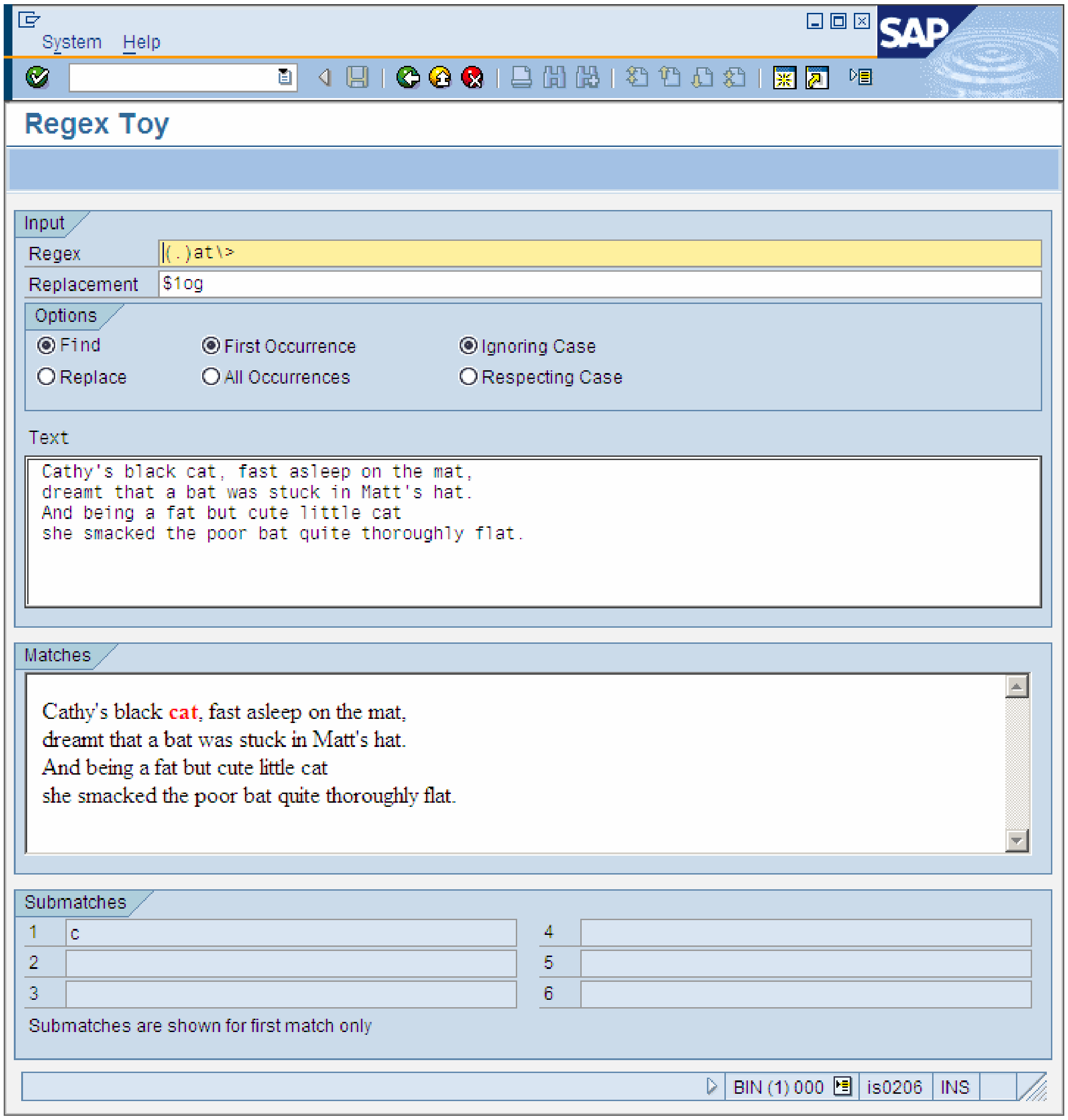
xxx**aaab**aabbaxxx

instead when using Perl-style pattern matching.

Simply speaking, POSIX uses backtracking to search exhaustively for the longest match for a given regex, whereas Perl stops as soon as a match is found. The crux is that **ABAP uses POSIX-style regexes**, while most other tools – including the ones mentioned above – use Perl-style regexes. Thus, using these tools for ABAP development can be extremely misleading.

The Regex Toy

Fortunately, there is no need to worry here – NetWeaver brings along its own regex tool called The Regex Toy, although you may need to upgrade your 2004s installation to a more recent support package (SP7, I believe) in order to use it. To invoke the Regex Toy, follow the link in the ABAP online documentation on regular expressions, or simply start report DEMO_REGEX_TOY. You'll be presented with the following screen:



Regex Toy

Input

Regex	(.)at\>
Replacement	$1og

Options

[✓ Find] Replace
[✓ First Occurrence] All Occurrences
[✓ Ignoring Case] Respecting Case

Text

```
Cathy's black cat, fast asleep on the mat,
dreamt that a bat was stuck in Matt's hat.
And being a fat but cute little cat
she smacked the poor bat quite thoroughly flat.
```

Matches

Cathy's black cat, fast asleep on the mat,
dreamt that a bat was stuck in Matt's hat.
And being a fat but cute little cat
she smacked the poor bat quite thoroughly flat.

Submatches

1	c	4	
2		5	
3		6	

Submatches are shown for first match only

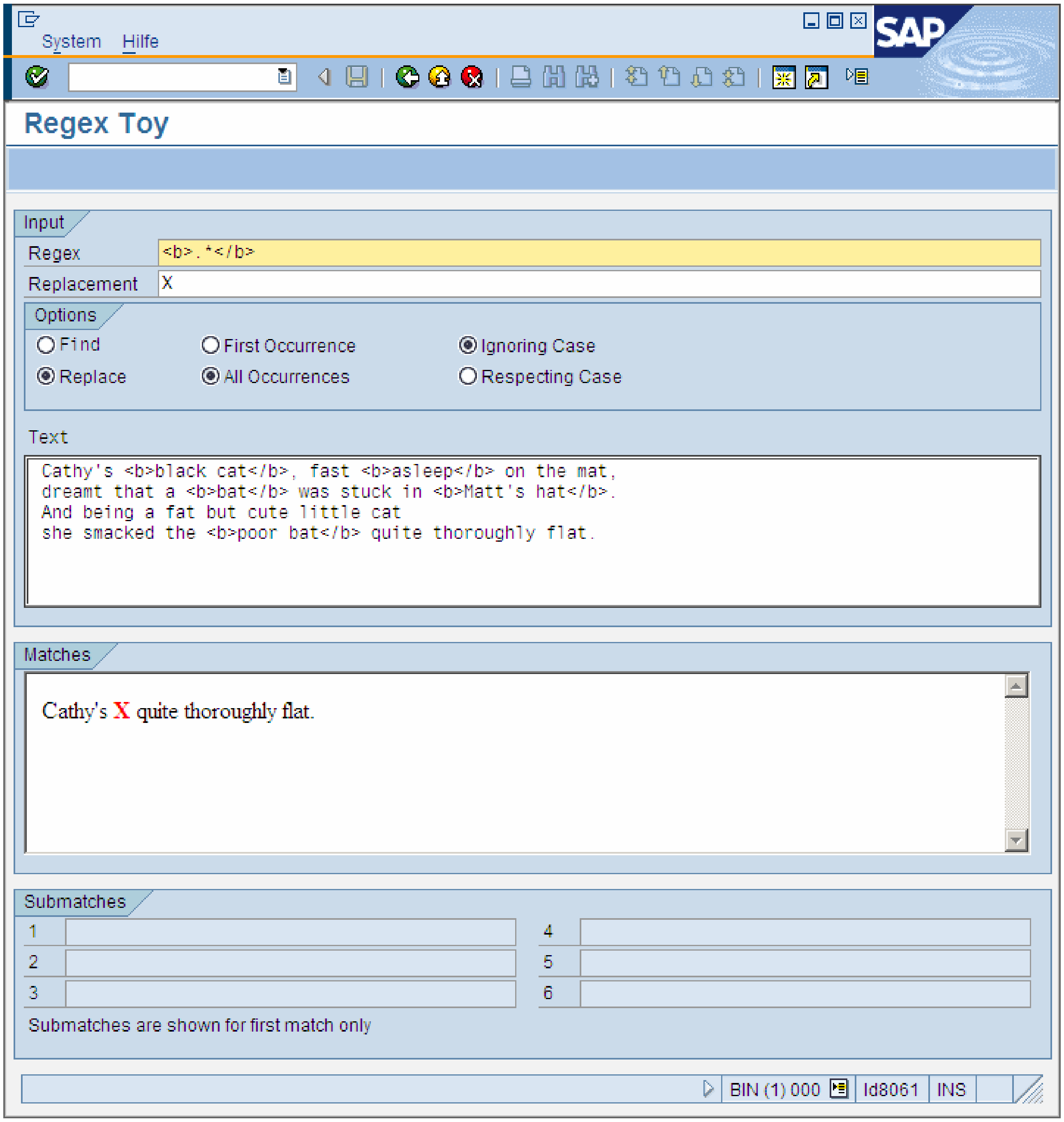
BIN (1) 000 is0206 INS

Really straightforward, isn't it? You enter your regex, some text and an optional replacement string at the top, select among a few options, and voila – your matches (or replacements) will be highlighted in red. The bottom part will also show any submatches that your very first match captured with parentheses.

Let's look at some examples. Suppose we want to redact all bold text from an HTML document, like this:

```
REPLACE ALL OCCURRENCES OF REGEX '<b>.*</b>'
           IN htmltext WITH 'X'.
```

Playing around with the Regex Toy quickly shows that this pattern will not do:



Regex Toy

Input

Regex	`<b>.*</b>`
Replacement	X

Options

( ) Find [✓ Replace]
( ) First Occurrence [✓ All Occurrences]
[✓ Ignoring Case] ( ) Respecting Case

Text

```
Cathy's <b>black cat</b>, fast <b>asleep</b> on the mat,
dreamt that a <b>bat</b> was stuck in <b>Matt's hat</b>.
And being a fat but cute little cat
she smacked the <b>poor bat</b> quite thoroughly flat.
```

Matches

Cathy's X quite thoroughly flat.

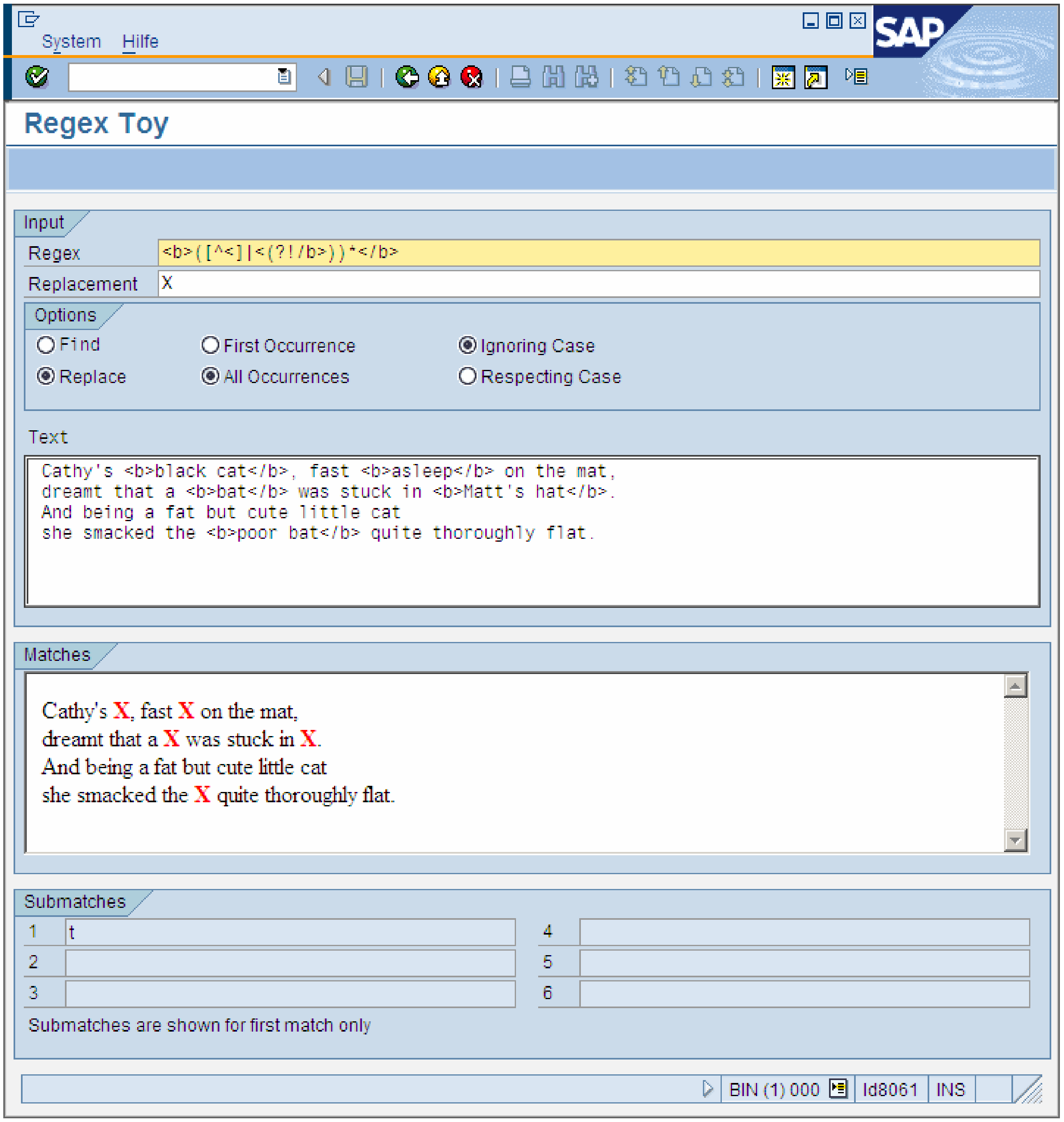
Submatches

1		4	
2		5	
3		6	

Submatches are shown for first match only

BIN (1) 000 Id8061 INS

We see that the `.*` after the <b> tag matches greedily anything up to the very last `</b>` tag – clearly, this is not what we had in mind. Fixing this is slightly involved, but regex `<b>([^<]|<(?!/b>))*</b>` will do:



Regex Toy

Input

Regex	`<b>([^<]|<(?!/b>))*</b>`
Replacement	X

Options

( ) Find [✓ Replace]

( ) First Occurrence [✓ All Occurrences]

[✓ Ignoring Case] ( ) Respecting Case

Text

```
Cathy's <b>black cat</b>, fast <b>asleep</b> on the mat,
dreamt that a <b>bat</b> was stuck in <b>Matt's hat</b>.
And being a fat but cute little cat
she smacked the <b>poor bat</b> quite thoroughly flat.
```

Matches

Cathy's X, fast X on the mat,
dreamt that a X was stuck in X.
And being a fat but cute little cat
she smacked the X quite thoroughly flat.

Submatches

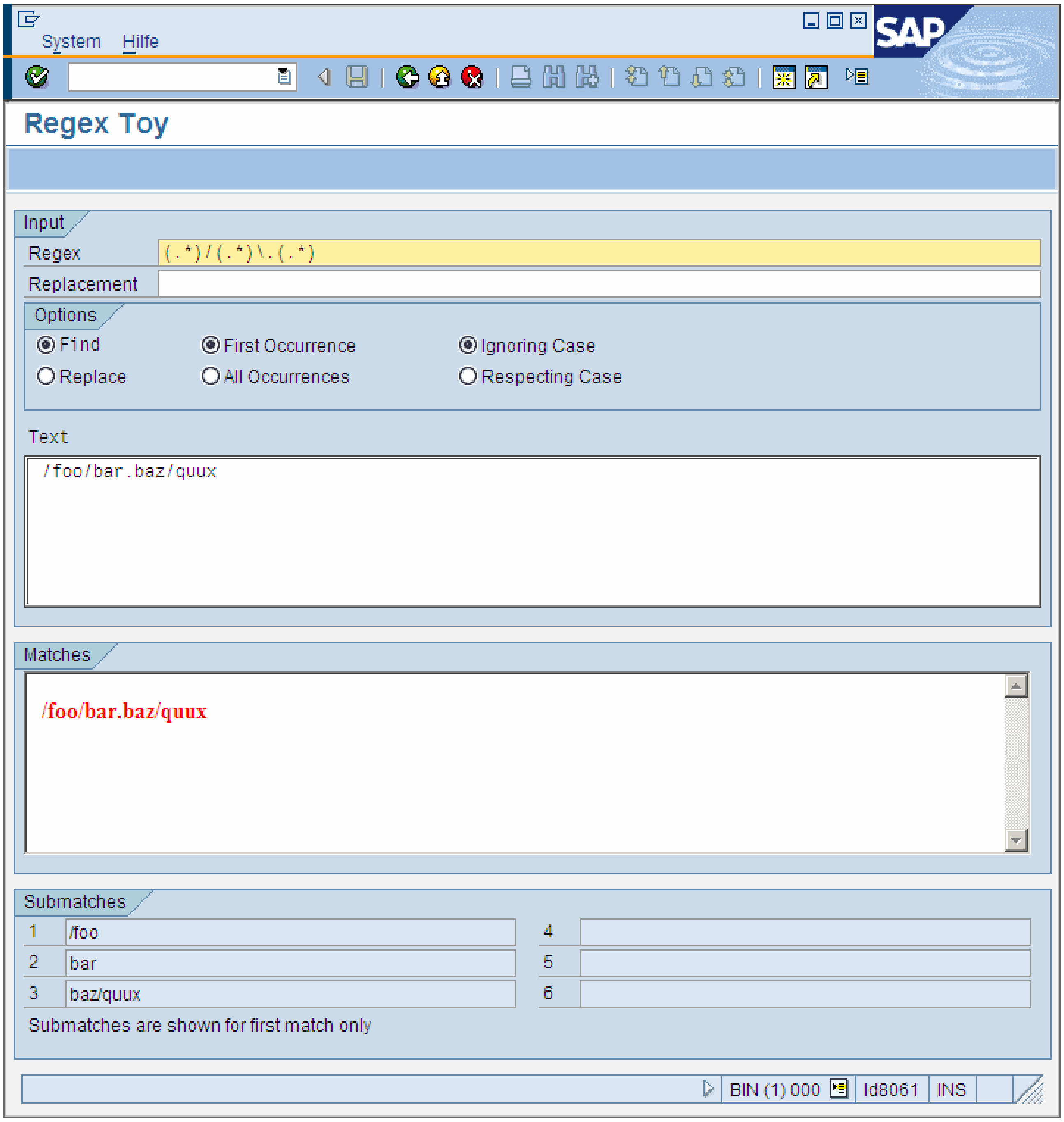
1	t	4	
2		5	
3		6	

Submatches are shown for first match only

As an example for extracting information with submatches, suppose that we want to write a regex that separates a file path such as `/foo/bar/quux.txt` into the directory part `/foo/bar`, the base name `quux`, and the extension `txt`. Our first try would be to match anything up to `/` as the directory, then anything up to `.` as the base name, and the remainder as the extension:

```
    FIND REGEX '(.*)/(.*)\.(.*)' IN text
         SUBMATCHES dir base ext.
```

We then use the Regex Toy to test this expression against a number of sample paths. Eventually, we will discover that our regex breaks for paths that have a dot inside the directory path but no extension:



Regex Toy

Input

Regex	`(.*)/(.*)\.(.*)`
Replacement	

Options

[✓ Find] Replace

[✓ First Occurrence] All Occurrences

[✓ Ignoring Case] Respecting Case

Text

```
/foo/bar.baz/quux
```

Matches

```
/foo/bar.baz/quux
```

Submatches

1	/foo	4	
2	bar	5	
3	baz/quux	6	

Submatches are shown for first match only

As we can see, everything after the dot is put erroneously into the third submatch, i.e., the extension. This flaw (along with others) is fixed with the improved regex `(.*/)?([^.]*)(\..*)?`.

Readers interested in the rationale for these two mildly complex regexes should keep an open eye for further articles on regular expressions on SDN.

Summary

Of course the Regex Toy is mostly that – a toy rather than a tool. It does, however, all that I needed at the time when I wrote it. Let me know of any killer features that you would like to see included in a future update. And I'm positively soliciting creative readers to send me a funny cat limerick to replace my crude poetry!

Related Content

1. ABAP Online Documentation (index search for "regex")
2. Eddy De Clercq: "Express Yourself Regularly", SDN
3. Wikipedia article on Regular Expressions

Copyright