

# iRules 101 - #10 - Regular Expressions



Joe Pruitt, 2007-20-12

---

While we don't recommend their use, regular expressions are sometimes a "necessary evil" during the development of iRules. This article will discuss the commands that support regular expressions and some tools you can use to help in coming up with the format you are looking for. Other articles in the series:

- [iRules 101 – #01 – Introduction to iRules](#)
- [iRules 101 – #02 – If and Expressions](#)
- [iRules 101 – #03 – Variables](#)
- [iRules 101 – #04 – Switch](#)
- [iRules 101 – #05 – Selecting Pools, Pool Members, and Nodes](#)
- [iRules 101 – #06 – When](#)
- [iRules 101 – #07 – Catch](#)
- [iRules 101 – #08 – Classes](#)
- [iRules 101 – #09 – Debugging](#)
- [iRules 101 – #10 – Regular Expressions](#)
- [iRules 101 – #11 – Events](#)
- [iRules 101 – #12 – The Session Command](#)
- [iRules 101 – #13a – Nested Conditionals](#)
- [iRules 101 – #13b – TCL String Commands Part 1](#)
- [iRules 101 – #14 – TCL String Commands Part 2](#)
- [iRules 101 – #15 – TCL List Handling Commands](#)
- [iRules 101 – #16 – Parsing String with the TCL Scan Command](#)
- [iRules 101 – #17 – Mapping Protocol Fields with the TCL Binary Scan Command](#)

## Regular Expressions

A regular expression is essentially a string that is used to describe or match a set of strings, according to certain syntax rules.

Regular expressions ("REs") come in two basic flavors: extended REs ("ERE"s) and basic REs ("BREs"). For you unix heads out there, EREs are roughly the same as used by traditional egrep, while BREs are roughly those of the traditional ed. The TCL implementation of REs adds a third flavor, advanced REs ("AREs") which are basically EREs with some significant extensions.

It is beyond the scope of this article to document the regular expression syntax. A great reference is included in the [re\\_syntax document](#) in the TCL Built-In Commands section of the TCL documentation.

## Regular Expression Examples

So what does a regular expression look like? It can be as simple as a string of characters to search for an exact match to "abc"

RE: {abc}

Or a builtin escape string that searches for all sequences of non-whitespace in a string

RE: {S+}

Or a set of ranges of characters that search for all three lowercase letter combinations

RE: {[a-z][a-z][a-z]}

Or even a sequence of numbers representing a credit card number.

```
{(?:3[47]\d{13})|(?:4\d{15})|(?:5[1-5]\d{14})|(?:6011\d{12})}
```

For more information on the syntax, see the [re\\_syntax manual page](#) in the TCL documentation, you won't be sorry.

Commands that support regular expressions

In the TCL language, the following built in commands support regular expressions:

[regexp](#) - Match a regular expression against a string

[regsub](#) - Perform substitutions based on regular expression pattern matching

[lsearch](#) - See if a list contains a particular element

[switch](#) - Evaluate one of several scripts, depending on a given value

iRules also has an operator to make regular expression comparisons in commands like "if", "matchclass" and "findclass"

[matches\\_regex](#) - Tests if one string matches a regular expression.

Think twice, no three times, about using Regular Expressions

Regular expressions are fairly CPU intensive and in most cases there are faster, more efficient, alternatives available. There are the rare cases, such as the Credit Card scrubber iRule, that would be very difficult to implement with string searches. But, for most other cases, we highly suggest you search for alternate methods. The "switch -glob" and "string match" commands use a "glob style" matching that is a small subset of regular expressions, but allows for wildcards and sets of strings which in most cases will do exactly what you need.

In every case, if you are thinking about using regular expressions to do straight string comparisons, please, please, please, make use of the "equals", "contains", "starts\_with", and "ends\_with" iRule operators, or the glob matching mentioned above. Not only will they perform significantly faster, they will do the exact same thing.

Here's an example:

```
BAD: if { [regexp {bcd} "abcde"] } {
```

```
BAD: if { "abcde" matches_regex "bcd" } {
```

```
BETTER: if { [string match "*bcd*" "abcde"] } {
```

```
BEST: if { "abcde" contains "bcd" } {
```

Conclusion

Regular expressions are available for those tricky situations where you just need to perform some crazy insane search like "string contains 123 but only if follows by "abc" or "def", or it contains a 5 digit number that is a valid US zip code". Also, if you need to find the exact location of a string within a string (for instance to replace one string for another), then `regexp` and `regsub` will likely work for you (if a Stream Profile doesn't). So, in some cases, a regular expression is likely the only option. Just keep in mind that even multiple string and comparison tests are more efficient than the most simple regular expression, so use them wisely.

[Get the Flash Player](#) to see this player.

F5 Networks, Inc. | 401 Elliot Avenue West, Seattle, WA 98119 | 888-882-4447 | f5.com

---

F5 Networks, Inc.  
Corporate Headquarters  
info@f5.com

F5 Networks  
Asia-Pacific  
apacinfo@f5.com

F5 Networks Ltd.  
Europe/Middle-East/Africa  
emeainfo@f5.com

F5 Networks  
Japan K.K.  
f5j-info@f5.com

---

©2016 F5 Networks, Inc. All rights reserved. F5, F5 Networks, and the F5 logo are trademarks of F5 Networks, Inc. in the U.S. and in certain other countries. Other F5 trademarks are identified at f5.com. Any other products, services, or company names referenced herein may be trademarks of their respective owners with no endorsement or affiliation, express or implied, claimed by F5. CS04-00015 0113