The last Tclsh Spot article showed how to write a simple HTML robot that to get the current price of a stock using the HTTP package and some simple regular expressions.

The final code looked like this:

```
package require http


foreach symbol $argv {
    set url "http://www.newsalert.com/free/stocknews?Symbol=$symbol"


    set id [::http::geturl $url]
    set data [::http::data $id]
    regexp {arts\?Symbol=(.+?)">(.+?)<}   $data match symbol price


    puts "$symbol last traded at $price"
    }
```

This is useful information, but there is a lot more info in the report from newsalert.com that it would be nice to get, and the `regexp` command seems like a good tool to use.

The interesting part of the page from newsalert.com looks like this:

```
 <td align="left" class="symprice"> <a href="/bin/charts?Symbol=SUNW">142 </a> $
 <td align="left" class="indexNumUp"> <img src="/gifs/uparrow2.gif"
width="9" height="8" alt="" border="0">6 </td>
 <td align="left" class="symprice">4.4 </td>
 <td align="left" class="symprice">12/3 </td>
 <td align="left" class="symprice">138 15/16 </td>
 <td align="left" class="symprice">143 3/8 </td>
 <td align="left" class="symprice">138 7/8 </td>
 <td align="left" class="symprice">11,781 </td>
```

The obvious pattern is that all the interesting data is in between a > and a < symbol after the `charts?Symbol=` string.

A simple brute force approach to this regular expression would be to match each `<` *`stuff`* `>` **`Data`** `<` *`stuff`* `>` pattern with a regular expression something like this:

```
<[^>]+?>([^<]+?)<[^>]+?>
```

This regular expression matches a < symbol, any other characters except a greater-than symbol up to the first > symbol (the <td align left..,tag), then any characters except a less-than symbol (the data) until the next < and any characters except the greater-than until the next > (the </td> tag).

This style of regular expression is familiar to most folks who have used regular expressions with sed or fgrep, and will work with all revisions of Tcl.

The regular expression code was rewritten for Tcl 8.1. Among other improvements (like support for Unicode), a new modifier was added to change the regular expression parsing behavior from using the maximum number of characters to match a regular expression to using the minimum number of characters to match the expression.

This lets us simplify the previous pattern to this;

```
<.+?>(.+?)<.+?>
```

This regular expression matches a < symbol, any other characters up the first > symbol, then any characters up to the next < and any characters until the next >.

We can concatenate as many copies of this pattern as we need to collect the change, percentage change, high, low, etc. This is conceptually simple, but creates a long incomprehensible regular expression.

Another new feature with the 8.1 and 8.2 Tcl interpreter is the `-expanded` flag. The `-expanded` flag causes the regular express parser to ignore whitespace and comments. Thus, instead of an ugly long regular expression, we can write

```
regexp -expanded {arts\?Symbol=(.+?)">  # Get the symbol
                  (.+?)                 # and the last sale price
                  <.+?>                 # Close the Href
                  .+?
                  <.+?>                 # Close the Table definition
                  .+?
                  <.+?>                 # Start a table definition
                  .+?
                  <.+?>                 # The image declaration
                  (.+?)                 # Absolute change
                  <.+?>                 # Close table} $page m symb pric chg
```

This is readable, but still longer than seems necessary.

The Tcl regular expression engine can be declared with repeating atoms. An atom can be a single character, or a regular expression enclosed in parentheses. The number of times the pattern can be matched is declared by following the atom with a value inside curly braces.

`{val}` Match the pattern exactly `val` times.

`{val,}` Match the pattern at least `val` times.

`{val,max}` Match the pattern at least `val` times, and at most `max` times.

This lets us shorten the regular expression by removing the repeated pattern of a tag followed by unnecessary characters.

```
regexp -expanded {arts\?Symbol=(.+?)">  # Get the symbol
                  (.+?)                 # and the last sale price
                  (<.+?>                # Close the Href and Close Table column
                  .+?){3}               #   start next table column
                  <.+?>                 # The image declaration
                  (.+?)                 # Absolute change
                  <.+?>                 # Close table} $page m symb pric dummy chg
```

The `dummy` variable is there to catch the part of the string that's matched by the duplicated regular expression.

We won't be using that data, so there is really no need to collect it.

A regular expression can be made "non-capturing" by following the left parentheses with a questionmark and colon, instead of continuing with the regular expression.

```
regexp -expanded {arts\?Symbol=(.+?)">  # Get the symbol
                  (.+?)                 # and the last sale price
                  (?:<.+?>              # Close the Href and Close Table column
                  .+?){3}               #   start next table column
                  <.+?>                 # The image declaration
                  (.+?)                 # Absolute change
                  <.+?>                 # Close table} $page m symb pric chg
```

This technique makes a fairly comprehensible regular expression, but we've only gotten as far as the price change. The code doesn't handle high, low, date, volume, etc.

More fields can be added to the regular expression by just extending this pattern, but that will get long.

We can use the match count descriptor to extract successive fields from a regular expression into variables by using a loop like this:

```
array set varnames {1 chg 2 pct 3 time 4 open 5 high 6 low 7 volume}
for {set i 1} {$i < 8} {incr i} {
 regexp -expanded "arts.Symbol=(.+?)\">  # Get the symbol
                  (.+?)                 # and the last sale price
                  (?:<.+?>              # Close the Href and Close Table column
                  .+?){3}               #   start next table column
                  (?:<.+?>                    # The image declaration
                  (.+?)                     # Absolute change
                  <.+?>.+?){$i}"      $page m symb pric $varnames($i)
}
```

This is fairly short, uses some nice new features of the `regexp` command. The only problem is that it parses the full HTML page to extract each and every item. That's only 8 passes, but still...

The data that we want to get from this page are the only characters not inside a tag. If we just strip the tag info away from the text, we'll be left with the data we want.

There's are other Tcl commands that uses the regular expression engine. One of these is `regsub` command.

Syntax: regsub *?options? expression string subSpec varName*

regsub	Copies *string* to the variable *varName*. If *expression* matches a portion of *string* then that portion is replaced by *subSpec*.
options	Options to fine tune the behavior of `regsub` May be one of:
	`-all` Replace all occurrences of the regular expression with the replacement string. By default only the first occurrence is replaced.
	`-nocase` Ignores the case of letters when searching for match.
	`--` Marks the end of options. Arguments which follow this will be treated as regular expressions, even if they start with a dash.
expression	A regular expression which will be compared to the target string.

string	A target string to which the regular expression will be compared.
subSpec	A string which will replace the regular expression in the target string.
varName	A variable in which the modified target string will be placed.

In this case, we can use the `regexp` command to strip away most of the HTML page, leaving us with several lines of html data that describe a single row in the table. Our script can use the `regsub` command to strip the tag information out of that string, and finally convert the multiple lines of data into a list with the `split` command.

```
regexp -expanded "arts.Symbol=(.+?)\">  # Get the symbol
                  (.+?)<tr>"      $page m symb data
regsub -all {<.+?>} $data {} lines
set dataList [split $lines \n]
```

This is a fairly small set of code for parsing the information out of these HTML pages. A robot using this parsing code would resemble this:

```
package require http


::http::config -proxyhost 56.0.0.2 -proxyport 8000


puts [format \
   {%-5s %-9s %-7s %-5s %-6s %-9s %-9s %-9s %-8s}\
  symb last change pct date open high low volume]


foreach symbol $argv {
    set url "http://www.newsalert.com/free/stocknews?Symbol=$symbol"


    set id [::http::geturl $url]
    set page [::http::data $id]


    regexp -expanded "arts.Symbol=(.+?)\">  # Get the symbol
                  (.+?)<tr>"      $page m symb data
    regsub -all {<.+?>} $data {} lines
    set dataList [split [string trim $lines] \n]


    puts [eval format \
      {{%-5s %-9s %-7s %-5s %-6s %-9s %-9s %-9s %-8s}} \
      $symbol $dataList]
 }
```

This robot will generate output resembling:

```
symb  last      change  pct   time   open      high      low       volume
sunw  142       6       4.4   12/3   138 15/16 143 3/8   138 7/8   24
hp    23 7/16   -1 1/16 -4.3  12/3   24 1/2    24 1/2    23 5/16   176
```

Again, this robot can be put into a crontab entry to put stock quotes into your mailbox.

But, now that I can collect daily quotes, I want to analyze and view the data. Next article will discuss saving the data and using the BLT graph widget to view it.