The last *Tclsh Spot* article introduced Tcl's `http` package, and showed how to use that package to retrieve an html page and display it using the `htmllib.tcl` package.

While we can retrieve and display HTML from PINE or ELM using the `htmlview.tcl` program, cruising the web is a task that's really better done with a dedicated browser. Surfing the web is an interactive process, and an application that's truly designed to be interactive rather than a static html viewing program will be more satisfying to use.

However, there are some web oriented tasks that I prefer to not do interactively. For instance, getting the current quote on several stocks by going to a website typing in each ID one at a time, clicking the button and waiting for it to load is about as amusing as watching rdist run.

This is a great job for a custom software robot that will parse the HTML page for information content, rather than for display instructions.

This article will describe a stock-quote retrieving robot, and introduce the Tcl regular expression commands to parse the page. (In fact, there are more generic ways to parse an HTML page, but regular expressions are just fine for simple dedicated robots.)

As you recall from last article, using the `http` package to retrieve a web page is pretty simple. We need to use the `package require` command to tell Tcl we'll be using the `http` package, and then use the `:http::geturl` command to retrieve a page.

Here's the code for retrieving a page with a quote for Sun Microsystems from the NewsAlert homepage.

```
set url "http://www.newsalert.com/free/stocknews?Symbol=sunw"

package require http

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

The good news is that getting the page is easy. The bad news is that the current price of Sun stock is 4 bytes out of 21K of data. Finding the two items of information (The stock symbol and the last price) in 21K of data is the fun part.

The interesting part of the page we retrieved from NewsAlert looks like this:

```
<td colspan="10"><table border="0" cellpadding="0" cellspacing="0" width="100%"><tr><td bgcolor="#6699CC"><img src="/images/spacer.gif" height="2" alt="" border="0"></td></tr></table></td>
</tr><tr >
<td align="center"><a href="/bin/headlines?Query=SUNW&SearchOption=ticker"><img src="/gifs/unchecked3.gif" width="13" height="13" border="0"></a></TD>
<td align="left" class="symprice"><a href="/bin/digest?Symbol=SUNW" onmouseover="status='Digest for Sun Microsystems Inc.';return true">SUNW</a></td>
<td align="left" class="symprice"><a href="/bin/charts?Symbol=SUNW">89 5/8</a></td>
<td align="left" class="indexNumD"><img src="/gifs/downarrow2.gif"
width="9" height="8" alt="" border="0">-3 3/4</td>
<td align="left" class="symprice">-4.0</td>
<td align="left" class="symprice">16:01</td>
<td align="left" class="symprice">94</td>
<td align="left" class="symprice">94</td>
<td align="left" class="symprice">88 15/16</td>
<td align="left" class="symprice">10,044</td>
</tr></table><table cellpadding="0" cellspacing="0" border="0" width="100%">
<tr>
```

If you're familiar with parsing strings in C using `strtok`, `strchr`, `strncpy` etc., it may seem obvious to parse the data using the Tcl `string` commands to find tokens and extract substrings from a larger string.

The Tcl `string` commands (;login: Vol 24, Num 3, June, 1999, P67) can be used to extract the information we want from a web page. However, while parsing HTML pages using `string first`, `string last` and `string range` commands is conceptually easy, the code tends to be rather long, and ends up with page specific strings to mark the start and end of interesting areas.

If we look at the page as a set of patterns, and extract the information based on the patterns in the HTML page, rather than individual match strings, we can make the extraction code more generic and robust.

Thinking in terms of patterns leads me to think of using regular expressions to describe the HTML page.

Tcl has supported basic regular expressions operating on standard ASCII text for many years. When Tcl version 8.1 was developed, the regular expression support was completely rewritten to support Unicode and more regular expression options. The regular expression support is based on Henry Spencer's implementation, which follows the POSIX 1003.2 specification and includes many of the Perl 5 extensions.

Tcl regular expressions come in three forms:

- **BRE:** POSIX *Basic* Regular Expressions
- **ERE:** POSIX *Extended* Regular Expressions
- **ARE:** *Advanced* Regular Expressions, generally a superset of EREs, with a few incompatibilities

This description of Regular Expressions is taken from Christopher Nelson's book *Tcl/Tk Programmer's Reference*, published by Osborne/McGraw-Hill. You should be able to order it by the time this issue of **;login:** is available. (See `www.purl.org/net/TclTkProgRef` for more information.)

> Tcl interprets regular expressions as AREs by default. BRE and ERE interpretation may be selected with embedded options.
>
> **Regular Expression Syntax**
>
> A regular expression (RE) is made up of *atoms* that match characters in a string, and constraints that limit where those matches occur. The simplest atom is a single character to be matched literally. For example, the regular expression "a" consists of one atom that matches the start of "abc" or "apple", the middle of "bat", or the end of "comma".
>
> Any single character can be matched with "." (dot). For example the RE **a.c** matches any three-character substring starting with *a* and ending with *c* anywhere within a string.
>
> One of several specific characters can be matched with a *class*. A class consists of square brackets surrounding a string of characters, collating elements, equivalence classes, or named character classes to match. Characters, collating elements, equivalence classes, and named classes may be freely mixed in a class, except that equivalence classes and named character classes may not be used as the end of a range.
>
> If the first character of a class is ^ (caret), the class is *negated* and matches any character *not* in the class. For example, **[^0-9]** matches any nondigit. A caret after the first character of a class has no special meaning.
>
> Characters may be listed explicitly (for example, **[0123456789]**) or may be described with an implicit range of characters, by specifying the first and last character in the range, separated by a dash (for example, **[0-9]**).
>
> *The following paragraph includes the Greek combination* "ae" *as a single-character symbol.*
>
> A Unicode *collating element* is a character or multi-character string that sorts as a single character, such as "ae" for æ. A collating element is represented by the multi-character string or the collating element's name surrounded by **[.** and **.]** such as **[.ae.]**.
>
> *The preceding paragraph includes the special character æ.*

The following paragraph includes several accented lower case o's. We can lose 3-4 of them if they are hard to typeset.

The following paragraph includes several instances of the greek letter pi.

A Unicode *equivalence class* is a set of characters that sort to the same position, such as o, ò, ó, ô, õ, ö. An equivalence class is represented by a member of the class surrounded by **[=** and **=]**, such as **[=o=]** for the preceding list of o variants. For example, if p and *pi* were members of an equivalence class, **[[=p=]]**, **[[=*pi*=]]**, and **[*ppi*]** all match either character.

> *Note: Character ranges, collating elements, and equivalence classes are highly locale-dependent, not portable, and not generally useful in regular expressions. Their support in **regexp** is a side effect of Tcl's general support of Unicode encoding.*

A *named character class* is specified with a class name between **[:** and **:]** and stands for all characters (not all the collating elements) belonging to that class. A named character class may not be used as the endpoint of a range. The POSIX.1 standard class names are listed in the following table. A locale may define others.

Name	Description
alnum	Alphanumeric characters
alpha	Alphabetic characters
blank	A blank character
cntrl	Control characters (e.g., ASCII characters less than 32 and 127)
digit	Decimal digits
graph	All printable characters except blank
lower	Lowercase alphabetic characters
print	Printable characters
punct	Punctuation
space	White space
upper	Uppercase alphabetic characters
xdigit	Hexadecimal digits

For example, **{[[:alpha:]]+[[:digit:]]+}** matches "abcd1234".

An atom can be *quantified* with one of several suffixes:

*	Specifies that the atom may match 0 or more times in the string
+	Specifies that the atom must match at least once, but may be repeated more times
?	Specifies that the atom may match zero or one occurrence.

> *Note: The + and ? quantifiers are not supported by BREs.*

Thus, **.*** matches any string, including an empty string, **a+b** matches one or more a's followed by a *b*, and **a[bc]?d** matches "ad", "abc", and "acd", but not "abcd".

Starting with Tcl revision 8.1, you can also explicitly specify *bounds* on how many instances of the atom match:

{*m*}	Specifies that the atom must match exactly *m* times
{*m*,}	Specifies that the atom must match *m* or more times
{*m*,*n*}	Specifies that the atom must match *m* to *n* times (inclusive). *m* must be less than or equal to *n*.

m and *n* may be in the range 0 to 255, inclusive.

> *Note: In basic regular expressions, the braces ({}) must be preceded by backslashes, like **\{m,n\}**.*

That gives us a short review/overview of how a regular expression can be constructed. The syntax of the Tcl regular expression command looks like this:

Syntax: `regexp` *?options? expression string ?matchVar? ?subMatchVar?*

`regexp` Returns 1 if *`expression`* has a match in *`string`*. If *`matchVar`* or *`subMatchVar`* arguments are present, they will be assigned values based on the matched substrings.

`options` Options to fine tune the behavior of `regexp`. May be one of:

- `-nocase` Ignores the case of letters when searching for a match.
- `-indices` Stores the location of a match, instead of the matched characters, in the *`subMatchVars`* variable.
- `-expanded` Use the expanded syntax that ignores whitespace and comments, allowing long regular expressions to be formatted and commented
- `-line` Enable newline-sensitive matching. Equivalent to specifying `-linestop` and `-lineanchor` options.

-linestop	Parses a dot (.) atom as all characters except newlines.
-lineanchor	Treates the beginning of string (^) and end-of-string ($) marker as beginning of line and end of line markers.
-about	Returns a list of information about the regular expression.
--	Marks the end of options. Arguments which follow this will be treated as a regular expressions even if they start with a dash.
expression	The regular expression to match with *string*
string	The string to search for the regular expression.
?matchVar?	If the regular expression matches the string, the entire matching string will be assigned to this variable.
?subMatchVarN?	The string matched by each of the substrings defined within parentheses in the regular expression will be assigned to these variables, in the order that the parenthesized expressions appear counted from left to right, and outer to inner.

The matchVar and subMatchVar variables make the regexp command very useful for pulling patterns out of strings.

For example, this little bit of line noise will extract the name and current price from the data:

```
regexp {arts\?Symbol=([^"]+)">([^<]*)} $data fullmatch name value
```

Note that the regular expression is enclosed in curly braces. Since many of the regular expression symbols also have meaning to the Tcl interpreter, they have to be distinguished from characters that the interpreter should evaluate. The Tcl interpreter will evaluate all characters in a string delimited with double quotes, but will not evaluate the characters in a string delimited with curly braces. If you want the Tcl interpreter to evaluate parts of a regular expression, but not others, you can enclose the string in double quotes, and escape the characters that you don't want evaluated with a backslash.

Examining the atoms from left to right:

arts.Symbol=	This is a match to the string arts?Symbol=, which occurs as part of the (more recognizable) string /bin/charts?Symbol=.
	The dot in arts.Symbol will match any character. In this case, that character will be the question mark.
	Because a question mark has meaning to the regular expression parser. The regular expression arts?Symbol= would not match the string /bin/charts?Symbol= The regular expression parser would evaluate the question mark as a quantifier (there must be 0 or 1 s atoms) instead of matching a question mark character.
	To match a literal question mark, we need to escape the question mark with a backslash. The regular expression: arts\?Symbol= would match the string /bin/charts?Symbol=,
([^"]+)	This subexpression will be used to set the value of the first sub-match variable (name).
	The class [^"] will match any character except a double-quote. The plus symbol (+) matches 1 or more characters that are not a double-quote.
	This subexpression matches the characters between the equals sign, and the first double quote - the symbol for this stock.
">	These two atoms match a double quote followed by a less-than symbol.
([^<]*)	This subexpression will be used to set the value of the second sub-match variable (value). It will match characters until the first less-than symbol.

The default behavior for the Tcl regular expression parser is to match the maximum number characters to a quantified expression. For example, this command

```
regexp {A(.*)A} AbcdAbcdAbcd full submatch
```

would extract the characters between the first and third A (the string "bcdAbcd") in the submatch variable, rather than the characters between the first and second A.

With the new regexp implementation, a question mark (?) can be used to change how a quantifier is used. If a question mark follows the quantifier +*, the regular expression engine will use the minimum number of characters necessary to match an expression, instead of the maximum.

For example, this regular expression

```
regexp {A(.*?)A} AbcdAbcdAbcd full submatch
```

places the characters bcd in the submatch variable.

We can simplify the regular expression for extracting the stock symbol and last price by using the dot atom, instead of a class to describe the characters we'll accept, and a question mark with the plus sign, to force the parser to match the smallest possible match, instead of the largest.

The new regular expression looks like this:

```
regexp {arts\?Symbol=(.+?)">(.+?)<} $data match symbol last
```

Here's a complete robot for getting the current stock prices:

```
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 robot will accept a list of stock symbols as command line arguments, and report the last trade price for each stock. You can run this script from a crontab and get quotes mailed to you when you'd like, without ever needing to click a button or view an advertisement.

The next *Tclsh Spot* article will look at more regular expression features, and ways to get the rest of the information out of the html page.