A Look at PerlClip

by Danny R. Faught

PerlClip is an eclectic little script I frequently use to generate test data for exploratory testing. PerlClip was born when I was showing James Bach how I use a single line of Perl code to generate many different types of test data that I can then copy and paste into the application I'm testing. He asked me if Perl could place data directly onto the clipboard so it didn't have to be copied manually. After a bit of research, I found out that it could. Later, James implemented this initial concept as a tool with several useful features and thus deserves the credit for making it a reality. (See the StickyNotes for references.)

How It Works

PerlClip uses a command line interface to accept a number of different commands. You can run it from the MS-DOS command line or a Unix-style shell. After each command, the test data you specified is placed directly onto the system clipboard. Then you switch back to the application you're testing and paste it right in. With some additional effort, you can place the data into a file for use in your testing.

Counterstrings

When I do exploratory testing, I often check to see how many characters I can stuff into each alphanumeric input field. You've probably tried this before. You either randomly pound on the keyboard or lean on one key for a long time. But if you do find a bug, how can you describe the process of reproducing it? I like to be more precise, so I use the PerlClip feature called "counterstrings." These are strings that help you determine their length.

After starting PerlClip, it asks for a pattern. To generate a counterstring twelve characters long, I type **counterstring 12:**

```
Pattern:
counterstring 12
*** Ready to Paste!
```

That's all there is to it. The string is now in my clipboard. All I have to do is paste it into the application I'm testing. This is the string that is created:

```
*3*5*7*9*12*
```

This curious-looking string is assembled using asterisks and numbers indicating the character position of the asterisk to the right of each number. When I paste this string into the input field and see **12*** at the end of the string, I know that all twelve characters were successfully pasted. It's important to count from the last asterisk, so if the string looks like ***3*5*7*9*12**, I start counting at the asterisk after the nine and count **9, 10, 11** to the end of the string for a total size of eleven characters.

Now, it's time to try a longer string to determine the actual limits of the input field. So I go back to PerlClip and ask for ten thousand characters using the **counterstring 10000** command. When I paste that into my target field in the application and scroll all the way to the right end of the field I see Figure 1.

I can see that the last asterisk is at character number ninety-nine. I count over one more character to the end of the string and find its length is one hundred. All of the characters after the first hundred were truncated, so now I know exactly how many characters are accepted in this field. I can compare this with the specifications and user documentation, if any. When I try this same test on the Mac OS version of the application, I find that I can't paste any part of a string that's longer than a hundred characters, which prompts me to explore whether this is standard behavior on Mac OS.

Figure 1: Using a counterstring to test the Tag Line feature in the Windows version of my client's application

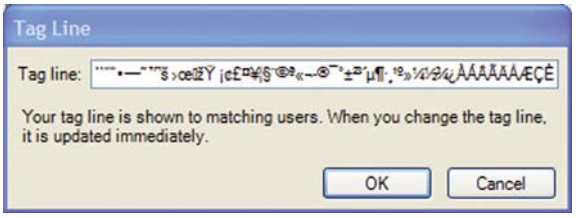

Figure 2: Pasting characters you didn't even know how to type

Bisection

You may have used the "bisection" or "binary search" technique to determine a precise failure point in a system. It's the same approach you use when finding a word in a dictionary, trying a page in between two endpoints, and then trying again in successively smaller sections of the book. PerlClip supports bisection with counterstrings using the "u" and "d" commands, which make the counterstring longer or shorter without the user's needing to track the string size by hand.

Allchars

For creating test data other than counterstrings, you can feed PerlClip any Perl code you can conjure up. Whatever data your code returns will be placed onto the clipboard. The simplest example is the built-in `$allchars` variable, which you can use like this:

```
Pattern:
$allchars
*** Ready to Paste!
```

The `$allchars` variable contains all

character codes between one and 255. You can use it to make sure that your application processes all input characters correctly, not only the traditional ASCII characters but also the codes higher than 127 that include non-English characters and other symbols.

Since `$allchars` generates 255 characters, I had a bit of trouble using it for a field that only allowed one hundred characters. When I paste the generated string, only the first hundred characters are used; the remaining characters are truncated. To test all of the characters in the set, I used the Perl function `substr`, which selects a substring:

```
Pattern:
substr($allchars,100,100)
*** Ready to Paste!
```

This extracted one hundred characters from the `$allchars` string, beginning at the one-hundredth character. I followed this selection with `substr($allchars,200)`, which gave me the last chunk of the string. By using three substrings, I covered all the characters in three passes and found two bugs in the process—bugs that would have been missed had I only tried the first one hundred characters. (See Figure 2.)

Text Files

The **textfile** command allows you to load a file from a disk and place the contents onto the clipboard. It works best when you have an easy-to-type path to the file on the disk; for example, if your data is your current working directory, you can type the filename without a directory path.

System Requirements

You can download PerlClip from www.satisfice.com/tools.shtml.

PerlClip is an open source tool, licensed under the GNU Public License. PerlClip currently runs on Windows and Mac OS X. It is available both as a Windows binary and as a script. If you run the perlclip.exe binary file on Windows, you don't need to have the Perl interpreter installed, so you can just install the one file on your test system and then you're ready to go. If you're using Mac OS X, use the perlclip.pl script. A suitable Perl interpreter was already available as part of every Mac OS X installation I've tried. At a terminal prompt, type **perl perlclip.pl** to start PerlClip.

If you're a Perl programmer, you can modify the perlclip.pl program. To run it on Windows, you'll need to install the Perl interpreter and the Win32::Clipboard module. ActiveState.com is the most popular place to get Perl for Windows. Use ActiveState's "ppm" tool to install modules after you have Perl installed.

Limitations

As nifty as it is, PerlClip has a few weaknesses. When I want to rapidly try different data lengths, "counterstring" is a long word to type. If I'm repeating it in the same PerlClip session, I usually can use the up arrow to edit a previously entered command and avoid typing it all again. But on Mac OS and often on the Cygwin shell on Windows, command line editing doesn't work. PerlClip relies on the command line editing features built in to MS-DOS.

Sometimes it's awkward for me to switch from my normal shell interface to PerlClip's command line and back. I find myself wishing for the ability to enter a command at the same time I invoke PerlClip. Then I could go right back to working in my shell. This feature does not yet exist.

PerlClip only accepts single-line commands. When I need to do something more elaborate, it may be easier to just write a Perl script from scratch.

The clipboard can't handle massively large strings of test data very well. A megabyte is usually fine. Ten megabytes or more might bring your system to a standstill. Also, it takes some extra work to get the generated data written to a file. That would be a nice addition to PerlClip.

The **u** and **d** bisection commands aren't very forgiving—if I type the wrong one, it's difficult to recover without some head scratching or starting over at the beginning. Also, bisection isn't supported for test data other than counterstrings.

Some people may be put off by the command line interface. It wouldn't be difficult to build a simple GUI on top of PerlClip.

To use anything beyond counterstrings and `$allchars`, you need a bit of Perl programming knowledge.

Summing It Up

As simple as the counterstrings concept is, I think it is one of the most interesting testing innovations I've seen in awhile. You can tell where I've been testing because you'll see counterstrings scattered all over the application. Adding the full power of Perl opens up an incredible range of possibilities for creating test data.

The PerlClip interface isn't well refined, but it gets the job done. Because it's open source, I can change the code myself if I feel compelled to improve it. **{end}**

---

*Danny R. Faught (that's *3*5* *. 2*4*6* in counterstrings) is an independent software testing consultant based in Fort Worth, Texas and a regular contributor on StickyMinds.com. Visit his Web site at www.tejasconsulting.com or contact him at faught@tejasconsulting.com.*

Thanks to James Bach for his help with this article.

Sticky Notes

For more on the following topics, go to www.StickyMinds.com/bettersoftware

- References