Perl and filePro

by Laura Brody

Beginner

Before you skip over this article because you think that it has nothing to do with filePro, or that Perl is useless on a DOS or Windows platform, read the next few paragraphs.

I have been threatening to learn Perl for a several years. This is no exaggeration – I am embarrassed to report that a "Mother of Perl" CD with at date of August 1996 on it resides in my CD rack. I made the decision to jump in, start learning and using it back in April. I was looking into CGI programs to add functionality to my web site and Perl was *everywhere*. Perl is the most common language used for CGI programming, hands down. Perl itself is free with massive collections of free scripts to do almost anything imaginable available for download on the internet.

A short time after I started to take a serious look at Perl and Ken was in the middle of coding new features for filePro v5.0, several stray thoughts of mine spontaneously converged to form a "really cool feature". I always thought that menu scripts were the only major portion of a filePro application which were not cross-platform compatible. Perl presented itself as the perfect solution to this nagging problem. Perl is powerful, mature, widespread, free, and available on every platform that filePro runs on. I told Ken about my idea and he agreed that it was a good solution. Over lunch, he added the code for `PFPERL` and for the menu editor to handle Perl scripts properly on all platforms.

As an added bonus, filePro developers could add Perl CGI programming to their list of services. Perl/CGI programming really isn't that difficult once you learn a few basics, assuming that you can already write an HTML page using a no frills text editor like vi or edit. The filePro Purity Test at http://www.hvcomputer.com/filepro/purity.html was my first Perl script of any complexity (and my first CGI script, period) and I am a far cry from a programming genius. (There's nothing like diving into the deep end of the pool when you want to learn to swim. <g>)

I will be covering CGI programming in future Perl articles in fPDJ, but if you are impatient, you could read about Perl/CGI Programming at http://cgi.resourceindex.com/Documentation/ and at http://www.virtualschool.edu/lang/perl/. You could also examine the source files of the filePro Purity Test which are in the public download area of my website.

At this point, it is my intention to have Perl as the defacto standard for filePro menu scripts from this day forward. No more writing shell scripts (and *trying* to write equivalent DOS batch) for the same application. Nuts to that. I don't know anyone who has the time to do the same thing twice. I vote for writing it once in Perl and being done with it.

With that in mind, it would be helpful for someone (I guess that would be me) to step up to the plate and give filePro developers the basic tools and knowledge to start using Perl in their menus. In an effort to encourage Perl usage by filePro developers, all of the Perl scripts in this article (and future enhancements to them) will be available on my website in the public download area. I want people to try them, examine them, use them and write their own Perl scripts.

The first thing you need is a copy of Perl (it is included on many Unix/Linux systems). Try typing "`perl -v`" on the command line. If it displays the current version number, etc. and it is v5.xxx you are ready to go. If you get a "file not found" message or the version number is v4.xxx or earlier, visit http://www.cpan.org/ports/index.html. Find your platform on the list and download the latest version for your system (the current version of Perl is v5.6), install it according to the documentation, and start exploring Perl. Don't worry if you don't have filePro v5.0 yet, you can still learn how you can use Perl in your filePro application. (And, using an external text editor, you can use those scripts with earlier versions of filePro.)

This article is a case of the teacher reading one chapter ahead of the students, so please forgive my lack of Perl programming expertise. (This means that experienced Perl programmers who read this article should be prepared to see some very amateurish code which will cause them to giggle uncontrol-

lably, but the code will do the job. TMTOWTDI*) With that disclaimer out of the way, let's explore Perl.

Perl was developed in 1986 by Larry Wall. The name, Perl, stands for "Practical Extraction and Report Language", but some people prefer Larry's other name of "Pathologically Eclectic Rubbish Lister". Larry Wall has the reputation of a rather unusual individual and his program reflects that individuality. The bible for Perl programmers is "Programming Perl" by Wall, Christiansen, and Schwartz, published by O'Reilly. (It's equivalent would be a filePro book written by Howard Wolowitz, Ken Brody, and Dave Roeger). It is known as "the camel book" by Perl hackers. Perl is a conglomeration of C and several Unix utilities (including shell, sed, awk) just as a camel appears to be a horse designed by committee. In my mind, the folks at O'Reilly chose the perfect animal to grace the cover of this book.

Perl has the reputation of being a cryptic and difficult language to learn. I feel that any language can be made quite unreadable by humans (see the Obfuscated filePro code on page 23). Some of Perl's syntax can appear quite bizarre to beginners. I suppose that it doesn't help Perl's reputation that there are Perl obfuscated code and Perl poetry contests (contestants write a working Perl program in the form of Haiku poetry and the like). Since I have C and Unix shell programming in my background, Perl felt quite familiar and almost comfortable. If you have programming experience with regular expressions (including writing your own edits in filePro), Perl will not look very foreign to you at all.

Perl:

```
#!/usr/bin/perl
print "Hello, world\n";
```

Shell:

```
echo "Hello, world\n"
```

C:

```
#include <stdio.h>
main()
{     printf ("Hello, world\n");
}
```

Figure 1

Here are some short examples of a Perl script, a shell script and a C program all doing the same thing.

Figure 1

This is a very simple example, but you can see some similarities already. Let's start examining the short Perl script. The first line is mandatory if you want to execute the script by its name, rather than as "`perl scriptname`". This tells the Unix/Linux shell where to find the Perl executable and that the file is a Perl script. On other systems, the Perl executable is in the path or is specified in an environmental variable. The second line sends the string "Hello, world" to STDOUT (usually the screen) followed by a newline.

Figure 2

The next example in figure 2 is a bit more useful. You can run it from a command prompt or from a filePro menu. It reads the environment for `PFPROG` and `FPPROG`, to locate the filePro executable pro-

```
#!/usr/bin/perl
# grab the following environmental variables
use Env qw(PFPROG FPPROG);
$PFPROG ||= $FPPROG; #if PFPROG has a value use it, otherwise try FPPROG.

@args = ( $PFPROG."/fp/dreport","test","-f","perltest","-a");
$RETCODE = (0xffff & system @args)/256; #divide by 256 to get actual code.

if ($RETCODE == 0)
      { print "Normal exit.\n"; }
else
      { printf "system (%s) returned %d:\n","@args",$RETCODE; }
```

Figure 2 – `menu.pl`

*Pronounced "tim-toady" – short for the popular Perl saying "There's More Than One Way To Do It."

```
::end:
@done::exit("37"):
```

Figure 3

grams. A command line for calling dreport with some parameters is built and put into the array "@args". A system call is made with those commands and the return code is placed into the variable "$RETCODE". The value is tested and if the return code was zero, "Normal exit." is displayed otherwise the command line and return code are displayed.

Figure 3

To see how filePro and Perl can pass information to each other, try adding the code in figure 3 into the "`perltest`" output format code. You will see that this Perl script will report that the return code was 37. (Note: You can use return codes from 0 to 255).

That little script is a handy starting place for most filePro programmers, but many of you need to read the contents of the fppath and config files at some time. No problem, examine figure 4 and I'll explain how it works.

Figure 4

At this point, you may be thinking "So what. I don't see what all of the fuss is about. I've been writing shell scripts more complicated than that for 20 years on my Xenix and Unix systems". But the exciting thing is that you can take this (or any) perl script, put it on a DOS/Windows system and run it without a single change (with the obvious exception of system calls to programs/scripts not available from every platform). Unix shell scripts aren't portable enough for the job, but Perl is.

Let me take a few moments to explain what some of the code is doing. I will not attempt to teach you all of Perl in this article, just enough for you to understand what this code does so that you can modify it for your own uses with minimal effort.

As I said earlier, the first line is mandatory for all Perl scripts no matter what platform you are running on, if you want to execute the script by its name, rather than with "`perl scriptname`". Anything that follows the '#' character is treated as a comment and ignored by Perl. Although Perl scripts can have any name and extension, it is customary to have an extension of "`.pl`". It is also customary to have the description at the top. In "real" Perl scripts, the documentation is included in "pod" format in the script itself. I have not had a chance to learn the syntax of pod just yet, so you will have to wait until the next issue of fPDJ for more information (after *I* learn how it works <g>).

Lines 20 and 21 tell Perl to include the modules english.pm and cwd.pm. This is similar to a C program `#include <>` statement. English.pm allows me to use the less cryptic names of the built-in variables. In this script, I use `$OSNAME` rather than `$^O` and `$ARG` rather than `$_`. Cwd.pm allows me to call `getcwd()` to get the current working directory on line 44.

The following section starts to get a little confusing. Line 24 defines a hash table named `%fpenv`. A hash table is a list of information with each entry having a string name and a value. The string name will be the environmental variable name and the value will be the value assigned to the environmental variable. i.e. in `ABE=ASCII`, the name is "`ABE`" with a value of "`ASCII`".

Line 25 loops through all of the environmental variables in memory (Perl automatically puts them into hash table `%ENV` for you) sorted by name. Line 26 pulls the value out of each entry. If the script is running on a DOS/Windows system, line 28 will change the name to all upper-case characters.

Line 30 puts the current environmental variable name into `$ARG` (commonly known as `$_`) so that the "if" statement on line 34 will use the environmental variable name to compare against the patterns `^ABE$`, `^PF` and `^FP`. This will match any environmental variable that matches exactly "`ABE`", or any variable which begins with "`PF`" or "`FP`". The following line converts any variables beginning with "`FP`" to "`PF`". I do this because `PF` variables have precedence over `FP` variables in filePro. Since the variables are being evaluated in alphabetical order, if there is both a `FP` and `PF` version of the same variable, the `PF` version will overwrite the converted `FP` value. (Also, you will only need to check the `PF` variable, and not have to check for both `PF` and `FP` every time you want to refer to it.) The line 36 simply adds the variable and it's corresponding value to the `%fpenv` hash table.

On line 43 through 46, I determine what the path

is to the fppath file and attempt to open it for reading. Since filePro looks for the fppath file in the root directory on DOS/Windows systems, I have to see if I have to look for the file there or in the traditional "`/etc/default/fppath`" location. On line 44, I append the first two characters of the current directory (the drive letter and "`:`") to "`/fppath`". The `substr()` works like the `mid()` in filePro in that it can be on either side of the "=", and you need to give it a string to work with, a starting point and number of characters to extract or insert. One thing to remember is that when it comes to string positions and arrays, filePro starts counting at "1", but Perl starts at "0". Therefore, line 49 places the first element of the array into the "`PFPROG`" entry of the `%fpenv` hash table.

I dump the entire contents of the fppath file into an array named "`@fppath`" on line 47 and remove the newline characters with Perl's `chomp` command on the next line. The notation "`||=`" is shorthand for "if the left side does not have a value, assign the right side to it". If "`PFPROG`" was an environmental variable and already in the hash table, it will not overwrite it with the value in the fppath file. (Just as filePro uses the environment variables to override the fppath entries.)

Since `PFCONFIG` can be used to override the standard location of the filePro config file, I check for it in the `%fpenv` hash on line 59. I set `$CONFIGFILE` to that value, if it exists, otherwise I set it to the standard location.

Now, I can locate the filePro config file and read the entries into `%fpenv`. I open the file on line 60 and start reading each line in on line 61. The `chomp` command will remove the newline character(s). There is one label in the config file, "`Colors:`", that I test for and ignore if it is found. The `split` command takes each line and returns two strings and puts them into `$env` and `$value`. It puts the characters until the "=" into `$env` and the remaining characters into `$value`. Under DOS/Windows, I force the characters in `$env` to upper-case. On line 70, I add the entry to the `%fpenv` hash table if that environmental variable is not already present.

The lines from 78 to 82 send the contents of the hash table to `STDOUT` (which is usually the screen) in alphabetical order so that you can see the results of the script.

Summary

I wrote these scripts to be a reasonable starting point for filePro developers to begin using Perl in their menu scripts. I would welcome comments, suggestions and improvements to these scripts.

To create the scripts in this article, I relied heavily on two books for syntax explanations and code examples: "Programming Perl" by Wall, Christiansen, and Schwartz; and "Perl Cookbook" by Christiansen and Torkington. I recommend both of them very highly. I found the FAQ pages at http://www.perl.com very helpful for a beginner Perl hacker too. As I explored Perl, I found numerous websites that were quite helpful and informative, so I created a page of these Perl resources on my website at http://www.hvcomputer.com/filepro/perl.html. Some of the bigger and better collections of Perl scripts, tutorial pages, books, magazines and newsgroups are on this page. Although it is far from comprehensive, it contains more than enough information to "make you dangerous".

Laura Brody is the founder of Hudson Valley Computer Associates, and is the mad genius behind the filePro Developer's Journal. She can be reached at <laura@hvcomputer.com>

"A computer lets you make more mistakes faster than any invention in human history -- with the possible exceptions of handguns and tequila."
Mitch Ratcliffe, _Technology Review_ April 1992

In retrospect,
"Let's get the goat drunk"
should have been my cue
to leave the party.

```
#!/usr/bin/perl
# This Perl script will read the /etc/default/fppath file
# and the environmental variables PF*, FP* to locate
# the filePro programs and data files. It will print out
# these values and the contents of the config file.
#
# Brought to you by: Laura Brody
#                on: 01 July 2000
# filePro Developer's Journal, Volume 1 Number 3.
# Copyright 2000, Hudson Valley Computer Associates, Inc.
#             http://www.hvcomputer.com
# This code may be freely distributed as long as this comment
# and author's name is in it. Please send any enhancements
# or suggestions to laura@hvcomputer.com.
#
# It requires the files:
# English.pm      (use less cryptic names of built-in variables),
# cwd.pm    (current working directory).
# You may need to set PERLLIB to the location of these files.
use English;
use Cwd;

# grab the following environmental variables:  ABE, PF* and FP*
%fpenv = ();      # define a hash table to store the info
foreach $k (sort keys %ENV) # read in all env vars in alphabetical order
{   $v = $ENV{$k};
    if ($OSNAME eq "MSWin32")
    { $k =~ tr/a-z/A-Z/; # force everything to upper case
    }
    $ARG = $k;
    # Note: Because FPxxx sorts before PFxxx, the PFxxx value will
    # override a matching FPxxx entry, which is the same behavior
    # as filePro.
    if ( m/^ABE$/ || m/^PF/ || m/^FP/ )
    {   $k =~ s/^FP/PF/;      # Convert FPxxx to PFxxx
      $fpenv{$k} = $v;
    }
}

# Read PFPROG, PFDATA, and PFDIR from the fppath file

# Decide the location of the fppath file
$PATHFILE = ($OSNAME eq "MSWin32")
      ? join "",substr(getcwd(),0,2), "/fppath"
      : "/etc/default/fppath";
if ( open (PATHFILE,"< $PATHFILE") )
{   @fppath = <PATHFILE>; # dump the contents into an array
    chomp(@fppath); # remove the newline characters.
    $fpenv{"PFPROG"} ||= $fppath[0]; #first line
    $fpenv{"PFDATA"} ||= $fppath[1]; #second line
    $fpenv{"PFDIR"}  ||= $fppath[2]; #third line
    close PATHFILE;
}

# Now that we know where the program and data files are located,
# we can read and display the contents of the config file.

# Note: PFCONFIG can override "$PFPROG/fp/lib/config"
```

```
59 $CONFIGFILE = $fpenv{"PFCONFIG"} || $fpenv{"PFPROG"}."/fp/lib/config";
60 if ( open (CONFIGFILE,"< $CONFIGFILE") )
61 {   while(<CONFIGFILE>)
62     { chomp;
63       # skip the "Colors:" label, display everything else.
64       if( ! /^Colors:/)
65       {     ($env,$value) = split(/=/,$ARG,2);
66             $env =~ s/^\s+//;    # remove any leading spaces
67             if ($OSNAME eq "MSWin32")
68             {   $env =~ tr/a-z/A-Z/; # force to upper case
69             }
70             $fpenv{$env} ||= $value;
71       }
72     }
73     close $CONFIGFILE;
74 }
75
76 # For debugging purposes: display the results
77
78 print join "", "-" x 20, " contents of fpenv ", "-" x 20, "\n";
79 foreach $key (sort keys %fpenv)
80 {     print "$key=$fpenv{$key}\n";
81 }
82 print join "", "-" x 20, " contents of fpenv ", "-" x 20, "\n";
```

Figure 4 – `fpinfo.pl`

Next issue

The theme for the next issue of the filePro Developer's Journal is "back to basics". This will include some basic tutorials, common programming situations and solutions, and neat tricks to add some pizzazz to your applications. We also plan on including an introduction to some of version 5.0's new features.

If you would like to submit articles that deal with topics suitable for filePro Developer's Journal, we would be most happy to consider them for publication. Be sure to follow the current author's guidelines, which are available on the web at: http://www.hvcomputer.com/fpdj/authors.html.

Manifest of files

The files included with this issue are stored in `DOS03.ZIP` (for DOS/Windows systems) and `unix03.tar` (for Unix/Linux systems). They both contain the identical information, but formatted appropriately for the platform. The files within the zip/tar file have the same basename, but the extension is different for each platform.

Basename	Article
`bar39`	3of9 Barcode by Jim Asman
`esak`	Barcodes, processing, etc by John Esak
`fpcgi`	fpCGI by Bob Haussmann
`HTMLpt3`	HTML Command, part 3
`obfus`	Obfuscated filePro code
`perl`	Perl and filePro by Laura Brody
`qwik`	Personal printer config from Bob Stockler
`uncgi`	UNCGI by Ted Dodd