Integrating legacy software with a Web front end

Netscape Enterprise Developer

May 1998

TABLE OF CONTENTS
TOPICAL INDEX
SEARCH

How to solve your legacy access problems with Web-based technology

By Scott Locklin

Summary
Cut expensive training and terminal costs and give your users a familiar interface by using Web pages to pass information back and forth to your legacy systems. Scott Locklin shows you how to use CGI to accomplish legacy-to-Web integration. *(1,850 words)*

Even with PC-based networks taking over the world, many corporations still have mission-critical software and databases sitting on older mainframes and other so-called legacy systems. Legacy software poses a number of problems to the MIS executive. Software revisions to legacy systems are generally time consuming and expensive. It's also possible that the expertise for writing the legacy software may no longer be available due to employee turnover -- not many new programmers these days are learning COBOL.

The platforms that legacy software run on may be vastly out of date, unsupported, or prohibitively expensive to maintain. Such platforms generally do not support a simple, modern user interface, and usually require expensive user training -- a time-waster in the sense that employees must be trained in *both* the legacy interface and the modern network desktop interface (i.e., Windows or MacOS).

Legacy software may also run on platforms that require proprietary networking equipment (such as the proprietary networking protocols used with IBM mainframes). Such old-style networks are often prohibitively expensive for a WAN or even a LAN, and are redundant in the sense that modern enterprise computing operations usually also have IP-based intranets and Internet access.

Legacy-to-Web integration is often a simple solution to many of these problems. Because Web-based connections function over standard IP networks, legacy-to-Web integration would remove the need for expensive proprietary networks running in parallel with the standard IP-based enterprise networks. Because most legacy software is written with the client/server model in mind (since it was written for mainframe/dumb terminal networks), integration into a Web-based client/server configuration is often a relatively simple task.

The Web browser user interface is not only familiar to most users, it is rapidly becoming ubiquitous; browsers are now available on platforms ranging from handhold computers to desktop computers, workstations, and powerful servers and supercomputers. Training costs, as well as the costs of maintaining terminal hardware and software, can be dramatically reduced by using the familiar Web browsers already on the desktops of most users.

Using Unix for legacy-to-Web migration

Derivatives of Unix are among the most popular and powerful operating systems presently available for this type of task. The Unix OS integrates naturally with IP networks, as historically IP networking was developed for networks of Unix systems. Because the universally popular C programming language was developed for and on Unix, there are terabytes of source code

written specifically for Unix-type operating systems; much or most of it is freeware.

The tool-based philosophy of Unix also allows for rapid development and integration of code on these platforms. Due to the ease of development on Unix systems, many of the legacy programming languages are available on Unix, making it possible to port the original source code directly; legacy languages such as LEXX, COBOL, Fortran, PL/1, Pascal, and BASIC are all well supported and even free for the various Unix flavors.

These advantages, combined with the historical development of the Web itself on Unix machines, the availability of Unix-style operating systems on powerful servers, and the high level of support for Web applications on Unix-type platforms make Unix the first choice of the Web developer seeking to open legacy software to a Web-based interface. Newer operating systems, such as Windows NT, are also becoming more competitive for this function, though their lack of support for high-end server hardware, closed programming models, and lack of available software make them less attractive for some applications.

It's also possible to retain the legacy server hardware and either run a Web server directly on it (assuming the hardware is robust, still supported by its parent company, supports TCP/IP, has a Web server ported to it, and is sufficiently powerful to support its user community), or integrate it with a Unix machine that runs the Web server and makes queries to the legacy hardware via custom network software.

Careful consideration must be given to these options as well as porting to Unix, paying close attention to issues such as robustness of and level of support for the legacy hardware, difficulty of Unix port, cost of administration of legacy hardware, life expectancy of the utility of the legacy software and a general cost-benefit analysis. Obviously, both technical staff and management should be involved in this decision, as it involves both technical and bottom-line issues.

The use of legacy hardware or mainframes as direct Web servers or co-processors for a standard Web server is beyond the scope of this article, the remainder of which will focus on legacy software that can be ported to run on a Unix platform.

Using CGI scripts to create the interface

CGI, or Common Gateway Interface, is the standard method for interfacing user-written programs to a Web server. Though the common parlance is "CGI script," there is no CGI scripting language, and the code could as easily be compiled as interpreted. Semantics aside, CGI is really just a set of environment variables that can be sent from the Web server program to a user-supplied program called by the Web server. The user-supplied program then acts on the contents of these environment variables.

In the client/server model we will be using, the script or program generally returns some HTML to the Web server, which returns it to the browser that called the CGI script in the first place. Though in principle one can use any programming language that allows passing of environment variables and printing to the standard output, the best language for such tasks is Perl. Perl is easy to learn if the programmer knows any Unix scripting or C programming, has powerful string manipulation and system integration features, and has an extensive library of functions and programs that are quite useful for Web scripting.

Porting an example in Fortran

As a simplified example of legacy to Web integration, I will give a quick and dirty illustration of how CGI works, while at the same time demonstrating a port of an extremely simple piece of legacy code written in Fortran. The same general principles will apply to porting any piece of legacy code written in any language to the Web via a client/server model -- though real-world situations will often be more complex, and will involve issues of security, CPU load, and other considerations that are beyond the scope of this article.

Let's say you have a legacy Fortran program that does something useful for engineers in the field. In the example below, I provide a trivial Fortran program for calculating how a translation through space and a thin lens will affect a ray of light. This program is incredibly simple, and could be ported to Perl directly, but it's useful for our examination. Here is the program code:

```
PROGRAM OPTICS
IMPLICIT NONE
REAL focal, distnc, ofst, angl, nwofst, nwangl
OPEN (UNIT=8,FILE='/home/httpd/tmp/tmp.data',STATUS='OLD')
READ(8,*) focal
READ(8,*) distnc
READ(8,*) ofst
READ(8,*) angl
CLOSE (UNIT=8)
nwofst = ofst + distnc * angl
nwangl = ofst/focal + (1. -  distnc/focal) * angl
PRINT *,'New Offset= ',nwofst,' New Angle= ',nwangl
END
```

To anyone with the misfortune of knowing Fortran, it should be obvious that this piece of code reads a value from a file, then writes a result to the standard output. This file will be compiled, the executable will be called optics.exe, and it will held in the same directory as the CGI scripts, */home/httpd/cgi-bin*.

Note that it is quite likely that the original code would have interactively prompted the user for the input values. A simple and portable way of inputting data is to use a data file instead.

On the Web server, you can build a page for an HTML form that users can fill out:

```
<html><head><title>Optics Calculation</title></head>
<body>
<h1> Optics Calculation </h1>
<form method="POST" action="http://www.myserver.com/cgi-bin/script.pl">
<P>enter the lens focal length(cm):<input type=text name=focal size=10><br>
<P>enter the propagation distance(cm):<input type=text name=distance
size=10><br>
<P>enter the angle (radians):<input type=text name=angle size=10><br>
<P>enter the offset from center(cm):<input type=text name=offset size=10><br>
<P><input type=submit value="submit"> * <input type=reset value="reset">
</form><hr>
<p><p>
</body></html>
```

This HTML document has some formatting information, some text, and some information to send to the Web server; particularly the name of the CGI script associated with this form (`action="http://www.myserver.com/cgi-bin/script.pl"`). Once this form is filled out and the Submit button is pressed, the entered data (among other things) is passed over the network to the Web server. The server program then executes the Perl CGI script referenced in the *action* variable, and passes the user-entered data and other CGI environment variables to the CGI script. Here's the CGI script itself:

```
#!/usr/bin/perl
# Note that many machines will have perl in /usr/local/bin/perl

# Get the data from the CGI environment variable

read(STDIN,$buffer,$ENV{'CONTENT_LENGTH'});

# These next lines of gibberish get rid of the string around the data and
# format the data into regular numbers. Perl/Unix voodoo happens here.

$inc = 1;
@pairs = split(/&/, $buffer);
foreach $pair (@pairs) {
  ($name, $value) = split(/=/, $pair);
  $value =~ tr/+/ /;
  $value =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;
  $value =~ s/<!--(.|\n)*-->//g;
```

```
  $value =~ s/,//;
  $value =~ s/\%//;
  $value =~ s/\$//;

# Check for any funky entries

   &funky_entry if ($value == 0);

# Create an array:
# Note I did it two ways so you can reference by a variable name or by an index

   $FORM{$name} = $value;
   $INDEX{$inc} = $value;
   $inc++;
}

# Write out the parameters for the Fortran code

open(OUTPUT,">/home/httpd/tmp/tmp.data");
$inc = 1;
while ($inc <= 4){
  printf OUTPUT ("%5.12E\n",$INDEX{$inc});
  $inc++;
}
close(OUTPUT);

# Call the Fortran routine and pipe the output to the variable $answers

open(RESULTS,"/home/httpd/cgi-bin/optics.exe|");
@answers = <RESULTS>;
close(RESULTS);

# Print beginning of HTML

print "Content-Type: text/html\n\n";
print "<html><head><title>Your Answer</title></head>\n";
print "<body><h1>The results:</h1>\n";

# Print the information

printf ("@answers\n");

#  Print the end of the HTML file

print "</body></html>\n";

# Funky Entry Subroutine

sub funky_entry {
 print "Content-Type: text/html\n\n";
 print "<html><head><title>Bad Entry</title></head>\n";
   print "<body><h1>Invalid Entry</h1>\n";
   print "Please enter only numbers, no text or\n";
   print "any other characters.\n";
   print "<p>\n";
   print "<hr>\n";
   print "</body></html>\n";
   exit;
}
```

You can see from the comments that the script takes the CGI input, uses some Unix "regular expressions" to break out the data from the passed environment variable, writes the parameters to a file, executes the Fortran code, then pipes the output of the Fortran code to the standard output, which is formatted HTML. The standard output of the script is then sent to the Web server, which returns it to the browser that posted the information from the HTML form. Though this was a trivial example, and there are housecleaning and security issues not addressed by this script, this basic outline of legacy code ported to a Web interface illustrates a general methodology for such tasks. Features such as graphical output can be easily added using Unix graphics utilities as well as Perl libraries. ■

Also this month in Netscape Enterprise Developer Go

Resources

- Discussion of security issues with CGI scripts http://www.w3.org/Security/Faq/www-security-faq.html
- CGI Perl scripts and modules http://www-genome.wi.mit.edu/WWW/tools/scripting/index.html
- Object-oriented Perl programming using CGI.pm
  http://www-genome.wi.mit.edu/ftp/pub/software/WWW/cgi_docs.html

Legacy language information files (lists various modern versions, including freeware versions):

- COBOL FAQs http://www.netcomuk.co.uk/~james/FAQ/cobol-faq.html
- FORTRAN FAQs http://www.fortran.com/fortran/FAQ/cont.html
- REXX information (including information on free ports of REXX)
  http://www.hursley.ibm.com/rexx/rexxfaq.htm
- Pascal FAQs http://funrsc.fairfield.edu/program/pascal/pascalfaq.html

---

About the author

Scott Locklin is a mad scientist working on Soft X-ray Fourier Transform Spectroscopy at LBNL's Advanced Light Source. His daily computer tasks include network and system administration, security issues, porting legacy code, parallel processing, and coding for real time operating systems. Reach Scott at scott.locklin@ne-dev.com.

What did you think of this article?

-Very worth reading	-Too long	-Too technical
-Worth reading	-Just right	-Just right
-Not worth reading	-Too short	-Not technical enough

Comments:

Name:

Email:

Company Name:

Send data

TABLE OF CONTENTS TOPICAL INDEX SEARCH

If you have problems with this magazine, contact webmaster@ne-dev.com

URL: http://www.ne-dev.com/ned-05-1998/ned-05-legacy.html
Last modified: Saturday, November 20, 1999