by Jeffreys Copeland and Haemer

MICHAEL AVETO

"All sorts of interesting things happened when programmers tried to [manually] copy subroutines…it therefore seemed sensible to have the computer copy the subroutines."
– Grace Murray Hopper on why she built the first compiler

"hello, world"
– Nearly every compiler writer since then

A Simple Web Script

If you haven't done any Web programming yet, but would like to try, this month's column is designed to help you get started. We'll create a Web page, a CGI script and a simple, LWP-based Web client. None of these will be sophisticated or difficult. On the way to making them, we'll show you how to make a simple front end for an arbitrary Web page. We'll start the usual way.

Hello, World

November 21, Haemer's birthday, is World Hello Day. "Hello, world" is a catchphrase that every programmer knows–a literary allusion to the very first page of *The C Programming Language*, by Brian W. Kernighan and Dennis M. Ritchie (published by Prentice-Hall Inc., now in its second edition, ISBN 0-13-110370-9), perhaps the single most influential book in programming:

1.1 Getting Started

The only way to learn a new programming language is by writing programs in it. The first program to write is the same for all languages:

Print the words

```
hello, world
```

This is the basic hurdle; to leap over it you have to be able to create the program text somewhere, compile it successfully, load it, run it, and find out where your output went. With these, mechanical details mastered, everything else is comparatively easy.

In C, the program to print "hello, world" is

```
main()
{
 printf("hello, world\n");
}
```

The line, "The first program to write is the same for all languages" wasn't true until K&R wrote it. Now, "hello, world" is a noun, an adjective and occasionally even a verb. We, for example, are about to show you how to "hello, world" in several new ways. There are many Web pages devoted to "hello, world" in hundreds of languages, for example, `http://www.latech.edu/~acm/HelloWorld.shtml`.

(We note that the other accepted way to demonstrate a programming language is to write the "99 bottles of beer" program. Examples of this in different languages are available at `http://www.ionet.net/~timtroyr/funhouse/beer.html`.)

We'll want a "hello, world" for each piece of our application: Web page, CGI script and Web client.

The Web Page

Here's the HTML for a "hello, world" Web page:

```
<HTML>
<HEAD><TITLE>Hello
        HTML</TITLE></HEAD>
<BODY>
<H1>hello, world</H1>
```

```
</BODY>
</HTML>
```

The hard part will be figuring out how to point your Web browser at it. You can start by putting it on your own machine:

```
$ cat << EOD > /tmp/hello
<HTML>
<HEAD><TITLE>Hello HTML</TITLE></HEAD>
<BODY>
<H1>hello, world</H1>
</BODY>
</HTML>
EOD
$ netscape file:/tmp/hello
```

Eventually, though, you'll need to figure out how to put it on the same machine as a Web server. This involves analogues to all the mechanical problems described in K&R. You'll need to at least do the following:

• Find a machine you have access to that's running a Web server. If you administer your own machine, you can install one. The most popular Web server is Apache (`http://www.apache.org`). More Web sites run the Apache server than all other Web servers combined, and its market share is growing. One thing that keeps this from an antitrust suit is that no one owns or sells it. It's open source and it's free. (One thing that keeps increasing Apache's market share, even on Windows NT-based machines, is that, unlike the corresponding Microsoft Corp. product, it's limited by hardware performance, not by the performance of the NT scheduler.)

• Figure out where to install it. Web servers don't let you get files from just any old place on the server's machine. That would be a tremendous security hole.

Although a URL like `http://woodcock/hello/hello.html` really is on `woodcock`, and the file is `hello.html`, the directory it's in really isn't `/hello/`. Instead, it's probably `/usr/local/etc/httpd/htdocs/hello/`. For better or for worse, you'll need to talk to the person who administers the server to find out where to put your page and what URL it will correspond to. This is analogous to writing a shell script and then installing it on your machine. Where you're allowed to install it will vary, and even if you install it in `/usr/local/bin`, you'll probably invoke it as `myscriptname` and not as `/usr/local/bin/myscriptname`. Moreover, invoking it without a full path doesn't mean there's a copy in your current working directory.

The analogy isn't exact because there's no analogy to the user-specified search path (a systems administrator can specify global search paths), but it gets the idea across.

• Figure out what the permissions and ownerships need to be. As with shell scripts, permissions and ownerships have to be set correctly. Here, too, these will vary with the installation. Sometimes, you may even need to conform to a naming convention, such as `foo.html`.

We warn you that getting through this list the first time can be annoyingly time-consuming. So was getting your first C program to compile and run, but it was so long ago that the mechanical difficulties have been forgotten. It may help to remind yourself that the HTML for the Web page is trivial; once you overcome the administrative hurdles, it's relatively smooth sailing. Asking a friend for help is probably the most painless way to proceed.

The CGI Script

Here's a "hello, world" CGI script:

```
#!/usr/local/bin/perl -w

print "Content-type: text/html\n\n";
print "hello, world\n";
```

A CGI script is a piece of code the server runs at the request of the client, which constructs a Web page on the fly.

When you use the AltaVista search engine to search for `+duoglide+music`, the URL it takes you to is `http://www.altavista.com/cgi-bin/query?pg=q&kl=XX&q=%2Bduoglide+%2Bmusic`. There is no page of HTML on the server called `query?pg=q&kl=XX&q=%2Bduoglide+%2Bmusic`; it is just a request to the server to run a program called `query`, with `pg=q&kl=XX&q=%2Bduoglide+%2Bmusic` as input (actually with a decoded version as command-line parameters, but we're trying to avoid that level of detail). The `query` program then queries its database, generates HTML on the fly and hands it back to the server, which returns it to you as though it were a static page of HTML sitting somewhere on its disk. Put another way, a CGI script isn't HTML, but it injects HTML into the output stream: "I'm not a Web page, but I play one on an HTTP port." This can be a little confusing at first.

The tiny protocol used for communicating between the server and the script is the CGI referred to in the name, the Common Gateway Interface.

Here, again, the really hard part will be finding out what machine you can put this on, where on the server it needs to be installed, what the permissions and ownerships need to be and whether the file needs to follow any particular naming conventions. One added piece of complexity is you need to make sure the script is syntactically legal. The message you see when a script won't run will be something like this:

```
The server encountered an internal error
or misconfiguration and was unable to
complete your request.
```

Not much of a clue. Here, frequent use of `perl -c` to precheck the syntax will be a big help.

The good news is the script is run by the server and lives on the server machine, just like a Web page. Because you've figured out how to install Web pages, getting the same information for your CGI scripts is easy. However, a caveat: A paranoid systems administrator may disallow CGI scripts because of security worries. We always encourage staying on good terms with your systems administrator.

This column won't be a tutorial on CGI, HTML, Web client programming or administering a Web server. For each of these topics, there are many books and online tutorials; some of them good. We will, however, spend a moment proselytizing for Lincoln Stein's CGI.pm.

The CGI program we showed you earlier handcrafts a CGI script. It's a pretty simple script, but CGI scripts quickly get more complicated. For anything much more complex, CGI.pm is the tool of choice. Here's a CGI.pm "hello, world":

```
#!/usr/local/bin/perl -Tw
use strict;
use CGI qw(:standard);

print header();
print start_html("Hello CGI");
print h1("hello, world");
print end_html();
```

When you install and run this, you'll notice that, unlike our earlier CGI script, the output looks like our original HTML page.

(We'll also pause to evangelize `use strict` and the flags `-T` and `-w`. These are safety features. If you're going to publish Web pages for the whole world to see, you really should take some serious precautions to ensure no one can shoot you in the foot.)

Examining Web Pages

So is the HTML that the CGI.pm generated really identical to the HTML we wrote by hand? Let's look.

If you're running Netscape, you can look at the raw HTML that your browser is interpreting by clicking on "View," and then "Page Source." Doing this for the hand-coded HTML page shows that the browser is seeing exactly what we wrote. Here's what the CGI-generated page reveals:

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML><HEAD><TITLE>Hello CGI</TITLE>
</HEAD><BODY><H1>hello, world</H1></BODY></HTML>
```

Browsers don't care about layout for HTML source, so the only differences are the first line, which is a comment (comments in HTML start with `<!`), and the different title, which we gave it on purpose.

Don't stop at just looking at the source in your browser. You can capture it as a file. Click on "File," then "Save As," and save the source ("Format for Saved Document: Source") as `file/tmp/cgi.output`. Then do this

```
$ netscape file:/tmp/cgi.output
```

This is an extremely good trick. See something on the Web you like? Capture the source, edit it with your favorite text editor until you get something that looks like what you want, using your browser to look at it while you make your edits, and save it. (If you don't like clicking the "Reload" button each time you want to see the results of your changes, see "We Use vi to Edit Web Pages," *SunExpert*, May 1997, Page 90.)

A Simple Web Front End

We can expand this idea of "new Web pages from old" into a twist on the popular UNIX pastime of front-ending commands. Let's suppose you frequently use a Web front end to a database, and the query page is more general-purpose and noisier than you would like. You now know enough to make a custom-tailored front end with very little work.

In our example, we'll use `http://www3.ncbi.nlm.nih.gov/Omim/searchomim.html`, the front end for searching articles published by Online Mendelian Inheritance in Man (OMIM), the National Center for Biotechnology Information's repository of information about human genetics.

The original OMIM Web page lets users limit their searches in a wide variety of ways. We would like an interface that searches the same database and still permits us to enter any keywords we choose, but limits our searches to the last seven days and returns all articles by default. While we're at it, we'd like to pare down the text on the screen and change the background color.

A little experimentation turns the original, 56-line form into the 19-line derivative shown in Listing 1.

Listing 1

```
1  <HTML>
2  <HEAD>
3  <TITLE>Updates from OMIM</TITLE>
4  </HEAD>
5  <body bgcolor="#EEFFFA">
6  <A HREF="/">
7  <img border=0 align=top
8    src="http://www3.ncbi.nlm.nih.gov/PMGifs/NCBI_logo.gif"
9    alt="NCBI GenBank"></A>
10 <HR>
11 <h3>Recent Articles from Online Mendelian Inheritance in Man</h3>
12 <FORM METHOD="POST"
13  ACTION="http://www3.ncbi.nlm.nih.gov/htbin-post/Omim/getmim">
14 Keyword(s):
15 <INPUT TYPE="text" VALUE="gene" NAME="search" SIZE=40>
16 <INPUT TYPE="hidden" NAME=search_time VALUE=7>
17 </FORM>
18 </BODY>
19 </HTML>
```

Line 3 changes the title to make it easier to find in our Forward list or bookmarks.

Line 5 changes the background color to the soothing "mint green" that David Siegel recommends (see `http://www.dsiegel.com/tips/wonk2/background.html`).

Line 11 changes the text to make the form say what it's really for, and line 14 reduces the number of words.

Lines 8 through 13 point to an image and a CGI script. In the original form, these were given as relative paths because they were on the same machine as the form. Here, they've been expanded to full URLs, to point to the original machine.

Listing 2

```
1  #!/usr/local/bin/perl -Tw
2  # $ID: query_omim,v 1.3 1999/02/01 00:10:26 jsh Exp $
3  use strict;

4  use HTTP::Request::Common qw(POST);
5  use LWP::UserAgent;

6  my $usage = "usage: $0 [keyword]\n";
7  my $keyword = @ARGV ? shift : "gene";

8  my $ua = LWP::UserAgent->new();
9  $ua->agent("Fezilla/v6.9 Irridium");
10 $ua->env_proxy();

11 my $ncbi = "http://www3.ncbi.nlm.nih.gov";
12 my $req = POST "$ncbi/htbin-post/Omim/getmim",
13  [ "search" => $keyword, "search_time" => 7 ];
14 my $resp = $ua->request($req);
15 die $resp->status_line(), "\n" if $resp->is_error();
16 (my $content = $resp->content()) =~ s(HREF=")($&$ncbi)g;
17 print $content
```

Line 15 is the data-entry field. We left it as is, but gave it a default value that is likely to be found in all entries: the keyword "gene." The rest of the options and choices have been eliminated.

Line 16 is an input field that the CGI script demands a value for, but we always want it to have the same value. Making it a hidden field and hard-coding its value achieves this end.

Conceptually, it's no different from aliasing `ls` to `ls -CF`, but it's nice to know that such tricks aren't confined to the command line.

Back to the Future: The Command Line

Speaking of the command line, if we're just interested in grabbing a bunch of data off the Web, why go through a form at all? Is there a way to write programs that would let us query Web databases from the command line without ever invoking a browser, say, like this?

```
$ query_omim fragile-X
```

Yes. This is a job for LWP, Gisle Aas' "Library for WWW access in Perl." Basically, LWP is a suite of modules that lets you create programs that act as tiny browsers and other sorts of user agents. The programs you write are typically special-purpose, but lack the enormous overhead of launching an 8-MB browser and the performance penalties that come from waiting while it retrieves and displays forms.

Listing 2 shows an example.

Lines 1 through 3 are boilerplate and lines 4 and 5 bring in the necessary modules. Lines 6 and 7 do argument processing; we've confined ourselves to a single search key because it's just an example.

Line 8 actually creates a browser "object." Pretty amazing. Line 9 names it, in case someone's collecting browser statistics at the other end. We could masquerade as Netscape, Lynx or some other browser, but why not be honest?

Line 10 gets the HTTP proxy information from the environment variable `$http_proxy`.

Line 11 contains the name of the site, to save a little typing. Lines 12 and 13 format a request, with the target address and the field names lifted right out of the source of the form we showed you earlier.

Line 15 sends the request and gathers the response as an object. Line 16 checks the exit status. If there was an error, it reports it. If not, we gather up the contents, discarding any HTTP header information, resolve the relative paths to absolute paths and print the result, which we can look at with our browser for further processing.

This should give you the basic idea.

This command-line utility doesn't need any special permissions, nor does it need to live in any special directories. It's just another Perl script.

Whew. That's a lot. If you had never done any Web programming before, you've now made a Web page and a CGI script, written a Web page replacement front end and a Web client. Not bad for one column. But not that difficult, either. Come to think of it, last month's math problem may have been harder. In fact, it may have been more difficult than we thought, see `http://cardit.et.tudelft.nl/~arlet/puzzles/putnam.html`. (The William Lowell Putnam Mathematical Competition is arguably the acme of national undergraduate math competitions.)

Then again, it may have been easier than we thought it was. Dr. James Flanagan of the University of Iowa Medical School, explained it to us by email:

Open lockers will be those numbered N, where N is an integer with an odd number of factors. Factors come in pairs, e.g.,

1 x N = N

A x B = N

so that 1, N, A and B are all factors of N. The only cases where there are an odd number of factors are those where N is a square of another integer, e.g., C x C = N.

So, the open lockers are all those whose number N is a square of another integer.

Until next time, happy trails.

Jeffrey Copeland (`copeland@alumni.caltech. edu`) *lives in Boulder, CO, and works at Softway Systems Inc. on UNIX internationalization. He spends his spare time rearing children, raising cats and being a thorn in the side of his local school board.*

Jeffrey S. Haemer (`jsh@usenix.org`) *works at QMS Inc. in Boulder, CO, building laser printer firmware. Before he worked for QMS, he operated his own consulting firm and did a lot of other things, like everyone else in the software industry.*

Note: The software from this and past Work columns is available at `http://alumni.caltech.edu/~copeland/work`.