CGI Programming

Phil Spector
Statistical Computing Facility
Department of Statistics
University of California, Berkeley

How a Web Browser Works

When a user enters a URL in a browser, the browser sends a request to a server for a particular resource, such as an HTML page or an image, and the server responds by sending some headers that describe what it's sending followed by the actual content.

If an HTML page contains images, flash animations, etc. the process is repeated for each item – the browser then displays everything in the appropriate way.

If the web server is configured to allow it, requests made to URLs in certain directories will run a program which generates the appropriate headers and content instead of just sending a static page. The mechanism that allows this is known as CGI (Common Gateway Interface), and such programs are known as CGI programs.

Browser to Server Communication

When a browser makes a request for a resource from a web server, what information is actually transmitted? We can use a simple program that sits will echo any information sent to it, and then point our browser at the program and see what happens.

In python, here's such a program, which I'll call `webecho`:

```
#!/usr/bin/python
import socket,os,sys

srvsocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
srvsocket.bind(("",1888))
srvsocket.listen(5)
while 1:
     clisocket,addr = srvsocket.accept()
     now = clisocket.recv(1024)
     clisocket.send(now)
     clisocket.close()
```

A Simple Request

With the `webecho` program running, let's make a simple request and see what happens. If I point a web browser at `http://localhost:1888/something`, here's what's displayed:

```
GET /something HTTP/1.1
Host: localhost:1888
User-Agent: Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.7.12)
            Gecko/20051010 Firefox/1.0.7 (Ubuntu package 1.0.7)
Accept: text/xml,application/xml,application/xhtml+xml,text/html;
        q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

HTML Forms

A variety of form elements are available through HTML, and are presented in an interactive learning tool at:

`http://www.w3schools.com/html/html_forms.asp`

While I'll show examples of some form elements, I suggest consulting the w3schools web page for additional details.

One important type of form element is the `type=hidden` form. When you have a multi-screen CGI program, using this type of form allows you to pass information between the different screens without the information being visible to the user.

Getting information to CGI Programs

What makes CGI programming interesting is that it can get information from users through various HTML form elements, like entry fields, drop-down menus, and file upload dialogs. How is that information transmitted to the server? The first step is producing a page that contains a form. Here's the html for a simple form that will talk to the webecho program:

```
<form action='http://localhost:1888' method=get>
<input type=text name=postvar><br>
<input type=submit value='GET'>
</form>
```

Here's what shows up in a web browser:

Getting information to CGI Programs (cont'd)

When the phrase "Hello, world" is entered in the form, and the submit button pressed, here's what appears in the browser window:

```
GET /?getvar=Hello%2C+world HTTP/1.1
Host: localhost:1888
User-Agent: Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.7.12)
            Gecko/20051010 Firefox/1.0.7 (Ubuntu package 1.0.7)
Accept: text/xml,application/xml,application/xhtml+xml,text/html;
        q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://springer/~s133ar/cform.html
```

Getting information to CGI Programs (cont'd)

The `GET` method sends information to the browser by adding name/value pairs, possibly separated by ampersands (`&`), to the URL after a question mark. These URLs are encoded to handle characters not allowed in URLs. URLs constructed this way will communicate with the web server regardless of the method used to send the information, and are the only way to invoke a CGI program without a surrounding form.

An alternative method of sending information to a web server that doesn't display the information in the web server's address bar is known as `POST`. Here's a form that uses this method:

```
<form action='http://localhost:1888' method=post>
<input type=text name=postvar><br>
<input type=submit value='POST'>
</form>
```

The POST method

Here's the result of submitting `Hello, world` through the form using the `POST` method:

```
POST / HTTP/1.1
Host: localhost:1888
User-Agent: Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.7.12)
            Gecko/20051010 Firefox/1.0.7 (Ubuntu package 1.0.7)
Accept: text/xml,application/xml,application/xhtml+xml,text/html;
        q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://springer/~s133ar/cform1.html
Content-Type: application/x-www-form-urlencoded
Content-Length: 22

postvar=Hello%2C+world
```

The POST method (cont'd)

The URL is no longer changed, and the name/value pair information is sent by the browser after it has sent the headers. The `POST` method can also be used for file uploads. Here's the HTML for a form which will accept a file name and upload it to the server:

```
<form action='http://localhost:1888' method=post
                enctype='multipart/form-data'>
<input type=file name='myfile'>
<input type=submit value='Upload'>
</form>
```

Here's how it looks in a browser (the Browse button is added automatically by the browser):

File Upload

Suppose I upload a small text file – here's what the browser sends to the server:

```
POST / HTTP/1.1
Host: localhost:1888
User-Agent: Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.7.12) Gecko/20051010 Firefox/1.0.7 (Ubuntu package 1.0.7)
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://springer/~s133ar/cform2.html
Content-Type: multipart/form-data; boundary=---------------------------174172229279482188344411240 6
Content-Length: 687

-----------------------------174172229279482188344411240 6
Content-Disposition: form-data; name="myfile"; filename="small.txt"
Content-Type: text/plain

When you type the address of a web page (i.e. a URL or Universal
Resource Locator) into a browser,or click a link which refers to
a URL, a request is made to a computer on the internet to send the
contents of a web page to your browser.  Web pages are written
in a language known as HTML (Hypertext Markup Language), and your
web browser knows how to translate HTML into text, pictures, links,
animations or whatever else the designer of the web page had in mind.

-----------------------------174172229279482188344411240 6--
```

File Upload (cont'd)

Uploading the file resulted in the addition of some headers, similar to those used to send attachments in email messages. The main difference is that even binary files can be transfered in this way, since the HTTP protocol does not disturb the eighth bit of its input the way that SMTP does.

Now that we've seen how information is transmitted from the browser to the server, let's consider the communication in the opposite direction.

Server to Browser Communication

The underlying principle behind CGI programs is that the standard output of your CGI program is sent to the browser, so your CGI programs must generate HTML to produce your desired output. But just as we saw that the browser inserts some headers before it sends information (if there is any) to the server, the server needs to send at least one header line to the browser, to let it know that HTML will follow. Thus, the first thing that any CGI program should do is to output a header like this:

```
Content-type: text/html
```

It's also the CGI program's responsibility to send a completely empty line to signal the end of the headers.

Cookies

Cookies are text stored on a remote computer when they access your CGI program. Cookies can serve as an alternative to hidden variables, or as way to remember users when they access your CGI program again.

Cookies are sent through outgoing headers as follows:

```
Set-Cookie: cookiename=cookievalue
```

This produces a cookie that expires when the user closes their browser.

To make persistent cookies, provide an expiration date as follows after the above specification:

```
; expires=Monday, 01-May-06 00:00:00 GMT
```

The cookie, sent as a name/value pair, will be passed to your CGI program automatically whenever a browser storing your cookie returns to your HTML page or CGI program.

The CGI Standard

The CGI standard insures that information is properly transfered between the browser and the web server, and between the web server and the browser, by running CGI programs in an appropriate environment.

1. Headers from the browser are converted to environmental variables.
2. Information from the browser that was after the headers (if any) is placed in standard input for the CGI program.
3. Standard output from the CGI program is directed to the web browser.

Each language that is appropriate for CGI scripting will provide tools to make this information available to your CGI program in a convenient form.

What do CGI Interfaces Provide?

The most important feature that a CGI interface provides is a means of getting the values of the CGI form variables into the programming environment. All programs with a CGI interface provide a way of getting this information regardless of the method (`GET` or `POST`) that was used.

For file uploads, there should an option to avoid reading the entire file into memory.

Since additional information is provided through environmental variables, there should be a simple way of accessing these variables.

Since standard output from the CGI program is directed to the browser, many CGI interfaces provide helper functions to generate HTML, as well as convenience functions to generate the required header lines, although it's often easier to do this by yourself.

Some Useful CGI Environmental Variables

Name	Contents
`HTTP_COOKIE`	Persistent data stored in cookies
`HTTP_REFERER`	URL of refering document
`HTTP_USER_AGENT`	Type of browser being used
`QUERY_STRING`	URL Fragment after `?`
`REMOTE_ADDR`	IP Address of user
`REMOTE_HOST`	Hostname of user
`REMOTE_USER`	Username, if authentication was used
`SERVER_NAME`	Hostname of server
`SERVER_PORT`	Port number of server
`SERVER_SOFTWARE`	Name and version of server software

Security of CGI Programs

Remember that CGI programs are accessible to anyone who can access the web server that hosts the program, so extra care is necessary to make sure your programs are secure.

The cardinal rule is to make sure that any user input that gets passed to the operating system does not contain any unusual characters.

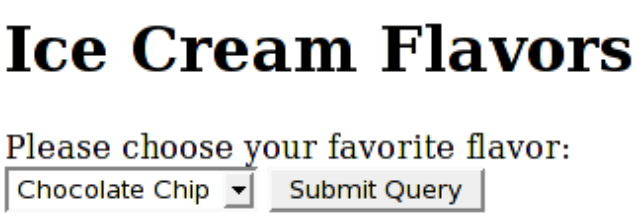
In fact, it may be wise to avoid calling any operating system commands that include user input.

It is not uncommon for CGI programs to set their command path (through environmental variables) to one limited to just those directories necessary for execution of the CGI program. On a UNIX system, an example would be a path of `/bin:/usr/bin`.

A Simple Example

A form to allow the user to choose an ice cream flavor is generated by the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<head title='Pick a Flavor'>
<html><body>
<h1>Ice Cream Flavors</h1>
Please choose your favorite flavor:<br>
<form action='http://localhost/~spector/cgi-bin/icecream.py' method='post'>
<select name="flavor" >
<option value="Chocolate Chip">Chocolate Chip</option>
<option value="Strawberry">Strawberry</option>
<option value="Rum Raisin">Rum Raisin</option>
<option value="Vanilla">Vanilla</option>
</select>
<input type='submit' name='Submit'>
</form>
</body></html>
```



Processing the Form: Perl

Here's a perl script to process the ice cream form:

```
#!/usr/bin/perl

use CGI;

$q = new CGI();
$flavor = $q->param('flavor');

print $q->header();
print $q->start_html('Ice Cream Example');
print $q->h1('Ice Cream Flavor');
print "$flavor is a good choice, I like it too";
print $q->end_html();
```

Processing the Form: Python

```
#!/usr/bin/python

import cgi

f = cgi.FieldStorage()
flavor = f['flavor'].value

print 'Content-type: text/html\n\n'
print '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">'
print '<head title="Ice Cream Example">'
print '<html><body>'
print '<h1>Ice Cream Flavors</h1>'
print '%s is a good choice, I like it too' % flavor
print '</body></html>'
```

Combo Forms

In the previous example, the proper action had to be coded into the HTML file that presented the form. In general, separating the form from the code may make it more difficult to maintain your program. In addition, it's often useful to generate HTML forms through a program, instead of hardcoding the form.

Combo forms are programs that first check to see if form data has been received – if not, they generate the necessary form, refering back to themselves as the action. They are especially handy for multi-form transactions, since they put all of the programs in a single file, and eliminate the need for separate static HTML pages.

Combo Form in Perl

```
#!/usr/bin/perl
use CGI;

$q = new CGI();

@flavors = ('Chocolate Chip','Strawberry','Rum Raisin','Vanilla');

if (not defined $q->param()){    # called directly
   print $q->header(),
         $q->start_html('Ice Cream Example'),
         $q->h1('Ice Cream Flavors');
   print $q->startform({action=>'icecream1.pl',method=>'POST'}),
         $q->popup_menu(-name=>'flavor',-values=>\@flavors),
         $q->submit(),
         $q->end_form(),$q->end_html();
}
else{

   $flavor = $q->param('flavor');

   print $q->header();
   print $q->start_html('Ice Cream Example');
   print $q->h1('Ice Cream Flavor');
   print "$flavor is a good choice, I like it too";
   print $q->end_html();
}
```

Combo Form in Python

```
#!/usr/bin/python
import cgi

f = cgi.FieldStorage()

print 'Content-type: text/html\n\n'
print '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">'

flavors = ('Chocolate Chip','Strawberry','Rum Raisin','Vanilla')

if len(f.keys()) == 0:   # called directly
        print '''<head><title>Pick a Flavor</title></head>
<html><body>
<h1>Ice Cream Flavors</h1>
Please choose your favorite flavor:<br>
<form action='icecream1.py' method='post'>
<select name="flavor" > '''
        for f in flavors:
              print '<option value="%s">%s</option>' % (f,f)
        print '''</select>
<input type='submit' name='Submit'>
</form>
</body></html>'''
else:
        flavor = f['flavor'].value
        print '''<head><title>Ice Cream Example</title></head>
<html><body>
<h1>Ice Cream Flavors</h1>
%s is a good choice, I like it too''' % flavor
        print '</body></html>'
```

Debugging CGI Scripts

1. Make sure the `Content-type` header line is being generated.
2. Make sure that the first line of the CGI program indicates the location of the executable that will run the program.
3. Make sure that the script is executable, and any files which it accesses have appropriate permissions for the user under which the CGI program will run.
4. Execute the script from the command line to find syntax errors
5. Errors are usually redirected to the server's error log, which may not be accessible. Redirecting standard error to standard output will display messages in the browser.
6. There may be special debugging facilities in the language of your choice.