A Brief Look at CGI with Perl

CGI (Common Gateway Interface) lets HTTP clients interact with programs across a network through a workstation.

CGI is a standard for interfacing applications with a Web server.

CGI applications can be written in many languages.

But Perl is often preferred because of

- its text-processing capabilities and
- its simple approach to communicating with files and processes.

A Web browser takes information from a user (usually via a HTML form) and sends it (via HTTP) to a Web server.

A server-side CGI program is then executed and sends information back

This information is typically an HTML Web page but could be other things, such as images.

The standard output from server-side applications or scripts is redirected (piped) to the CGI then sent over the Internet to a Web browser for rendering.

Because CGI is an interface, it can’t be directly programmed – a script must be used to interact with it.

When a server receives a request for a CGI program, it executes that file rather than returning it.

The `method` attribute of a `form` tag specifies the technique (`GET` or `POST`) for passing data to the server.

`GET` is the default.

Both techniques code the form data into a text string – called a *query string* – when the submit button is clicked.

With the `GET` method, the browser appends a `?` then the query string to the end of the CGI program's URL.

A disadvantage is that some servers truncate characters in the URL string beyond a certain limit.

The `POST` method passes the query string through the standard input (STDIN) of the CGI program.

The name-value pair of each form element is coded as an assignment in the query string.

These assignments are separated by `&`'s.

E.g.,

```
name=Albert&payment=visa
```

A special character is coded as a `%` followed by the two-character hex representation of its ASCII code.

E.g., a space is coded as `%20` (ASCII 32).

Some browsers replace a space with a +.

So, instead of

```
name=Albert%20Esterline
```

we might see

```
name=Albert+Esterline
```

We can inspect the query string using raw Perl, without the CGI module.

(Generally, novice CGI programmers should use the CGI module or an equivalent whenever possible.)

The value of the environment variable `REQUEST_METHOD` is either `GET` or `POST`, depending on how the program was requested.

If it's `GET`, then the server moves the query string from the URL string to the environment variable `QUERY_STRING`.

If it's `POST`, the `read` function can be used to read the query string from STDIN.

Environment variable `CONTENT_LENGTH` is the number of bytes to be read.

```
$request_method = $ENV{'REQUEST_METHOD'};
if ($request_method eq "GET") {
    $query_string = $ENV{'QUERY_STRING'};
}
elsif ($request_method eq "POST") {
    read(STDIN, $query_string,
         $ENV{'CONTENT_LENGTH'});
}
else {
    print "Error - request method is illegal \n";
}
```

We next convert the query string to its separate name-value assignments:

```
@name_value_pairs = split(/&/, $query_string);
```

We can loop over these assignments with

```
foreach $name_value (@name_value_pairs) {
…
}
```

Within the loop, we split the assignment into a two-element list:

```
($name, $value) = split (/=/, $name_value);
```

It remains

- to convert `+`'s into spaces and
- to convert hex-coded special characters to their decimal ASCII codes.

We use the translate operator to convert `+`'s to spaces:

```
$value =~ tr/+/ /;
```

Converting hex-coded characters is much more difficult.

We need a pattern matcher to find hex-coded characters.

The following character class matches any hex digit (`0-9`, `A-F` or `a-f`):

```
[\dA-Fa-f]
```

We want to match – and remember the hex part, so surround it with `(…)`’s) – patterns of the form `%xx`, where each `x` is a hex digit.

So the pattern matcher is

```
/%([\dA-Fa-f][\dA-Fa-f])/
```

The replacement we want for a string matching this pattern can be constructed using the `pack` function:

```
pack("C", hex($1))
```

The `hex` operator is needed so that the matched string is interpreted as a hex number.

The `C` code indicates a (single-byte) decimal ASCII code.

Since we want to execute `pack("C", hex($1))` before performing the substitution, we need a modifier `e` after the substitution.

Since we want this substitution to apply to every hex-coded character in the string, we also need the global modifier, `g`.

So the statement to convert hex-coded characters is

```
$value =~
  s/%([\dA-Fa-f][\dA-Fa-f])/pack("C", hex($1))/eg;
```

Example:

The following HTML document has a form with text fields with names "`name`", "`street`", and "`city`".

It has four radio buttons, all with name "payment" but with four different values ("`visa`", "`mc`", "`discover`", and "`check`").

When the submit button is clicked, the query string is sent to the program `query_string.pl`, which is listed next.

This uses the techniques just described to decode the query string.

```
<html>
<head>
<title> Sales CGI program </title>
</head>
<body>

<form action = "/cgi-bin/query_string.pl"
      method = "POST">

<p> Buyer's Name:
    <input type = "text"  name = "name"
           size = "30"> </p>
<p> Street Address:
    <input type = "text"  name = "street"
           size = "30"> </p>
<p> City, State, Zip:
    <input type = "text"  name = "city"
           size = "30"> </p>

<p> Payment Method: <br/>
<input type = "radio"  name = "payment"
       value = "visa" checked> Visa <br/>
<input type = "radio"  name = "payment"
       value = "mc"> Master Card <br/>
<input type = "radio"  name = "payment"
       value = "discover"> Discover <br/>
<input type = "radio"  name = "payment"
       value = "check"> Check <br/> <br/>
</p>

<input type = "submit"  value = "Submit Order">
<input type = "reset"  value = "Clear Order Form">

</form>
</body>
</html>
```

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

# This is query_string.pl
# It is a CGI program to decode and display the form
# data from the popcorn Web document

# First produce the header of the HTML return value

print "Content-type: text/html\n\n";
print "<html><head>\n";
print "<title> Display query string data”;
print “</title></head> \n";
print "<body>\n";

# Determine the request method, get the query string

$request_method = $ENV{'REQUEST_METHOD'};
if ($request_method eq "GET") {
    $query_string = $ENV{'QUERY_STRING'};
}
elsif ($request_method eq "POST") {
    read(STDIN, $query_string,
         $ENV{'CONTENT_LENGTH'});
}
else {
    print "Error - request method is illegal \n";
}

# Split the query string into the name/value pairs

@name_value_pairs = split(/&/, $query_string);
```

Continued next page

Continued from previous page

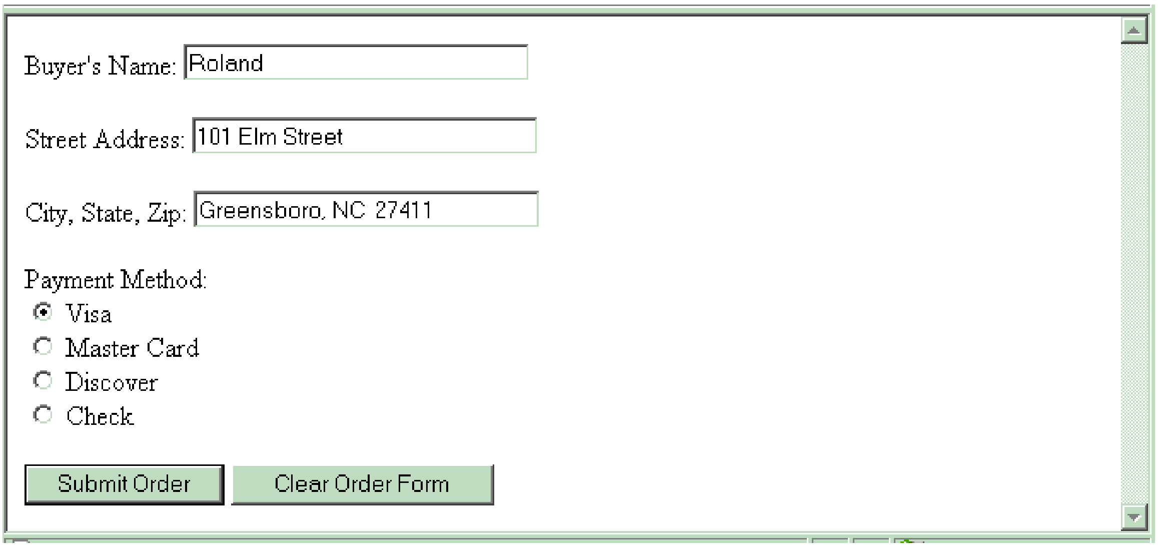
```
# Split the pairs into names and values and translate
# the values into text (decode hex characters
# and translate +'s to spaces)

foreach $name_value (@name_value_pairs) {
    ($name, $value) = split (/=/, $name_value);
    $value =~ tr/+/ /;
    $value =~
     s/%([\dA-Fa-f][\dA-Fa-f])/pack("C", hex($1))/eg;
    print "The next name/value pair is: “;
    print “$name, $value <br>\n";
}

# Generate the trailer HTML

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

The following shows the browser window with the form filled out.



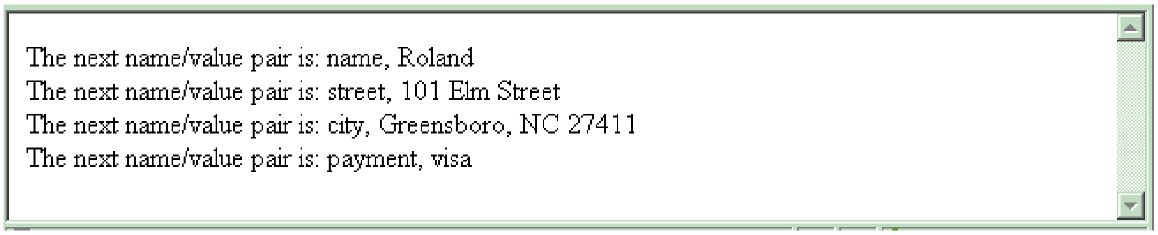
Buyer's Name: Roland

Street Address: 101 Elm Street

City, State, Zip: Greensboro, NC 27411

Payment Method:
[✓ Visa]
Master Card
Discover
Check

The following is the browser’s rendering of what is sent to it when this form is submitted.



```
The next name/value pair is: name, Roland
The next name/value pair is: street, 101 Elm Street
The next name/value pair is: city, Greensboro, NC 27411
The next name/value pair is: payment, visa
```

We typically start a Perl CGI program with the two lines

```
#!perl
use CGI qw/:standard/;
```

The first line causes the text file with the Perl program to be executed by the Perl interpreter.

The `use` statement allows Perl programs to include libraries.

The CGI library contains keywords that represent HTML tags.

It also causes the output of `print` statements to be redirected to the client.

We also include the following before any other print statements:

```
print header;
```

This uses the header function from the CGI library, and includes the following in the header:

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

This indicates that the browser must display the returned information as a HTML document.

Form Processing and Business Logic

`form` elements let users enter data that’s sent to a Web server for processing.

In the `form` tag, we use `method = “POST”` and set the value of the `action` attribute to the name of the Perl program that processes the data.

If a form element has `name` attribute value name, then the value for this element can be accessed in the target Perl CGI program with

```
param(name)
```

Example:

```
<html>
<head>
<title>Simple</title>
</head>

<body>
<form method = "POST"
      action = "/cgi-bin/simpleform.pl">
 <p>First name:
   <input type = "text" name = "firstn"></p>
 <p>Last name:
   <input type = "text" name = "lastn"></P>
 <input type = submit>
</form>
</body>
</html>
```

```
#!perl

use CGI qw/:standard/;

$fname = param(firstn);
$lname = param(lastn);

print header;

print "Hello $fname $lname!\n";
```

```
# data.pl

use Win32::ODBC;
use CGI qw/:standard/;

my $querystring = param(QUERY);
$DSN = "Products";

print header;

if (!($Data = new Win32::ODBC($DSN)))
{
   print "Error connecting to $DSN\n";
   print "Error: " . Win32::ODBC::Error() . "\n";
   exit;
}

if ($Data->Sql($querystring))
{
   print "SQL failed.\n";
   print "Error: " . $Data->Error() . "\n";
   $Data->Close();
   exit;
}

print "<BODY>";
print "<P> Search Results </P>";

$counter = 0;

print "<TABLE>";
```

Continued next page

Continued from previous page

```
while($Data->FetchRow())
{

   %Data = $Data->DataHash();
   @key_entries = keys(%Data);

   print "<TR>";

   foreach $key( keys( %Data ) )
   {
      print "<TD>$Data{$key}</TD>";
   }
   print "</TR>";
   $counter++;
}
print "</TABLE>";
print "<BR>Your search yielded”,
      “<B>$counter</B> results.";
print end_html;

$Data->Close();
```