Unit 9 – Using Perl For CGI Programming

The Common Gateway Interface

- XHTML is a markup language and ***does not really allow for interactive computing*** (beyond loading additional pages).
- Web require computation and database access.
- This requires interaction with the server.
- Interaction with the server requires a set of protocols to govern it.
- **Common Gateway Interface** (**CGI**) is an example of such a protocol.
- CGI, which stands for Common Gateway Interface, is not a programming language but a protocol.

CGI and LAMP

- **_LAMP_** is an acronym for a **solution stack** (short for ***L***inux, ***A***pache, ***M***ySQL and **P** for ***P***erl, ***P***HP or ***P***ython) principal components to build a viable general purpose **web server**.
- CGI is part of a popular approach to server side computation frequently called LAMP
- There are two other common approaches:
  - **ASP** (***A***ctive ***S***erver ***P***ages), where XHTML is combined with programs or links to external programs which are compiled into classes.
  - **Java servlets** and **JSP** (***J***ava ***S***erver ***P***ages), which are similar to ASP and somewhat analogous to Java applets.

Running CGI

- An HTTP request to run a CGI program is made, identifying a file on the server as a CGI program. They are so indentified by their addresses or file name extensions.
- Instead of returning the file, it is executed on the server, returning output to the requesting client.
- In most cases, the output is an XHTML document, with the server writing most of the header.

- Interaction is commonly through a form.

Running Perl CGI Programs

- Running a Perl CGI program requires that perl, the Perl interpreter be called. For UNIX or Linux, that requires the line

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

  be include at the beginning of the file. (This isn't necessary for Windows-based servers.)
- The program can be called as a hypertext link, or as a side-effect of a **`Submit`** button.

1. Explain how HTML pages are created dynamically using CGI.

Calling a Perl CGI Program

- An XHTML document can call a Perl CGI program by including

```
<a href = "reply.cgi">
    Click here to run the CGI program reply.cgi
</a>
```

- A form can invoke a Perl CGI program by giving its address in the **`action`** attribute of the **`form`** tag:

```
<form action = "./cgi-bin/popcorn.cgi"
    method = "post" >
```

reply.html

```
<!-- to call a simple Perl CGI program -->

    <?xml version = "1.0" encoding = "utf-8"?>
    <!DOCTYPE html PUBLIC "-//w3c//DTD XHTML 1.1//EN"
       "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns = "http://www.w3.org/1999/xhtml">
    <head> <title> XHTML to call the Perl CGI program, reply.cgi </title>
    </head>
    <body>
        <p> This is our first Perl CGI example
            <br /> <br />
            <a href = "reply.cgi">
                Click here to run the CGI program, reply.cgi
            </a>
        </p>
    </body>
</html>
```

reply.cgi

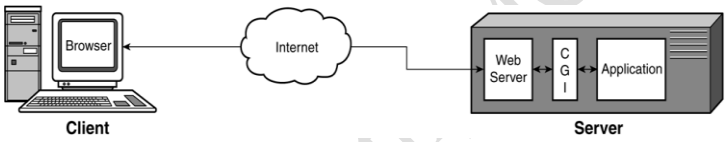
```
#!/usr/bin/perl -w

# reply.cgi - This CGI program returns a greeting to the client

print "Content-type: text/html \n\n",

print "<?xml version = '1.0' encoding - 'utf-8'?> \n" ;
print "<!DOCTYPE html PUBLIC '-//w3c//DTD XHTML 1.1//EN' \n" ;
print "'http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd'>\n" ;
print " <html xmlns = 'http://www.w3.org/1999/xhtml'> \n" ;
print "<head><title> reply.cgi example </title></head>\n" ;
print "<body>\n" ;
print "<h1>Greetings from your Web server! </h1>\n" ;
print "</body></html>";
```

What is query string? Explain how query string is transmitted to the server with get and post method. --8

What is CGI? How parsing of data is done for GET and POST methods. --10

CGI – Definition:

CGI is a "Common Gateway Interface" for communication between the web server and CGI scripts that can be run on the server.



CGI programs (or scripts) are commonly written in
PERL Script, Shell Script, C or C++ program

Query Strings

- A string query is a string containing data associated with a request to the Web server.
- The request is typically made using **GET** and **POST** methods.
- query strings can be handwritten and used directly with **GET** and **POST**
- Mainly form data and are constructed by the browser when the ***Submit button is clicked.***

Query Strings Format

- For every element (widget) with a value, the name and value is coded as a character string in the format of '***name = value***' and included in the query string:

```
payment = discover
```

- If there is more than one widget, they are separated by ampersands (**&**)

```
Caramel=7&payment=discover
```

▲

- Special characters can be coded using a percent sign and their ASCII value (**%20** for ***space*** and **%21** for **!**)

Payment=visa & saying=Eat %20your %20fruit % 21

`get` Method

- ***The action of a form is always get or post, with `get` being the default.***
- When **`get`** is used, the browser attaches the query string to the **URL** of the CGI program, so the form data is transmitted to the server with the URL.
- The server removes the query string from URL and places it in the environmental variable **QUERY_STRING**, where program can access it.
  - **`get`** also be used to pass parameters to server when there is no form.
  - **Disadvantage**- some servers limit the length of URL and will truncate.

`post` Method

- The post method passes the query string ***through standard input*** to the CGI program so the CGI programs just reads it as STDIN.

- There is no length limitation for the query string with the post methods, so it's a better choice if there are several widgets.

Example: Placing Data in 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 = the request method is illegal \n";
  exit(1);
}
```

What is the perpose of the shortcuts in CGI.PM? --06

Purpose:

- With the Perl CGI.pm module, ***you can create dynamic Web pages quickly and easily*** from within the powerful Perl programming language.
- CGI.pm contain functions to perform common tasks like writing commonly used HTML tags and for creating form elements such as buttons and lists.
- Following declaration gives access to a particular module of CGI.pm

```
use CGI ":standard";
```

Common **CGI.pm** Functions

br

– returns a break tag **
**

```
print br;          #places a break in the XHTML document.
```

h1

- places the string in a largest size header
- Example `print h1("This is the real stuff");`

–creates the XHTML code:

```
<h1> This is the real stuff </h1>
```

textarea

- places a **textarea** in the HXTML with the attributes passed as a **hash** literal
- Example:

```
print textarea( -name => "Description",
                - rows => "2",
                -cols => "35" );
```

produces:

```
<textarea name = "Description" rows = "2" cols = "35" >
</textarea>
```

a

- creates a hypertext link or another anchor

```
print ( a { -href => "fruit.html" },
          "Press here for fruit descriptions" );
```

produces

```
<a href = "fruit.html" >
    Press here for fruit descriptions
</a>
```

ol and li

- **ol** – creates an ordered list:

```
print ol ( li ( {-type =>"square"}, ["milk", "bread", "cheese"] ) );
```

produces

```
<ol>
    <li type="square"> milk </li>
    <li type="square"> bread </li>
    <li type="square"> cheese </li>
</ol>
```

radio_group

– sets up an entire radio group

```
print radio_group ( -name=>"colors",
                    -value => ['blue', 'green", 'yellow', 'red'],
                    -default => 'blue');
```

produces

```
<input type="radio" name = "colors" value="blue" checked /> blue
<input type="radio" name = "colors" value="green" /> green
<input type="radio" name = "colors" value="yellow" /> yellow
<input type="radio" name = "colors" value="red" /> red
-----------------------------------
```

start_html

- produces the boilerplate for the opening HTML;

```
print start_html("Paul's Gardening Supplies");
```

produces

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional //EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"lang="en-US" xml:lang="en-US">
<head><title> Paul's Gardening Supplies </title>
<meta http-equiv="Content-Type" content="text/html"; charset=iso-8859-1">
</head>
<body>
```

header

- produces boilerplate for the beginning of an XHTML document

```
print header
```

produces

```
Content-type: text/html; charset=ISO-8859-1
blank line
```

param

- allows the program separate name/value pairs that appear in a query string.
- If the query string is

```
name=Bob%20Brumbel&payment=discover
```

– The statement

```
my $myName = param("name")
```

assigns the variable `$myName` the value `"Bob Brumbel"`

– The statement

```
my ($myName, $salary) = ( param("name"),
                          param("payment") );
```

would assign `"Bob Brumbel"` to `$myName` and `"discover"` to `$salary`

```
end_html
```

- produces closing tags for an XHTML document

```
print end_html;
```

Produces :

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

What is Cookie? Where is it stored? Explain how to create and delete a cookie. –10

What is Cookie? Where is it stored? What is form of the value of a cookie? --06

What is Cookie? Explain creating, reading and deleting cookie. --10

Why need Cookies

- **A session** is the time span during which a browser interacts with a specific server.
- **Hypertext Transfer Protocol (HTTP)** ***stores no information*** from a session that the next session can use, despite the fact that this information may be useful.
- ***Cookies are a useful of saving information about a session***.
- The server is given this information when the browser makes subsequent requests of the same server.

What Is A Cookie?

- ***Definition:***
  - ***A cookie is a small object of information used to maintain state information between the web server and the client browser..***
- ***Cookies can be passed from browser to server or from server to browser*** in an HTTP message header.
- ***Cookies are assigned a lifetime***; when it reaches the end of that lifetime, it is deleted from the client machine.
- Every browser includes all the cookies that the host machine has stored that are associated with the server.
- ***Cookies are stored as text*** and can be deleted by the host machine.

Creating Cookies

- `CGI.pm` supports cookies through the `cookie` function.
- The cookie function can create cookies:

```
cookie( -name => aCookieName,
        -value => aValue
        -expires => aLifeTimeValue );
```

- **Calling `cookie` without parameters *will return a hash*** with all the cookies in the HTTP header of the current request.

Returning Specific Cookies

- To retrieve a specific cookie, the name of that cookie is given as a parameter:

```
$age = cookie('age');
```

- To display all the cookies (both names and values), we could write:

```
print "Cookie Name \t Cookie Value <br />;
foreach $name( keys cookie() )
{
        print "$name \t cookie($name) <br />";
}
```

Write perl program which uses built-in function to get current system time and date and formats that information and print on screen. --10

Getting the Time

- If we wanted the date and time, we could use **time** to provide the seconds since January 1, 1970.
- The function **localtime** calls **time** and returns 9 values:

```
($sec, $min, $hour, $mday, $mon, $year, $wday,
     $yday, $iddat) = localtime;
```

  - **mday** is the day of the month
  - **mon** is the month (**0** to **11**)
  - **year** is the year since 1900
  - **wday** is the day of the week (**0** to 6)
  - **yday** is the day of the year
  - **isdst** is a Boolean telling whether this is Daylight Saving Time

time_date.pl

```
# Input: None
# Output: The nine values return by localtime

 ($sec, $min, $hour, $mday, $mon, $year, $wday,
             $yday, $iddat) = localtime;

print "seconds = $sec\n";
print "minutes = $min\n";
print "hour = $hours\n";
print "day_of_the_month = $mday\n";
print "month = $mon\n";
print "year = $year\n";
print "day_of_week = $wday\n";
print "day_of_year = $yday\n";
print "daylight_Saving = $isdst\n";
```

Getting the Day of the Week

```
$day_of_week =
  ( qw( Sunday Monday  Tuesday Wednesay Thursday
        Friday Saturday ) ) [ (localtime)[6] ];
```

Creating A Greeting For Visitors

1. Get the cookie named **last_time**
2. Get the current day of the week, month, day of the month and put them in a cookie called last_time
3. Put the cookie in the header of the return document
4. If there is no existing cookie, produce a welcome message for the first-time visitor
5. If there was a cookie, produce a welcome message that includes the previous visit's day of the week, month and day of the month

day_cookie.pl

```
#!/usr/bin/perl -w
# A CGI-Perl program to use a cookie to remember the day
# of the last login from a user and display it when run
use CGI ":standard";
@last_day = cookie('last_time');
            #>>> Get the existing day cookie, if there was one

$weekDay =  ( qw ( Sunday Monday Tuesday Wednesday
  Thursday Friday Saturday)) [ (localtime) [6] ] ;
            #>>> Get the day of a week

$month= ( qw (January February March April May June  July August
  September October November December)) [ (localtime) [4] ];
            #>>> Get the month

$day_of_month = (localtime)[3];
            #>>> Get the day of a month

@day_stuff = ($day_of_week, $day_of_month, $month);

#>>>  make the new cookie
$day_cookie = cookie(   -name => 'last_time',
                        -value => \@day_stuff,
                        -expires => '+5d'
                    );
```

```
#>>> start Producing the return document, First, put the cookie in the new header
print header (-cookie => $day_cookie);

print start_html('This is day_cookie.pl');

#>>> If there was no day cookie, this is the first visit
if (scalar(@last_day) == 0)
{
  print "Welcome to you on your first visit to our
  site <br />";
}   # Otherwise, welcome the uyser back and give the date of the last visit
else
{
  ($day_of_week, $day_of_month, $month) = @last_day;
  print "Welcome back! <br />"
  print  "Your last visit was on "
  print  "$day_of_week, $month $day_of_month<br />";
}
print end_html;
```