CGI NOTES

CGI stands for Common Gateway Interface. All web servers have CGI capabilities. The CGI is the means for web pages to communicate with the server directly, with power to read and rewrite information on the server hard drive, whether in a database or in a more ordinary directory or file.

CGI scripts can be written in any programming language, including C, C++, Java, Perl and PHP.

Perl and PHP preferred.

Security: CGI scripts have the power to rewrite or erase a web server's hard drive,

bin: Web servers have something called the "cgi-bin", which is simply a directory on the server which contains all of the CGI scripts for that server. These CGI scripts may be accessed if you know the URL for the script and its "cgi-bin" directory.

CGI scripts usually come with one of two dot-extensions on the file name, ".cgi" (for a generic CGI script) and ".pl" (for a Perl script specifically).

Static vs Dynamic HTML

 A plain HTML document that the Web daemon retrieves is static, which means it exists in a constant state: a text file that doesn't change. A CGI program, on the other hand, is executed in real-time, so that it can output dynamic information.

CGI – creates a gateway that can allow people to obtain html pages on your os

Key: CGI program should not take too long to process

How to connect to a script:

Use a FORM tag

Two attributes: METHOD and ACTION

Attribute: **METHOD**  
Value: **get** (default) or **post**  
Description: this attribute defines the manner in which the form information is conveyed to the script.

Attribute: **ACTION**  
Value: a URL (to the CGI script in the cgi-bin in question)  
Description: the ACTION attribute of the FORM tag connects that form to a CGI script; when the submit button for the form is pressed, the web browser seeks out the CGI script at the URL indicated by the value of ACTION and causes the web server to execute that CGI script.

<form method="post" action="myScript.pl">

<!-- one or more form elements, with their accompanying HTML context -->

<!-- a submit button -->

</form>

BOTH method AND action attributes are REQUIRED to connect from a form to CGI script

Perl

Practical Extraction and Report Language

It's original purpose was to monitor large software projects and generate reports. It was initially developed on Unix.

most of the information that users send to servers is text, such as usernames, passwords, and email addresses.

Rules:

All statements must end with semi-colon

perl is case-sensitive

first line: #!/user/local/bin/perl This tells hills to execute the program that follows use Perl (this line is machine dependent)

Comments: Use # for comments

Variables: start with $ example: $in { ‘choice’ }

Print Statement:

print “<H2> dd </h2”;

print <<”\_END\_”; #output the following info until the statement \_END\_; is reached

Debugging:

note: cgiwrap – allows ordinary users to run their own CGI scripts a gateway that allows more secure user access to CGI programs on an HTTPd server than is provided by the http server itself.

Goal: make certain that any CGI script runs with the permissions of the user who installed it and not those of the server

 /cgi-bin/cgiwrap/ (or phpwrap) for urls ending in ".php" or ".pl".

How to run perl interpreter:

 Go to your cgi-bin subdirectory by entering:   
  
cd public\_html/cgi-bin  
perl testerr4.pl (where testerr4.pl is the name of your Perl script)  
  
You should now be able to locate the error in your perl script. In this example, the Perl script had

PHP:

It's official name is PHP: Hypertext Preprocessor, and it is a server-side scripting language. When your Web browser accesses a URL, it is making a request to a web server. When you request a PHP page, for example, http://fog.ccsf.edu/~srubin/formA.php, the Web server wakes up the PHP parsing engine and says, "Hey! You've got to do something before I send a result back to this person's Web browser."  
  
Then the PHP parsing engine runs through the PHP code found in formA.php, and returns the resulting output. This output is passed back to the Web server as part of the HTML code in the document, which in turn is passed to your browser, which displays it to you.

What does PHP do?

Here are some common uses of PHP:

Perform system functions: create, open, read from, write to, and close files on your system; execute system commands; create directories; and modify permissions.

Gather data from forms: save the data to a file, send data via e-mail, return manipulated data to the user.

Access databases and generate content on-the-fly, or create a web interface for adding, deleting, and modifying elements within your database.

Set cookies and access cookie variables.

Start sessions and use session variables and objects.

Use PHP user authentication to restrict access to sections of your web page.

Create images on-the-fly.

Encrypt Data.

HTML + PHP

<form action="formA.php" method="post">

Regarding the form statement, notice that the action's value does not refer to your php directory, because the server will know to automatically look in the php directory for any files having a file extension of .php. You must NOT include the reference to the php directory in the path; the special cgiwrap that Apache uses when it sees the .php extension presumes that directory implicitly. Note that other servers may not require your php files to be in a special sub-directory called php.

If your hw8b.html has as its URL:   
  
http://hills.ccsf.edu/~USERID/cnit132/homework/hw8/hw8b.html  
  
then the php directory is 3 directories up from hw8b.html and your form statement would be:

Your php file must reside in a directory, called php, which is a sub-directory of your public\_html directory.

Be sure to upload your php file in ascii transfer mode.

PERL EXAMPLE:

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>

<title>forms page</title>

</head>

<body>

<form method="post" action="test.pl">

<h3>A Simple Form</h3>

<p>Which course was your favorite?</p>

<p>  
<select name="fav">  
<option>CNIT 131</option>  
<option selected="selected">CNIT 132</option>   
<option>CNIT 133</option>   
</select>  
</p>

<p>Which course was your least favorite?</p>

<p><input type="radio" name="least" value="cnit131" checked="checked" />CNIT 131

<input type="radio" name="least" value="cnit132" />CNIT 132

<input type="radio" name="least" value="cnit133" />CNIT 133</p>

<p>Comments/Questions<br />

<textarea name="comments" rows="4" cols="40"></textarea></p>

<p><input type="submit"> <input type="reset" /></p>

</form>

</body>

</html>

Here is the perl script, named test.pl, that actually processes the submitted information from the form and returns the processed information as a web page.

#!/usr/local/bin/perl

do "cgi-lib.pl" ||die "Fatal Error: Can't load cgi library";

#calls subroutine in cgi-lib.pl library

&ReadParse;

# The above line reads the stream of name=value pairs from the form   
# and puts the values in a perl table called in

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

# in perl a \n is a "carriage return, next line"   
# NOTE: the text/html indicates the MIME type of  
# information that the   
# script is sending back to the browser thru the Web server

# The items within the braces below refer to the name contents found within the form.

# The next line says to output the following information, until   
# the statement \_END\_; is reached

print<<"\_END\_";

<html>

<head>

<title>Form Page</title>

</head>

<body>

<p>You indicated that your favorite class is $in{'fav'}</p>

<p>You indicated that your least favorite class is $in{'least'}</p>

<p>The value of comments is $in{'comments'}</p>

</body>

</html>

\_END\_  
;

PHP

Your PHP script must have a file extension of .php. Here is the content of my formA.php.

<?php $msg = "E-MAIL SENT FROM FOG\n";

$msg .= "Sender's Name:

$\_POST[sender\_name]\n";

$msg .= "Sender's E-mail:

$\_POST[sender\_email]\n";

$msg .= "Sender's Age:

$\_POST[age]\n"; $msg .= "Message:

$\_POST[message]\n\n";

$to = "srubin45@comcast.net";

$subject = "Form Feedback from Fog";

$mailheaders = "From:

$\_POST[sender\_email]\n";

$mailheaders .= "Reply-To:

$\_POST[sender\_email]\n\n";

mail($to, $subject, $msg, $mailheaders); ?>

<html>

<head>

<title>The following email has been sent to Steve Rubin</title>

</head>

<body bgcolor="#f5f5dc">

<p><b>Your Name: </b>

<?php echo

"$\_POST[sender\_name]"; ?> <br /><br />

<b>Your E-mail Address: </b>

<?php echo "$\_POST[sender\_email]"; ?> <br /><br />

<b>Your Age: </b> <?php echo

"$\_POST[age]"; ?> <br /><br />

<b>Your Message:</b>

<?php echo "$\_POST[message]"; ?>

</p>

</body>

</html>

Explanation of the above code:

The code in formA.php will send an email back to you and and return the form information back to the user's screen.  
  
A. Begin a PHP block and start building a message string:

<?php  
$msg = "E-MAIL SENT FROM FOG\n";   
  
NOTE: The message E-MAIL SENT FROM FOG will be the first line in the message text of the email that is sent and can be changed to anything you like. \n is a carriage return.   
  
  
B. Continue building the message string by adding an entry for the sender's name:

$msg .= "Sender's Name: $\_POST[sender\_name]\n";

NOTE: The .= indicates concatenation. Your variable after $\_POST must match exactly the variable name in your form input text box for sender's name. For example in my HTML form page, I have:  
  
<input type="text" name="sender\_name" size="40">   
  
NOTE: You don't have to use the variable sender\_name in your form or in this php file. What matters is that the names must MATCH.   
  
C. Continue building the message string by adding an entry for the sender's e-mail address, an entry for age, and an entry for the message:   
  
$msg .= "Sender's E-mail:   
$\_POST[sender\_email]\n";   
$msg .= "Sender's Age:  
$\_POST[age]\n";   
$msg .= "Message:   
$\_POST[message]\n\n";   
  
NOTE: The extra spaces before $\_POST are there so that the messages will be better aligned vertically when you are reading the email. The variable names after $\_POST (in this case, sender\_email, age, and message) must match exactly the variable name in your form input text box, radio buttons, and textarea. For Example, in my HTML form page, I have:  
  
<input type="text" size="25" name="sender\_email" />   
<input type="radio" name="age" value="0-20" />Under 21   
<textarea rows="5" cols="20" name="message"></textarea>   
  
D. Create a variable to hold the recipient's e-mail address (substitute your own):  
  
$to = "srubin45@comcast.net";   
  
E. Create a variable to hold the subject of the e-mail:  
  
$subject = "Form Feedback from Fog";   
  
NOTE: You can change the string inside the quotes to anything you like.  
  
  
F. Create a variable to hold additional mailheaders:   
  
$mailheaders = "From: $\_POST[sender\_email]\n";   
  
NOTE: This will put the sender's email address in the from field of the sent email that comes to you. Note also that the variable name inside of the square brackets must match exactly the variable name in your form input text box for sender's email address.  
  
  
G. Add to the $mailheaders variable:  
  
$mailheaders .= "Reply-To: $\_POST[sender\_email]\n\n";   
  
NOTE: the variable name inside of the square brackets must match exactly the variable name in your form input text box for sender's email address.  
  
  
H. Add the mail() function:  
  
mail($to, $subject, $msg, $mailheaders);   
  
I. Close your PHP block:

?>

NOTE: You're not done yet. Although this code will send the email, you should return something to the user's screen so that they know the form has been sent. Otherwise, they might sit there and continually click the Send button.  
  
J. The next 6 lines are normal HTML statements.  
  
K. Add the text label for the Your Name field and display the user's input:  
  
<p><b>Your Name: </b>   
<?php echo "$\_POST[sender\_name]"; ?>   
  
NOTE: sender\_name must match exactly the variable name in your form input text box for sender's name.   
  
<?php echo "$\_POST[sender\_name]"; ?>   
returns to the screen the value of that form name variable.   
  
  
L. Add the text label for the Your E-mail Address field and display the user's input:  
  
<b>Your E-mail Address: </b>   
  
<?php echo "$\_POST[sender\_email]"; ?>   
  
NOTE: sender\_email must match exactly the variable name in your form input text box for sender's email.  
  
  
M. Add the text label for the Age field and display the user's input:  
  
<b>Your Age: </b>   
<?php echo "$\_POST[age]"; ?>   
  
NOTE: age must match exactly the variable name in your form input text box for sender's age.  
  
  
N. Add the text label for the Message field and display the user's input:  
  
<b>Your Message:</b>   
<?php echo "$\_POST[message]"; ?>  
  
NOTE: message must match exactly the variable name in your form textarea for message.  
  
  
O. Add the closing HTML statements.