

CGI Programming

What is "CGI"?

- Common Gateway Interface
- A means of running an executable program via the Web.
- CGI is not a Perl-specific concept. Almost any language can produce CGI programs
 - even C++ (gasp!!)
- However, Perl does have a **very** nice interface to creating CGI methods

How Does it Work?

- A program is created, like normal.
- The program must be made user-executable
 - `#!/usr/bin/env perl`
 - `chmod u+x filename.cgi`
- A user visits a web page.
- The web browser contacts the server where the CGI program resides, and asks it to run the program
 - passes Parameters to the program
 - similar to command line arguments
- The server runs the program with the parameters passed in
- The server takes the output of the program and returns it to the web browser.
- The web browser displays the output as an HTML page
 - or text, or jpeg, or whatever

Forms

- Most (not all) CGI scripts are contacted through the use of HTML forms.
- A form is an area of a web page in which the user can enter data, and have that data submitted to another page.
- When user hits a submit button on the form, the web browser contacts the script specified in the form tag.

Creating a Form

```
<form method="post" action="file.cgi">
```











...

```
<input type="submit" value="Submit Form">
```

```
</form>
```

- Method attribute specifies how parameters are passed to the CGI program. "post" means they're passed in the HTTP header (and therefore aren't seen anywhere). "get" means they're passed through the query string of the URL itself, and therefore seen in the address bar in the web browser.
- Action attribute specifies which program you want the web browser to contact.
- **<input>** is a tag used to accept User data. type="submit" specifies a Submit button. When user clicks this button, browser contacts file specified in action attribute.

Form Input Types

- Many different ways of getting data from user. Most specified by **<input>** tag, type specified by **type** attribute
- text  a text box
- checkbox  a check box
- radio  a Radio button
- password  password field
 - (text box, characters display as *****)
- hidden  hidden field (nothing displayed in browser)
- submit  Submit button. Submits the form
- reset  Reset button. Clears form of all data.
- button  A button the user can press
 - (usually used w/ javascript. *shudder*)
- file  field to upload a file
- image  an image user can click to submit form

Other Attributes of `<input>`

- **name** \mathbb{L} name of input field.
- **value** \mathbb{L} value returned from checks & radios; text of submits and buttons; contents of text, password, and hidden
- **size** \mathbb{L} width of text or password
- **checked** \mathbb{L} radio or checkbox 'turned on'
- **src** \mathbb{L} URL of an image

Inputs that Don't Use `<input>`

- `<textarea name="...">...</textarea>`
 - Multi-line text field. Specify **rows** and **cols** attributes, and type any pre-existing text between the tags.
- `<select name="...">...</select>`
 - create a drop-down menu.
 - `<option value="...">...</option>`
 - Options in the drop down menu. value attribute is the value of the select input when submitted; text between tags is what is displayed in the browser for this option

Great. We can input. Now what?

- Now, we can write a CGI program that takes those inputs, executes some code, and returns an HTML page
- As with all perl scripts, first line of the program MUST be the shebang: `#!/usr/bin/env perl`
- Next, need to include all the CGI methods. These are stored in CGI.pm
 - functions we care about stored in the 'standard' tag from `%EXPORT_TAGS`
- `use CGI ':standard';`

Outputting from CGI

- Just as CGI program took 'input' from user via web browser, it outputs back to user via web browser as well.
- STDOUT is redirected to the web browser that contacted it.
- This means you don't have to learn any new output functions. `print()` will now throw data to the web browser.

Beginning a CGI Program

- ```
#!/usr/bin/env perl
use strict;
use warnings;
use CGI qw/:standard/;
print header('text/html');
```
- `header()` prints the HTTP header to web browser. The argument is the MIME type of the forthcoming data.
  - Defaults to `text/html`, so you can usually just leave the argument out.

---

---

---

---

---

---

---

## Now Create your Output

- Remember, you're building an HTML page in the output. So you must follow the HTML format:
- ```
print "<html>\n<head>\n<title>My CGI
Program</title>\n</head>\n<body>\n";
```
- CGI.pm gives you a better way to do this. We'll get to it soon.

Where'd Those Inputs Go?

- They were passed into the CGI program as parameters. You can retrieve them using the **param()** function.
- Called in scalar context without any arguments, returns the number of parameters passed to the script
 - `my $num_params = param();`
- Called in list context without an argument, returns names of all parameters.
 - `my @params = param();`
- Called in scalar context, takes name of one parameter, and returns value of that parameter
 - `my $name = param('name');`
- Called in list context w/ an argument, returns array of all values for that parameter (ex, for checks and dropdowns)
 - `my @colors = param('color');`

Dump ()

- subroutine defined in CGI.pm. Retrieves list of parameters and creates an HTML list of all parameters and all values.
- Like most CGI functions, doesn't print anything. You must manually print the return value of the function call.
- `print Dump;`

HTML Shortcuts

- CGI.pm gives you methods to create HTML code without actually writing HTML.
- most HTML tags are aliased to CGI functions.
- unpaired tags:
 - `print br();` # sends `
` to browser
- paired tags:
 - `print strong('Bold text');`
#`Bold text`
 - `print em('Italic');`
#`Italic`

More shortcuts

- For tags with attributes, place name/value attributes in a hash reference as the first argument. The string enclosed in tag is the second argument. Ex:
- ```
print a(
 {href=>'sample.html', target=>'top'},
 'Sample file'
);
```
- Produces:
- ```
<a href="sample.html"
target="top">Sample file</a>
```
- You may think this is needless amounts of extra learning, with no time or length benefits
 - You're probably right. In this case.

start_html() and end_html()

- start_html() can take one parameter, the title.
- ```
print start_html("My title");
```

  - ```
<html><head><title>My title
</title></head><body>
```
- Can also take named parameters, with attributes to give to <body> tag:
- ```
print start_html (-title=>'My
Title', -bgcolor=>'Red');
```

  - ```
<html><head><title>My title
</title></head><body bgcolor="Red">
```
- ```
print end_html;
```

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

HTML Form Shortcuts

- For full list of Form shortcuts, see "FORM ELEMENTS" in `perldoc CGI`
 - or <http://search.cpan.org/dist/CGI.pm/CGI.pm>
- Each one takes parameters as name/value pairs. Name starts with a dash. Most parameters are optional
- ```
print startform(-method=>'post', -action=>
'foo.cgi')
```

  - default method is post. default action is current script
- produces: 

```
<form method="post" action="foo.cgi">
```
- ```
print endform();
```

 # produces

```
</form>
```

Input Shortcuts

- `textfield(-name=>"MyText",
-default =>"This is a text box")`
– `<input type="text" name="MyText" value="This is a text box">`
- All HTML Form input shortcuts are similar. Again, see the perldoc for full list and description.

Programmer Beware

- "default" in input methods is value used *first* time script is loaded only. After that, they hold the values they had the last time the script was run.
- to override (ie, force default value to appear), set parameter `-override=>1` inside input method:
- `textfield(-name=>"foo",
-default =>"bar",
-override=>1);`

Avoid Conflicts

- Some HTML tags have same name as internal Perl functions. Perl will get very confused if you try to use the CGI shortcut methods to print these tags...
- `<tr>` – table row. conflicts with `tr()`
– use `Tr()` or `TR()` instead
- `<select>` – dropdown list. conflicts with `select()`.
– use `Select()` instead
- `<param>` – pass parameters to Java applet – conflicts with `param()`.
– use `Param()` instead
- `<sub>` – Make text subscripted. Conflicts with `sub` keyword.
– Use `Sub()` instead

All in One file

- One common method of CGI programming is to make both the form and the response all in one script. Here's how you do that...
- ```
#!/usr/bin/env perl
use strict;
use warnings;
use CGI qw/:standard/;
print header;
if (!param()){ #no parameters
 print start_html("Here's a form");
 print startform;
 #create your form here...
 print endform;
} else { #parameters passed, form submitted
 print start_html("Here's the result");
 #display the results of the submitting form
}
```

---

---

---

---

---

---

---

## What button was pushed?

- If you want to determine which button the user pushed to activate the CGI script, check the parameter list.
- Only the button that was pushed will be in the parameter list. Other buttons do not exist.
- ```
<input type="submit" name="btn1" value="Push Me!">
<input type="submit" name="btn2" value="No, Push Me!">
if (param('btn1')) { print "btn1 clicked" }
if (param('btn2')) { print "btn2 clicked" }
<input type="submit" name="btn" value="One">
<input type="submit" name="btn" value="Two">
if (param('btn') eq 'One') {
    print "One Clicked"
}
if (param('btn') eq 'Two') {
    print "Two Clicked"
}
```

CGI on CS

- The CGI-enabled server is `cgi2.cs.rpi.edu`
- To run your scripts on `cgi2.cs`
 - set shebang: `#!/usr/bin/env perl`
 - make file user-executable
 - put the file in `public.html/cgi-bin` directory
 - Make sure `public.html` and `cgi-bin` are world-executable
 - go to `http://cgi2.cs.rpi.edu/~rcsid/cgi-bin/filename.cgi`
 - see "Steps For CGI" in handouts for detailed process
- If all goes well, your script's output will be displayed.
- If all does not go well, you'll get HTTP 500

Debugging

- Debugging a CGI program can be a very frustrating task.
- If there are any compilation errors whatsoever, the web browser will simply display **500 Internal Server Error**
- It will also tell you that "more information may be found in the server logs".
- Those server logs are located on the CS System at: **/servers/cgi2/logs/error.log**
- There is, of course, a nicer alternative...

CGI::Carp

- Enables you to view your compilation errors in the browser, rather than searching for them in the server's error logs.
- ```
#!/usr/bin/env perl
use strict;
use warnings;
use CGI qw/:standard/;
use CGI::Carp (
 'warningsToBrowser',
 'fatalsToBrowser'
);
print header;
warningsToBrowser(1);
```
- fatal errors will now be displayed in the browser.
- Warnings will be displayed as HTML comments (view the source of your page to see them)

---

---

---

---

---

---

---

## Debugging process

- If you get HTTP 500, even with using CGI::Carp...
- Try running your script from the command line
  - ssh into cgi2.cs.rpi.edu, change to your public.html/cgi-bin directory, and run your script
    - Run it \*directly\*: **./prog.cgi**, not **perl prog.cgi**
- If that appears to work without problems, then check the server logs
  - try **grep**'ing for your username or filename:  
**grep lallip error.log**  
**grep my\_cgi.cgi error.log**

---

---

---

---

---

---

---