COMP284 Scripting Languages

Lecture 8: Perl (Part 7)

Handouts (8 on 1)

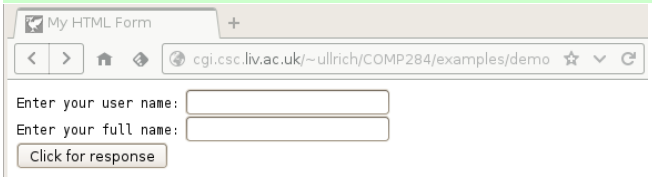
Ullrich Hustadt

Department of Computer Science
School of Electrical Engineering, Electronics, and Computer Science
University of Liverpool

Client requests

In the following we focus on client requests that are generated using HTML forms

```
<!DOCTYPE html>
<html>
<head><title>My HTML Form</title></head>
<body>
<form action="http://cgi.csc.liv.ac.uk/cgi-bin/cgiwrap/ullrich/demo"
 method="post">
<label>Enter your user name: <input type="text" name="username"></label><br>
<label>Enter your full name: <input type="text" name="fullname"></label><br>
<input type="submit" value="Click␣for␣response">
</form>
</body>
</html>
```



Contents

1. CGI
   - Overview
   - CGI I/O
2. The Perl module CGI.pm
   - Motivation
   - HTML shortcuts
   - Forms

Encoding of input data

- Input data from an HTML form is sent URL-encoded as sequence of key-value pairs: `key1=value1&key2=value2&...`

  Example:
  `username=dave&fullname=David%20Davidson`

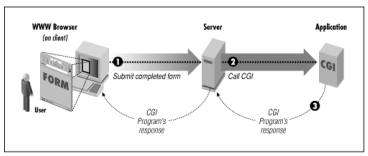
- All characters except A-Z, a-z, 0-9, -, _, ., ~ (unreserved characters) are encoded
- ASCII characters that are not unreserved characters are represented using ASCII codes (preceded by %)
  - A space is represented as %20 or +
  - + is represented as %2B
  - % is represented as %25

`Catherine+0%27Donnell`

Common Gateway Interface — CGI

The Common Gateway Interface (CGI) is a standard method for web servers to use an external application, a CGI program, to dynamically generate web pages

1. A web client generates a client request, for example, from a HTML form, and sends it to a web server
2. The web server selects a CGI program to handle the request, converts the client request to a CGI request, executes the program
3. The CGI program then processes the CGI request and the server passes the program's response back to the client



Request methods: GET versus POST

The t forms
are

- G
  - Form data is appended to the URI in the request

    `<scheme> "://" <server-name> ":" <server-port>`
    `<script-path> <extra-path> "?" <query-string>`

  - Form data is accessed by the CGI program via environment variables

Example:

```
GET /cgi-bin/cgiwrap/ullrich/demo?username=dave&
fullname=David+Davidson HTTP/1.1
Host: cgi.csc.liv.ac.uk
```


Client requests

In the following we focus on client requests that are generated using HTML forms

```
<!DOCTYPE html>
<html>
<head><title>My HTML Form</title></head>
<body>
<form action=
 "http://cgi.csc.liv.ac.uk/cgi-bin/cgiwrap/ullrich/demo"
 method="post">
<label>Enter your user name:
    <input type="text" name="username"></label><br>
<label>Enter your full name:
    <input type="text" name="fullname"></label><br>
<input type="submit" value="Click␣for␣response">
</form>
</body>
</html>
```


Request methods: GET versus POST

The two main request methods used with HTML forms are GET and POST:

- POST:
  - Form data is appended to end of the request (after headers and blank line)
  - Form data can be accessed by the CGI program via standard input
  - Form data is not necessarily URL-encoded (but URL-encoding is the default)

Example:

```
POST /cgi-bin/cgiwrap/ullrich/demo HTTP/1.1
Host: cgi.csc.liv.ac.uk

username=dave&fullname=David+Davidson
```

Environment variables: GET

Env variable	Meaning
QUERY_STRING	The query information passed to the program
REQUEST_METHOD	The request method that was used
PATH_INFO	Extra path information passed to a CGI program
PATH_TRANSLATED	Translation of PATH_INFO from virtual to physical path
SCRIPT_NAME	The relative virtual path of the CGI program
SCRIPT_FILENAME	The physical path of the CGI program

Example (1):

```
GET http://cgi.csc.liv.ac.uk/cgi-bin/cgiwrap/ullrich/demo/more/dirs?
    username=dave&fullname=David+Davidson
QUERY_STRING        username=dave&fullname=David+Davidson
REQUEST_METHOD      GET
PATH_INFO           /more/dirs
PATH_TRANSLATED     /users/www/external/docs/more/dirs
SCRIPT_NAME         /cgi-bin/cgiwrap/ullrich/demo
SCRIPT_FILENAME     /users/loco/ullrich/public_html/cgi-bin/demo

STDIN
# empty
```


Environment variables: GET

Env variable	Meaning
QUERY_STRING	The query information passed to the program
REQUEST_METHOD	The request method that was used
PATH_INFO	Extra path information passed to a CGI program
PATH_TRANSLATED	Translation of PATH_INFO from virtual to physical path
SCRIPT_NAME	The relative virtual path of the CGI program
SCRIPT_FILENAME	The physical path of the CGI program

Example (2):

```
GET http://cgi.csc.liv.ac.uk/cgi-bin/cgiwrap/ullrich/demo/more/dirs?
    username=2%60n+d%2Bt+e+s%27t&fullname=Peter+Newton
QUERY_STRING        username=2%60n+d%2Bt+e+s%27t&fullname=Peter+Newton
REQUEST_METHOD      GET
PATH_INFO           /more/dirs
PATH_TRANSLATED     /users/www/external/docs/more/dirs
SCRIPT_NAME         /cgi-bin/cgiwrap/ullrich/demo
SCRIPT_FILENAME     /users/loco/ullrich/public_html/cgi-bin/demo

STDIN
# empty
```


Environment variables: POST

Env variable	Meaning
QUERY_STRING	The query information passed to the program
REQUEST_METHOD	The request method that was used
SCRIPT_NAME	The relative virtual path of the CGI program
SCRIPT_FILENAME	The physical path of the CGI program

Example:

```
POST /cgi-bin/cgiwrap/ullrich/demo
Host: cgi.csc.liv.ac.uk

username=2%60n+d%2Bt+e+s%27t&fullname=Peter+Newton
QUERY_STRING
# empty
REQUEST_METHOD      POST
SCRIPT_NAME         /cgi-bin/cgiwrap/ullrich/demo
SCRIPT_FILENAME     /users/loco/ullrich/public_html/cgi-bin/demo

STDIN               username=2%60n+d%2Bt+e+s%27t&fullname=Peter+Newton
```


More environment variables

Env variable	Meaning
HTTP_ACCEPT	A list of the MIME types that the client can accept
HTTP_REFERER	The URL of the document that the client points to before accessing the CGI program
HTTP_USER_AGENT	The browser the client is using to issue the request
REMOTE_ADDR	The remote IP address of the user making the request
REMOTE_HOST	The remote hostname of the user making the request
SERVER_NAME	The server's hostname
SERVER_PORT	The port number of the host on which the server is running
SERVER_SOFTWARE	The name and version of the server software

CGI programs and Perl

- CGI programs need to process input data from environment variables and STDIN, depending on the request method
  - ⇝ preferably, the input data would be accessible by the program in a uniform way
- CGI programs need to process input data that is encoded
  - ⇝ preferably, the input data would be available in decoded form
- CGI programs need to produce HTML markup/documents as output
  - ⇝ preferably, there would be an easy way to produce HTML markup

In Perl all this can be achieved with the use of the `CGI.pm` module
`http://perldoc.perl.org/CGI.html`

CGI.pm HTML shortcuts

- CGI.pm provides so-called HTML shortcuts that create HTML tags

a	address	applet	b	body	br	center	code
dd	div	dl	dt	em	font	form	
h1	h2	h3	h4	h5	h6	head	header
html	hr	img	li	ol	p	pre	strong
sup	table	td	th	tr	title	tt	ul

- HTML tags have attributes and contents

```
<p align="right">This is a paragraph</p>
```

- HTML shortcuts are given
  - HTML attributes in the form of a hash reference as the first argument
  - the contents as any subsequent arguments

```
                                    ␣paragraph")
```


CGI.pm HTML shortcuts: Examples

```
Code:   print p
Output
```

```
Code:   print p('');
Output: <p></p>

Code:   print p({-align=>right},"Hello␣world!");
Output: <p align="right">Hello world!</p>

Code:   print p({-class=>right_para,-id=>p1},"Text");
Output: <p class="right_para" id="p1">Text</p>
```


CGI.pm HTML shortcuts: Nesting vs Start/End

- Nested HTML tags using nested HTML shortcuts

```
Code:   print p(em("Emphasised")."␣Text"), "\n";
Output: <p><em>Emphasised</em> Text</p>
```

- Nested HTML tags using start_*tag* and end_*tag*:

```
use CGI qw(-utf8 :all *em *p);

print start_p(), start_em(), "Emphasised", end_em(),
      "␣Text", end_p(), "\n";
Output: <p><em>Emphasised</em> Text</p>
```

The following start_*tag*/end_*tag* HTML shortcuts are generated automatically by CGI.pm:

```
start_html(), start_form(), start_multipart_form()
  end_html(),   end_form()    end_multipart_form()
```

All others need to be requested by adding :*tag* to the CGI.pm import list

CGI.pm Forms

- HTML forms are created using `start_form` and `end_form`

```
print start_form({-method=>request_method,
                  -action=>uri});
form_elements
print end_form;
```

- HTML form elements are again created using HTML shortcuts

textfield	textarea	password_field
filefield	hidden	scrolling_list
popup_menu	optgroup	
image_button	checkbox	checkbox_group
radio_group	reset	submit

  - optgroup creates an option group within a popup menu
    ⇝ optgroup occurs nested inside popup_menu
  - All other HTML shortcuts for HTML form elements will occur independently of each other within a form

Making it work

For CGI programs to work on our systems you must proceed as follows:

1. Your home directory must be 'world executable'
2. You must have a directory
   `$HOME/public_html/cgi-bin/`
   Your `public_html` and `cgi-bin` directory must be both readable and executable by everyone
3. Your CGI script must be placed in
   `$HOME/public_html/cgi-bin/`
   and must be executable by everyone
4. The CGI script can then be accessed using the URL
   `http://cgi.csc.liv.ac.uk/cgi-bin/cgiwrap/<user>/<script>`
   or `http://cgi.csc.liv.ac.uk/cgi-bin/cgiwrapd/<user>/<script>`
   where `<user>` is your user name
   and `<script>` is the filename of the script
   (`cgiwrapd` provides debugging output, but does not reveal all errors)

CGI.pm Forms: Examples

```
print textfield({-name=>'username',
                 -value=>'dave',
                 -size=>100,
                 -maxlength=>500});
```

- `-name` specifies the name of the text field
  and is the only required argument of `textfield`
- `-value` specifies a default value that will be shown in the text field
- `-size` is the size of the text field in characters
- `-maxlength` is the maximum number of characters that the text field will accept

Output:

```
<input type="text" name="username"
       value="dave" size="100" maxlength="500" />
```


CGI.pm Forms: Examples

```
print submit({-name=>'submit',
              -label=>'Click␣for␣response'});
```

- `-name` is an optional argument that allows to distinguish submit buttons from each other
- `-label` or `-value` is an optional argument that determines the label shown to the user and the value passed to the CGI program

Output:

```
<input type="submit" name="submit"
       value="Click␣for␣response" />
```


CGI.pm Forms: Example

```
#!/usr/bin/perl

use CGI qw(-utf8 :all);

print header(-charset=>'utf-8'),
      start_html({-title=>'My␣HTML␣Form',
                  -author=>'u.hustadt@liverpool.ac.uk',
                  -style=>'style.css'});

print start_form({-method=>"GET",
                  -action=>"http://cgi.csc.liv.ac.uk/".
                           "cgi-bin/cgiwrap/ullrich/demo"});
print textfield({-name=>'username',
                 -value=>'dave',
                 -size=>100});
print br();
print textfield({-name=>'fullname',
                 -value=>'Please␣enter␣your␣name',
                 -size=>100});
print br();
print submit({-name=>'submit',
              -value=>'Click␣for␣response'});
print end_form, end_html;
```


Accessing and processing data

- Perl provides a hash %ENV that stores the information stored in environment variables
- Processing %ENV is done in the standard way for hashes

```
print "The␣request␣method␣used␣is␣",
      $ENV{'REQUEST_METHOD'}, br(), "\n";

foreach $key (keys %ENV) {
  print "The␣value␣of␣$key␣is␣$ENV{$key}", br(), "\n";
}
```

Output:

```
The request method used is GET
The value of SCRIPT_NAME is /cgi-bin/cgiwrap/ullrich/demo
The value of SERVER_NAME is cgi.csc.liv.ac.uk
The value of SERVER_ADMIN is root@localhost
The value of HTTP_ACCEPT_ENCODING is gzip,deflate
```


Accessing and processing data

- C … access the input data
- o …
- F

  key1=*value1*&*key2*=*value2*&*key3*=*value3*&...

  representing the input data of a HTML form

  `param('key1')` `param('key2')` `param('key3')` ...

  will return

  value1 *value2* *value3*

  while `param()` returns the list `('key1','key2','key3', ...)`
- The values returned by `param` have already been decoded
- `param('key')` returns the empty string if *value* is empty
- `param('key')` returns `undef` if *key* is not among the key-value pairs of the request
- This does not depend on whether the request method is GET or POST

Accessing and processing data

- `CGI.pm` provides the `param` routine to access the input data of HTML forms

```
print "The␣value␣of␣username␣is␣",
      param('username'), br(), br(), "\n";
print "The␣value␣of␣fullname␣is␣",
      param('fullname'), br(), br(), "\n";

foreach $key (param()) {
   print "The␣value␣of␣$key␣is␣",param($key), br(), "\n";
}
```

Output:

```
The value of username is dave
The value of fullname is David Davidson

The value of submit is Click for response
The value of username is dave
The value of fullname is David Davidson
```

Accessing and processing data: UFT-8

- The pragma -utf8 in

```
use CGI qw(-utf8 :all);
```

makes makes CGI.pm treat all param() values as UTF-8 strings

- Alternatively, specific param() values can be decoded using the decode subroutine of the Encode module

```
use Encode;
my $fullname = decode("utf8",param('fullname'));
```

- With

```
binmode(STDOUT, ":encoding(utf-8)");
print header(-charset=>'utf-8');
```

we ensure that the web page we produce is sent to the brower using UTF-8 encoding

Accessing and processing data: Security

- Do not trust any data accessed via param (beware code injection)
  Example:

```
print "The value of username is ",param('username'),"\n";
```

together with input

```
<script>window.location="http://malware_site/"</script>
```

for username, would redirect the browser to malware_site.

- Check whether the data has the format expected

```
if (param('username') !~ /^[a-zA-Z0-9]+$/s) {
  print "Not a valid user name"
} else {
  print "The value of username is ",param('username'),"\n";
}
```

or sanitise the input using the CGI.pm routine escapeHTML

```
  print "The value of username is ",
        escapeHTML(param('username')),
```

or even better, do both

CGI.pm Scripts: Example (Part 1)

```
use CGI qw(-utf-8 :all *table);
binmode(STDOUT, ":encoding(utf-8)");

print header(-charset=>'utf-8'), "\n",
      start_html({-title=>'Form Processing',
                  -author=>'u.hustadt@liverpool.ac.uk'});

if (!defined(param('username'))) {
  # This branch is executed if the user first visits this page/script
  print start_form({-method=>"POST"});
  print textfield({-name=>'username', -value=>'dave',
                   -size=>100}), "\n";
  print br(), "\n";
  print textfield({-name=>'fullname',
                   -value=>'Please enter your name',
                   -size=>100}), "\n";
  print br(), "\n";
  print submit({-name=>'submit',
                -value=>'Click for response'}), "\n";
  print end_form;
} else {
  # This branch is executed if the client request is generated
  # by the form
```


CGI.pm Scripts: Example (Part 2)

```
  # (We are in the else-branch now)

  print start_table({-border=>1});
  print caption("Inputs");
  foreach $key (param()) {
    print Tr(td('PARAM'),td($key),td(escapeHTML(param($key))));
   }
  foreach $key (keys %ENV) {
    print Tr(td('ENV'),td($key),td(escapeHTML($ENV{$key})));
  }
  print end_table;
}
print end_html;
```


CGI.pm Scripts: Example (Part 3)

Page produced on the first visit

Page produced on submission of the form

Inputs

PARAM	username	dave
PARAM	fullname	David Davidson
PARAM	submit	Click for response
ENV	REQUEST_METHOD	POST
ENV	QUERY_STRING	
ENV	SCRIPT_FILENAME	/users/loco/ullrich/public_html/cgi-bin/lect09.pl
ENV	SERVER_NAME	cgi.csc.liv.ac.uk
ENV	HTTP_REFERER	http://cgi.csc.liv.ac.uk/cgi-bin/cgiwrap/ullrich/lect09.pl

Revision

Read

- Chapter 11: Perl Modules

of

R. L. Schwartz, brian d foy, T. Phoenix:
Learning Perl.
O'Reilly, 2011.

- http://perldoc.perl.org/CGI.html