Creating Dynamic Websites with CGI and Mason

Example Programs and Reference

Jon Warbrick
University of Cambridge Computing Service
jon.warbrick@ucs.cam.ac.uk

January 2006

Introduction

These notes contain copies of all the example programs used during the course, together with some related reference material. They also contain a brief summary of the content of each section of the course, but they ***don't*** include all the material covered and will not form a substitute for careful note taking during the sessions.

Most of the example programs were created to demonstrate particular aspects of web application programming - I make no claim in respect of the quality of the programming or of the HTML that they generate, or of the appropriateness of any of these programs for any particular use. In these notes, sections of some programs are presented in bold type -- this is either to draw attention to important sections of the programs or to identify changes from previous versions of the same program.

Some of the example programs used in the course are based on, or adapted from, example programs by Lincon Stein (in '*The Official Guide to Programming with CGI.pm*', John Wiley & Sons, Inc, 1998), Scott Guelich, Shishir Gundavaram and Gunther Birznieks (in '*CGI Programming with Perl*', O'Reilly & Associates, July 2000) and Rachel Coleman (University of Cambridge MISD). My thanks to them all for the inspiration.

The course has an associated web site at

```
http://www-uxsup.csx.cam.ac.uk/~jw35/courses/cgi-and-mason/
```

which includes up-to-date copies of these notes and of the course slides, and machine-readable copies of all the example programs.

Getting started

Some simple CGI programs; the rational for using Perl; the need to escape some HTML characters.

Example 1 - simple.html

A very straight-forward HTML document.

```
<html>
<head>
<title>A first HTML document</title>
</head>
<body>
<h1>Hello World</h1>
<p>Here we all are again</p>
</body>
</html>
```

Example 2 - simple.cgi

A simple CGI programs that produces the same result as the HTML above.

```
#!/usr/bin/perl -Tw
use strict;

print "Content-type: text/html; charset=iso-8859-1\n";
print "\n";

print "<html>\n";

print "<head>\n";
print "<title>A first CGI program</title>\n";
print "</head>\n";

print "<body>\n";
print "<h1>Hello World</h1>\n";
print "<p>Here we all are again</p>\n";
print "</body>\n";

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

The result of running simple.cgi at a terminal.

```
$ ./simple.cgi
Content-type: text/html; charset=iso-8859-1

<html>
<head>
<title>A first CGI program</title>
</head>
<body>
<h1>Hello World</h1>
<p>Here we all are again</p>
</body>
</html>
```

Example 3 - date.cgi

Much like simple.cgi but including some dynamic content, in this case the current date and time.

```
#!/usr/bin/perl -Tw
use strict;

my $now = localtime();

print "Content-type: text/html; charset=iso-8859-1\n";
print "\n";

print "<html>\n";

print "<head>\n";
print "<title>A second CGI program</title>\n";
print "</head>\n";

print "<body>\n";
print "<h1>Hello World</h1>\n";
print "<p>It is $now</p>\n";
print "</body>\n";

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

Example 4 - date2.cgi

As for date.cgi, but using escapeHTML from CGI.pm to escape any problem characters in the date.

```
#!/usr/bin/perl -Tw
use strict;

use CGI qw/escapeHTML/;

my $now = localtime();

print "Content-type: text/html; charset=iso-8859-1\n";
print "\n";

print "<html>\n";

print "<head>\n";
print "<title>A second CGI program</title>\n";
print "</head>\n";

print "<body>\n";
print "<h1>Hello World</h1>\n";
print "<p>It is ";
print escapeHTML($now);
print "</p>\n";
print "</body>\n";

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

Some standards

We need to know about HTTP and the 'Common Gateway Interface' itself. We have already seen how CGI programs send data to the client, but now we also discover about the environment variables that CGI programs can use and about CGI headers.

Example 5 - env_named.cgi

All of the 17 'named' CGI environment variables.

```
#!/usr/bin/perl -Tw
use strict;

use CGI qw/escapeHTML/;

my @vars = ('SERVER_SOFTWARE', 'SERVER_NAME',      'GATEWAY_INTERFACE',
            'SERVER_PROTOCOL', 'SERVER_PORT',      'REQUEST_METHOD',
            'PATH_INFO',       'PATH_TRANSLATED', 'SCRIPT_NAME',
            'QUERY_STRING',    'REMOTE_HOST',      'REMOTE_ADDR',
            'AUTH_TYPE',       'REMOTE_USER',      'REMOTE_IDENT',
            'CONTENT_TYPE',    'CONTENT_LENGTH');

print "Content-type: text/html; charset=iso-8859-1\n";
print "\n";

print "<html>\n";

print "<head>\n";
print "<title>Named CGI environment variables</title>\n";
print "</head>\n";

print "<body>\n";
print "<h1>Named CGI environment variables</h1>\n";

print "<p>This webserver is '";
print escapeHTML($ENV{SERVER_NAME});
print "', the request method was '";
print escapeHTML($ENV{REQUEST_METHOD});
print "' and the query string was '";
print escapeHTML($ENV{QUERY_STRING} || '(empty)');
print "'</p>\n";

for my $var (@vars) {
  print "$var: ";
  print escapeHTML($ENV{$var} || '(empty)');
  print "<br />\n";
}

print "</body>\n";

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

Example 6 - env_http.cgi

All the CGI environment variables derived from HTTP headers.

```
#!/usr/bin/perl -Tw
use strict;

use CGI qw/escapeHTML/;
```

```
print "Content-type: text/html; charset=iso-8859-1\n";
print "\n";

print "<html>\n";

print "<head>\n";
print "<title>HTTP CGI environment variables</title>\n";
print "</head>\n";

print "<body>\n";
print "<h1>HTTP CGI environment variables</h1>\n";

print "<p>This browser calls itself '";
print escapeHTML($ENV{HTTP_USER_AGENT});
print "'</p>\n";

for my $var (sort keys %ENV) {
  if ($var =~ /^HTTP_/) {
    print escapeHTML($var);
    print ": ";
    print escapeHTML($ENV{$var} || '(empty)');
    print "<br />\n";
  }
}

print "</body>\n";

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

Example 7 - random2.cgi

This program displays a random photograph by using the 'Location' CGI header to tell the webserver to return the content of a different document.

```
#!/usr/bin/perl -Tw
use strict;

my ($docroot, $pict_dir, @pictures, $num_pictures,
    $lucky_one, $buffer);

$docroot  = "/var/www/html";
$pict_dir = "cgi-course-examples/img";

chdir "$docroot/$pict_dir"
            or die "Failed to chdir to picture directory: $!";
@pictures = glob('*.png');
$num_pictures = $#pictures;
$lucky_one = $pictures[rand($num_pictures-1)];
die "Failed to find a picture" unless $lucky_one;

print "Location: /$pict_dir/$lucky_one\n";
print "\n";
```

Example 8 - random3.cgi

This program is similar to random2.cgi, but uses the 'Location' CGI header to return a redirect message to the browser telling it to collect a different document.

```
#!/usr/bin/perl -Tw
use strict;

my ($docroot, $pict_dir, @pictures, $num_pictures,
    $lucky_one, $buffer, $server);
```

```
$server   = "mnementh.csi.cam.ac.uk";
$docroot  = "/var/www/html";
$pict_dir = "cgi-course-examples/img";

chdir "$docroot/$pict_dir"
           or die "Failed to chdir to picture directory: $!";
@pictures = glob('*.png');
$num_pictures = $#pictures;
$lucky_one = $pictures[rand($num_pictures-1)];
die "Failed to find a picture" unless $lucky_one;

print "Location: http://$server/$pict_dir/$lucky_one\n";
print "\n";
```

Example 9 – errors.cgi

This program demonstrates 'proper' error reporting using the CGI 'Status' header.

```
#!/usr/bin/perl -Tw
use strict;

my ($file, $buffer);
$file = '/var/www/msg.txt';

if ((localtime(time))[1] % 2 == 0) {
  error (403, "Forbidden",
         "You may not access this document at the moment");
elsif (!-r $file) {
  error(404, "Not found",
        "The document requested was not found");
else {
  unless (open (TXT, $file)) {
    error (500, "Internal Server Error",
           "An Internal server error occurred");
  }
  else {
    print "Content-type: text/plain\n";
    print "\n";
    while (read(TXT, $buffer, 4096)) {
      print $buffer;
    }
    close TXT;
  }
}

sub error {
  my ($code,$msg,$text) = @_;
  print "Status: $code $msg\n";
  print "Content-type: text/html; charset=iso-8859-1\n";
  print "\n";
  print "<html><head><title>$msg</title></head>\n";
  print "<body><h1>$msg</h1>\n";
  print "<p>$text</p></body></html>\n";
}
```

Getting information from the URL

What URLs are, and how to extract information from them in a CGI program – both a single value and a set of name/value pairs.

Example 10 - photo.cgi

A CGI program that expects one item of information (a photograph number) in the query string.

```
#!/usr/bin/perl -Tw
use strict;

my @titles = ('What we did on our holidays', 'Liege &amp;amp; Lief',
              'Full House', 'Jewel In The Crown',
              'Who Knows Where The Time Goes?');

my $number = $ENV{QUERY_STRING};

my $title = 'Unknown';
if ($number =~ /^\d+$/ and $number >= 0 and $number <= 4) {
  $title = $titles[$number]
}

print "Content-type: text/html; charset=iso-8859-1\n";
print "\n";

print "<html>\n";

print "<head>\n";
print "<title>$title</title>\n";
print "</head>\n";

print "<body>\n";
print "<h1>$title</h1>\n";

if ($title eq 'Unknown') {
  print "<p>Sorry - unknown picture.</p>";
}
else {
  print "<img src=\"img/photo$number.png\" alt=\"$title\" />\n";
}

print "</body>\n";

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

Example 11 - photo2.cgi

A variant on photo.cgi that expects two items of information (a photograph number and the name of someone to credit) as name/value pairs in the query string.

```
#!/usr/bin/perl -Tw
use strict;

use CGI qw/escapeHTML param/;

my @titles = ('What we did on our holidays', 'Liege &amp;amp; Lief',
              'Full House', 'Jewel In The Crown',
              'Who Knows Where The Time Goes?');

my $number = param('photo');
my $credit = param('credit');

my $title = 'Unknown';
if (defined($number) and $number =~ /^\d+$/ and
    $number >= 0 and $number <= 4) {
  $title = $titles[$number]
}
```

```
print "Content-type: text/html; charset=iso-8859-1\n";
print "\n";

print "<html>\n";

print "<head>\n";
print "<title>$title</title>\n";
print "</head>\n";

print "<body>\n";
print "<h1>$title</h1>\n";

if ($title eq 'Unknown') {
  print "<p>Sorry - unknown picture.</p>";
}
else {
  print "<img src=\"img/photo$number.png\" alt=\"$title\" />\n";
  print "<p>&copy; " . escapeHTML($credit) . "</p>\n" if $credit;
}

print "</body>\n";

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

Forms

More than we ever wanted to know about HTML forms, the elements that then can contain and the difference between GET and POST.

Example 12 - search.html

A simple example of an HTML form.

```
<html>

<head>
<title>Book search</title>
</head>

<body>
<h1>Book Search</h1>
<p>Please enter details for the book you are looking for:</p>

<form action="environment.cgi">
<p>Author: <input type="text" name="author" /></p>
<p>Title:  <input type="text" name="title" /></p>
<p><input type="submit" value="Search" /></p>
</form>

</body>

</html>
```

Example 13 - form-elements.html

Examples of the various elements that can appear in an HTML form.

```
<html>
```

```
<head>
<title>Example form elements</title>
</head>

<body>

<h1>Example form elements</h1>

<form action="environment.cgi">

<p>

  Name: <input type="text" name="surname" value="Name" />
  <br />
  Password: <input type="password" name="pwd" value="foobar" />

</p>

<hr />

<p>

  <input type="radio" name="drink" value="tea" />Tea
  <input type="radio" name="drink" value="coffee"
                              checked="checked" />Coffee
  <br />
  <input type="checkbox" name="milk"  value="yes" />Milk
  <input type="checkbox" name="sugar" value="yes" />Sugar

</p>

<hr />

<p>

Note there's nothing here --&gt;
<input type="hidden" name="state" value="New York" />

</p>

<hr />

<p>

  <select name="contact">
    <option selected="selected">Simon</option>
    <option value="Peggy">Dave</option>
    <option>Rick</option>
    <option>Chris</option>
    <option>Gerry</option>
    <option>Rob</option>
  </select>

   

  <select name="contact" size="4">
    <option selected="selected">Simon</option>
    <option value="Peggy">Dave</option>
    <option>Rick</option>
    <option>Chris</option>
    <option>Gerry</option>
    <option>Rob</option>
  </select>

   
```

```
  <select name="contact" size="4" multiple="multiple">
    <option selected="selected">Simon</option>
    <option value="Peggy">Dave</option>
    <option>Rick</option>
    <option>Chris</option>
    <option>Gerry</option>
    <option>Rob</option>
  </select>

</p>

<hr />

<p>

  <textarea name="Comments" cols="50" rows="3">
Across the evening sky, all the birds are leaving
But how can they know it's time for them to go?
Before the winter fire, I will still be dreaming
I have no thought of time
  </textarea>

</p>

<hr />

<p>

  <input type="submit" name="submit" value="Do Stuff" />
  <input type="image" name="find" value="Finding"
                            src="b1.png" alt="[FIND]" />
  <input type="reset"  name="why"    value="Defaults" />
  <input type="button" name="button" value="A button" />

</p>

</form>

</body>

</html>
```

Example 14 - view-request.html

A picture viewer request form.

```
<html>
<head>
<title>Picture Viewer</title>
</head>

<body>

<form action='viewer.cgi'>

<p>Your name: <input type='text' name='name' /></p>

<p>Select a picture:
<select name="photo">
    <option value="0">What we did on our holidays</option>
    <option value="1">Liege &amp;amp; Lief</option>
    <option value="2">Full House</option>
    <option value="3">Jewel In The Crown</option>
    <option value="4">Who Knows Where The Time Goes?</option>
</select>
```

```
</p>

<p>
<input type='submit' name='show' value='Show' />
<input type='reset' value='Reset' />
</p>

</form>

</body>

</html>
```

Example 15 - viewer.cgi

A CGI program that processes requests from view-request.html

```
#!/usr/bin/perl -Tw
use strict;

use CGI qw/escapeHTML param/;

print "Content-type: text/html; charset=iso-8859-1\n";
print "\n";

print "<html>\n";

print "<head>\n";
print "<title>Photo display</title>\n";
print "</head>\n";

print "<body>\n";

do_display();
print "<p><a href='view-request.html'>Request another picture</a></p>\n";

print "</body>\n";

print "</html>\n";

# ---

sub do_display {

  my $name   = param('name');
  my $number = param('photo');

  print "<center>\n";
  print "<h1>A photo for ", escapeHTML($name), "</h1>\n" if $name;
  print "<p>\n";
  if (defined($number) and $number =~ /^\d+$/
      and $number >= 0 and $number <= 4) {
   print "<img src=\"img/photo$number.png\" alt=\"photo $number\" />\n";
  }
  else {
    print "Sorry - unknown picture.\n";
  }
  print "</p>\n";
  print "</center>\n";

}
```

Example 16 - viewer2.cgi

A program like viewer.cgi but which also creates its own request form.

```
#!/usr/bin/perl -Tw
use strict;

use CGI qw/escapeHTML param/;

print "Content-type: text/html; charset=iso-8859-1\n";
print "\n";

print "<html>\n";

print "<head>\n";
print "<title>Photo display</title>\n";
print "</head>\n";

print "<body>\n";

do_display() if param;
do_form();

print "</body>\n";

print "</html>\n";

# ---

sub do_display {

  my $name   = param('name');
  my $number = param('photo');

  print "<center>\n";
  print "<h1>A photo for ", escapeHTML($name), "</h1>\n" if $name;
  print "<p>\n";
  if (defined($number) and $number =~ /^\d+$/
      and $number >= 0 and $number <= 4) {
   print "<img src=\"img/photo$number.png\" alt=\"photo $number\" />\n";
  }
  else {
    print "Sorry - unknown picture.\n";
  }
  print "</p>\n";
  print "</center>\n";

  print "<hr />\n";

}

# ---

sub do_form {

  my $script = $ENV{SCRIPT_NAME};

  print "<form action='$script'>\n";
  print "<p>Your name: <input type='text' name='name' /></p>\n";
  print "<p>Select a picture:\n";
  print "<select name='photo'>\n";
  print "   <option value='0'>What we did on our holidays</option>\n";
  print "    <option value='1'>Liege &amp; Lief</option>\n";
  print "    <option value='2'>Full House</option>\n";
  print "    <option value='3'>Jewel In The Crown</option>\n";
```

```
  print "    <option value='4'>Who Knows Where The Time
Goes?</option>\n";
  print "  </select>\n";
  print "</p>\n";
  print "<p>\n";
  print "<input type='submit' name='show' value='Show' />\n";
  print "<input type='reset' value='Reset' />\n";
  print "</p>\n";
  print "</form>\n";

}
```

Example 17 - viewer3.cgi

As for viewer2.cgi, but using CGI.pm's form shortcuts to create the form.

```
#!/usr/bin/perl -Tw
use strict;

use CGI qw/:standard/;

print "Content-type: text/html; charset=iso-8859-1\n";
print "\n";

print "<html>\n";

print "<head>\n";
print "<title>Photo display</title>\n";
print "</head>\n";

print "<body>\n";

do_display() if param;
do_form();

print "</body>\n";

print "</html>\n";

# ---

sub do_display {

  my $name   = param('name');
  my $number = param('photo');

  print "<center>\n";
  print "<h1>A photo for ", escapeHTML($name), "</h1>\n" if $name;
  print "<p>\n";
  if (defined($number) and $number =~ /^\d+$/
      and $number >= 0 and $number <= 4) {
   print "<img src=\"img/photo$number.png\" alt=\"photo $number\" />\n";
  }
  else {
    print "Sorry - unknown picture.\n";
  }
  print "</p>\n";
  print "</center>\n";

  print "<hr />\n";

}

# ---
```

```
sub do_form {

  my $labels = {0 => 'What we did on our holidays',
                1 => 'Liege & Lief',
                2 => 'Full House',
                3 => 'Jewel In The Crown',
                4 => 'Who Knows Where The Time Goes?'};

  print start_form(-method=>'GET'),
        p("Your name:",
          textfield(-name=>'name')),
        p("Select a picture:",
          scrolling_list(-name=>'photo',
                         -size=>1,
                         -values=>[0,1,2,3,4],
                         -labels=>$labels)),
        p(submit(-name=>'Show'),
          reset(-name=>'Reset')),
        end_form;

}
```

Example 18 - viewer4.cgi

As viewer3.cgi, but submitting the form using POST. Note that there is only one change.

```
#!/usr/bin/perl -Tw
use strict;

use CGI qw/:standard/;

print "Content-type: text/html; charset=iso-8859-1\n";
print "\n";

print "<html>\n";

print "<head>\n";
print "<title>Photo display</title>\n";
print "</head>\n";

print "<body>\n";

do_display() if param;
do_form();

print "</body>\n";

print "</html>\n";

# ---

sub do_display {

  my $name   = param('name');
  my $number = param('photo');

  print "<center>\n";
  print "<h1>A photo for ", escapeHTML($name), "</h1>\n" if $name;
  print "<p>\n";
  if (defined($number) and $number =~ /^\d+$/
      and $number >= 0 and $number <= 4) {
   print "<img src=\"img/photo$number.png\" alt=\"photo $number\" />\n";
  }
```

```
  else {
    print "Sorry - unknown picture.\n";
  }
  print "</p>\n";
  print "</center>\n";

  print "<hr />\n";

}

# ---

sub do_form {

  my $labels = {0 => 'What we did on our holidays',
                1 => 'Liege & Lief',
                2 => 'Full House',
                3 => 'Jewel In The Crown',
                4 => 'Who Knows Where The Time Goes?'};

  print start_form(-method=>'POST'),
        p("Your name:",
          textfield(-name=>'name')),
        p("Select a picture:",
          scrolling_list(-name=>'photo',
                         -size=>1,
                         -values=>[0,1,2,3,4],
                         -labels=>$labels)),
        p(submit(-name=>'Show'),
          reset(-name=>'Reset')),
        end_form;

}
```

Security

We review the issues involved in creating CGI programs that do not have major security problems. This includes the problem of using data 'from outside' when opening files, executing commands, creating SQL statements, and inside generated HTML. We also look at the possibility of CGI scripts being misused and at other security issues.

Debugging CGIs

What the CGI fails to define; how to track down problems in CGI programs

Example 19 - fatal.cgi

How to use the Perl CGI::Carp module to get error messages displayed by the browser.

```
#!/usr/bin/perl -Tw
use strict;

use CGI qw/escapeHTML/;
use CGI::Carp qw/fatalsToBrowser/;

my $now = localtome();
```

```
print "Content-type: text/html; charset=iso-8859-1\n";
print "\n";

print "<html>\n";

print "<head>\n";
print "<title>A second CGI program</title>\n";
print "</head>\n";

print "<body>\n";
print "<h1>Hello World</h1>\n";
print "<p>It is ";
print escapeHTML($now);
print "</p>\n";
print "</body>\n";

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

Introducing Mason

A brief introduction to Mason and review of what makes it useful.

Example 20 - mason.html

A simple Mason document.

```
<html>

<head>
<title>A first Mason document</title>
</head>

<body>
<h1>Hello World</h1>
<p>Here we all are again</p>
</body>

</html>
```

Example 21 - date.html

A Mason page including some dynamic content

```
% my $now = localtime();

<html>

<head>
<title>A second Mason document</title>
</head>

<body>
<h1>Hello World</h1>
<p>It is <% $now %></p>
</body>

</html>
```

Mason from 10,000 feet

A whistle-stop tour through some Mason features.

Example 22 - hs-mason.html

Using a component library to apply the University House Style

```
<& /hs/htmlhead.mason,
         title=>'A first Mason document' &>

<& /hs/banner.mason,
         heading=>'Computing Service',
         department=>'YES',
&>

<body>
<h1>Hello World</h1>
<p>Here we all are again</p>
</body>

<& /hs/footer.mason,
         credit=>"University of Cambridge Computing Service",
         webmaster=>"webmaster\@ucs.cam.ac.uk" &>

<& /hs/htmltail.mason &>
```

Passing information to components

Everything you wanted to know about passing arguments to components.

Autohandlers and Dhandlers

Wrapping content, and producing multiple 'pages' from one component.

Example 23 - autohandler.hason, wrap1.html, wrap2.html

Using an autohandler to automatically 'wrap' content.

```
<html>

<head>
<title>A standard title</title>
</head>

<body>

<p>Standard header</p>

% $m->call_next;

<p>Standard footer</p>

</body>
```

```
</html>
```

```
<h1>Wrapped document 1</h1>

<p>Here is the content</p>
```

```
<h1>Wrapped document 2</h1>

<p>Here is the content</p>
```

Example 24 - dhandler.mason

Using a Dhandler.

```
<html>

<head>
<title>Dhandler example</title>
</head>

<body>
<h1>This is a trivial Dhandler example</h1>
<p>The rest of the path information is <% $m->dhandler_arg() |h %></p>
</body>

</html>
```

Doing 'CGI' things in Mason

How do do all the useful things we learnt about above under CGI, but in Mason.

Example 25 - viewer2.html

Simple photo display, Mason style.

```
<html>

<head>
<title>Photo display</title>
</head>

<body>

% if (defined($name) and defined($photo)) {
<center>
<h1>A photo for <% $name |h %></h1>
<p>
%   if ($photo =~ /^\d+$/ and $photo >= 0 and $photo <= 4) {
<img src="img/photo<% $photo |u %>.png" alt="photo <% $photo |h %>" />
%   } else {
Sorry - unknown picture.
%   }
</p>
</center>
<hr />
% }
```

```
<form action=''>
<p>Your name: <input type='text' name='name' /></p>
<p>Select a picture:
<select name='photo'>
  <option value='0'>What we did on our holidays</option>
  <option value='1'>Liege &amp;amp; Lief</option>
  <option value='2'>Full House</option>
  <option value='3'>Jewel In The Crown</option>
  <option value='4'>Who Knows Where The Time Goes?</option>
</select>
</p>
<p>
<input type='submit' name='show' value='Show' />
<input type='reset' value='Reset' />
</p>
</form>

</body>

</html>

<%args>
$name => undef
$photo => undef
</%args>
```

Example 26 - viewer3.html

As above, but with 'sticky' fields.

```
<html>

<head>
<title>Photo display</title>
</head>

<body>

% if (defined($name) and defined($photo)) {
<center>
<h1>A photo for <% $name |h %></h1>
<p>
%   if ($photo =~ /^\d+$/ and $photo >= 0 and $photo <= 4) {
<img src="img/photo<% $photo |u %>.png" alt="photo <% $photo |h %>" />
%   } else {
Sorry - unknown picture.
%   }
</p>
</center>
<hr />
% }

<& /cgi.mason:init, %ARGS &>

<form action=''>
<p>Your name: <& /cgi.mason:textfield, name=>"name" &></p>
<p>Select a picture:
<& /cgi.mason:popup_menu, name=>"photo",
     values=>[0, 1, 2, 3, 4],
     labels=>{0 => 'What we did on our holidays',
              1 => 'Liege &amp; Lief',
              2 => 'Full House',
              3 => 'Jewel In The Crown',
```

```
              4 => 'Who Knows Where The Time Goes?'} &>
</p>
<p>
<input type='submit' name='show' value='Show' />
<input type='reset' value='Reset' />
</p>
</form>

</body>

</html>

<%args>
$name => undef
$photo => undef
</%args>
```

Example 27 - info1.html

Getting information about the request via the Mason and Apache APIs.

```
<html>

<head>
<title>Request information</title>
</head>

<body>
<h1>Information from this request</h1>
<p>Server name: <% $r->get_server_name() |h %></p>
<p>Request method: <% $r->method() |h %></p>
<p>Remote user:
<% $r->can('user') ? $r->user() : $r->connection->user() |h %> </p>
<p>User-agent header: <% $r->headers_in()->{'User-agent'} |h %></p>
<p>Refering page: <% $r->headers_in()->{'Referer'} |h %></p>
<p>Refering page (from environment): <% $ENV{HTTP_REFERER} |h %></p>
</body>

</html>
```

Example 28 - text.html

Overriding the default content type, in this case to display a '.html' file as text.

```
%# Override default content type
% $r->content_type('text/plain; charset=utf-8');

<html>

<head>
<title>A first Mason document</title>
</head>

<body>
<h1>Hello World</h1>
<p>Here we all are again</p>
</body>

</html>
```

Example 29 - random3.html

Redirects

```
<%perl>

my ($docroot, $pict_dir, @pictures, $num_pictures,
    $lucky_one, $buffer, $server);

$server   = "mnementh.csi.cam.ac.uk";
$docroot  = "/var/www/html";
$pict_dir = "mason-course-examples/img";

chdir "$docroot/$pict_dir"
            or die "Failed to chdir to picture directory: $!";
@pictures = glob('*.png');
$num_pictures = $#pictures;
$lucky_one = $pictures[rand($num_pictures-1)];
die "Failed to find a picture" unless $lucky_one;

$m->redirect("http://$server/$pict_dir/$lucky_one");

</%perl>
```

Example 30 - forbidden.html

How to return a non-200 status.

```
<html>

<head>
<title>Intermittent restriction</title>
</head>

<body>
<h1>Sometimes this is restricted</h1>
<p><% 60 - (localtime(time))[0] |h %> seconds until it vanishes</p>
</body>

</html>


<%init>

if ((localtime(time))[1] % 2 == 0) {
  $m->clear_buffer();
  $m->abort(403);
}

</%init>
```

Example 31 - panic.html

Sending custom headers.

```
% $r->headers_out->{'X-panic'} = 'Now!';

<html>

<head>
<title>Including additional headers</title>
</head>
```

```
<body>
<h1>Extra headers</h1>
<p>This document is served with an 'X-Panic' header</p>
</body>

</html>
```

Example 32 - syntax.html, runtime.html, confusion.html

Examples of syntax and run-time errors.

```
<html>

<head>
<title>A broken Mason document</title>
</head>

<body>

% my $broken = "This is a mistake';

<h1>Hello World</h1>
<p>Here we all are again</p>
</body>

</html>
```

```
<html>

<head>
<title>A second broken Mason document</title>
</head>

<body>

% my $now = localtome();

<h1>Hello World</h1>
<p>Here we all are again</p>
</body>

</html>
```

```
% if ('a' eq 'b') {

<html>

<head>
<title>A third broken Mason document</title>
</head>

<body>

<h1>Hello World</h1>
<p>Here we all are again</p>
</body>

</html>

% ]
```

Example 33 - logging.html

Example Apache API logging calls.

```
<html>

<head>
<title>Logging example</title>
</head>

<body>
<p>Messages have been written to the log.</p>
</body>

</html>

<%init>
  $r->log->emerg('A emergency!');
  $r->log->alert('Something needs attension');
  $r->log->crit('A critical error');
  $r->log->error('Something went wrong');
  $r->log->warn('You might want to know...');
  $r->log->notice('Take note');
  $r->log->info('For your information...');
  $r->log->debug('In foobar loop, no widgits');
</%init>
```

Useful techniques

Various useful techniques for CGI or Mason programming.

Example 24 - mailer.html, send_message.mason

```
<html>

<head>
<title>Sending e-mail...</title>
</head>

<body>
<h1>Sending e-mail...</h1>

<p>
<& send_message.mason,
       from => 'jw35@cam.ac.uk',
       to => 'jon.warbrick@ucs.cam.ac.uk',
       subject => 'A test message',
       message => 'Hello World!'
&>
</p>

</body>

</html>
```

```
<%args>
$from
$to
```

```
$subject
$message
</%args>

<%perl>

use English qw(-no_match_vars);
use Net::SMTP;

my $result;

my $msg = "From: $from\n"
          . "To: $to\n"
          . "Subject: $subject\n"
          . "\n"
          . $message;


eval {
  my $smtp = Net::SMTP->new('ppsw.cam.ac.uk', Debug => 1)
                        or die "connect";
  $smtp->mail($from)    or die "mail";
  $smtp->to($to)        or die "to";
  $smtp->data()         or die "data";
  $smtp->datasend($msg) or die "datasend";
  $smtp->dataend()      or die "dataend";
  $smtp->quit()         or die "quit";
  $result = "Message to $to successfully sent";
};
if ($EVAL_ERROR) {
  $result = "Message not sent: failed $EVAL_ERROR";
}

</%perl>

<% $result |h %>
```

Example 35 - lotr.html

Interfacing to databases.

```
<html>

<head>
  <title>The characters</title>
</head>

<body>
<center>

% if (defined($results)) {

<h1>Characters</h1>

<table border="1">
  <tr>
    <th>Name</th>
    <th>Race</th>
  </tr>
%   foreach my $result (@$results) {
  <tr>
    <td><% $result->[0]|h %></td>
    <td><% $result->[1]|h %></td>
  </tr>
%   }
```

```
</table>

% }

<form action="">
  <p>Chose a Middle Earth race:
  <br />
  <& /cgi.mason:popup_menu, name=>'race',
                            values=>\@values,
                            labels=>\%labels &>
  <& /cgi.mason:submit, name=>'Submit', label=>'Submit Query' &>
  </p>
</form>

</center>
</body>

</html>

%# -----------------------------------------

<%args>
$race => undef
</%args>

%# -----------------------------------------
<%init>

use DBI;

$m->comp('/cgi.mason:init', %ARGS);

my %attr = ( RaiseError => 1, PrintError => 0, AutoCommit => 1 );
my $dbh = DBI->connect("DBI:SQLite:dbname=/var/www/html/mason-course-
examples/lotr", "user", "pwd", \%attr);

my $sth = $dbh->prepare ("SELECT name, id FROM race ORDER BY name");
$sth->execute;
my @values;
my %labels;
while ( my ($name, $race) = $sth->fetchrow_array) {
  push @values,$race;
  $labels{$race}=$name;
}

my $results = undef;
if (defined($race)) {
  my $sth = $dbh->prepare ("SELECT characters.name, race.name
                            FROM characters, race
                            WHERE characters.race = race.id
                              AND race.id = ?
                            ORDER BY characters.name");
  $sth->execute($race);
  $results = $sth->fetchall_arrayref;
}

$dbh->disconnect;

</%init>

%# -----------------------------------------
```

Example 36 - lookup.html

Interfacing to lookup.

```
<html>

<head>
  <title>lookup directory lookup</title>
</head>

<body>

<h1>Directory lookup</h1>

% if ($message) {
<p><% $message |h %></p>
% }

% if ($person) {
<p>
CRSid: <% $person->uid() |h %><br />
Name: <% $person->name() |h %><br />
Registered name: <% $person->cn() |h %><br />
E-mail addresses: <% join(', ',$person->allmail()) |h %></br />
</p>
<hr />
% }

<p>
<form action=''>
Lookup: <& /cgi.mason:textfield, name=>"id", default => $user &>
<input type='submit' name='show' value='Show' />
</form>
</p>

<p>This page created by <% $user |h %></p>

</body>

</html>

%# ----------------------------------------

<%args>
$id => undef
</%args>

%# ----------------------------------------
<%init>

use English qw(-no_match_vars);
use Ucam::Directory;
$m->comp('/cgi.mason:init', %ARGS);
my $message = undef;

my $user = $r->can('user') ? $r->user() : $r->connection->user();
$id = $user unless $id;

if (!$user) {
  $r->log->error("Attempted access by unauthenticated user");
  $m->clear_buffer();
  $m->abort(403);
}

my $dir = new Ucam::Directory(ldapserver => "ldap.lookup.cam.ac.uk");
die ("Failed to connect to the directory") unless $dir;

my $person = eval { $dir->lookup_person($id) };
$message = "Failed to lookup '$id'" if $EVAL_ERROR;
```

```
</%init>

%# -----------------------------------------
```

Closing Remarks

Some comments on the problems with CGI, and on possible solutions; assorted references.

REFERENCE

CGI Environment Variables

Extracted from `http://hoohoo.ncsa.uiuc.edu/cgi/interface.html`

The following environment variables are not request-specific and are set for all requests:

- `SERVER_SOFTWARE`

  The name and version of the information server software answering the request (and running the gateway). Format: name/version

- `SERVER_NAME`

  The server's hostname, DNS alias, or IP address as it would appear in self-referencing URLs.

- `GATEWAY_INTERFACE`

  The revision of the CGI specification to which this server complies. Format: CGI/revision

The following environment variables are specific to the request being fulfilled by the gateway program:

- `SERVER_PROTOCOL`

  The name and revision of the information protocol this request came in with. Format: protocol/revision

- `SERVER_PORT`

  The port number to which the request was sent.

- `REQUEST_METHOD`

  The method with which the request was made. For HTTP, this is "GET", "HEAD", "POST", etc.

- `PATH_INFO`

  The extra path information, as given by the client. In other words, scripts can be accessed by their virtual pathname, followed by extra information at the end of this path. The extra information is sent as PATH_INFO. This information should be decoded by the server if it comes from a URL before it is passed to the CGI script.

- `PATH_TRANSLATED`

  The server provides a translated version of PATH_INFO, which takes the path and does any virtual-to-physical mapping to it.

- `SCRIPT_NAME`

  A virtual path to the script being executed, used for self-referencing URLs.

- `QUERY_STRING`

  The information which follows the ? in the URL which referenced this script. This is the query information. It should not be decoded in any fashion. This variable should always be set when there is query information, regardless of command line decoding.

- `REMOTE_HOST`

  The hostname making the request. If the server does not have this information, it should set REMOTE_ADDR and leave this unset.

- `REMOTE_ADDR`

  The IP address of the remote host making the request.

- `AUTH_TYPE`

  If the server supports user authentication, and the script is protects, this is the protocol-specific authentication method used to validate the user.

- `REMOTE_USER`

  If the server supports user authentication, and the script is protected, this is the username they have authenticated as.

- `REMOTE_IDENT`

  If the HTTP server supports RFC 931 identification, then this variable will be set to the remote user name retrieved from the server. Usage of this variable should be limited to logging only.

- `CONTENT_TYPE`

  For queries which have attached information, such as HTTP POST and PUT, this is the content type of the data.

- `CONTENT_LENGTH`

  The length of the said content as given by the client.

In addition to these, the header lines received from the client, if any, are placed into the environment with the prefix `HTTP_` followed by the header name. Any - characters in the header name are changed to _ characters. The server may exclude any headers which it has already processed, such as Authorization, Content-type, and Content-length. If necessary, the server may choose to exclude any or all of these headers if including them would exceed any system environment limits.

Standards

- CGI: `http://hoohoo.ncsa.uiuc.edu/cgi/`

- HTML 4.01: `http://www.w3.org/TR/html4/`
- XHTML 1.0: `http://www.w3.org/TR/xhtml1/`
- HTTP 1.1: RFC 2616 l HTTP 1.0: RFC 1945
- URI generic syntax: RFC 2393
- RFCs are available from
    - `ftp://ftp.rfc-editor.org/in-notes/rfc<nnnn>.txt` (official)
    - `http://www-uxsup.csx.cam.ac.uk/netdoc/rfc/rfc<nnn> .txt` (local)
    - `http://www.faqs.org/rfcs/rfc<nnnn>.html` (pretty)

Books

- CGI Programming with Perl (2nd Edition). Scott Guelich, Shishir Gundavaram, Gunther Birznieks. O'Reilly. 1-56592-419-3
- The Official Guide to Programming with CGI.pm. Lincoln Stein. Wiley & Sons. 0-471-24744-8
- Learning Perl, 3rd Edition. Randal L. Schwartz, Tom Phoenix. O'Reilly. 0-596-00132-0
- Programming Perl, 3rd Edition. Larry Wall, Tom Christiansen, Jon Orwant. O'Reilly. 0-596-00027-8
- Programming the Perl DBI. Alligator Descartes, Tim Bunce. O'Reilly. 1-56592-699-4
- HTML & XHTML: The Definitive Guide, 5th Edition. Chuck Musciano, Bill Kennedy. O'Reilly. 0-596-00382-X
- Writing Apache Modules with Perl and C. Lincoln Stein, Doug MacEachern. O'Reilly. 1-56592-567-X
- Embedding Perl in HTML with Mason. Dave Rolsky and Ken Williams. O'Reilly. 0-59600-225-4

... and any number of other books from O'Reilly, John Wiley and others.

Other resources

- World Wide Web Security FAQ: `http://www.w3.org/Security/faq/www-security-faq.html`
- Apache Tutorial: Dynamic Content with CGI: `http://httpd.apache.org/docs-2.0/howto/cgi.html`
- Apache Module mod_cgi: `http://httpd.apache.org/docs-2.0/mod/mod_cgi.html`
- Apache suEXEC Support: `http://httpd.apache.org/docs-2.0/suexec.html`
- The mod_perl website: `http://perl.apache.org/`
- The Mason website: `http://www.masonhq.com/`

- The HTML::Mason man pages on any system with Mason installed - start with
  `man HTML::Mason::Devel`