CGI Scripting for Programmers: Introduction

Example Programs and Reference

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

June 2005

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 CGI 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/
```

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.

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";
```

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 7 – 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 8 – 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 9 – 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 10 – 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 11 – 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 12 – 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 13 – 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;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 14 – 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 15 – 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;

}
```

CGI Headers Revisited

We take a detailed look at the three special CGI headers – Content-Type, Location, Status.

Example 16 – random.cgi

This program returns a random image by reading an image file and returning its content marked with an appropriate content type.

```
#!/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 "Content-type: image/png\n";
print "\n";

binmode STDOUT;
open (IMAGE, $lucky_one)
  or die "Failed to open image $lucky_one: $!";
while (read(IMAGE, $buffer, 4096)) {
  print $buffer;
}

close IMAGE;
```

Example 17 – random2.cgi

This program has the same effect as random.cgi, but does it 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 18 – 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 19 – 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";
}
```

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.

Configuring Webservers

How to configure Apache and IIS to execute CGI programs, and the dangers of allowing users on a multi-user machine to run their own CGI programs.

Debugging CGIs

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

Example 20 – 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";
```

Templating

The benefits of using a templating system; some example systems.

Example 21 – template.ttml

An example template using the template Toolkit language.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<head>
<title>Congratulations!!</title>
</head>

<body>

<h1>Congratulations [% name FILTER html %]</h1>

<p>Congratulations [% name FILTER html %], we are pleased
to tell you that you have just been allocated
$[% value FILTER html %] in our prize draw. All you need
to do is contact us at our address below to claim your prize.
</p>

<p>
[% FOREACH line = address -%]
[% line FILTER html %]<br />
[% END -%]
</p>

</body>
</html>
```

Example 22 – template.cgi

A program to invoke template.ttml.

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

use Template;

my $data = { name => 'Jon Warbrick',
             value => "1,000,000",
             address => ['123, The Street', 'Anytown', 'Aynwhere',
                         'ZZ1 1ZZ']
           };

my $tt = Template->new or
      die "Failed to create new template: " . Template->error();

my $html;
$tt->process("template.ttml",$data,\$html) || die $tt->error();

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

print $html;
```

Sending email

Sending email is a common requirement for a CGI program, but it's surprisingly hard.

Example 23 – Net-SMTP.pl

Using the Perl Net::SMTP module with ppsw.cam.ac.uk as a smarthost to send mail.

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

use Net::SMTP;

my $from    = 'jw35@cam.ac.uk';
my $to      = 'jon.warbrick@ucs.cam.ac.uk';
my @message = ("From: $from",
               "To: $to",
               "Subject: A test message",
               "",
               "Hello World!");
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";
  foreach my $line (@message) {
    $smtp->datasend("$line\n") or die "datasend";
  }
  $smtp->dataend()            or die "dataend";
  $smtp->quit()               or die "quit";
};
if ($@) {
  die "Message not sent: $@ failed\n";
}
```

Example 24 – sendmail.cgi

Using a local copy of sendmail to send mail.

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

$ENV{PATH} = $ENV{BASH_ENV} = '';

my $from    = 'jw35@cam.ac.uk';
my $to      = 'jon.warbrick@ucs.cam.ac.uk';
my @message = ("From: $from",
               "To: $to",
               "Subject: A test message",
               "",
               "Hello World!");

open(SENDMAIL, "|/usr/sbin/sendmail -oi -t")
  or die "Failed to open sendmail: $!\n";

foreach my $line (@message) {
  print SENDMAIL "$line\n";
}

close SENDMAIL or warn $! ? "Error closing sendmail pipe: $!\n"
                          : "Error $? from sendmail pipe";
```

Maintaining State

The options for maintaining state in the otherwise stateless world of HTTP and CGI

Example 25 – loan.cgi

A longish example of how you can store state in hidden fields.

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

# This program from 'The Official Guide to Programming with
# CGI.pm', Lincon Stein, 1998

# script: loan.cgi
use CGI qw/:standard :html3/;

# this defines the contents of the fill out forms
# on each page.
@PAGES = ('Personal
Information','References','Assets','Review','Confirmation');%FIELDS =
('Personal Information' => ['Name','Address','Telephone','Fax'],
           'References'           => ['Personal Reference 1','Personal
Reference 2'],
           'Assets'               => ['Savings Account','Home','Car']
           );
# accumulate the field names into %ALL_FIELDS;
foreach (values %FIELDS) {
    grep($ALL_FIELDS{$_}++,@$_);
}


# figure out what page we're on and where we're heading.
$current_page = calculate_page(param('page'),param('go'));
$page_name = $PAGES[$current_page];

print_header();
print_form($current_page)         if $FIELDS{$page_name};
print_review($current_page)       if $page_name eq 'Review';
print_confirmation($current_page) if $page_name eq 'Confirmation';
print end_html;

# CALCULATE THE CURRENT PAGE
sub calculate_page {
    my ($prev,$dir) = @_;
    return 0 if $prev eq '';    # start with first page
    return $prev + 1 if $dir eq 'Submit Application';
    return $prev + 1 if $dir eq 'Next Page';
    return $prev - 1 if $dir eq 'Previous Page';
}

# PRINT HTTP AND HTML HEADERS
sub print_header {
    print header,
    start_html("Your Friendly Family Loan Center");
}

# PRINT ONE OF THE QUESTIONNAIRE PAGES
sub print_form {
    my $current_page = shift;
    print "Please fill out the form completely and accurately.",
       start_form,
       hr;
    draw_form(@{$FIELDS{$page_name}});
    print hr;
    print submit(-name=>'go',-value=>'Previous Page')
```

```
        if $current_page > 0;
    print submit(-name=>'go',-value=>'Next Page'),
       hidden(-name=>'page',-value=>$current_page,-override=>1),
       end_form;
}

# PRINT THE REVIEW PAGE
sub print_review {
    my $current_page = shift;
    print "Please review this information carefully before submitting it.
",
       start_form;
    my (@rows);
    foreach $page ('Personal Information','References','Assets') {
        push(@rows,th({-align=>'left'},em($page)));
        foreach $field (@{$FIELDS{$page}}) {
            push(@rows,
                 TR(th({-align=>'left'},$field),
                    td(param($field)))
                 );
            print hidden(-name=>$field);
        }
    }
    print table({-border=>1},caption($page),@rows),
       hidden(-name=>'page',-value=>$current_page,-override=>1),
       submit(-name=>'go',-value=>'Previous Page'),
       submit(-name=>'go',-value=>'Submit Application'),
       end_form;
}

# PRINT THE CONFIRMATION PAGE
sub print_confirmation {
    print "Thank you. A loan officer will be contacting you shortly.",
       p,
       a({-href=>'../source.html'},'Code examples');
}


# CREATE A GENERIC QUESTIONNAIRE
sub draw_form {
    my (@fields) = @_;
    my (%fields);
    grep ($fields{$_}++,@fields);
    my (@hidden_fields) = grep(!$fields{$_},keys %ALL_FIELDS);
    my (@rows);
    foreach (@fields) {
        push(@rows,
             TR(th({-align=>'left'},$_),
                td(textfield(-name=>$_,-size=>50))
                )
             );
    }
    print table(@rows);

    foreach (@hidden_fields) {
        print hidden(-name=>$_);
    }
}
```

Example 26 – cookie.cgi

An example of storing state in cookies.

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

# This program from 'The Official Guide to Programming with
# CGI.pm', Lincon Stein, 1998
```

```
use CGI qw/:standard :html3/;

# Some constants to use in our form.
@colors=qw/aqua black blue fuschia gray green lime maroon navy olive
    purple red silver teal white yellow/;
@sizes=("<default>",1..7);

# recover the "preferences" cookie.
%preferences = cookie('preferences');

# If the user wants to change the background color or her
# name, they will appear among our CGI parameters.
foreach ('text','background','name','size') {
    $preferences{$_} = param($_) || $preferences{$_};
}

# Set some defaults
$preferences{'background'} = $preferences{'background'} || 'silver';
$preferences{'text'} = $preferences{'text'} || 'black';

# Refresh the cookie so that it doesn't expire.
$the_cookie = cookie(-name=>'preferences',
                     -value=>\%preferences,
                     -path=>'/',
                     -expires=>'+30d');
print header(-cookie=>$the_cookie);

# Adjust the title to incorporate the user's name, if provided.
$title = $preferences{'name'} ?
    "Welcome back, $preferences{name}!" : "Customizable Page";

# Create the HTML page.  We use several of the HTML 3.2
# extended tags to control the background color and the
# font size.  It's safe to use these features because
# cookies don't work anywhere else anyway.
print start_html(-title=>$title,
                 -bgcolor=>$preferences{'background'},
                 -text=>$preferences{'text'}
                 );

print basefont({-size=>$preferences{size}}) if $preferences{'size'} > 0;

print h1($title);

# Create the form
print hr,
    start_form,

    "Your first name: ",
    textfield(-name=>'name',
              -default=>$preferences{'name'},
              -size=>30),br,

    table(
          TR(
             td("Preferred"),
             td("Page color:"),
             td(popup_menu(-name=>'background',
                           -values=>\@colors,
                           -default=>$preferences{'background'})
                ),
             ),
          TR(
             td(''),
             td("Text color:"),
             td(popup_menu(-name=>'text',
                           -values=>\@colors,
```

```
                        -default=>$preferences{'text'})
                )
            ),
        TR(
            td(''),
            td("Font size:"),
            td(popup_menu(-name=>'size',
                        -values=>\@sizes,
                        -default=>$preferences{'size'})
                )
            )
        ),

    submit(-label=>'Set preferences'),
    end_form,
    end_html;
```

The Perl DBI

How to interface to a database from Perl

Example 27 – lotr.cgi

A simple Perl DBI program that extracts and displays data from a database.

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

use CGI qw/:standard :html3 center/;
use DBI;

print header,
      start_html (-title=>"The characters");

my %attr = ( RaiseError => 1,
             PrintError => 0,
             AutoCommit => 1,
           );
my $dbh = DBI->connect("DBI:SQLite:dbname=lotr",
                    "user", "pwd", \%attr);

do_list() if param;
do_form();

$dbh->disconnect;
print end_html;

sub do_list {

    my $race = param('race');
    my $select = '';
    $select = 'AND race.id = ' . $dbh->quote($race)
                                          if ($race =~ /^\d$/);

    my $sth = $dbh->prepare ("SELECT characters.name, race.name
                              FROM characters, race
                              WHERE characters.race = race.id
                              $select
                              ORDER BY characters.name");
    $sth->execute;
    my $results = $sth->fetchall_arrayref;
    print center(
            h1("Characters"),
            table({border=>1},
              Tr({align=>"center"},
                [ th( [ 'Name', 'Race' ] ),
                  map { td($_) } @$results
                ]
              )
            )
          );

}

sub do_form {

    my $sth = $dbh->prepare ("SELECT name, id
                              FROM race
                              ORDER BY name");
    $sth->execute;

    my @values = ('*');
    my %labels = ('*' => 'All');
```

```
    while ( my ($name, $race) = $sth->fetchrow_array) {
        push @values,$race;
        $labels{$race}=$name;
    }

    print center(
              start_form,
              p(
                "Chose a Middle Earth race: ",
                br,
                popup_menu(-name=>'race',
                           -values=>\@values,
                           -labels=>\%labels),
                submit,
              ),
              end_form,
          );

}
```

Caching

What CGI authors need to know about web caching.

Example 28 – caching.cgi

A modified version of date.cgi from earlier which allows its output to be cached for 30 seconds.

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

use CGI qw/:all/;

my $now = localtime();

print "Content-type: text/html; charset=iso-8859-1\n";
print "Cache-control: max-age=30\n";
print "\n";

print "<html>\n";

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

path_info

How to use the information provided in path_info, and the dangers of ignoring it.

Example 29 – bottomless.cgi

A demonstration of how failing to handle additional path information can create a document tree of infinite length.

```
#!/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 Bottomless document tree</title>\n";
print "<meta name='robots' content='index,nofollow' />\n";
print "</head>\n";

print "<body>\n";
print "<h1>A Bottomless document tree</h1>\n";
print "<p>Here we have a <a href='tar/pit.html'>relative\n";
print "link</a>.</p>\n";
print "</body>\n";

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

File Uploads

How it's possible to upload entire files from HTML forms

Example 30 – upload.html

An example upload form

```
<html>
<head>
<title>Upload Example</title>
</head>

<body>
<h1>Upload Example</h1>

<p>Upload a file:</p>

<form method="post" action="upload.cgi"
      enctype="multipart/form-data">
<p>Save as: <input type="text" name="save_as" /></p>
<p><input type="file" name="upload" value="" size="60" /></p>
<p><input type="submit" name="submit" value="Upload File" /></p>
</form>

</body>
</html>
```

Example 31 – upload.cgi

A form that processes submissions from upload.html.

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

use CGI qw/:standard :html3/;

$CGI::DISABLE_UPLOADS = 0;
$CGI::POST_MAX        = 1024 * 1024;

print header,
      start_html('File upload'),
      h1('File upload');

print_results();

print end_html;

sub print_results {

  my $length;
  my $file = param('upload');
  if (!$file) {
    print "No file uploaded.";
    return;
  }
  print p(
          b('Save as:'),
          escapeHTML(param('save_as')),
        ),
        p(
          b('Uploaded file name:'),
          escapeHTML($file)
        ),
```

```
        p(
          b('File MIME type:'),
          escapeHTML(uploadInfo($file)->{'Content-Type'})
      );
  my $fh = upload('upload');
  while (<$fh>) {
    $length += length($_);
  }
  print p(
          b('File length:'),
          $length
        );
}
```

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

... 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