Working with Perl modules

Lesson 2

CGI Programming

Automated Creation of Web Pages

- CGI.pm provides many methods to simplify the creation of web pages out of perl scripts
- Module contained in core perl
- Main application is creation of dynamic web pages
- Main features
  - processing of CGI parameter lists
  - supports new HTML or arbitrary XML tags
  - not restricted to usage in CGI scripts
  - support for forms, cookies, style sheets

Other Perl based Tools for HTTP

- Modules for integration into apache (Apache::xxx)
  - (not covered, see L.Stein, Writing Apache Modules with perl and C, O'Reilly 1999)
- Other Module families for WWW programming
  - LWP: API for usage of the HTTP protocol (libwww)
  - URI: Dealing with Uniform Resource Locators (URL)
  - HTML: Analysis and processing of HTML pages
  - many more modules in the categories XML, CGI, …
- Perl scripts for downloading and mirroring
  - lwp-mirror, lwp-download, lwp-request, lwp-rget, w3mir

CGI.pm Basics

- CGI.pm methods bound to a CGI object

  `use CGI; $q=new CGI; print $q->start_html;`
- Usage inconvenient, import of the methods as functions using the tags defined within CGI.pm

  `use CGI qw( :standard );# :html3 for tables`

  `print start_html; # same as object call`
- Nearly all HTML tags have function equivalent

  `<H1> => h1(); <UL>…</UL> => ul(…);`

  all upper/lower case variations equivalent: `Ul()`, `UL()`

  there is already a `tr` therefore use `Tr()` for `<TR>`

CGI.pm Basics (2)

- Functions for new tags can be provided easily

```
use CGI qw( :standard new);# function new
print new('text'); # yields <NEW>text</NEW>
```

- Start and end tags can be generated separately by functions **`start_xxx`** and **`end xxx`**
  - at least one of the functions has to be imported

```
use CGI qw( :standard start_ul);
print start_ul,'text',end_ul;#<UL>text</UL>
```

HTML Syntax Conversion

- HTML attributes get converted to anon hash (arg 1)

```
<H1 ALIGN="LEFT"> =>   h1({-align=>left});
```

- HTML contents can be filled into further arguments

```
<H1>a test</H1>   =>   h1('a','test');
```

- HTML lists can be bundled into one function call

`li('red'), li('yellow'), li('blue')` becomes

`li(['red', 'yellow', 'blue'])`

Introductory Example

```
use CGI qw /:standard/;
print start_html,
      h1("a first test"),
      end_html;
```

becomes

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML><HEAD><TITLE>Untitled Document</TITLE>
</HEAD><BODY><H1>a first test</H1></BODY>
</HTML>
```

Web Page Generation

- Offline creation of pages using a perl script with CGI.pm by **`perl`** ***`script_name`*** **`1`**
  - parameter required, otherwise console input expected
- CGI.pm for creation of dynamic pages (CGI script)
  - script has to be executable by webserver account
  - Webserver has to be configured properly
    - directory for CGI scripts with proper access control
    - naming schema for CGI scripts (e.g. extension .pl)
  - script has to fulfil security requirements (taint safe)

HTTP Protocol Issues

- First example as content of a HTML page o.k.
- Not suited as CGI script, server and browser need to read HTTP header
- HTTP Header contains status code and other fields
  - gets added by the **`header`** method

```
print header; #prints Content-type: text/html
```

- Several HTML features can be controlled using arguments in the **`header`** call
  - content type, language, expiration, authentication,...

HTTP Header Generation

- Header generation controlled by anonymous hash as first (and only) parameter for the **`header`** method
  - -type, -expires, -status, -content-encoding etc.
- To generate a header that forces a password dialog and sets expiration time of one day:

```
print header(-type => 'text/html',
     -expires => '+1d',
     -status => '401 Authentication required'
    '-auth-type' => 'Basic');
```

Debugging

- CGI error messages are written to the web server logfile, usually restricted access to that file
- Several options to debug without the error logfile
  - offline testing using options -w and -T and under **`use strict;`** pragma
  - process critical parts under **`eval`** control, report errors (contained in `@!`) to users
  - use CGI::Carp to redirect errors to browser **`use CGI::Carp qw(fatalsToBrowser);`**

Creating a dynamic HTML Page

```
print header,                        # output Content-Type:
      start_html({-title =>'HTML example',
                  -bgcolor =>'gray'}),
      h2('a list demo'),
      ul(li([i('italic'),            # anon array for list
             b('bold'),
             tt('fixed width font')])),
      hr, p,
      h2('link generation'),
      'Start page of ',
      a({href=>'http://www.desy.de'},'DESY'),
      end_html;                      # separate start/end tags
```

Tables

- Created with the **`table`**, **`Tr`**, **`th`** and **`td`** functions
- Creation in one go sometimes difficult
    - subdivide task (see example on next page)
- Can be done nevertheless (elegant but hard to read)

```
#example from M.Schilli, Linux Magazin 3/98 (in German)
$content = [ ["column 1", "column 2", "column 3"],
             [1,2,3], [4,5,6], [7,8,9] ];
print table(Tr(map {th($_)} shift @$content), "\n",
            map {Tr(map {th($_)} $_)."\n"} @$content);
```

Tables, the easy way

```
#example from M.Schilli, Linux Magazin 3/98 (in German)
use CGI qw /:standard :html3/;
print header;
print start_html, hr, h2('A table demo');
foreach $row (1..2) {
    $rowcontent = "";
    foreach $col (1..2) { $rowcontent .= td("field $row/$col"); }
    $tablecontent .= Tr($rowcontent) . "\n";
}
print table(
    {-border => 1, -bgcolor => 'orange'}, "\n",
    Tr(th("column 1"), th("column 2")), "\n",
    $tablecontent
);
```

Forms

- Enclosed in the `start_form` and `end_form` functions
- Widgets available as function calls

popup_menu	radio_group	
textfield	textarea	
scrolling_list	checkbox_group	
checkbox	submit	reset

- For an exhaustive example see demo in Mod2.pl

A simple forms demo

```
use CGI qw/:standard :html3/;
print start_form,
    checkbox('-name'     => 'color',
             '-checked' => 'checked',
             '-value'   => 'yes',
             '-label'   => 'Yes?'),
     submit('-name'  => 'submit_button',
            '-value' => 'Send'),
     reset(),
     end_form;
```

Parameter Processing

- Function **`param(`*`key`*`)`** returns CGI parameter ***`key`*** passed to the script
- Function without args retrieves **all** parameters
- ***`Key`*** corresponds to **`-name`** attribute in form

```
 if (defined param('color')) {
     $color = param('color'); ...
   }
```

- depending on context **`param`** returns scalar or array

```
 @colors = param('color');
```

Remembering state

- The HTTP protocol is stateless
- Remembering the state between subsequent invocations of a script requires additional tools
  - cookies, generated by the server, held in the browser
  - hidden fields in parameter passing, not displayed
  - combination of hidden field parameters or cookies that act as keys and a database on the server to look up the values
- Cookies can be switched off, hidden fields can be suppressed by explicitly CGI with parameters

Remembering state (2)

- Hidden fields
  - get transferred using **`hidden()`**
- Cookies
  - query values using **`$val=cookie(-name=>'`*`id`*`');`**
  - retrieve the names of all cookies: **`@ids=cookie();`**
  - set using **`cookie(-name=>'`*`id`*`',-value=>'val');`**
- Very few cookies per visited site are good practice
  - use **`-domain`** attribute to have site wide cookies
  - further attributes like expiry date should also be set

A cookie example

```
use CGI qw/:standard/;
if(defined ($id=cookie(-name => 'ID'))) { # Cookie is set!
   print header();
   print b("Welcome back, customer with ID $id!");
} else {
   # new customer
   $id = unpack ('H*', pack('Nc', time, $$ % 0xff));

   $cookie = cookie('-name'    => 'ID',
                    '-value'   => $id,
                    '-expires' => '+1h',
                    '-domain'  => '.ifh.de');
   print header('-cookie' => $cookie);
   print b("Welcome, you get customer ID $id");
}
```

Incremental Updates (NPH Scripts)

- Incremental updates done by NPH scripts
  - perform a server push
  - perl output has to be unbuffered
  - header attributes determine output in parts
  - support by module CGI::Push
  - **not useable**, incompatible with HTTP/1.1 and SSL

Dynamic updates using Client pull

- Refresh algorithm of server gets used
  - CGI generates header with **`-refresh=>$time`** attribute and URL pointing to itself
  - Script gets called again after **`$time`** seconds
  - refresh cycle is stopped when header is called without new **`-refresh`** attribute
  - Parameter passing within URL possible

Dynamic Updates Example

```
use CGI qw/:standard/;
$time = 1;
$count = param('count');
$count ||= 10;    $count--;
if ( $count ) {
  print header(-refresh => "$time; URL=$ENV{SCRIPT_NAME}?
   count=$count");
  my $date = localtime(time);
  print start_html('test'), h1($date),"\n", end_html;
} else {
  print header,
        start_html('no further testing, sorry'),
        h1('the clock is broken now'), end_html;
}
```

Topics not covered

- There is support for the following topics in CGI.pm
  - URL redirection (better done with mod_perl)
  - Cascading Style Sheet
  - Javascript Support
  - Image Maps
  - Frames

Literature

- Official Guide to Programming with CGI.pm, Lincoln Stein, Wiley (1998)
- **`perldoc CGI`**
- http://stein.cshl.org/~lstein/talks/perl_conference/cute_tricks
- http://www.perl.com/reference/query.cgi?cgi