B336 Advanced Internet Computing

Implementing a Web Client

Learning Objective

- Understand the basic requirements in implementing a web client.
- Use the LWP library in Perl to write your own simple web client.

Lecture Outline

- Preliminary object-oriented concepts in Perl
- Perl's LWP library
- A Basic Web Client
- HTTP::Request, HTTP::Response, and LWP::UserAgent

A Basic Web Client

- Here’s your solution to Ass 1 part III …well, almost!

```
#!/usr/bin/perl

# file get_url.pl
# Figure 9.1: Fetch a URL using LWP’s object-oriented interface

use strict;
use LWP;

my $url = shift;

my $agent    = LWP::UserAgent->new;
my $request  = HTTP::Request->new(GET => $url);

my $response = $agent->request($request);
$response->is_success or die "$url: ",$response->message,"\n";

print $response->content;
```

A few things about Perl...

- Before we delve into the example, a few things about Perl to make the syntax a bit more understandable:
    - Accessing classes in libraries and packages.
    - Syntax for object-oriented programming in Perl.

Accessing Elements in Packages

- To access anything within a certain package (or in a module), we do two things:
  - Load the package using **use** or **require**.
  - Use the double colon to refer to package.
- Eg.

```
use MyPackage ;
$x = MyPackage::my_subroutine ;
$y = MyPackage::$my_variable ;
```

Accessing Elements in Packages

- Sometimes you may see multiple levels of dereferencing:

```
MyPackage::ASubPackage::mysubroutine ;
```

- A module is basically a package where the file name is the same as the package name.

- A library is a more generic term which could mean a package, a module, a set of modules, etc.

Classes in Packages

- The predefined classes in Perl are stored as packages.

- So, to access a class in one of those packages, we use something like:

```
MyPackage::MyClass
```

Syntax for OO in Perl

- Perl object-oriented features are built up from its references, packages and subroutines.
    - An object is basically a reference.
    - Classes is basically comes in packages.
    - A method is basically a subroutine in a package.

Creating Objects in Perl

- To create a new object from a class, we use something like:

```
$new_obj = MyPackage::MyClass->new ;
```

or

```
$new_obj = MyPackage::MyClass->new (init_data =>
                                    “value”) ;
```

Assign newly created object to variable

Call the constructor method for the class

Set a data member of the class to a value

Invoking Methods of Objects

- To get an object to perform one of its methods, we use something like:

```
MyPackage::MyClass->my_method(param1 => “value”)
          or die (“Error Message.”) ;
```

The LWP Library

- The Library for Web access in Perl is a set of modules which provides a unified API for interacting with Web (and other important Internet services).

- The library contains a set of classes to support:
    - requesting documents from web servers
    - posting data to a web server
    - parse HTML documents
    - handle cookies
    - ...

The LWP Library

- Besides the main LWP module, the LWP distribution file (`libwww*`) usually contains a whole lot of other important modules as well, such as HTTP.

The Basic Web Client

- Let’s look at our basic web client again.
- It does something very simple, which all web browsers do:
  - fetch the document specified by a URL.
  - It uses HTTP to do the transport - this means it will only be able to communicate with a HTTP server.
- To keep things simple, it reads the URL from the command line, and prints the URL to the command line.

The Basic Web Client

- The basic structure of the script is:

```
<COMMENTS - don't care for now!>

<LOAD MODULES - take for granted for now!>

<DETERMINE URL GIVEN TO THE SCRIPT>

<CREATE A USER-AGENT CAPABLE OF COMMUNICATING WITH A HTTP SERVER>
<CREATE A REQUEST OBJECT USING THE URL>

<HAVE THE USER_AGENT USE THE REQUEST OBJECT TO GET A RESPONSE>

<IF THE RESPONSE IF SUCCESSFUL, DISPLAY THE CONTENTS OF THE
RESPONSE>
```

HTTP::Request

- When sending out a HTTP request, you must at least specify:
    - the method (GET, POST, HEAD, etc)
    - the URL (or the more general term URI, Universal Resource *Identifier*) you want to retrieve.

- Refer back to lectures in week 1 for information about methods and URLs.

Basic syntax for creating a new HTTP::Request object

```
$r = HTTP::Request->new($method, $uri, [$header, [$content]]) ;
```

- Eg.

  $request = HTTP::Request->new
  (HEAD, ‘http://red.murdoch.edu.au’) ;

  $request = HTTP::Request->new
  (GET => ‘http://red.murdoch.edu.au:15678’) ;

Basic syntax for accessing the headers of a HTTP::Request object

```
@values = $r->header($field1)
```

- Eg.

```
$request = HTTP::Request->new
      (GET => ‘http://www.murdoch.edu.au’) ;
@values = $request->header(“Content-type”) ;
```

To go through every header in a HTTP::Request object

```
$request->scan (\&sub)
```

- Eg.

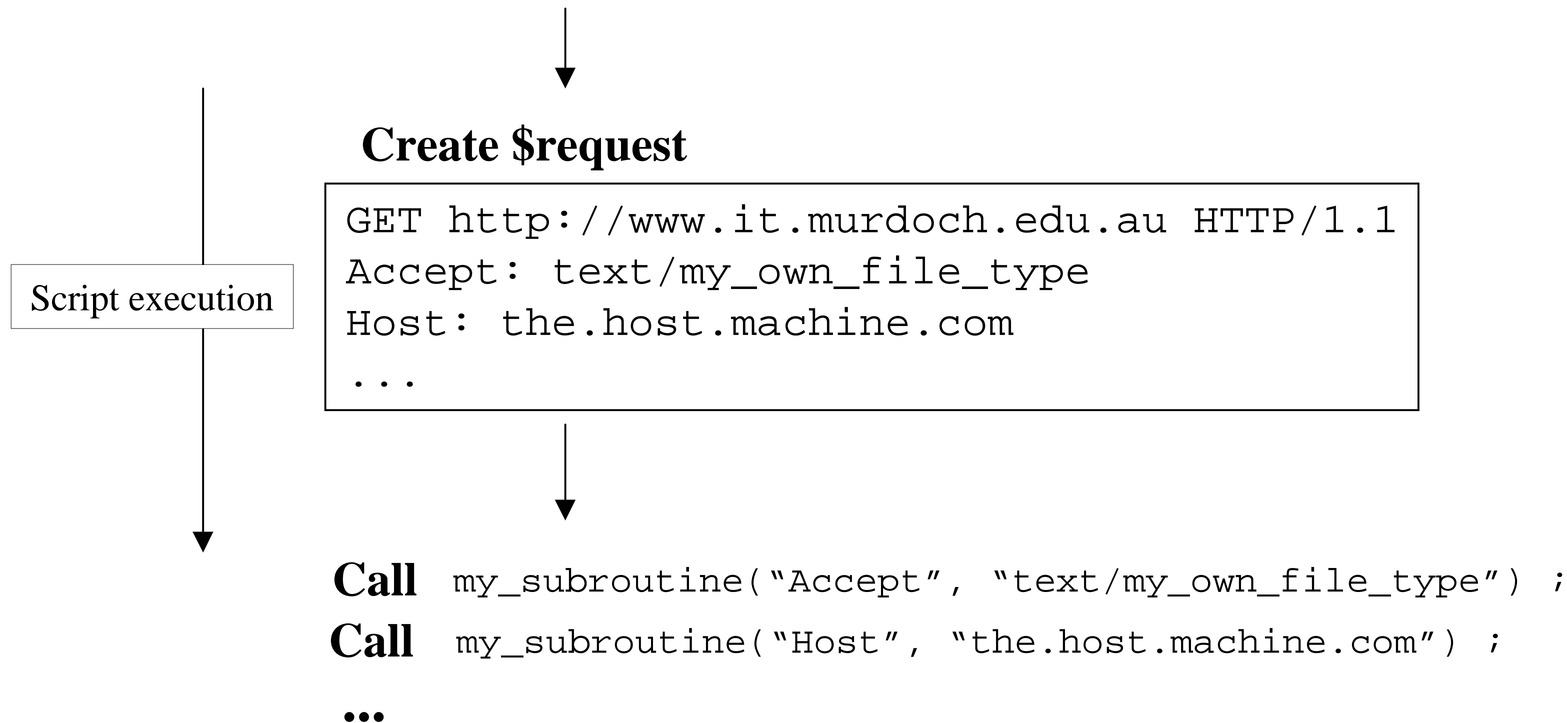
```
$request = HTTP::Request->new
                        (GET => ‘http://www.murdoch.edu.au’) ;
$request->header(Accept => ‘text/my_own_file_type’) ;
$request->header(Host => ‘the.host.machine.com’) ;

$request->scan (\&my_subroutine) ;

sub my_subroutine {
        my ($fieldname, $fieldvalue) = @_ ;
        …
}
```

HTTP::Request->scan

- What happens to the previous example script?



Note: what `my_subroutine` does with the parameters is for you to define yourself.

Common HTTP::Response Status

- See page 254 or section THREE in the Unit Reader.

To get data about the status line of the HTTP::Response

```
$str = $response->status_line ;
$code = $response->code ;
```

- Eg.

```
if (($response->code < 300) && ($response->code > 299))
{
        …
}
```

Another way to check the status of the HTTP::Response

- Eg.

```
if ($response->is_success) {
        …
} elsif ($response->is_redirect) {
        …
```

Header information in the HTTP::Response object

- Since both HTTP::Request and HTTP::Response both contain the HTTP::Header object, all the header manipulation methods we mentioned before is relevant for HTTP::Response as well.

LWP::UserAgent

- The LWP::UserAgent class is responsible for
  - submitting HTTP::Request objects to HTTP servers, and
  - receiving and encapsulating the response.

A possible additions to the basic client

- To repeated request from a certain web server
  - for stress testing

```perl
#!/usr/bin/perl

use strict ;
use LWP ;

my $url = shift ;

my $agent = LWP::UserAgent->new ;
my $request = HTTP::Request->new(GET => $url) ;

my $n = 0 ;
while (1) {
  sleep 1 ;
  my $response = $agent->request($request) ;
  if ($response->is_success) {
    print "Request #$n from $url\n" ;
    $n++ ;
  } else {
    die "$url: ", $response->message,"\n" ;
  }
}

# print $response->content ;
```

An Even Simpler Solution

- Does everything `get_url.pl` does!

```
#!/usr/bin/perl

use LWP::Simple;

my $url = shift;
getprint($url);
```

- But doesn’t allow you to manipulate the requests and responses.

Is there more to the Web Client?

- Do not be deceived into believing that what we saw today is all that a web client does. It does A LOT MORE.
  - And not just fancy graphic displays.
  - Look at some of the other example code in section 3 of your Unit Reader for other things a web client should handle.

- However, what we saw today does serve to illustrate one of the most important aspects of a web client: the ability to create a HTTP request, and to receive and handle the HTTP response.