



B336 Internet Systems Programming

Implementing a Web Client

(Week 3 Lectures)



Lecture Objectives

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



Lecture Objectives

- Relevance to unit objectives:
 - Learning objective 2: Writing software

- Relevance to assessments:
 - Lab week 4
 - assignment 1



Lecture Outline

- Perl's LWP library
- A Basic Web Client
- HTTP::Request, HTTP::Response, and LWP::UserAgent



The LWP Library

- The Library for Web access in Perl (LWP)
 - a set of modules which provides a unified API for interacting through the Web.
 - Also includes other important Internet services like FTP.
- 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
 - etc.



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.



A Basic Web Client (get_url.pl)

```
#!/usr/bin/perl

# file get_url.pl
# 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;
```



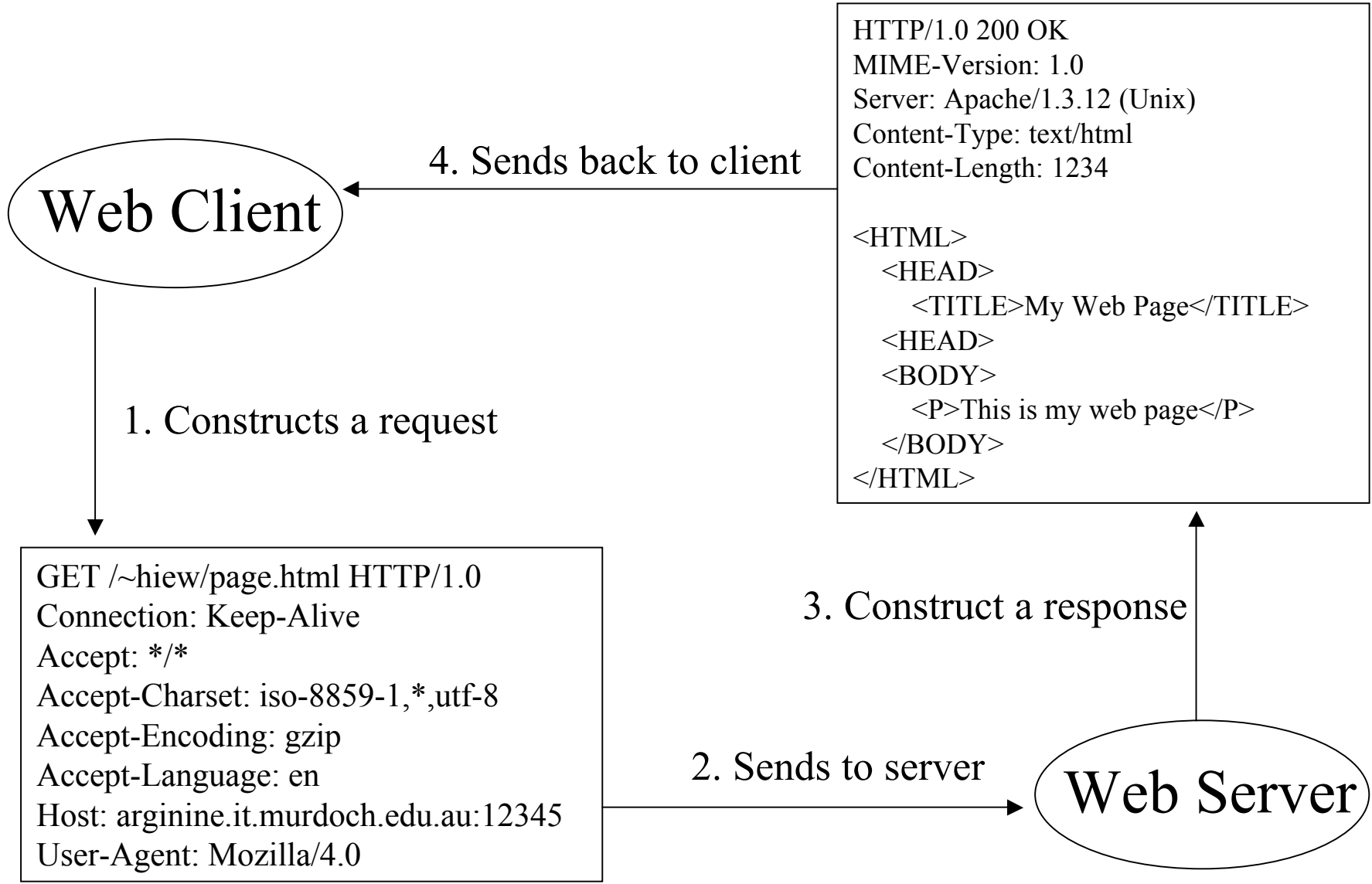
Acknowledgement

- The `get_url.pl` script was written by Lincoln Stein (a very influential Perl programmer).
- It was published in the book “Network Programming with Perl”.
 - Sections of the book is available in your Unit Reader.



The Basic Web Client

- The script does something very simple, which all web browsers do:
 - fetch the resource specified by a URL.
 - Send a HTTP request message, then
 - Wait for a response, then
 - Process the response when it arrives
 - It uses HTTP to do the transport - this means it will only be able to communicate with a HTTP server.





The Basic Web Client

- The basic structure of the script (from page 5) 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 IS 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://www.murdoch.edu.au') ;
```

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



Request headers + content in a string

```
$request->as_string ;
```

- Eg.

```
$request = HTTP::Request->new  
    (HEAD, 'http://www.murdoch.edu.au') ;  
print $request->as_string ;
```



Accessing the headers of a HTTP::Request object

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

- Eg.

```
$request = HTTP::Request->new  
    (GET => 'http://www.murdoch.edu.au') ;  
my @values = $request->header('Content-Type') ;
```



Header values in an array

- The standard return value for the header() method is as an array of strings.
 - This is because the standard HTTP header can potentially contain more than one value.
 - If a header contains only one value, it will be in the first element of the array.



All Request Headers in a string

```
$request->headers_as_string ;
```

- Eg.

```
$request = HTTP::Request->new  
    (HEAD, 'http://www.murdoch.edu.au') ;  
print $request->headers_as_string ;
```



To process every header in a HTTP::Request object

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

- **Example script:**

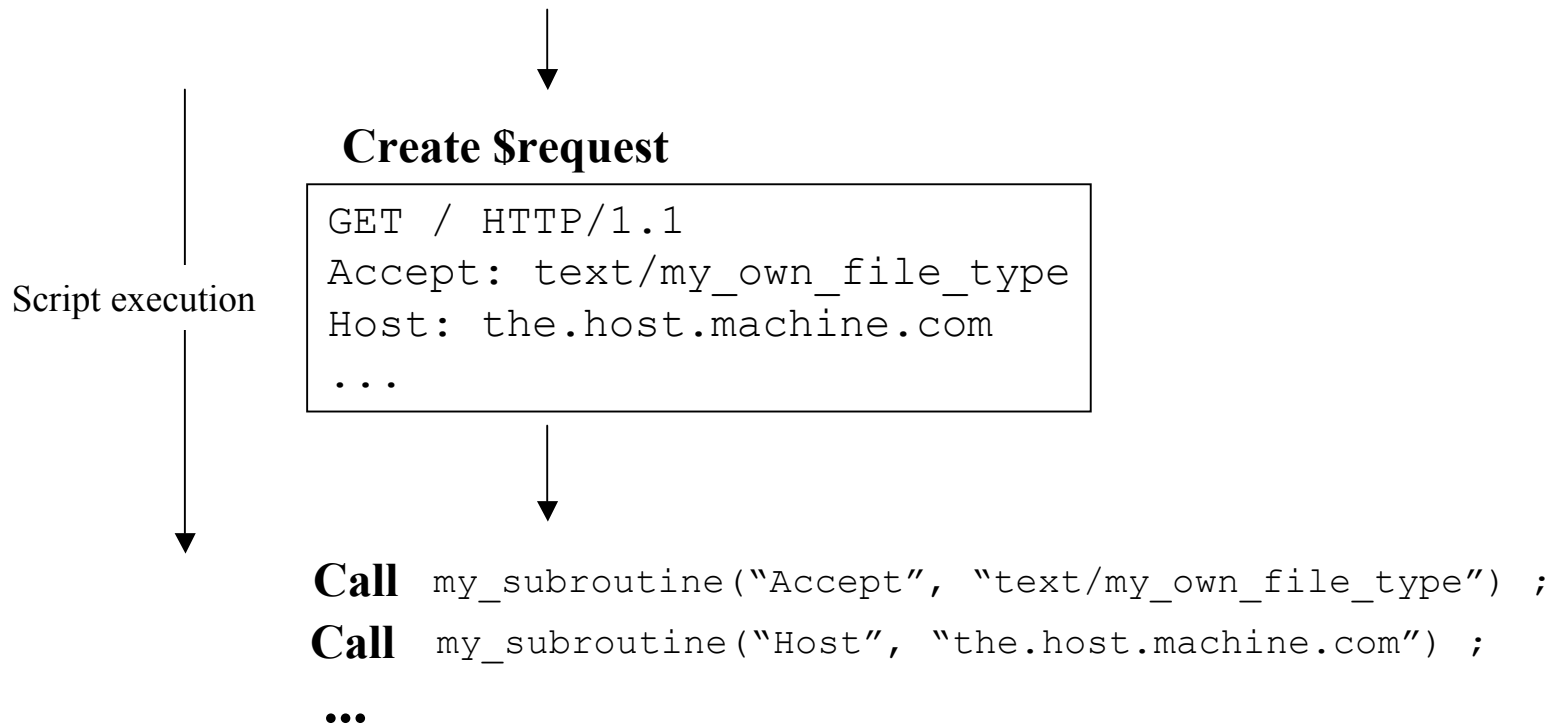
```
$request = HTTP::Request->new
    (GET => 'http://www.murdoch.edu.au') ;
$request->header(Accept => 'text/my_own_file_type') ;
$request->header(Host => 'the.host.machine.com') ;

$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.



HTTP::Response

- HTTP::Request is a class for encapsulating request messages. The equivalent for response messages is HTTP::Response.
- Some common HTTP::Response status
 - See page 254 of Unit Reader 4.



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

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

- Example script:

```
if ( ($response->code < 300) &&  
    ($response->code > 199) )  
{  
    ...  
}
```



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

- Example script:

```
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 for HTTP::Request is relevant for HTTP::Response as well.



Example methods for HTTP::Response

```
print $response->as_string ;
```

```
print $request->headers_as_string ;
```

```
my @svr= $response->header('Server') ;  
print "The name and version of the server is  
      $svr[0]\n" ;
```



LWP::UserAgent

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



Example additions to the basic client

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

```
#!/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 number $n from $url\n" ;
        $n++ ;
    } else {
        die "$url: ", $response->message, "\n" ;
    }
}

# print $response->content ;
```



Additional
code



An Even Simpler Web Client

- The following script 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 a Web Client?

- What we saw today is NOT 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 4 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.



A Graphical User Interface?

- To keep things simple, we will not deal with a graphical user interface (GUI) for our client
 - We will use basic input-output through the command line only.
- Those of you already familiar with GUI programming will find it is not that hard to build a GUI around what we will develop in this unit
 - In Perl, you can add a GUI using windowing toolkits like Tk.
 - Here is an example simple graphical web browser in Java
 - <http://www.cafeaulait.org/slides/gsdw/wcp/>

SimpleWebBrowser.java

```
import javax.swing.text.*;
import javax.swing.*;
import java.io.*;
import java.awt.*;

public class SimpleWebBrowser {

    public static void main(String[] args) {

        // get the first URL
        String initialPage = "http://metalab.unc.edu/javafaq/";
        if (args.length > 0) initialPage = args[0];

        // set up the editor pane
        JEditorPane jep = new JEditorPane();
        jep.setEditable(false);
        jep.addHyperlinkListener(new LinkFollower(jep));

        try {
            jep.setPage(initialPage);
        }
        catch (IOException e) {
            System.err.println("Usage: java SimpleWebBrowser url");
            System.err.println(e);
            System.exit(-1);
        }

        // set up the window
        JScrollPane scrollPane = new JScrollPane(jep);
        JFrame f = new JFrame("Simple Web Browser");
        f.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
        f.getContentPane().add(scrollPane);
        f.setSize(512, 342);
        f.show();
    }
}
```

A simple Web Browser in Java (with GUI)

Ref: <http://www.cafeaulait.org/slides/gsdw/wcp/122.html>

Handling Links for by SimpleWebBrowser.java

LinkFollower.java

```
import javax.swing.*;
import javax.swing.event.*;

public class LinkFollower implements HyperlinkListener {

    private JEditorPane pane;

    public LinkFollower(JEditorPane pane) {
        this.pane = pane;
    }

    public void hyperlinkUpdate(HyperlinkEvent evt) {

        if (evt.getEventType() == HyperlinkEvent.EventType.ACTIVATED) {
            try {
                pane.setPage(evt.getURL());
            }
            catch (Exception e) {
            }
        }
    }
}
```

Ref: <http://www.cafeaulait.org/slides/gsdw/wcp/121.html>



Is a GUI always necessary?

- Sometimes a GUI for a web client is neither necessary nor **desirable**.
 - Many web clients are used manually (eg. spiders used by search engines).
 - Eg. I can call `get_url.pl` from another script, or redirect/pipe the results from `get_url.pl`, etc.
- This same principle applies to keeping the script I/O to standard input and standard output.
 - Avoid adding a text-based interactive user menu, such as those some of you learnt in B102 or B104.



Trying out the scripts

- The scripts given in this lecture is available in the account ~hiew on gryphon. To copy, execute and modify the scripts, do the following:

```
gryphon:~> cp ~hiew/examples/web/* .  
gryphon:~> perl get_url.pl http://www.murdoch.edu.au  
...  
gryphon:~> pico get_url.pl  
...
```



Further Reading

- Required Reading
 - Unit Reader 4: Web Clients
- LWP and HTTP module references
 - In the documentation that comes with your Perl installation.
 - <http://www.perldoc.com/>



In next week's lecture...

- Example perl code for a web server.