



The Kalmanovitz Library and
The Center for Knowledge Management

Library for World Wide Web in Perl

Gilberto da Gente

Bioinformatics Resource Specialist

Wednesday, April 9th 2008

University of California, San Francisco



LWP

- “LWP (short for ‘Library for World Wide Web in Perl’) is a set of Perl modules and object-oriented classes for getting data from the Web.”
- LWP lets you do web automation.
- LWP can fetch web pages, submit forms, authenticate, and extract information from HTML and other file formats.
- Used for both static web pages and dynamic programs

WWW

- Three things made the Web possible:
 - HTML for encoding documents,
 - HTTP for transferring them, and
 - URLs for identifying them.
- To fetch and extract information from web pages
 - Construct a URL for the page you wish to fetch
 - Make an HTTP request for it and decode the HTTP response
 - Parse the HTML to extract information.

A Sample URL

[http://www.ncbi.nlm.nih.gov/blast/Blast.cgi?
CMD=Put&PROGRAM=blastn&DATABASE=
nr&FILTER=L&QUERY=AF123456](http://www.ncbi.nlm.nih.gov/blast/Blast.cgi?CMD=Put&PROGRAM=blastn&DATABASE=nr&FILTER=L&QUERY=AF123456)

CMD

Put : submit a query

PROGRAM

blastn : run BLASTn

DATABASE

nr : search against nr

FILTER

L : turn low complexity filtering on

QUERY

AF123456 : accession, GI, or FASTA

URL encoded characters

- The only characters allowed in the path portions of a URL are the US-ASCII **alphanumeric** characters and these permitted punctuation characters:
 - - _ . ! ~ * ' , : @ & + \$ () /
- For a query component, the same rule holds, except that the only punctuation characters allowed are these:
 - - _ . ! ~ * ' ()

URL encoded characters

- Any other characters must be *URL encoded*,
 - Expressed as a percent sign followed by the two hexadecimal digits for that character. So if you wanted to use a space in a URL, it would have to be expressed as %20, because space is character 32 in ASCII, and the number 32 expressed in hexadecimal is 20.

URI::Escape

- Modular part of LWP that provides the `uri_escape()` function to help you build URLs in Perl

```
use URI::Escape;  
encoded_string = uri_escape(raw_string);
```

- To build the name, age, and country query string:

```
$n = uri_escape("Hiram Veeblefeetzer");  
$a = uri_escape(35);  
$c = uri_escape("Madagascar");  
$query = "name=$n+age=$a+country=$c";  
print $query;
```

```
name=Hiram%20Veeblefeetzer+age=35+country=Madagascar
```

HTTP

- HTTP stands for **Hypertext Transfer Protocol**.
- It's the network protocol used to deliver virtually all files and other data on the World Wide Web.
- It is a server/client protocol: the server has the information, and the client wants it.
- In regular web surfing, the client is a web browser such as Mozilla or Internet Explorer.
- The URL for a document identifies the server
- The server returns either in error ("file not found") or success (in which case the document is attached).

HTTP Request and Response

- An HTTP client opens a connection and sends a request message to an HTTP server;
- The server then returns a response message, usually containing the resource that was requested.
- After delivering the response, the server closes the connection

Format of the request and response messages

- An initial line,
- Zero or more header lines,
- A blank line (i.e. a CRLF by itself)
- An optional message body (e.g. a file, or query data, or query output).

Initial Request/Response Line

- A request line has three parts, separated by spaces:
 - A method name
 - The local path of the requested resource
 - The HTTP version being used
- A response line, called the status line, also has three parts separated by spaces:
 - The HTTP version
 - A response status code
 - The status code description

Initial Line Examples

GET /path/to/file/index.html HTTP/1.0

method name

local path

HTTP version

HTTP/1.0 200 OK

HTTP version

Status code

code description

200

OK

404

Not Found

301

Moved Permanently

302

Moved Temporarily

303

See Other (HTTP 1.1 only)

500

Server Error

Header Lines

- Header lines provide information about the request or response, or about the object sent in the message body.

form "Header-Name: value"

User-agent: Mozilla/3.0Gold

Server: Apache/1.2b3-dev

Last-Modified: Fri, 31 Dec 1999 23:59:59 GMT

Message Body

- An HTTP message may have a body of data sent after the header lines.
- In a request, this is where user-entered data or uploaded files are sent to the server.
- In a response, this is where the requested resource is returned to the client

Examples

```
POST /path/script.cgi HTTP/1.0
```

```
From: frog@jmarshall.com
```

```
User-Agent: HTTPTool/1.0
```

```
Content-Type: application/x-www-form-urlencoded
```

```
Content-Length: 32
```

```
home=Cosby&favorite+flavor=flies
```

```
GET /path/file.html HTTP/1.0
```

```
From: someuser@jmarshall.com
```

```
User-Agent: HTTPTool/1.0
```

```
[blank line here]
```

```
HTTP/1.0 200 OK
```

```
Date: Fri, 31 Dec 1999 23:59:59 GMT
```

```
Content-Type: text/html
```

```
Content-Length: 1354
```

```
<html>
```

```
<body>
```

```
</body>
```

```
</html>
```

GET

- Get is the simplest and most common type of HTTP request.
- Form parameters are supplied in the URL
 - Note that urls have character limitations
- There is never a body to the request.
- In Perl, **LWP::Simple** module's `get()` function takes a URL and returns the body of the document:

```
use LWP::Simple;  
$content = get("http://www.anyserver.com/");  
die "Couldn't get it!" unless defined $content;
```

LWP::Simple

- LWP::Simple has several functions for quickly fetching a document with a get request
- For full access to every part of an HTTP transaction—request headers and body, response status line, headers and body—you have to go beyond LWP::Simple

`get($url)`

Returns undef if it fails

`head($url)`

Returns \$content_type,
\$document_length, \$modified_time,
\$expires, \$server

LWP::Simple

getstore(\$url, \$file)

getprint(\$url)

mirror(\$url, \$file)

```
if (mirror("http://www.sn.no/", "foo") ==  
RC_NOT_MODIFIED) { ... }
```

```
if (is_success(getprint("http://www.sn.no/"))) { ... }
```

is_success(\$rc)

True if response code indicated a successful request.

is_error(\$rc)

True if response code indicated that an error occurred.

POST

- Post is the most robust of HTTP request.
- Form parameters, files, and binary data can be supplied in the message body.
- There are usually extra headers to describe this message body,
 - Content-Type:
 - Content-Length:
- The HTTP response is normally program output, not a static file
- The *request URI* is not a resource to retrieve;
 - it's usually a program to handle the data you're sending.

Get

vs.

Post

- Parameters encoded into the URL being requested.
 - Limited as a result of URL character limitations.
 - Usage can result in really ugly URLs.
 - Can not send files or binary data
 - In Perl use `LWP::Simple`
- Parameters encoded in the message body of the request
 - Able to send files and even binary data.
 - In Perl use `LWP::UserAgent`

LWP::UserAgent

- The LWP::UserAgent is a class implementing a simple World-Wide Web user agent in Perl.
 - It acts like a browser
- It brings together the HTTP::Request, HTTP::Response and the LWP::Protocol classes that form the rest of the core of LWP

```
Use LWP::UserAgent  
$browser = LWP::UserAgent->new;
```

```
$response = $browser->get('http://search.cpan.org/');
```

```
if ($response->is_success) {  
    print $response->content}  
else { die $response->status_line; }
```

LWP::UserAgent Parameters

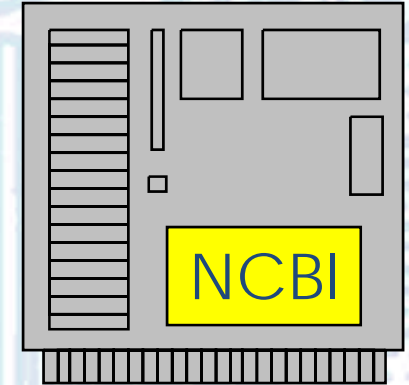
<u>Parameter</u>	<u>Default Value</u>
agent	"libwww-perl/#.###"
conn_cache	undef
cookie_jar	undef
from	undef
max_size	undef
parse_head	1
protocols_allowed	undef
protocols_forbidden	undef
requests_redirectable	['GET', 'HEAD']
timeout	180

```
Setting values:    $browser->timeout(newval);  
Reading values:  $value = $browser->timeout( );
```

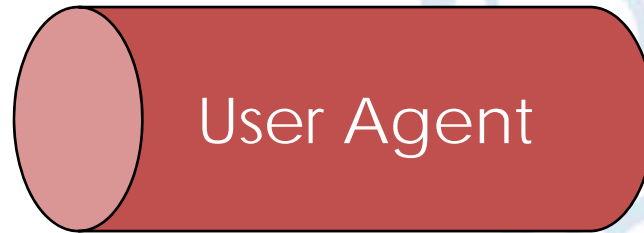
POSTing a URL



HTTP Post:
Supports URLs of
arbitrary length



HTTP
Request



HTTP
Response

```
$ua = LWP::UserAgent->new
```

```
$req = new HTTP::Request POST
```

```
$response = $ua->request($req)
```

HTTP::Request

- HTTP::Request->new(\$method, \$uri, \$header, \$content)
- \$browser->method(\$val) This is used to get/set the method attribute.
 - "get"
 - "head"
 - "put"
 - "post"
- \$browser->uri(\$val) This is used to get/set the uri attribute.
- \$browser->header(\$field => \$value) This is used to get/set header values
- \$browser->content(\$content) This is used to get/set the content
- \$browser->content_type(\$content_type) This is used to get/set the content type

HTTP::Response

`$response->is_success()`

`$response->status_line()`

```
if ($response->is_success) {  
    print $response->content;  
}  
else {  
    print STDERR $response->status_line, "\n";  
}
```

`$response->content()`

`$response->content_type()`

Another GET example

```
#!/usr/bin/perl -w
use strict;
use LWP::Simple;

my $catalog =
get("http://www.oreilly.com/catalog");
my $count = 0;
$count++ while $catalog =~ m{Perl}gi;
print "$count\n";
```

A *POST* Example

```
#!/usr/bin/perl -w
use LWP;

$params{'db'} = "nucleotide";
$params{'term'} = "mouse[orgn]";
$base = "http://eutils.ncbi.nlm.nih.gov/entrez/eutils/";

$url_params = "db=$params{'db'}&term=$params{'term'}";
$url = $base . "esearch.fcgi";

#create user agent
$ua = new LWP::UserAgent;
$ua->agent("epost_file/1.0 " . $ua->agent);

#create HTTP request object
$req = new HTTP::Request POST => "$url";
$req->content_type('application/x-www-form-urlencoded');
$req->content("$url_params");

#post the HTTP request which creates an HTTP response object
$raw = $ua->request($req);
$content = $raw->content();
print $content;
```

Advance BLAST POST Example

```
my %params = @_;  
my $ua = LWP::UserAgent->new;  
my ($req, $response, $html, $rid);  
my $search_args;  
  
$search_args = "CMD=Put&PROGRAM=$params{'program'}";  
$search_args .= "&DATABASE=$params{'database'}&EXPECT=$params{'expect'}";  
$search_args .= "&FILTER=$params{'filter'}&HITLIST_SIZE=$params{'hitlist_size'}";  
$search_args .= "&QUERY=$params{'query'}&ENTREZ_QUERY=$params{'entrez_query'}";  
$search_args .= "&QUERY_FROM=$params{'query_from'}&QUERY_TO=$params{'query_to'}";  
$search_args .= "&WORD_SIZE=$params{'word_size'}";  
  
#submit search  
$req = new HTTP::Request POST => 'http://www.ncbi.nlm.nih.gov/blast/Blast.cgi';  
$req->content_type('application/x-www-form-urlencoded');  
$req->content($search_args);  
  
$response = $ua->request($req);
```

Resources

<http://search.safaribooksonline.com/home>

<http://www.library.ucsf.edu/edtech/class/mbhandouts/>

<http://search.cpan.org/>

