NAME

lwpcook - The libwww-perl cookbook

DESCRIPTION

This document contain some examples that show typical usage of the libwww-perl library. You should consult the documentation for the individual modules for more detail.

All examples should be runnable programs. You can, in most cases, test the code sections by piping the program text directly to perl.

GET

It is very easy to use this library to just fetch documents from the net. The LWP::Simple module provides the *get()* function that return the document specified by its URL argument:

```
use LWP::Simple;
 $doc = get 'http://search.cpan.org/dist/libwww-perl/';
```

or, as a perl one-liner using the *getprint()* function:

```
 perl -MLWP::Simple -e 'getprint '" -P " -- http://search.cpan.org/dist/libwww-perl/
```

or, how about fetching the latest perl by running this command:

```
perl -MLWP::Simple -e '
getstore "ftp://ftp.sunet.se/pub/lang/perl/CPAN/src/latest.tar.gz",
"perl.tar.gz"'
```

You will probably first want to find a CPAN site closer to you by running something like the following command:

```
 perl -MLWP::Simple -e 'getprint '" -P " -- http://www.cpan.org/SITES.html
```

Enough of this simple stuff! The LWP object oriented interface gives you more control over the request sent to the server. Using this interface you have full control over headers sent and how you want to handle the response returned.

```
use LWP::UserAgent;
$ua = LWP::UserAgent->new;
$ua->agent("$0/0.1 " . $ua->agent);
# $ua->agent("Mozilla/8.0") # pretend we are very capable browser

$req = HTTP::Request->new(
    GET => 'http://search.cpan.org/dist/libwww-perl/');
$req->header('Accept' => 'text/html');

# send request
$res = $ua->request($req);

# check the outcome
if ($res->is_success) {
print $res->decoded_content;
}
else {
print "Error: " . $res->status_line . "\n";
}
```

The lwp-request program (alias GET) that is distributed with the library can also be used to fetch documents from WWW servers.

HEAD

If you just want to check if a document is present (i.e. the URL is valid) try to run code that looks like this:

```
use LWP::Simple;

if (head($url)) {
# ok document exists
}
```

The *head()* function really returns a list of meta-information about the document. The first three values of the list returned are the document type, the size of the document, and the age of the document.

More control over the request or access to all header values returned require that you use the object oriented interface described for GET above. Just s/GET/HEAD/g.

POST

There is no simple procedural interface for posting data to a WWW server. You must use the object oriented interface for this. The most common POST operation is to access a WWW form application:

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

my $req = HTTP::Request->new(
     POST => 'http://rt.cpan.org/Public/Dist/Display.html');
$req->content_type('application/x-www-form-urlencoded');
$req->content('Status=Active&Name=libwww-perl');

my $res = $ua->request($req);
print $res->as_string;
```

Lazy people use the HTTP::Request::Common module to set up a suitable POST request message (it handles all the escaping issues) and has a suitable default for the content_type:

```
use HTTP::Request::Common qw(POST);
use LWP::UserAgent;
$ua = LWP::UserAgent->new;

 my $req = POST 'http://rt.cpan.org/Public/Dist/Display.html',
[ Status => 'Active', Name => 'libwww-perl' ];

print $ua->request($req)->as_string;
```

The lwp-request program (alias POST) that is distributed with the library can also be used for posting data.

PROXIES

Some sites use proxies to go through fire wall machines, or just as cache in order to improve performance. Proxies can also be used for accessing resources through protocols not supported directly (or supported badly :-) by the libwww-perl library.

You should initialize your proxy setting before you start sending requests:

```
use LWP::UserAgent;
$ua = LWP::UserAgent->new;
$ua->env_proxy; # initialize from environment variables
# or
 $ua->proxy(ftp  => 'http://proxy.myorg.com');
 $ua->proxy(wais => 'http://proxy.myorg.com');
$ua->no_proxy(qw(no se fi));

my $req = HTTP::Request->new(GET => 'wais://xxx.com/');
```

```
print $ua->request($req)->as_string;
```

The LWP::Simple interface will call *env_proxy()* for you automatically. Applications that use the $ua->*env_proxy()* method will normally not use the $ua->*proxy()* and $ua->*no_proxy()* methods.

Some proxies also require that you send it a username/password in order to let requests through. You should be able to add the required header, with something like this:

```
use LWP::UserAgent;

$ua = LWP::UserAgent->new;
$ua->proxy(['http', 'ftp'] => 'http://username:password@proxy.myorg.com');

$req = );" -P HTTP::Request->new('GET'," -- http://www.perl.com

$res = $ua->request($req);
print $res->decoded_content if $res->is_success;
```

Replace `proxy.myorg.com`, `username` and `password` with something suitable for your site.

ACCESS TO PROTECTED DOCUMENTS

Documents protected by basic authorization can easily be accessed like this:

```
use LWP::UserAgent;
$ua = LWP::UserAgent->new;
 $req = HTTP::Request->new(GET => 'http://www.linpro.no/secret/');
$req->authorization_basic('aas', 'mypassword');
print $ua->request($req)->as_string;
```

The other alternative is to provide a subclass of *LWP::UserAgent* that overrides the *get_basic_credentials()* method. Study the *lwp-request* program for an example of this.

COOKIES

Some sites like to play games with cookies. By default LWP ignores cookies provided by the servers it visits. LWP will collect cookies and respond to cookie requests if you set up a cookie jar.

```
use LWP::UserAgent;
use HTTP::Cookies;

$ua = LWP::UserAgent->new;
$ua->cookie_jar(HTTP::Cookies->new(file => "lwpcookies.txt",
autosave => 1));

# and then send requests just as you used to do
 $res = $ua->request(HTTP::Request->new(GET => ));" -P " -- http://no.yahoo.com/
print $res->status_line, "\n";
```

As you visit sites that send you cookies to keep, then the file *lwpcookies.txt* will grow.

HTTPS

URLs with https scheme are accessed in exactly the same way as with http scheme, provided that an SSL interface module for LWP has been properly installed (see the *README.SSL* file found in the libwww-perl distribution for more details). If no SSL interface is installed for LWP to use, then you will get "501 Protocol scheme 'https' is not supported" errors when accessing such URLs.

Here's an example of fetching and printing a WWW page using SSL:

```
use LWP::UserAgent;

my $ua = LWP::UserAgent->new;
 my $req = HTTP::Request->new(GET => 'https://www.helsinki.fi/');
my $res = $ua->request($req);
```

```
 if ($res->is_success) {
 print $res->as_string;
 }
 else {
 print "Failed: ", $res->status_line, "\n";
 }
```

MIRRORING

If you want to mirror documents from a WWW server, then try to run code similar to this at regular intervals:

```
 use LWP::Simple;

 %mirrors = (
     'http://www.sn.no/'
                      => 'sn.html',
     'http://www.perl.com/'
                   => 'perl.html',
     'http://search.cpan.org/distlibwww-perl/'
=> 'lwp.html',
 'gopher://gopher.sn.no/' => 'gopher.html',
 );

 while (($url, $localfile) = each(%mirrors)) {
 mirror($url, $localfile);
 }
```

Or, as a perl one-liner:

```
  perl -MLWP::Simple -e ," -P 'mirror(" -- http://www.perl.com/
"perl.html")';
```

The document will not be transferred unless it has been updated.

LARGE DOCUMENTS

If the document you want to fetch is too large to be kept in memory, then you have two alternatives. You can instruct the library to write the document content to a file (second $ua->*request()* argument is a file name):

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

 my $req = HTTP::Request->new(GET =>
     'http://www.cpan.org/authors/Gisle_Aas/libwww-perl-6.02.tar.gz');
 $res = $ua->request($req, "libwww-perl.tar.gz");
 if ($res->is_success) {
 print "ok\n";
 }
 else {
 print $res->status_line, "\n";
 }
```

Or you can process the document as it arrives (second $ua->*request()* argument is a code reference):

```
 use LWP::UserAgent;
 $ua = LWP::UserAgent->new;
 $URL = 'ftp://ftp.unit.no/pub/rfc/rfc-index.txt';

 my $expected_length;
```

```
my $bytes_received = 0;
my $res =
$ua->request(HTTP::Request->new(GET => $URL),
sub {
my($chunk, $res) = @_;
$bytes_received += length($chunk);
unless (defined $expected_length) {
$expected_length = $res->content_length || 0;
}
if ($expected_length) {
printf STDERR "%d%% - ",
100 * $bytes_received / $expected_length;
}
print STDERR "$bytes_received bytes received\n";

# XXX Should really do something with the chunk itself
# print $chunk;
});
print $res->status_line, "\n";
```

COPYRIGHT