LibCurl and Perl 6

Curt Tilmes

Curt.Tilmes@nasa.gov

Philadelphia Perl Mongers

2017-06-12

Introduction

- `curl` is a popular command line tool for transferring data with URLs

Introduction

- `curl` is a popular command line tool for transferring data with URLs
- `libcurl` is a C library encompassing the functionality:

  libcurl is a free and easy-to-use client-side URL transfer library, supporting DICT, FILE, FTP, FTPS, Gopher, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMTP, SMTPS, Telnet and TFTP. libcurl supports SSL certificates, HTTP POST, HTTP PUT, FTP uploading, HTTP form based upload, proxies, cookies, user+password authentication (Basic, Digest, NTLM, Negotiate, Kerberos), file transfer resume, http proxy tunneling and more!

  libcurl is free, thread-safe, IPv6 compatible, feature rich, well supported, fast, thoroughly documented and is already used by many known, big and successful companies.

Introduction

- `curl` is a popular command line tool for transferring data with URLs
- `libcurl` is a C library encompassing the functionality:

  libcurl is a free and easy-to-use client-side URL transfer library, supporting DICT, FILE, FTP, FTPS, Gopher, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMTP, SMTPS, Telnet and TFTP. libcurl supports SSL certificates, HTTP POST, HTTP PUT, FTP uploading, HTTP form based upload, proxies, cookies, user+password authentication (Basic, Digest, NTLM, Negotiate, Kerberos), file transfer resume, http proxy tunneling and more!

  libcurl is free, thread-safe, IPv6 compatible, feature rich, well supported, fast, thoroughly documented and is already used by many known, big and successful companies.

- `LibCurl` is a Perl 6 module using the NativeCall capability of Perl 6 to interface directly with `libcurl`

Overview

`libcurl` supports two interfaces, both of which are mirrored into `LibCurl`:

- **Easy**
  - "The easy interface is a synchronous, efficient, quickly used and... yes, easy interface for file transfers."
  - The libcurl bindings are available via a low-level interface in `LibCurl::EasyHandle`. A friendly Perl 6 class makes them available through a high level interface as `LibCurl::Easy`.

Overview

`libcurl` supports two interfaces, both of which are mirrored into `LibCurl` :

- **Easy**
  - "The easy interface is a synchronous, efficient, quickly used and... yes, easy interface for file transfers."
  - The libcurl bindings are available via a low-level interface in `LibCurl::EasyHandle` . A friendly Perl 6 class makes them available through a high level interface as `LibCurl::Easy` .
- **Multi**
  - "The multi interface is the asynchronous brother in the family and it also offers multiple transfers using a single thread and more."
  - The multi bindings are similarly available via a low-level interface in `LibCurl::MultiHandle` , and wrapped in a high level interface as `LibCurl::Multi` .

LibCurl::Easy examples

```
use LibCurl::Easy;
print LibCurl::Easy.new(URL => 'http://example.com').perform.content;
```

LibCurl::Easy examples

```
use LibCurl::Easy;
print LibCurl::Easy.new(URL => 'http://example.com').perform.content;
```

Let's break it up into three phases:

1 `.new()` Construct a new LibCurl::Easy Object

You can pass in many options (~80 so far implemented) to control the nature of the desired transfer. The only option that is required is `URL` . You can also add options to the handle later.

LibCurl::Easy examples

```
use LibCurl::Easy;
print LibCurl::Easy.new(URL => 'http://example.com').perform.content;
```

Let's break it up into three phases:

1 `.new()` Construct a new LibCurl::Easy Object

You can pass in many options (~80 so far implemented) to control the nature of the desired transfer. The only option that is required is `URL` . You can also add options to the handle later.

2 `.perform()` Perform the transfer

LibCurl::Easy examples

```
use LibCurl::Easy;
print LibCurl::Easy.new(URL => 'http://example.com').perform.content;
```

Let's break it up into three phases:

1 `.new()` Construct a new LibCurl::Easy Object

You can pass in many options (~80 so far implemented) to control the nature of the desired transfer. The only option that is required is `URL` . You can also add options to the handle later.

2 `.perform()` Perform the transfer

3 `.content()` Examine the handle after the transfer, and, in this case, return the content. This is optional. For an upload (PUT/POST, or FTP upload), you might check status.

LibCurl::Easy examples

```
use LibCurl::Easy;
print LibCurl::Easy.new(URL => 'http://example.com').perform.content;
```

Let's break it up into three phases:

1 `.new()` Construct a new LibCurl::Easy Object

You can pass in many options (~80 so far implemented) to control the nature of the desired transfer. The only option that is required is `URL` . You can also add options to the handle later.

2 `.perform()` Perform the transfer

3 `.content()` Examine the handle after the transfer, and, in this case, return the content. This is optional. For an upload (PUT/POST, or FTP upload), you might check status.

Most methods perform some action, then return the same handle, so you can easily chain methods as in this example.

Shortcuts

Because those basic actions are so frequent, there are some shortcuts which perform them:

```
use LibCurl::HTTP :subs;
print get 'http://example.com';
```

Shortcuts

Because those basic actions are so frequent, there are some shortcuts which perform them:

```
use LibCurl::HTTP :subs;
print get 'http://example.com';
```

There is also a version that decodes as JSON into a data structure:

```
use LibCurl::HTTP :subs;
print jget('http://example.com/something-that-returns-json')<someval>;
```

Setting options on an Easy handle

- At construction:

```
my $curl = LibCurl::Easy.new(someoption => 'something');
```

Setting options on an Easy handle

- At construction:

```
my $curl = LibCurl::Easy.new(someoption => 'something');
```

- Explicitly with `.setopt()`

```
$curl.setopt(someoption => 'something');
```

Setting options on an Easy handle

- At construction:

```
my $curl = LibCurl::Easy.new(someoption => 'something');
```

- Explicitly with `.setopt()`

```
$curl.setopt(someoption => 'something');
```

- Directly with `FALLBACK` method:

```
$curl.someoption('something');
```

Setting options on an Easy handle

- At construction:

```
my $curl = LibCurl::Easy.new(someoption => 'something');
```

- Explicitly with `.setopt()`

```
$curl.setopt(someoption => 'something');
```

- Directly with `FALLBACK` method:

```
$curl.someoption('something');
```

You can shortcut with `:someoption` or `:!someoption` for Boolean options.

These all return the handle, so you can chain them.

For the most part, these are identical to `libcurl` options `CURLOPT_SOMETHING` , just remove the `CURLOPT_` and lowercase.

Options:

CAinfo CApath URL accepttimeout-ms accept-encoding address-scope append autoreferer buffersize certinfo cookie cookiefile cookiejar cookielist customrequest dirlistonly failonerror followlocation forbid-reuse fresh-connect ftp-skip-pasv-ip ftp-use-eprt ftp-use-epsv ftpport header http-version httpauth httpget httpproxytunnel infilesize low-speed-limit low-speed-time maxconnects maxfilesize maxredirs max-send-speed max-recv-speed netrc nobody noprogress nosignal password post postfields postfieldsize protocols proxy proxyauth proxyport proxytype proxyuserpwd range redir-protocols referer resume-from ssl-verifyhost ssl-verifypeer timecondition timeout timeout-ms timevalue unrestricted-auth use-ssl useragent username userpwd verbose wildcardmatch

Options

Some fun ones:

```
my $curl = LibCurl::Easy.new(:verbose, URL => "http://...");
```

When debugging always set `:verbose` and you'll get a dump of the actual HTTP transaction, very handy.

Options

Some fun ones:

```
my $curl = LibCurl::Easy.new(:verbose, URL => "http://...");
```

When debugging always set `:verbose` and you'll get a dump of the actual HTTP transaction, very handy.

`:followlocation` = automatically follow a `Location:` header as part of an HTTP 3xx response.

Options

Some fun ones:

```
my $curl = LibCurl::Easy.new(:verbose, URL => "http://...");
```

When debugging always set `:verbose` and you'll get a dump of the actual HTTP transaction, very handy.

`:followlocation` = automatically follow a `Location:` header as part of an HTTP 3xx response.

`:nobody` = Doesn't transfer the body of the request, for HTTP this is equivalent to a `HEAD` request.

Options

Some fun ones:

```
my $curl = LibCurl::Easy.new(:verbose, URL => "http://...");
```

When debugging always set `:verbose` and you'll get a dump of the actual HTTP transaction, very handy.

`:followlocation` = automatically follow a `Location:` header as part of an HTTP 3xx response.

`:nobody` = Doesn't transfer the body of the request, for HTTP this is equivalent to a `HEAD` request.

`proxy` = Set a proxy to use for the transfer.

Options

Some fun ones:

```
my $curl = LibCurl::Easy.new(:verbose, URL => "http://...");
```

When debugging always set `:verbose` and you'll get a dump of the actual HTTP transaction, very handy.

`:followlocation` = automatically follow a `Location:` header as part of an HTTP 3xx response.

`:nobody` = Doesn't transfer the body of the request, for HTTP this is equivalent to a `HEAD` request.

`proxy` = Set a proxy to use for the transfer.

`private` = Store any Perl object you want access to later.

Header Options

There are a few special options that set headers (useragent, referer, cookie), there are some extra options for headers: `Content-MD5` , `Content-Type` , `Content-Length` , `Host` , `Accept` , `Expect` , `Transfer-Encoding` .

```
$curl.Host('somewhere.com');  # or $curl.setopt(Host => 'somewhere.com')
$curl.Content-MD5('...');     # or $curl.setopt(Content-MD5 => '...')
```

Header Options

There are a few special options that set headers (useragent, referer, cookie), there are some extra options for headers: `Content-MD5` , `Content-Type` , `Content-Length` , `Host` , `Accept` , `Expect` , `Transfer-Encoding` .

```
$curl.Host('somewhere.com');  # or $curl.setopt(Host => 'somewhere.com')
$curl.Content-MD5('...');     # or $curl.setopt(Content-MD5 => '...')
```

You can also add any other headers you like:

```
$curl.set-header(X-My-Header => 'something', X-something => 'foo');
```

Header Options

There are a few special options that set headers (useragent, referer, cookie), there are some extra options for headers: `Content-MD5` , `Content-Type` , `Content-Length` , `Host` , `Accept` , `Expect` , `Transfer-Encoding` .

```
$curl.Host('somewhere.com');  # or $curl.setopt(Host => 'somewhere.com')
$curl.Content-MD5('...');     # or $curl.setopt(Content-MD5 => '...')
```

You can also add any other headers you like:

```
$curl.set-header(X-My-Header => 'something', X-something => 'foo');
```

Clear all ***except*** the libcurl special headers:

```
$curl.clear-header;
```

Upload/Download

- `download => 'myfile'`
- `upload => 'myfile'`
- `send => 'something'`
- `send => $mybuf`

If you *don't* specify a download filename, it will stash all incoming content in `$curl.buf` .

You can also access that content decoded as a `UTF-8` `Str` with `$curl.content` .

You can change encoding if you want `$curl.content('utf-16')` .

Info

- After a transfer completes (successfully or otherwise), you can access a lot of information about the transfer. Similar to options, there are several methods to get that information:

```
say $curl.getinfo('effective-url');
say $curl.getinfo('response-code');
say $curl.getinfo(<effective-url response-code>);  # Hash with those keys
say $curl.getinfo;   # Hash of all info fields
say $curl.response-code;
```

Info

- After a transfer completes (successfully or otherwise), you can access a lot of information about the transfer. Similar to options, there are several methods to get that information:

```
say $curl.getinfo('effective-url');
say $curl.getinfo('response-code');
say $curl.getinfo(<effective-url response-code>);  # Hash with those keys
say $curl.getinfo;   # Hash of all info fields
say $curl.response-code;
```

Fields currently defined are:

appconnect_time certinfo condition-unmet connect-time content-type cookielist effective-url ftp-entry-path header-size http-connectcode httpauth-avail lastsocket local-ip local-port namelookup-time num-connects os-errno pretransfer-time primary-ip primary-port proxyauth-avail redirect-url request-size response-code rtsp-client-cseq rtsp-cseq-recv rtsp-server-cseq rtsp-session-id size-download size-upload speed-download speed-upload ssl-engines total-time

Received headers

After a transfer, you can also check out the headers returned by the server:

```
say $curl.get-header('Content-Length');

say $curl.receiveheaders<Content-Length>;  # Hash of all headers

say $curl.Content-Length;
```

Errors

- Most real errors will throw an `X::LibCurl` exception
- The `failonerror` option will force an exception on an HTTP code ≥ 400 (not usually an error from the `LibCurl` perspective).
- You can check response code with $curl.response-code.

Debugging

- `:verbose` option will just dump some good stuff to `STDOUT`
- Create a debug subroutine:

```
sub debug(LibCurl::Easy $easy, CURL-INFO-TYPE $type, Buf $buf) {...}
$curl.setopt(debugfunction => &debug);
```

Gets called periodically:

- CURLINFO_TEXT
- CURLINFO_HEADER_IN
- CURLINFO_HEADER_OUT
- CURLINFO_DATA_IN
- CURLINFO_DATA_OUT
- CURLINFO_SSL_DATA_IN
- CURLINFO_SSL_DATA_OUT

Transfer progress

You can enable the simple `curl` progress printing by `:!noprogress`. (Yes, this seems backwards..)

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  364M  100  364M    0     0   104M      0  0:00:03  0:00:03 --:--:--  104M
```

You can also install your own progress function:

```
sub xferinfo(LibCurl::Easy $easy, $dltotal, $dlnow, $ultotal, $ulnow)
{...}

$curl.setopt(xferinfofunction => &xferinfo);
```

Multi-part Form POSTing

```
my $curl = LibCurl::Easy.new(URL => 'http://...');

# normal field
$curl.formadd(name => 'fieldname', contents => 'something');

# upload a file from disk, give optional filename or contenttype
$curl.formadd(name => 'fieldname', file => 'afile.txt',
              filename => 'alternate.name.txt',
              contenttype => 'image/jpeg');

# Send a Blob of contents, but as a file with a filename
$curl.formadd(name => 'fieldname', buffer => 'some.file.name.txt',
              bufferptr => "something".encode);

$curl.perform;
```

Multi Interface

Construct an `Easy` handle for each desired transfer, then `perform` them all simultaneously.

```
use LibCurl::Easy;
use LibCurl::Multi;

my $curl1 = LibCurl::Easy.new(:verbose, :followlocation,
                     URL => 'http://example.com',
                     download => './myfile1.html');

my $curl2 = LibCurl::Easy.new(:verbose, :followlocation,
                     URL => 'http://example.com',
                     download => './myfile2.html');

LibCurl:Multi.new.add-handle($curl1, $curl2).perform;

say $curl1.statusline;
say $curl2.statusline;
```

Multi Interface Async

```
use LibCurl::Easy;
use LibCurl::Multi;

my $curl1 = LibCurl::Easy.new(:followlocation,
                              URL => 'http://example.com',
                              download => 'myfile1.html');

my $curl2 = LibCurl::Easy.new(:followlocation,
                              URL => 'http://example.com',
                              download => 'myfile2.html');

sub callback(LibCurl::Easy $easy, Exception $e)
{
    die $e if $e;
    say $easy.response-code;
    say $easy.statusline;
}

my $multi = LibCurl::Multi.new(callback => &callback);

$multi.add-handle($curl1, $curl2);

$multi.perform;
```

Conclusion

- Perl 6 implementation still in development, please try it out and let me know what you like/don't like.
- Multi interface is in particular could be extended to support a more perl6-ish interface.

Conclusion

- Perl 6 implementation still in development, please try it out and let me know what you like/don't like.
- Multi interface is in particular could be extended to support a more perl6-ish interface.

Thank You!

Slides produced with remark