Doing more with Dropbox

Box It!

The proprietary Dropbox service has become a popular way to exchange large files. The Dropbox web API also supports scripts, like the one in this article that picks up files from behind a firewall. *By Mike Schilli*

Recently, an intern at Yahoo offered to lend me a digital audio book and suggested that she could "dropbox" it for me. Although I do try to keep in touch with today's youth, I was surprised to hear that the service offered by dropbox.com has become some kind of ubiquitous standard, to the point that "to dropbox" is now a verb, just like "to google."

Users can download the Dropbox client binary (Windows, Mac, and Linux are supported) from the website at dropbox.com; this reveals a brand-new, local Dropbox folder on their desktops. If you drag some files into the folder, a software tool automagically launches in the background and gradually uploads the new content to the Dropbox server without further interaction, as long as your Internet connection is up. Other clients belonging to the same user, or authorized buddies, located elsewhere on the Internet synchronize just as magically by downloading the shared files, giving the user a permanently up-to-date folder with important data on all of their computers. If you happen to be on a computer without a Dropbox client, all you need is a browser to view and manipulate the files on the Dropbox website (Figure 1).

Open Source Preferred

I must admit that I really didn't relish the thought of launching a binary on my computer at home without having seen the source code first. Fortunately, Dropbox also offers a Web API that gives a paranoid penguin friend like myself an option for putting the free service to productive use.

Sensibly, dropbox.com wants to train the users of API-driven programs to avoid typing their usernames and passwords in some third-party user interface and, because of this, relies on the OAuth protocol [2]. To register a user with today's featured Perl application, the code first picks up a request token and request secret from the Dropbox website, revealing its developer token and developer secret in the process.

Using the request token in the URL, the application then points the user's browser to the Dropbox website, where the user is prompted to enter their username and password if they are not already logged in (Figure 2).

After a successful login, Dropbox asks the user if they really want the "Perl Test Client" application to access their Dropbox data (Figure 3). If the user agrees, the Dropbox website points the browser back to the application and returns an access token in the URL. The application can then store the token and handle jobs on behalf of the user on his Dropbox account until the token expires.

Wireframe Web Server

Because a Perl command-line application doesn't normally run a browser interface, the `dropbox-init` script in Listing 1 puts together a minimal web server on *http://localhost:8082*, driven by the CPAN Mojolicious::Lite module. It responds to the paths `/` and `/callback`, featuring a starting point and a landing page after the user has successfully logged on to the Dropbox website, respectively.

The source for the HTML output rendered by the server is located in the `__DATA__` segment starting in line 88. The handler for the `'/'` path in line 44 refers to a symbol named `index` in line 57, which Mojolicious resolves to the `@@ index.html.ep` marker in line 90 and picks the corresponding HTML snippet consisting of a simple HTML link pointing the user to the Dropbox login site. The `%`

MIKE SCHILLI

Mike Schilli works as a software engineer with Yahoo! in Sunnyvale, California. He can be contacted at *mschilli@perlmeister.com*. Mike's homepage can be found at *http://perlmeister.com*.

`layout 'default';` instruction creates a well-formed HTML document from it.

When the user launches the `dropbox-init` Mojolicious application from the command line and types `http://localhost:8082` into the browser, the minimal UI (Figure 4) appears, containing just a single link to the dropbox.com login page.

The Net::Dropbox::API module, also from CPAN, nicely abstracts Dropbox access and OAuth authorization. After the API developer sets the combination of the developer key and secret they picked up from the Dropbox Developer site [3] in the constructor call in line 33, a later call to the `login()` method in line 51 accesses the Dropbox server behind the scenes, picks up a request token and secret, and returns a Login URL. The web application then presents this to the user who can click on it to log in to Dropbox.

The Mojolicious application then stores the retrieved values for the request token/secret in the global variables `$REQUEST_TOKEN` and `$REQUEST_SECRET`, where it can access them later

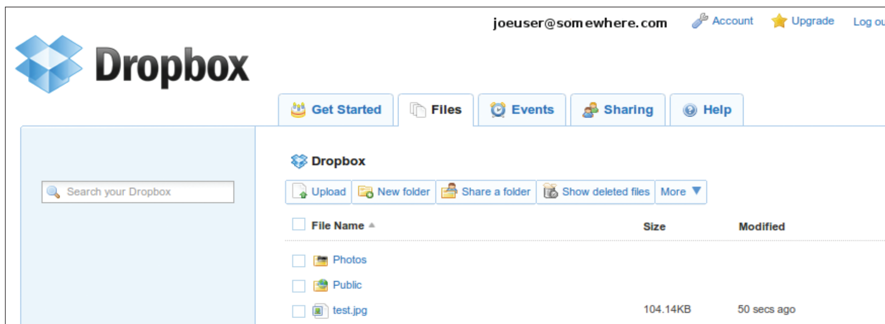

Figure 1: The Dropbox web interface.

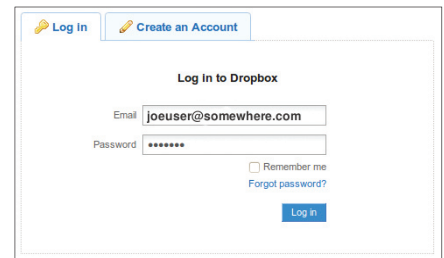

Figure 2: The Perl application points to dropbox.com, where the user enters their credentials to authenticate.

LISTING 1: dropbox-init

```
001 #!/usr/local/bin/perl -w
002 #############################
003 # dropbox-init --
004 #    collect dropbox token
005 # Mike Schilli, 2011
006 # (m@perlmeister.com)
007 #############################
008 use strict;
009 use Mojolicious::Lite;
010 use Net::Dropbox::API;
011 use YAML
012   qw(LoadFile DumpFile);
013
014 my $dev_key =
015   "iyaiu823ajksgwf";
016 my $dev_secret =
017   "zlkj32lkj2kl3dp";
018 my $listen =
019   "http://localhost:8082";
020 my ($home) = glob '~';
021 my $CFG_FILE =
022   "$home/.dropbox.yml";
023
024 my $CFG = {};
025 $CFG = LoadFile($CFG_FILE)
026   if -f $CFG_FILE;
027
028 @ARGV = (
029  qw(daemon --listen), $listen
030 );
031
032 my $box =
033   Net::Dropbox::API->new(
034   {
035   key     => $dev_key,
036   secret => $dev_secret,
037  }
038   );
039
040 my $REQUEST_TOKEN;
041 my $REQUEST_SECRET;
042
043 #############################
044 get '/' => sub {
045 #############################
046  my ($self) = @_;
047
048  $box->callback_url(
049   "$listen/callback");
050  $self->stash->{login_url} =
051    $box->login;
052
053  $REQUEST_TOKEN =
054    $box->request_token;
055  $REQUEST_SECRET =
056    $box->request_secret;
057 } => 'index';
058
059 #############################
060 get '/callback' => sub {
061 #############################
062  my ($self) = @_;
063
064  $box->auth(
065   {
066    request_token =>
067      $self->param(
068     'oauth_token'),
069    request_secret =>
070      $REQUEST_SECRET
071   }
072  );
073
074  $CFG->{access_token} =
075    $box->access_token();
076  $CFG->{access_secret} =
077    $box->access_secret();
078
079  DumpFile $CFG_FILE, $CFG;
080
081  $self->render_text(
082   "Token saved.",
083   layout => 'default');
084 };
085
086 app->start;
087
088 __DATA__
089 #############################
090 @@ index.html.ep
091 % layout 'default';
092 <a href="<%= $login_url %>"
093 >Login on dropbox.com</a>
094
095 @@ layouts/default.html.ep
096 <!doctype html><html>
097   <head><title>Token Fetcher
098   </title></head>
099     <body>
100       <pre>
101       <%== content %>
102       </pre>
103     </body>
104 </html>
```

when – assuming that the login was successful – dropbox.com sends the browser back to the minimal server with the callback URL set in line 48. When control is returned to the Mojolicious application (Figure 5), it only needs to grab the access token from the list of parameters attached to the callback URL and, in combination with the request token secret stored previously, now has the keys required to mess about with the user's Dropbox account to its heart's content. But the wireframe web application only stores the two keys in the `~/.dropbox.yml` YAML file (Figure 6) to allow applications called later to pick them up from here without having the user jump through login hoops again.

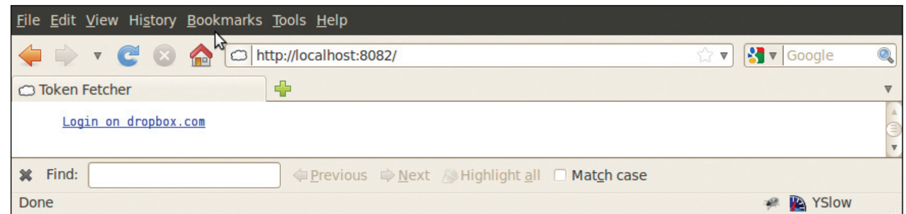

Figure 4: The Mojolicious server serves up a login link that points to dropbox.com.

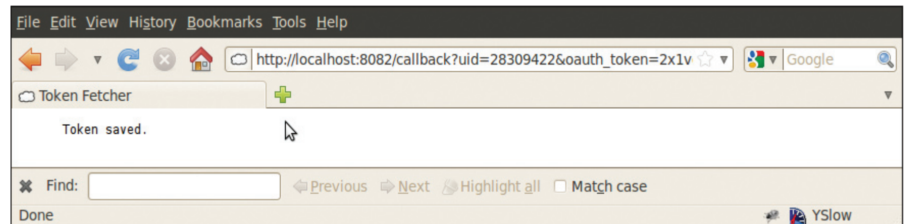

Figure 5: After the user has completed the login, dropbox.com returns control to the Mojolicious server, which stores the token data in a YAML file.

Drone Operations

For ease of access to the user credentials and transparent Dropbox control, Listing 2 defines a `MyDropbox` class, derived from Net::Dropbox::API, with a constructor that immediately serves up the keys stored in the YAML file to the Dropbox module so that applications that use it only need to worry about their own operations, and not about authentication issues. This process also allows for operations in drone mode – that is, without needing user interaction or even a web browser.

Listing 3 shows a simple sample application that pulls in `MyDropbox`, sets the context to `"dropbox"` (production operations instead of the `"sandbox"` test environment) and then calls `list()` to list the files in the `Photos` folder.

A nested structure similar to Figure 7 is returned, and the programmer can then extract the Dropbox content, garnished with file names, file size, last change date, and many other juicy bits of information. Other methods offered by the Dropbox API are uploading and downloading, and even file deletion. One test discovers whether a specific part of the file hierarchy has changed since the last request. There are no bounds to the developer's imagination here, as long as you don't exceed the current daily limit of 5,000 requests.

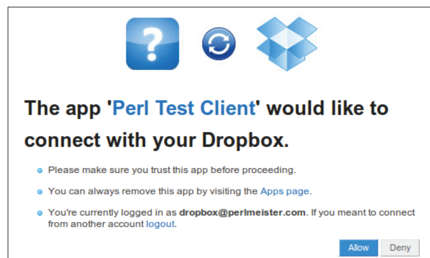

Figure 3: The user confirms that they trust the Perl application to manage their Dropbox data.

LISTING 2: MyDropbox.pm

```
01 #############################
02 # MyDropBox -- Dropbox access
03 # with stored credentials
04 # Mike Schilli, 2011
05 # (m@perlmeister.com)
06 #############################
07 package MyDropbox;
08 use strict;
09 use base 'Net::Dropbox::API';
10 use YAML qw(LoadFile);
11
12 my $dev_key =
13   "iyaiu823ajksgwf";
14 my $dev_secret =
15   "zlkj321kj2kl3dp";
16 my ($home) = glob '~';
17 my $CFG_FILE =
18   "$home/.dropbox.yml";
19
20 #############################
21 sub new {
22 #############################
23   my ($class) = @_;
24
25   my $box =
26     Net::Dropbox::API->new(
27    {
28     key     => $dev_key,
29     secret  => $dev_secret,
30    }
31     );
32
33   my $cfg =
34     LoadFile($CFG_FILE);
35   $box->access_token(
36    $cfg->{access_token});
37   $box->access_secret(
38    $cfg->{access_secret});
39
40   bless $box, $class;
41 }
42
43 1;
```

Ghost Updater

As a practical application of the API, I recently implemented a ghost updater. On every system I use, my Git repository clones are always up to date, thanks to the GitMeta tool [4] that I introduced in this column last year. Unfortunately, I occasionally forget to execute `git push` on my home computer, and this means that I have some local changes that I have not uploaded to the Git server and which I can't easily access using my laptop in a hotel room because my home computer lives behind a firewall.

So, I wrote the Dropbox daemon script in Listing 4, which runs on my home

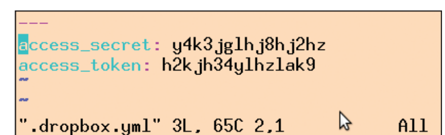

Figure 6: The YAML file with the access token and access secret.

LISTING 3: dropbox-dump

```
01 #!/usr/local/bin/perl -w
02 #############################
03 # dropbox-dump --
04 # Dump dropbox content
05 # Mike Schilli, 2011
06 # (m@perlmeister.com)
07 #############################
08 use strict;
09 use MyDropbox;
10 use Data::Dumper;
11
12 my $box = MyDropbox->new();
13 $box->context("dropbox");
14
15 my $href =
16   $box->list("/Photos");
17
18 $Data::Dumper::Indent = 1;
19 print Dumper($href);
```

computer. It is launched by `dropbox-gitgetter start`, then puts itself in the background and periodically (every 60 seconds) checks the `gitgetter/requests.txt` file in the Dropbox folder for changes. If the Dropbox user somewhere on Web needs a file from their home computer, they just need to append the desired file path to the `gitgetter/request.txt` file in the Dropbox.

Within a minute, the daemon running on the home computer then notices the change, extracts the new request line, and checks whether it really relates to a file in the Git repositories all located under a common path. If so, it picks up the required file from the local file system on the home computer, uses the Dropbox API to put it into the Dropbox, deletes the request from `requests.txt`, and deposits a modified version of the request file in the Dropbox.

Because Dropbox refuses to upload empty files, the daemon leaves a comment at the head of the file if there are no requests left to complete.

Listing 4 runs in the background after the `start` command and, thanks to App::Daemon from CPAN, also supports `stop` and `status` to shut down the daemon or determine its status. It uses Log4perl to log its activities in the `/tmp/dropbox-gitgetter.log` file, and `-X` launches the script in the foreground to facilitate troubleshooting if something doesn't work as intended.

In the infinite loop starting in line 33, Listing 4 calls the Dropbox object's `list()` method every 60 seconds to discover whether anything has changed below the `/gitgetter` Dropbox directory. But instead of asking the Dropbox server to send the whole directory hierarchy across the wire, the script uses an efficient hashing method. On completed requests, besides the desired results, the Dropbox API also returns a 32-byte hex string that reflects the status of file hierarchy for `list()` requests. If you want to save bandwidth, you can include the hash with the next call, and the Dropbox server will simply return an HTTP Code 403 ("not modified") instead of data if there were no recent modifications, and the script can go back to sleep until the next round.

```
 'icon' => 'folder',
 'bytes' => 0,
 'thumb_exists' => bless( do{\(my $o = 0)}, 'JSON::XS::Boolean' ),
 'contents' => [
  {
    'icon' => 'folder',
    'bytes' => 0,
    'thumb_exists' => $VAR1->{'thumb_exists'},
    'path' => '/git-retriever',
    'is_dir' => $VAR1->{'contents'}[0]{'is_dir'},
    'modified' => 'Fri, 13 May 2011 04:26:03 +0000',
    'size' => '0 bytes',
    'revision' => 28
  },
  {
    'icon' => 'folder_photos',
    'bytes' => 0,
    'thumb_exists' => $VAR1->{'thumb_exists'},
    'path' => '/Photos',
    'is_dir' => $VAR1->{'contents'}[0]{'is_dir'},
    'modified' => 'Mon, 09 May 2011 02:33:59 +0000',
    'size' => '0 bytes',
    'revision' => 18
                                                    25,1          50%
```

Figure 7: The list() call returns a data structure with Dropbox content.

Beware of Baddies

If there's news, the `request_handler()` function gets called in line 51 and uses the `getfile()` method in line 64 to pick up the content of the `gitgetter/requests.txt` file from the Dropbox server via the API. A `for` loop then iterates over the lines of the file, removing comments and ignoring empty lines. The `realpath()` function from the treasure trove of the Cwd module, makes sure that no malicious characters are hiding out in the identified lines that could trick the daemon into wandering down spurious paths and serving up the files that reside in them. To accomplish this, the script appends the requested path to the previously specified Git directory and then checks if a `realpath` of the result is still a location inside the Git directory.

This is important to prevent an attacker who has compromised the Dropbox account, or even taken control of the Dropbox server, mapping the entire local filesystem; at most, they can then see the explicitly authorized path to the Git repositories, the server versions of which are publicly available anyway in my case.

If the daemon authorizes the release of a file, `putfile()` in line 94 drops it into the Dropbox's `/gitgetter` directory. After processing requests, the `blurt` command of the Sysadm::Install CPAN module in line 102 writes an empty `requests.txt` with a comment line back to the Dropbox to avoid the daemon again pushing the file next time around.

Installation

The rat's tail of required modules is either available via your distribution's package manager (e.g., libapp-daemon-perl, liblog-log4perl, etc., for Ubuntu), or you can use a CPAN shell.

The CPAN Net::Dropbox::API module didn't support the hash-based Dropbox

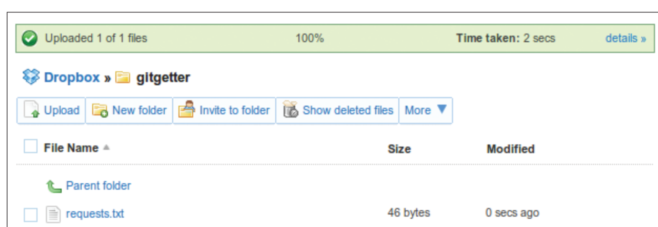

Figure 8: A request to the dropbox daemon to upload a forgotten file.

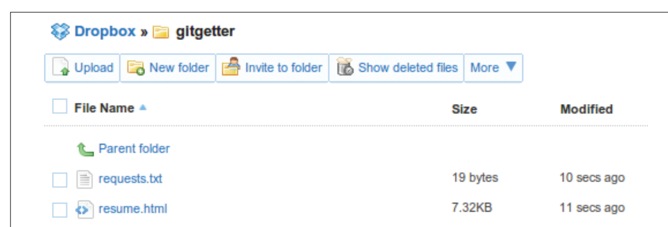

Figure 9: Later, the daemon has smuggled in the required file.

query for modified files when I wrote this article, so I simply forked the project on Github, added the function, and asked the author to add the patch to the main line. If this hasn't happened by the time this issue goes to press, you can do so by downloading the tarball with the modified module from Github [5].

To use the Dropbox API, programmers need a developer key, which you can request [3] after supplying your email address. You need to modify the `$dev_key` and `$dev_secret` variables in the listings, but then you're up and running.

Be Careful!

The advantage of Dropbox is quite obviously its simple handling. Even computer newbies can store data and pick it up elsewhere later. An example of a procedure for communications between developers and web designers uses Dropbox to make changes to a Git repository available to project contributors who find using a repository too challenging [6].

Incidentally, although dropbox.com ensures users that the files it stores are encrypted, and not even readable for Dropbox staff, a report [7] claims that this is not true. Whatever the case may be, the old adage still applies: Confidential data is still best kept on your local disk, and the cloud still needs to work out the kinks of cloudy security concepts.

INFO

[1] Listings for this article: *http://www.linux-magazine.com/Resources/Article-Code*

[2] OAuth: *http://en.wikipedia.org/wiki/Oauth*

[3] Dropbox for Developers: *https://www.dropbox.com/developers/quickstart*

[4] "Managing Git" by Michael Schilli, *Linux Magazine*, September 2010, pg. 50: *http://www.linux-magazine.com/Issues/2010/118/PERL-MANAGING-GIT*

[5] Net::Dropbox 1.4_01, including the hash function: *http://github.com/mschilli/Net--Dropbox/tarball/1.4_01*

[6] Dropbox + git = Designer Luv" by Ken Mayer: *http://pivotallabs.com/users/ken/blog/articles/1637-dropbox-git-designer-luv*

[7] "Dropbox Lied to Users About Data Security, Complaint to FTC Alleges": *http://www.wired.com/threatlevel/2011/05/dropbox-ftc/*

LISTING 4: dropbox-gitgetter

```
001 #!/usr/local/bin/perl -w
002 #############################
003 # dropbox-gitgetter -- Get
004 # forgotten git updates
005 # via dropbox.com
006 # Mike Schilli, 2011
007 # (m@perlmeister.com)
008 #############################
009 use strict;
010 use MyDropbox;
011 use App::Daemon
012   qw( daemonize );
013 use Log::Log4perl qw(:easy);
014 use HTTP::Status
015   qw(:constants);
016 use File::Temp qw(tempfile);
017 use Cwd qw(realpath);
018 use Sysadm::Install qw(:all);
019 use File::Basename;
020
021 daemonize();
022
023 my $mod_hash      = undef;
024 my $poll_interval = 60;
025
026 my $box = MyDropbox->new();
027 $box->context("dropbox");
028
029 my ($home) = glob "~";
030 my $gitdir =
031   realpath "$home/git";
032
033 while (1) {
034  my @hash_args = ();
035  @hash_args =
036    (hash => $mod_hash)
037    if defined $mod_hash;
038
039  my $href = $box->list(
040   {@hash_args},
041   "/gitgetter"
042  );
043
044  if (
045   $href->{http_response_code}
046   eq HTTP_NOT_MODIFIED)
047  {
048   DEBUG "Not modified";
049  } else {
050   $mod_hash = $href->{hash};
051   request_handler($box);
052  }
053
054  DEBUG
055 "Sleeping ${poll_interval}s";
056  sleep $poll_interval;
057 }
058
059 #############################
060 sub request_handler {
061 #############################
062  my ($box) = @_;
063
064  my $content = $box->getfile(
065   "gitgetter/requests.txt");
066
067  my $pushed = 0;
068
069  for my $line (split /\n/,
070   $content)
071  {
072   $line =~ s/#.*//;
073   next if $line =~ /^\s*$/;
074   DEBUG
075     "Found request: '$line'";
076
077   my $file = realpath(
078    "$gitdir/$line");
079   if ($file !~ /^$gitdir/) {
080    ERROR
081      "Path $file denied.";
082    next;
083   }
084
085   DEBUG "Delivering $file";
086
087   if (!-f $file) {
088    ERROR
089      "$file doesn't exist";
090    next;
091   }
092
093   my $href =
094     $box->putfile($file,
095    "gitgetter");
096   $pushed++;
097  }
098
099  if ($pushed) {
100   my ($fh, $tmpfile) =
101     tempfile(UNLINK => 1);
102   blurt
103     "# pending requests\n",
104     $tmpfile;
105   my $href =
106     $box->putfile($tmpfile,
107    "gitgetter",
108    "requests.txt");
109  }
110 }
```