Evernote productivity tool API

Remember This

Evernote, a highly structured, digital notepad quickly replicates ideas jotted down and web pages captured to different locations and devices. An API supports program-controlled access via Facebook's Thrift library. We provide a how-to in Perl. *By Mike Schilli*

Ideas often come to me in the strangest places – for example, on the train or in the shower. Typically, these ideas quickly disappear if I don't jot them down right away. But, instead of scribbling ideas on sticky notes, today's creative users capture their scribblings with distributed digital applications, which protect your data and allow you to search heaps of documents in a matter of seconds.

Needle in a Haystack

Evernote is one such commercial application. It offers a basic version that is free for moderate use [2] and comes with a browser interface and virtual notebook apps for mobile devices, such as the iPhone or iPad. Notes, in Evernote's sense of the word, can be formatted text with images, audio files, or screenshots of websites that you grab and paste in (Figure 1). Users group their notes in "notebooks," which in turn can be organized in subfolders known as "stacks."

Evernote's unique selling point is continuous and unobtrusive synchronization between the devices involved (Figure 2). A change that you make on the desktop in your browser is replicated within a couple of seconds in the browser running on your laptop, assuming it is also logged in to your Evernote account. And, the Evernote app even stores the data locally on a MacBook or on mobile devices such as the iPad, thus supporting offline operation.

The simple structuring model used in Evernote invites inventive tinkerers to glue together customized productivity tools based on the rudimentary design elements. Some users report [3] that they have set up calendar functionality by putting dates into note subjects so Evernote will display them sorted by title – now they can organize their time in a "Getting Things Done" approach [4].

Figure 1 shows the Evernote notes I took while working on this article. I found a sample application for the Evernote API with the Thrift framework on Stackoverflow.com and quickly archived it with the Evernote Web Clipper as the entry point for my research. I also found a PDF with a whitepaper on the Thrift framework and a couple of Perl examples on Apache.org. Armed with this collection, I could easily answer questions later by referring to the clipped texts or

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*.

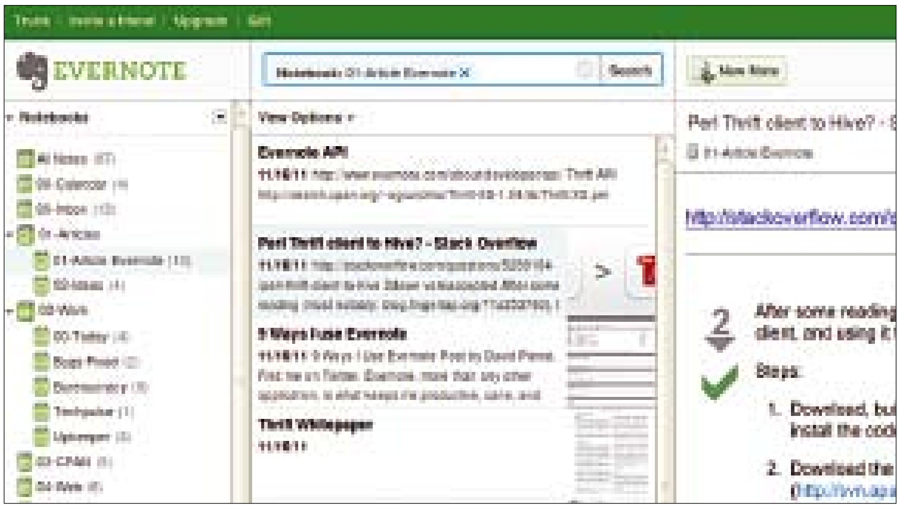


Figure 1: Web clippings stored in Evernote notebooks while I was writing this article.

Figure 2: Evernote synchronizes the stored information seamlessly between various portable devices – with an iPad in this case.

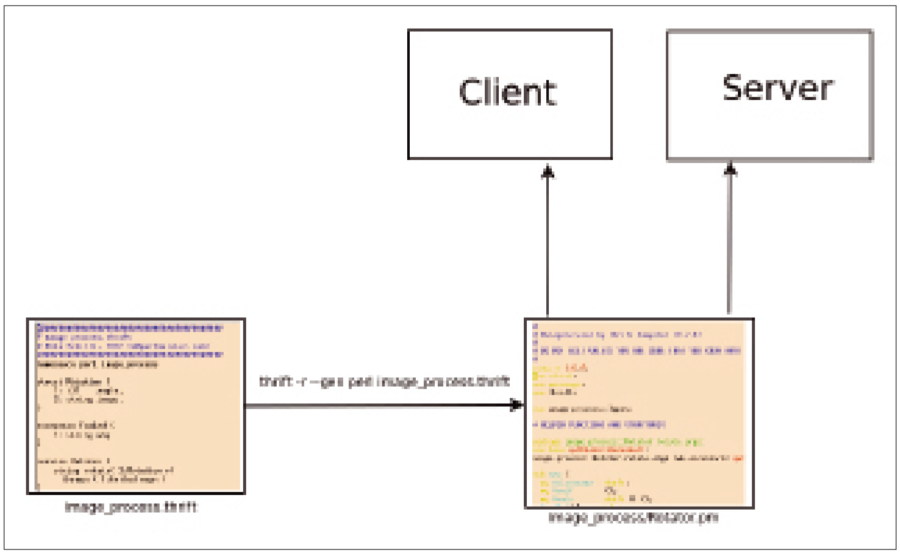


Figure 3: The Thrift compiler generates Perl code from the Thrift definition; the code is used by the client and the server in this application.

by accessing the web links that I had stored.

API Replaces GUI

Sometimes you don't want to use a GUI to make a note of your good ideas, which is what prompted me to look for a command-line tool. Fortunately, Evernote offers a web API to its service and uses Facebook's Thrift protocol to handle communications between clients and the server [5]. This decision probably was made for performance reasons because the binary protocol is leaner than communication pushing XML objects back and forth, especially for images.

A Lean Format

As Figure 3 shows, communicating with Thrift requires describing the data structures exchanged between client and server in the easily readable Thrift format in a `.thrift` file. The Thrift compiler then references the file to create library functions for a number of programming languages, from C++ to Java, and including scripting languages such as Perl, Ruby, Python, PHP, and JavaScript, as well as more exotic languages such as Erlang. This abstraction layer protects the application programmer from the headaches of reinventing cross-platform data transfers for every new project.

An example of using Thrift in Listing 1 shows the data structures and service definitions in `image_process.thrift` for a server that rotates image files through 90 degrees. The client sends the binary data of a JPG file to the server, and the latter uses the `convert` program from the ImageMagick package to perform the rotation and send the results back to the client, again in binary format. The client stores the JPG data on the local disk and notifies the user of the successful conversion or outputs an error message.

Compiler Writes Code

Thrift only supports a few data types, but they are powerful and portable. Besides simple 32- or 64-bit integers, users can pack data in `struct`s or use maps similar to Perl's hash types. Listing 1 defines a `Rotation` structure that accepts an integer for the desired angle of rotation and a string with the binary data of the image to rotate. The `Rotator` service defined in line 12 defines the `rotate()` method, which expects the `Rotation`

LISTING 1: image_process.thrift

```
01 namespace perl image_process
02
03 struct Rotation {
04   1: i32    angle,
05   2: string image,
06 }
07
08 exception Failed {
09      1: string why
10 }
11
12 service Rotator {
13  string rotate( 1:Rotation r)
14    throws ( 1:Failed oops )
15 }
```

structure described above as its enumerated first parameter and returns a string with the modified image data to the client.

Thrift throws exceptions if something goes wrong. Line 8 defines an exception of the type `Failed`, which contains a string named `why` with an explanation of what went wrong. If you download the Thrift distribution [6] and compile it using `sh ./configure` and `make`, you will have an executable by the name of `thrift`. This is the Thrift compiler, which turns Thrift files into glue code in the desired target language. If the build fails because you are missing some packages for the more exotic languages, you can disable them using the `configure --disable-xxx` option. The following command line

```
thrift -r --gen perl image_process.thrift
```

creates the Perl glue that you need for a straightforward exchange of data between the client and the server. The generated files are located in the `gen-perl/image_process` subdirectory.

The client in Listing 2 loads the resulting Thrift wrapper in line 17 and then defines a communication channel via a Unix socket on port 9001 of `localhost`, on which the server will listen later. The utility modules Thrift::Socket, Thrift::BufferedTransport, and Thrift::BinaryProtocol, which the client pulls in to set up the communication channel, are included in the Thrift distribution's `perl/lib` directory.

Line 31 instantiates a `RotatorClient` object using the Thrift data definition shown before; the object's Perl code is also auto-generated by `thrift`.

Thrift Throws Exceptions

The `eval` block in line 38 captures any exceptions that might occur while talking to the server, and the subsequent test in the `if` clause of line 59 prints any exception objects from the server, including the error texts they contain. After opening the `transport` in line 39, the client loads the image file specified at the command line from the filesystem by running `slurp` in the Sysadm::Install module from CPAN.

The `image_process::Rotation` type object instantiated in line 46 uses the `image()` and `angle()` accessors to put together the data structure to transfer. The only thing left is to call the `rotate` method with the Rotation structure in line 51 and snap up the results in the string returned by the server. The `blurt()` function, also courtesy of the CPAN Sysadm::Install module, then writes the rotated image data to a new file that uses a file name starting with `rotated-*`.

The corresponding server in Listing 3 defines the `RotateHandler` package for handling the client requests; it is based on the auto-generated `image_process::RotatorIf` class. In its `rotate()` method, which it runs thanks to Thrift magic when the client uses `rotate()` to send a request, the server creates two temporary files, extracts the image data from the rotation object passed to it, and pushes the bytes into the first file.

The `tap` command issued in line 44 runs the ImageMagick `convert` utility with the `-rotate` option, which writes the rotated resulting file to the second temporary file. If this fails, line 51 creates

LISTING 2: rotate-client

```
01 #!/usr/local/bin/perl -w
02 #############################
03 # rotate-client
04 # Mike Schilli, 2012
05 #   (m@perlmeister.com)
06 #############################
07 use strict;
08 use Sysadm::Install
09   qw(slurp blurt);
10 use Thrift;
11 use Thrift::BinaryProtocol;
12 use Thrift::Socket;
13 use
14   Thrift::BufferedTransport;
15
16 use lib 'gen-perl';
17 use image_process::Rotator;
18
19 my $socket =
20   Thrift::Socket->new(
21  "localhost", 9001);
22
23 my $transport =
24   Thrift::BufferedTransport
25   ->new($socket, 1024, 1024);
26
27 my $protocol =
28   Thrift::BinaryProtocol
29   ->new($transport);
30 my $client =
31   image_process::RotatorClient
32   ->new($protocol);
33
34 my ($image) = @ARGV;
35 die "usage: $0 image"
36   if !defined $image;
37
38 eval {
39  $transport->open();
40
41  my $image_data =
42    slurp $image;
43
44  my $action =
45    image_process::Rotation
46    ->new();
47  $action->image($image_data);
48  $action->angle(90);
49
50  my $rotated_image_data =
51    $client->rotate($action);
52
53  blurt $rotated_image_data,
54    "rotated-$image";
55
56  $transport->close();
57 };
58
59 if ($@ =~ m/image_process/
60  and exists $@->{why})
61 {
62  die $@->{why};
63 } elsif ($@) {
64  die $@;
65 }
```

an exception object, which is then thrown in line 53. Thrift magic picks up the exception and transfers it to the client, which throws it again. The `rotate()` method returns the image data in line 56. It is then picked up by the Thrift layer, wrapped up, and handed over to the client without the application logic having to lift a finger.

The main program beginning in line 60 simply uses predefined Thrift modules and calls `Thrift::ForkingServer` to launch a server that listens for client requests on port 9001 and forks a parallel process each time to handle an incoming request. After launching the server in a separate terminal, you can type the following client side:

```
./rotate-client image.jpg
```

After a short wait, the command comes back and drops a file named `rotated-image.jpg` into your current directory. Following Unix tradition, no output is generated if all goes well.

Notebook API

To address the Evernote API that I mentioned previously with the Thrift framework, developers first need to pick up an API key [2]. Because the new utility is a command-line script and not a web application, you need to select *Client Application* (Figure 4). You are then issued a "Consumer Key" and a "Consumer Secret" and can then play around with them on *sandbox.evernote.com*. You can go live on *evernote.com* after completing your tests and submitting a request to Evernote, which usually gets approved within a couple of hours.

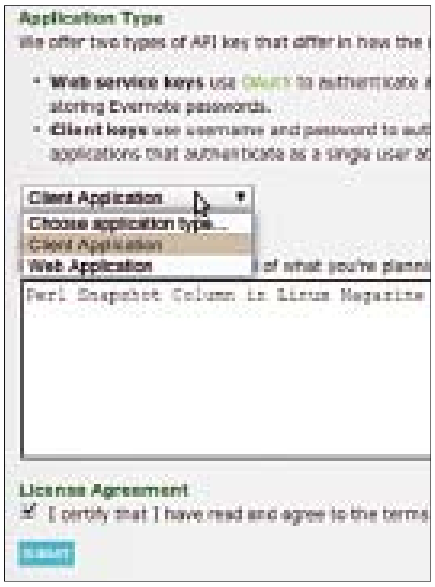
The Evernote Developer site also has a link to an SDK in ZIP format that contains many language modules, including a prebuilt Thrift wrapper for Perl. To convert the Thrift definitions of the Evernote data structures into Perl code using `thrift`, you need to do:

```
thrift -r --gen perl evernote-api-1.19/↩
  thrift/UserStore.thrift
thrift -r --gen perl evernote-api-1.19/↩
  thrift/NoteStore.thrift
```

assuming that you unpacked the SDK in `evernote-api-1.19`. The auto-generated `.pm` files are then stored in `gen-perl` below your current working directory.

Upgrading

Unfortunately, Thrift uses the obsolete `new Class()` notation to generate the Perl


Application Type
We offer two types of API key that differ in how the
- **Web service keys** use OAuth to authenticate
  storing Evernote passwords.
- **Client keys** use username and password to aut
  applications that authenticate as a single user ac

of what you're plann
Perl Snapshot Column in Linux Magazine

License Agreement
☑ I certify that I have read and agree to the terms

Figure 4: Developers can pick up the required API key from the Evernote Developer site.

LISTING 3: rotate-server

```
01 #!/usr/local/bin/perl -w
02 #############################
03 # rotate-server
04 # Mike Schilli, 2012
05 #    (m@perlmeister.com)
06 #############################
07 use strict;
08 use Thrift::Socket;
09 use Thrift::Server;
10
11 use lib 'gen-perl';
12 use image_process::Rotator;
13
14 #############################
15 package RotateHandler;
16 #############################
17 use base
18   qw(image_process::RotatorIf);
19 use Sysadm::Install
20   qw(slurp blurt tap);
21 use File::Temp qw(tempfile);
22
23 #############################
24 sub new {
25 #############################
26   my ($class) = @_;
27
28   return bless {}, $class;
29 }
30
31 #############################
32 sub rotate {
33 #############################
34   my ($self, $rotation) = @_;
35
36   my ($fh1, $infile) =
37     tempfile(UNLINK => 1);
38   my ($fh2, $outfile) =
39     tempfile(UNLINK => 1);
40
41   blurt $rotation->{image},
42     $infile;
43   my ($stdout, $stderr, $rc) =
44     tap "convert", "-rotate",
45     $rotation->{angle},
46     $infile, $outfile;
47
48   if ($rc != 0) {
49     my $x =
50       image_process::Failed
51       ->new();
52     $x->why($stderr);
53     die $x;
54   }
55
56   return slurp $outfile;
57 }
58
59 #############################
60 package main;
61 #############################
62 use Log::Log4perl qw(:easy);
63 Log::Log4perl->easy_init(
64   $DEBUG);
65
66 my $port = 9001;
67 my $handler =
68   RotateHandler->new();
69 my $processor =
70   image_process::RotatorProcessor
71   ->new($handler);
72 my $serversocket =
73   Thrift::ServerSocket->new(
74   $port);
75 my $forkingserver =
76   Thrift::ForkingServer->new(
77   $processor, $serversocket);
78
79 DEBUG
80 "Server starting on port $port";
81 $forkingserver->serve();
```

Mike:
Listing 4 has not been "modified" to fit our column width. -rls

LISTING 4: evernote-add

```
01 #!/usr/local/bin/perl -w
02 ###################################
   ########
03 # en-add - Add a note to Evernote
04 # Mike Schilli, 2012 (m@
   perlmeister.com)
05 ###################################
   ########
06 use strict;
07 use Thrift;
08 use Thrift::HttpClient;
09 use Thrift::BinaryProtocol;
10
11 use lib 'gen-perl';
12 use EDAMUserStore::Constants;
13 use EDAMUserStore::UserStore;
14 use EDAMNoteStore::NoteStore;
15 use EDAMErrors::Types;
16 use EDAMTypes::Types;
17
18 my $username        =
   "perlsnapshot";
19 my $password        = "*******";
20 my $consumer_key    =
   "perlsnapshot";
21 my $consumer_secret =
   "****************";
22
23 my( $message ) = @ARGV;
24 die "usage: $0 note" if !defined
   $message;
25
26 my $evernote_host = "evernote.com";
27 my $user_store_uri =
28     "https://$evernote_host/edam/
   user";
29 my $note_store_uri_base =
30     "https://$evernote_host/edam/
   note/";
31
32 my $http_client =
33   Thrift::HttpClient->new($user_
   store_uri);
34 my $protocol =
   Thrift::BinaryProtocol->new(
35   $http_client);
36
37 my $client =
38   EDAMUserStore::UserStoreClient->n
   ew(
39   $protocol);
40
41 my $version_ok =
42   $client->checkVersion(
   "perlsnapshot", 1,
43   19 );
44
45 if ( !$version_ok ) {
46   die "Version not ok";
47 }
48
49 my $result =
50   $client->authenticate( $username,
51   $password, $consumer_key,
52   $consumer_secret );
53
54 my $user = $result->user();
55
56 my $note_store_uri =
57   $note_store_uri_base .
   $user->shardId();
58
59 my $note_store_client =
60   Thrift::HttpClient->new($note_
   store_uri);
61
62 my $note_store_protocol =
63   Thrift::BinaryProtocol->new(
64     $note_store_client);
65
66 my $note_store =
67   EDAMNoteStore::NoteStoreClient->n
   ew(
68     $note_store_protocol);
69
70 my $notebooks =
71   $note_store->listNotebooks(
72     $result->authenticationToken()
   );
73
74 my $inbox_guid;
75
76 for my $notebook (@$notebooks) {
77   if ( $notebook->name() eq "Inbox"
   ) {
78     $inbox_guid = $notebook->guid();
79     last;
80   }
81 }
82
83 if ( !defined $inbox_guid ) {
84   die "No Inbox notebook found";
85 }
86
87 my $note = EDAMTypes::Note->new();
88 $note->title( $message );
89 $note->content();
90
91 my $created =
92   $note_store->createNote(
93   $result->authenticationToken(),
   $note );
94
95   # move new note to "Inbox"
96 $note_store->copyNote(
97   $result->authenticationToken(),
98   $created->guid(), $inbox_guid );
```

code, and the Perl version on my lab machine, `perl-5.10.1`, wouldn't swallow it. I did this to solve the problem:

```
find gen-perl -name '*.pm' \
  -exec perl -p -i -e \
  's/\bnew (.*?)\(/$1->new(/g;' {} \;
```

This command line rummages through all the auto-generated `.pm` files and replaces the obsolete syntax with the more common `Class->new()` format. The script in Listing 4 should work without any trouble now. The access credentials are given in lines 18 through 21; you will want to store them elsewhere for security reasons in a production script – preferably in an encrypted password safe.

Still Compatible?

Line 19 expects a single command-line argument as the note title; subjects with more than one word need to be quoted:

```
evernote-add ↩
  "Don't forget to buy the milk"
```

The script contacts the Evernote server, creates a new note with the title "Don't forget to buy the milk," leaves the body empty, and inserts the note into a notebook by the name of "Inbox" that I created previously (Figure 5).

In contrast to the image rotation test client created before, the script uses `Thrift::HttpClient` to communicate with the Evernote website via HTTP. The Thrift glue defines `EDAMUserStore::UserStoreClient`, and Listing 4 instantiates this client object for user authentication on Evernote in line 38. The `checkVersion` method called in line 42 uses the `EDAM_VERSION_MAJOR` and `EDAM_VERSION_MINOR` constants from the auto-generated code to check to see whether the SDK version is still compatible with the Evernote website.

EDAM stands for "Evernote Data Access and Management" and provides two different communication classes for interaction with the Evernote service. `EDAMUserStore::UserStoreClient` helps to authenticate the user with their user name, the password, the consumer key, and the consumer secret. If the Evernote server accepts this combination, it returns an authorization token, which the application can then use for a limited period of time for requests with the

EDAMNoteStore::NoteStoreClient. The latter is used for browsing and making modifications to the user's Evernote notepad.

If successful, the authenticate() method called in line 50 returns an object whose user() method provides a user object. The user object's shardId() method returns the user partition on Evernote. The user is assigned to the partition and has to append its short form to the basic URL of the web API when submitting requests. The authenticationToken() method returns the token that the application needs to submit along with any requests.

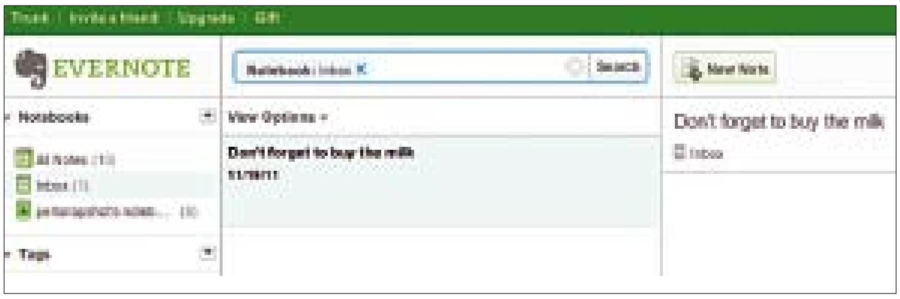
Browsing Notes

Line 71 runs listNotebooks() against the Evernote server; this command returns the names of all of the notebooks in the user's account. The server returns a reference to a Perl array, which the for loop in lines 76-81 iterates against. Each notebook object supplies the notebook name by calling the name() method. A unique ID to reference the notebook later is provided by guid(); it is required later to dump any new notes into it.

Line 77 now checks for every listed notebook if its name is "Inbox". Then, it interrupts the loop after storing the GUID for "Inbox" in the $inbox_guid variable. Line 87 creates a new note object with the title specified on the command line, simply by calling the EDAMTypes::Note class constructor and then setting the title appropriately via the title accessor. The call to content() intentionally leaves the content of the note empty.

The NoteStoreClient object's createNote() method sends the new note along with the authentication token to the server in line 92. If everything works out okay, the server returns a note object in the $created variable; its guid() method returns the GUID of the newly created note.

To make sure the new note ends up in the Inbox notebook, as shown in Figure 5, line 96 needs to send it there by using the copyNote() method and specifying the authentication token, the current GUID of the new note, and the GUID of the previously determined Inbox notebook.



Don't forget to buy the milk
11/18/11

Don't forget to buy the milk
Inbox

Figure 5: The note injected by the Perl script sitting in the *Inbox* folder.

Unlimited Versatility

Inventive users can whip up many other practical applications with the Evernote API. The Evernote app lets users export individual notebooks only, so it might be useful to create a backup script that works its way through your notebooks, extracting the content of the notes and storing the results in an XML-formatted backup file. If you accidentally delete a note in Evernote or mangle it beyond recognition, the free version won't let you recover an older version – so you'll be able to recover from mistakes on the cheap with this trick. If you use Evernote every day, you will definitely appreciate an intelligent approach to saving your brilliant ideas for posterity. ■■■

INFO

[1] Listings for this article: *http://www.linuxpromagazine.com/Resources/Article-Code*
[2] Evernote: *http://evernote.com*
[3] "9 Ways I Use Evernote" by David Pierce: *http://www.digitizd.com/2009/04/23/9-ways-i-use-evernote/*
[4] Allen, David, *Getting Things Done: The Art of Stress-Free Productivity*. Penguin, 2002
[5] Thrift: Scalable Cross-Language Services Implementation: *http://thrift.apache.org/static/thrift-20070401.pdf*
[6] Thrift Project: *http://thrift.apache.org/*

Layout: Info box needs to be re-aligned. Thnx. -rls

1/3 ad
with bleed