Judith: The code font is still difficult to see on the printed page. Can we get it darker?? -rls

Judith: I decreased the font size of @T so the whole title would fit. If it's too small, just use "Whispers." Thnx. -rls

A listening bot for IRC

Whispers from Beyond

A listening bot on an IRC channel wakes up when it hears certain keywords and notifies a defined user via instant messaging. *By Mike Schilli*

Open source projects like Catalyst use IRC channels to provide support; experts wait for user requests and then step in to give help. That said, IRC chat can make it difficult for the helpers to focus on their ongoing work. And, if the channel is full, conversations are always in full swing. The Perl bot I will describe in this article listens on a specific IRC channel and notifies its master when certain keywords occur.

The first step in creating an IRC bot, is fairly simple. After all, the CPAN Bot::BasicBot module that I've covered before provides an easily extensible framework for any kind of IRC bot. But how can the bot attract the attention of its hard-working user? Instant messaging with pop-up dialogs is one useful approach, and Pidgin provides a versatile client that supports common protocols such as Yahoo! Messenger, Google Talk, AIM, or MSN.

Chat via Web API

Some time ago, Yahoo! opened a web API [1] to its Messenger service whereby users would first log in and then use HTTP requests to exchange messages with other Yahoo! Messenger users. The bot script introduced here, `irc2ym`, joins an IRC channel and then just shuts up and listens (Figure 1). If a chat user mentions one of the keywords (Figure 2) in the `~/.irc-keywords` file, the bot launches the `ymsend` script, which logs into the Messenger Web API and sends the eavesdropped text message to a predefined Messenger account (Figure 3). The Messenger service then notifies the user, who immediately interrupts work, turns to the IRC channel, and contributes expert knowledge to help hapless newbies find their way around.

Sniffing Messages

Listing 1 [2] derives a `YMBot` class from the Bot::BasicBot base class on CPAN and overloads its `said()` method, which the bot calls whenever a user says something on an IRC channel. Along with a reference to the object, the bot passes a hash data structure to the method, containing the username in the `who` field and the message text in `body`.

In this callback method, the bot then calls the `keyword_match()` function, defined in line 58, and the function compares the message text with a dictionary of keywords parsed from the `~/.irc2ym-keywords` file (Figure 4). The script parses the entries in the file and stores them in the global `@KEYWORD_LIST` array. If one of the regular expressions stored in the `@KEYWORD_LIST` array fits the bill, line 27 of the same file triggers the `ymsend` script in the same directory. This script accepts the message text at the command line, logs in to the Web API, performs a couple of authorization steps based on the OAuth protocol, and finally sends the message text to the user defined in `$recipient` in line 11 of Listing 2.

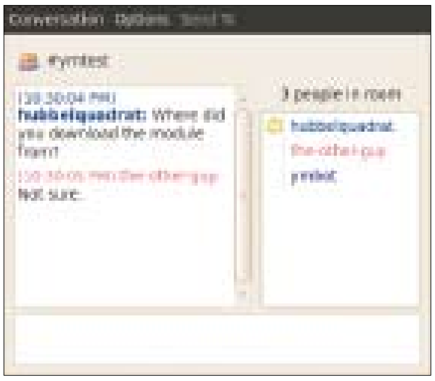
The script needs to jump through a few authentication hoops first, requiring the name of the sending Messenger user, their password, an API key that you need to retrieve from the Yahoo! Developer Network [3], and a shared secret for the application.

OAuth Jungle

The OAuth protocol [4] [5] lets an authenticated user pass a token to an application, which then acts on behalf of the user for a certain period of time. The beauty of the concept is that users don't have to tell the third-party application their password directly. The protocol authenticates just like other online offers at Yahoo's login screen, which then issues the token for the application to use. The concept makes a lot of sense with web applications, because users get trained never to enter their credentials on third-party sites, but

Mike:
irc2ym-keywords?? -rls

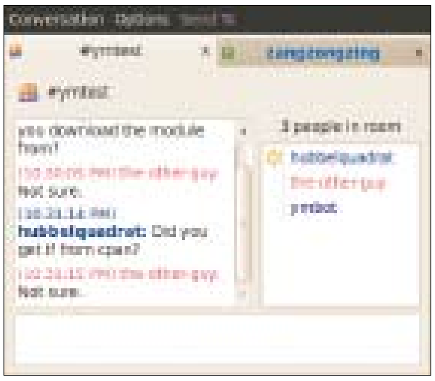
MIKE SCHILLI
ware
nny-
con-
tacted at *mschilli@perlmeister.com*. Mike's homepage can be found at *http://perlmeister.com*.



(10:30:04 PM)
hubbelquadrat: Where did you download the module from?
(10:30:05 PM) the-other-guy
Not sure.

hubbelquadrat
the-other-guy
ymbot

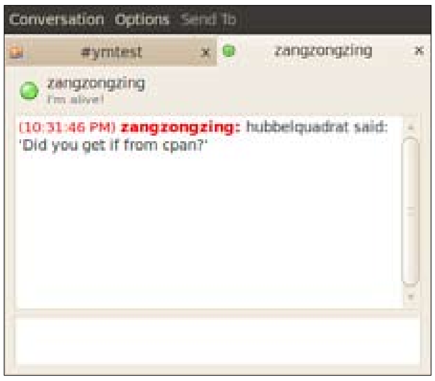
Figure 1: The ymbot does nothing until somebody mentions one of the predefined keywords.



you download the module from?
(10:30:05 PM) the-other-guy
Not sure.
(10:31:14 PM)
hubbelquadrat: Did you get if from cpan?
(10:31:15 PM) the-other-guy
Not sure.

hubbelquadrat
the-other-guy
ymbot

Figure 2: An IRC participant named "hubbelquadrat" mentions the "cpan" keyword, and the eavesdropping bot notifies the user.



(10:31:46 PM) **zangzongzing:** hubbelquadrat said: 'Did you get if from cpan?'

Figure 3: The bot has forwarded the message to the Y! Messenger user.

only on the original login screen of the provider. This issue is less evident with desktop applications like my script, which need the user's password anyway to authenticate at the login site behind the scenes.

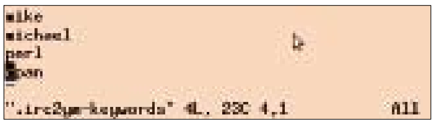
In the case of Y! Messenger, the token allows the application (i.e., the script) to send messages to the IM network and receive responses for one hour. Because the script runs very rarely and immediately quits after sending the message, storing the token wouldn't offer significant advantages. Thus, the script re-authenticates against the Yahoo login page, passing in a username and password (hard coded as `$user` and `$password` in `ymsend`) with every run, then picks up a new access token and uses it to run the send command in the web API.

In line 45, the `ymsend` script logs in the user as `$user` and `$passwd` at the URL stored in `$login_url`. Yahoo! sends back a request token in the body of the response.

The script then sends the token and the API key, with a matching secret key, `secret`, to the next URL, `$auth_token_url`, which then generates an access token, `oauth_token`, and an `oauth_token_secret`. The web server response uses

LISTING 1: irc2ym

```
01 #!/usr/local/bin/perl -w
02 use strict;
03 use local::lib;
04
05 #############################
06 package YMBot;
07 #############################
08 use base qw( Bot::BasicBot );
09 use FindBin qw($Bin);
10
11 my $ymsend = "$Bin/ymsend";
12 my ($home) = glob "~";
13 my $KEYWORD_LIST_FILE =
14   "$home/.irc2ym-keywords";
15 my @KEYWORD_LIST = ();
16
17 keyword_list_read();
18
19 #############################
20 sub said {
21 #############################
22  my ($self, $data) = @_;
23
24  if ( keyword_match(
25        $data->{body}) ) {
26
27   my $rc = system( $ymsend,
28     "$data->{who} said: " .
29     "'$data->{body}'" );
30
31   warn "$ymsend failed: $!"
32     if $rc;
33  }
34
35  return $data;
36 }
37
38 #############################
39 sub keyword_list_read {
40 #############################
41  if (!open FILE,
42      "<$KEYWORD_LIST_FILE") {
43   warn "$KEYWORD_LIST_FILE ",
44        "not found";
45   return;
46  }
47
48  while (<FILE>) {
49   chomp;
50   s/#.*//;
51   next if /^\s*$/;
52   push @KEYWORD_LIST, $_;
53  }
54  close FILE;
55 }
56
57 #############################
58 sub keyword_match {
59 #############################
60  my ($said) = @_;
61
62  for
63    my $regex (@KEYWORD_LIST)
64  {
65   return 1
66     if $said =~ /$regex/i;
67  }
68  return 0;
69 }
70
71 #############################
72 package main;
73 #############################
74 use Bot::BasicBot;
75
76 my $bot = YMBot->new(
77  server =>
78    "irc.freenode.com",
79  channels => ["#ymtest"],
80  nick     => "ymbot",
81  name     => "Relay to Y!M",
82  charset  => "utf-8",
83 );
84
85 $bot->run();
```



```
mike
michael
perl
ryan
"~/.irc2ym-keywords" 4L, 23C 4,1          All
```

Figure 4: The list of keywords to which the IRC bot will react.

the format `field=value&field=value ...`, which the script simply stores in a URI object in line 86 as a made-up query part of the URL. It then tells the `query_form` method to parse the object – this works because the data are formatted exactly like a URL using query parameters.

The combination of token and secret identifies the application as authorized by the user to use the web service on his behalf. The script then passes these on to the Messenger web service using the `$session_url`, which starts a new Messenger session and logs in the `$user` into the Yahoo! Messenger network. Once the session has started, other IM users see the user appear in their buddy lists, and the script uses the POST method in lines 148-155 to send the message passed in at the command line to the Messenger user defined in `$recipient` (who should be logged in). This last step involves encoding the request in JSON format as in:

```
{ message : "the message" }
```

If the message text also contains quotes, these non-standard characters must be encoded correctly. The `qquote` function exported by the Sysadm::Install CPAN module makes light work of this task.

Creating the Auth Token

To create an authentication token with a secret for the newly created application (i.e., the `ymsend` script), the API developer must click through *My Projects* and *New Project* (Yahoo! account required) on the Yahoo! Developer Network [3]. These steps will take you to the pop-up box shown in Figure 5. Because this is not a web application running in a browser, but a desktop client, you'll need to select *Or an application using these APIs: BOSS, Contacts, Mail, ...* .

In the form that appears, the developer must enter a short name (e.g., *irc2ymessenger*) and a couple of words of explanation as the description (Figure 6). The *Kind of Application* drop-down box must be set to *Client/Desktop* (not *Web-based*).

Below *Access Scopes*, you can then select *This app requires access to private user data*, then in the mass of sub-items that appears, just select the *Read/Write* option below the entry for *Yahoo! Messenger* (see Figure 7).

After accepting the conditions of use, you'll be given the keys you need to put the messenger client together (Figure 8). Cut and paste these into the strings in lines 15 and 16 of the `ymsend` script to set the `$api_key` and `$secret` variables.

In line 10 of the script, you'll also need to enter the password for the Messenger account sending the message. The username in the example is `zangzongzing`. If you don't have an account yet, you can simply press the *Sign Up* link to let yahoo.com take you to the account registration page.

After this, you only need to create a list of keywords in `~/.irc-keywords` and launch the `irc2ym` bot. The bot could take up to 20 seconds to log in to the preset channel on a heavily used IRC server, but then the bot will appear in the online list as *ymbot*.

Popular IRC clients include Irssi (for the command line), or Pidgin, the jack of all trades, which will display an ongoing chat once you are logged into the IRC channel.

If a channel participant uses one of the predefined keywords, `ymsend` will wake up and use the Messenger protocol to send the message to the predefined (and hopefully logged in) IM user, `$recipient`, in a dialog window. Now, it's time to help the newbies! ■■■

Mike: irc2ym-keywords?? -rls

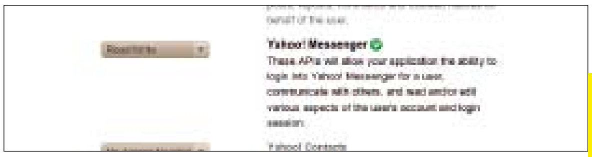

Yahoo! Messenger
These APIs will allow your application the ability to login into Yahoo! Messenger for a user, communicate with others, and read and/or edit various aspects of the user's account and login session.
Yahoo! Contacts

Figure 7: The application requires read/write access to Yahoo! Messenger data.

Judith: These figs are OK. -rls

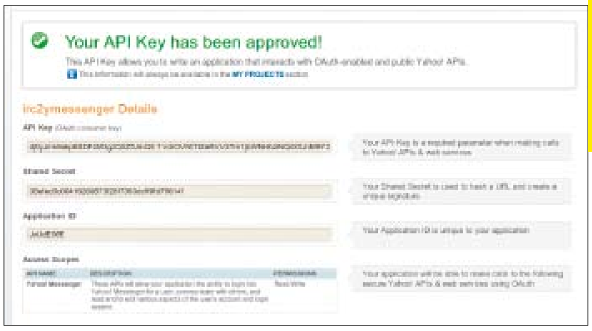


Figure 8: The ready-made API keys for creating the Y! Messenger client.

Judith: Figure 6 should be token-create-form. -rls

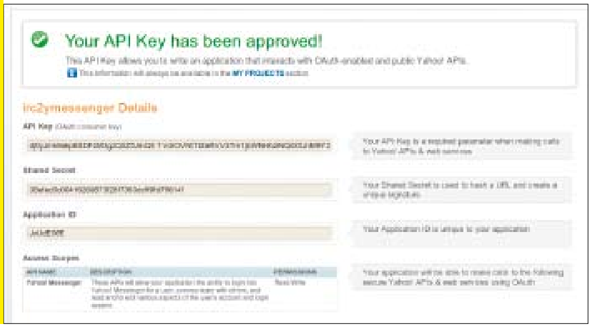


Figure 5: Developers need to request a consumer key for a desktop client application.

Judith: This is Figure 5. (Use the Fig. 5 caption above.)-rls

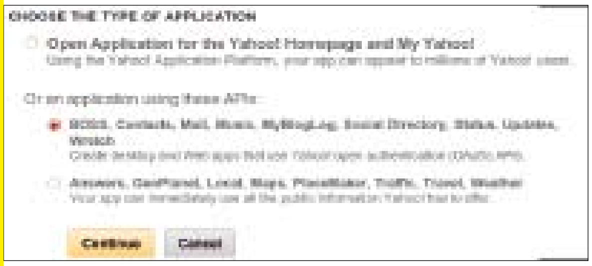

CHOOSE THE TYPE OF APPLICATION
Open Application for the Yahoo! Homepage and My Yahoo!
Or an application using these APIs:
BOSS, Contacts, Mail, Music, MyBlogLog, Social Directory, Status, Updates, Vitality
Answers, GeoPlanet, Local, Maps, PlaceMaker, Traffic, Travel, Weather
Continue
Cancel

Figure 6: The developer must request an authentication token for a desktop client application.

INFO

[1] Yahoo! Messenger IM API: *http://developer.yahoo.com/messenger/guide/ch02.html*

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

[3] Yahoo! Developer Network: *http://developer.yahoo.com/*

[4] Documentation for authentication token: *http://developer.yahoo.com/messenger/guide/chapterintrotomessengersdk.html*

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

LISTING 2: ymsend

```
#!/usr/local/bin/perl -w
use strict;
use LWP::UserAgent;
use Sysadm::Install
  qw(qquote);
use URI;
use JSON;

my $user   = "zangzongzing";
my $passwd = "*********";
my $recipient =
  "mikeschilli";

my $api_key =
  "******************";
my $secret = "*************";

my $login_url =
"https://login.yahoo.com/WSLogin/V1/get_auth_token";
my $auth_token_url =
"https://api.login.yahoo.com/oauth/v2/get_token";
my $session_url =
"http://developer.messenger.yahooapis.com/v1/session";
my $message_url =
"http://developer.messenger.yahooapis.com/v1/message/yahoo/$recipient";

my ($msg) = join ' ', @ARGV;

die "usage: $0 message"
  unless length $msg;

my $ua =
  LWP::UserAgent->new();

my $url =
  URI->new($login_url);

$url->query_form(
 login  => $user,
 passwd => $passwd,
 oauth_consumer_key =>
   $api_key
);

my $resp = $ua->get($url);

if ($resp->is_error()) {
 die
 "Can't get request token: ",
   $resp->message(), " ",
   $resp->content();
}

my ($request_token) =
  ($resp->content() =~
   /RequestToken=(.*)/);

$url =
  URI->new($auth_token_url);

$url->query_form(
 oauth_consumer_key =>
   $api_key,
 oauth_nonce =>
   int(rand 10000000),
 oauth_signature =>
   "$secret&",
 oauth_signature_method =>
   "PLAINTEXT",
 oauth_timestamp => time(),
 oauth_token =>
   $request_token,
 oauth_version => "1.0"
);

$resp = $ua->get($url);

if ($resp->is_error()) {
 die
"Can't get access token: ",
   $resp->message(), " ",
   $resp->content();
}

my $u = URI->new();
$u->query($resp->content());
my %form = $u->query_form;

$session_url =
  URI->new($session_url);

$session_url->query_form(
 oauth_consumer_key =>
   $api_key,
 oauth_nonce =>
   int(rand 10000000),
 oauth_signature =>
  "$secret&" .
  $form{oauth_token_secret},
 oauth_signature_method =>
   "PLAINTEXT",
 oauth_timestamp => time(),
 oauth_token =>
   $form{oauth_token},
 oauth_version => "1.0"
);

$resp = $ua->post(
 $session_url,
 Content_Type =>
  "application/json; " .
  "charset=utf-8",
 Content =>
  q[
  { "presenceState"   : 0,
    "presenceMessage" : "I'm alive!"
  }]);

if ($resp->is_error()) {
 die "Can't get session: ",
   $resp->message(), " ",
   $resp->content();
}

my $data = from_json(
 $resp->content());

$message_url =
  URI->new($message_url);

$message_url->query_form(
 oauth_consumer_key =>
   $api_key,
 oauth_nonce =>
   int(rand 10000000),
 oauth_signature =>
  "$secret&" .
  $form{oauth_token_secret},
 oauth_signature_method =>
   "PLAINTEXT",
 oauth_timestamp => time(),
 oauth_token =>
   $form{oauth_token},
 oauth_version => "1.0",
 sid => $data->{sessionId},
);

$resp = $ua->post(
 $message_url,
 Content_Type =>
   "application/json; " .
   "charset=utf-8",
 Content => '{ "message" : '
   . qquote($msg) . ' }'
);

if ($resp->is_error()) {
 die "Can't send message: ",
   $resp->message(), " ",
   $resp->content();
}
```