Configure Pidgin to play personal sounds

LISTEN TO THE SIGNALS

Perl helps Pidgin, the jack of all trades among instant messaging clients, assign different sounds to different communication partners and play them when a message arrives.

BY MICHAEL SCHILLI

If you use instant messaging at work, you might be familiar with the problem of incoming messages interrupting your concentration. Many of these messages could probably wait until later. On the other hand, if your manager sends you a message, you need to respond quickly. But how can you filter for urgent messages?

Pidgin, a messaging client [2] that supports all the popular communications protocols, such as Yahoo Messenger, AOL AIM, or IRC, offers what are known as "Buddy Pounces," which perform a specific action on a specific event. To set this up, you can access the dialog box via *Tools | Buddy Pounces* (see Figure 1). Among other things, you can play a predefined sound file or run an external program upon receiving a message ("Buddy sends a message" event) or when a specific communications partner signs on. To do so, you need to fill out the form shown in Figure 1, giving details of the IM service you are using, your

Figure 1: The "Pounce" function in Pidgin runs the ~/bin/pounce-sound script when IM buddy "bigboss" triggers an event.

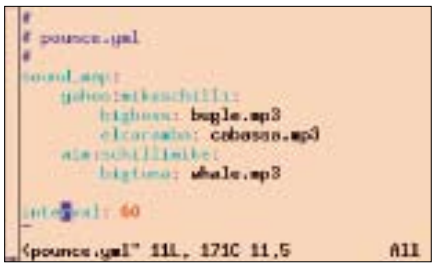

Figure 2: From the data in this YAML file, the script pounce-yml-to-xml generates the XML file in Figure 5.

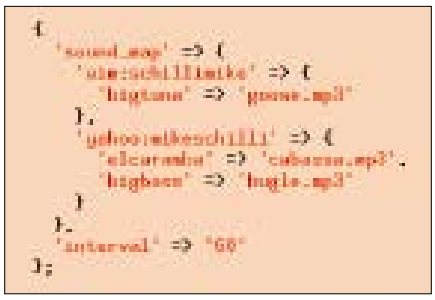

Figure 3: Using LoadFile() to parse the YAML data returns this data structure in Perl.

own account, and the name of your IM partner whose activity will trigger the event. To avoid having Pidgin enable the handler once only and then deleting your definition, it is important to check the *Recurring* box.

Slick Sound Deal

It just so happens that I went on a wild spending spree on Amazon.com a couple of weeks ago and bought 500 sound effects as downloadable MP3 files at the bargain price of US$ 2.59 (note the price has bounced back to US$ 7.99 since then) [3]. What a great opportunity to assign these sound effects to my favorite IM buddies! Once I've hidden the Pidgin chat window, only a uniquely identifiable sound for a select partner will interrupt my flow of thoughts. The rest of the world can wait.

To begin, I'll assign a bugle call (*bugle.mp3*) to the manager, a Samba rattle (*cabassa.mp3*) to a guy at work who is mad about dancing, and the sound of wild geese flying overhead (*goose.mp3*) to a friend on AIM who always has the latest gossip.

Regular readers of this column will understand that I wasn't in the least inclined to type all this stuff into Pidgin's dialog box for each buddy and to repeat the process for every single Pidgin instance, say, on my netbook and my home machine. Fortunately, Pidgin stores the pounce definitions in an easily readable XML file, *~/.purple/pounces.xml*.

Instead, the Perl *pounce-yml-to-xml* script expects a compactly formatted YAML file, like that in Figure 2, and converts it without much ado into the more convoluted XML format that Pidgin favors. The YAML file, *pounces.yml*, is stored in a newly created directory, *~/.pidgin-pounces*, where users can enter a sound file for each of their online buddies.

Runs with the Hare, Hunts with the Hounds

Because Pidgin uses a variety of IM protocols and users are typically logged in to multiple accounts on multiple servers at the same time, the *sound_map* entry in the YAML file is followed by a key that lists the IM provider and the user account with the provider in a colon-separated list. Below this level of the hierarchy are the online buddy mappings to the sound files that I want Pidgin to play. If you look at Figure 2, you will see that my screen name on the Yahoo messenger service is *mikeschilli*, and if *elcaramba* sends me a message, Pidgin will hammer out a hot Latin American beat in the form of *cabassa.mp3*.

Figure 4: Pidgin stores configured pounces in an XML file in the .purple folder below your home directory.

Filtering for Sound Overkill

Besides the buddy-to-sound mappings in *sound_map*, the YAML file also sets the *interval* parameter to avoid having the system play a sound whenever a message arrives. If *interval* is set to *60*, the minimum interval between two sounds is 60 seconds. If multiple messages arrive in this period, the system will automatically suppress the sound it would normally play to avoid exposing the poor user to sound overkill during work hours.

YAML data can be imported as is into Perl; the *LoadFile()* function exported from the YAML module converts them into the Perl data structure shown in Figure 3. The next step is to convert the data into Pidgin's bloated XML format.

From YAML to XML

The XML format used by Pidgin (Figure 4) defines a construct that is surrounded

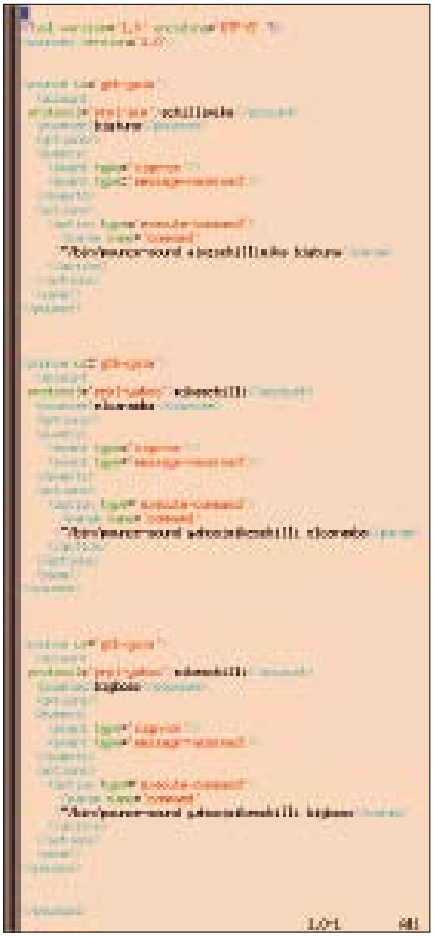

Figure 5: The XML generated by pounce-yml-to-xml, which, for example, tells Pidgin to call the external pounce-sound script when a message from bigboss reaches mikeschilli.

by <*pounce*> tags for each pounce. This totality of tags resides inside an umbrella tag called <*pounces*>, which can be seen with a bit of reverse engineering. Simply define multiple pounces in Pidgin's GUI and then inspect the new XML file this automatically creates (Figure 5).

The *pounce-yml-to-xml* script converts the compact YAML file from Figure 2 into explicit XML instructions for Pidgin then stores them in *~/.purple/pounces.xml* for Pidgin to pick up. The *for* loop beginning in line 21 iterates over the *keys* in the *sound_map* hash in the YAML file, which has a *protocol:account*-style format. At the hierarchy level, this is followed by another hash that maps my buddies to sounds. The *keys* function returns a list of these buddies, and line 34 calls the *mk_pounce()* function defined in lines 62-89 for each mapping entry. The parameters that the function expects are the name of the buddy, the account name (*recv* because this account will receive messages from the buddies), and the protocol to use (*yahoo*, *aim*, or similar). The function returns an XML string that matches a single pounce entry in Pidgin's *pounces.xml* configuration file.

Finicky Pidgin

To write the file in XML format, I used the CPAN Template module, which is a template processor that normally helps generate dynamic web pages. In the *pounce-yml-to-xml* script, it simply expands *[% variable %]*-style macros, and the practical *FOR* construct in the Template language helps avoid an awkward mix of Perl and XML code in line 45.

I started off experimenting with XML::Simple for this, but creating XML that looks as if Pidgin created it is more or less impossible. And Pidgin isn't exactly robust when it comes to the sequence of data in the XML file. If you want to see a really nasty Pidgin crash, just move the *pouncee* entry from the start of the *pounce* structure to the end. Anyway, the XML template works just fine, and before slipping the new file to Pidgin, the script calls the *mv* function from Sysadm::Install, to push aside a *pounce.xml* file that Pidgin might already be managing, and renames it to *pounces.xml.old*.

The *process()* method in line 56 expects an array of pounce XML strings, which the *FOR* loop in the Template code then iterates over. It inserts the parameterized XML in the correct places and then overwrites Pidgin's *pounces.xml* with the new data. The results, which you can see in Figure 5, are a lengthy slab of XML for every combination of protocol, account, and buddy. The external *pounce-sound* program calls all the instances of this XML snippet with these parameters.

Shorten Sounds

The sound files you want to map to your online buddies must be installed in the *sounds* directory below the *.pidgin-pounces* directory. The sounds shouldn't be more than a couple of seconds in length. If you need to curtail a longer

Listing 1: pounce-yml-to-xml

```
01 #!/usr/local/bin/perl -w
02 use strict;
03 use Sysadm::Install qw(:all);
04
05 use YAML qw(LoadFile);
06 use Template;
07
08 my ($home) = glob "~";
09 my $path =
10   "$home/.pidgin-pounce";
11 my $yaml = LoadFile(
12   "$path/pounce.yml");
13
14 my $xml_file =
15  "$home/.purple/pounces.xml";
16 my $SOUND_CMD =
17   "~/bin/pounce-sound";
18
19 my @pounces = ();
20
21 for my $account (
22   keys
23    %{ $yaml->{sound_map} } ) {
24
25   my ( $proto, $recv ) =
26     split /:/, $account;
27
28   for my $buddy (
29     keys %{
30       $yaml->{sound_map}
31             ->{$account} }) {
32
33     push @pounces,
34       mk_pounce( $buddy,
35       $recv, $proto );
36   }
37 }
38
39 binmode STDOUT, ":utf8";
40
41 my $xml = q{
42 <?xml version='1.0'
43       encoding='UTF-8' ?>
44 <pounces version='1.0'>
45 [% FOR pounce IN pounces %]
46   [% pounce %]
47 [% END %]
48 </pounces>
49 };
50
51 my $tmpl = Template->new();
52
53 mv $xml_file, "$xml_file.old"
54   if -f $xml_file;
55
56 $tmpl->process( \$xml,
57   { pounces => \@pounces },
58   $xml_file )
59   or die $tmpl->error;
60
61 ############################
62 sub mk_pounce {
63 ############################
64   my ( $buddy, $recv, $prot )
65     = @_;
66
67   return qq{
68 <pounce ui='gtk-gaim'>
69   <account
70    protocol='prpl-$prot'
71   >$recv</account>
72   <pouncee>$buddy</pouncee>
73   <options/>
74   <events>
75     <event type='sign-on'/>
76     <event
77    type='message-received'/>
78   </events>
79   <actions>
80     <action
81      type='execute-command'>
82       <param name='command'
83       >$SOUND_CMD $prot:$recv
   $buddy</param>
84     </action>
85   </actions>
86   <save/>
87 </pounce>
88     }
89 }
```

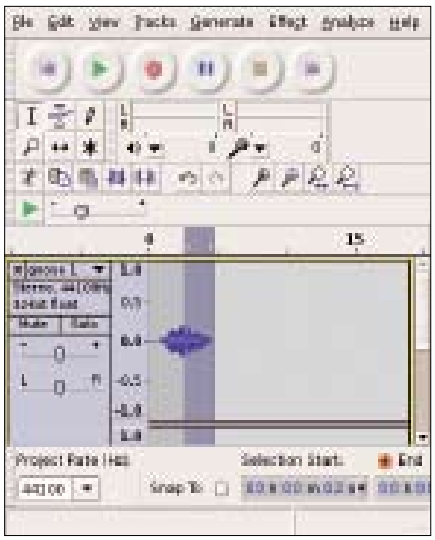

Figure 6: The Audacity tool trims the sound to the right size and allows it to gently fade out by applying the Fade In/Out effects.

sound file, just launch Audacity, trim the sound to the required length, and apply the fade in and fade out effects to ensure smooth audio (see Figure 6).

The *pounce-sound* script, called by Pidgin when an event occurs, expects a protocol and account combination, along with the nick of the sending user, as its command-line parameters. It reads in the YAML file and determines whether a special sound already exists for the user. If so, it uses the *play* utility of the handy *sox* package to output the sound via the soundcard. For example, if Pidgin receives a message from *bigboss* via the Yahoo protocol and addressed to a logged in user called *mikeschilli*, it calls:

```
pounce-sound yahoo:mikeschilli bigboss
```

The script *pounce-sound* then checks the YAML file, discovers that the *bugle.mp3* file is mapped for this case, and plays the file.

The only unusual thing about *pounce-sound* is the call to module Data::Throttler, which avoids annoying the user by playing a sound too often. Data::Throttler creates a YAML file in *~/.pidgin-pounce/throttler.yml* and remembers the number and time of the previous calls. The *try_push()* function stores a key comprising the protocol, account name, and buddy name and returns FALSE if the buddy was active within the last *interval* seconds. It lets other buddies through – until they overstep their quota, of course.

Installation

The *pounce-sound* script, called by Pidgin when an event occurs, must be saved in the *bin* directory below your home directory. Then you need to run the *chmod +x* command to make it executable.

The CPAN YAML, Template, and Sysadm::Install CPAN modules are available in many distributions via their package managers. The Data::Throttler module file can be installed in a CPAN shell by entering *perl -MCPAN -e'install Data::Throttler'*. The *play* utility that the *pounce-sound* program uses to output the MP3 file is part of the *sox* package, which you can install on Ubuntu with:

```
sudo apt-get install sox
```

Your sound files, with fade in and fade out as appropriate, must reside in the *sounds* subdirectory below *~/.pidgin-pounces*. Then, you need to modify the YAML file in Figure 2 to reflect your *interval* preference, stop the Pidgin process if it is running, and call the *pounce-yml-to-xml* script. If you want to sound a warning when messages arrive, but not when your buddies log on, you can delete the *sign-on* event from the XML in line 75 of *pounce-yml-to-xml*.

When it's restarted, Pidgin grabs the automatically generated XML file and, if asked to do so, shows you the new configuration in the Pounce dialog. When a defined event occurs, Pidgin calls the pounce defined for it, which in turn, sounds the bugle to interrupt your power nap. Okay, boss, I'm almost done! ■

Listing 2: pounce-sound

```
01 #!/usr/local/bin/perl -w
02 use strict;
03 use YAML qw(LoadFile);
04 use Data::Throttler;
05
06 my ( $proto, $buddy ) =
07   @ARGV;
08
09 die
10  "usage: $0 proto:recv buddy"
11   if !defined $buddy;
12
13 my ($home) = glob "~";
14 my $path =
15   "$home/.pidgin-pounce";
16
17 my $yaml = LoadFile(
18   "$path/pounce.yml");
19
20 my $throttler =
21   Data::Throttler->new(
22     max_items => 1,
23     interval =>
24       $yaml->{interval},
25     backend         => "YAML",
26     backend_options => {
27       db_file =>
28         "$path/throttle.yml",
29     },
30   );
31
32 if ( !$throttler->try_push(
33     key => "$proto:$buddy"
34   )) {
35     # rate limit reached, skip it
36   exit 0;
37 }
38
39 if ( exists
40      $yaml->{sound_map}->
41        {$proto}->{$buddy} ) {
42   my $sound =
43     $yaml->{sound_map}
44     ->{$proto}->{$buddy};
45   system(
46    "play $path/sounds/$sound"
47   );
48 }
```

INFO

[1] Listings for this article: *ftp://www.linux-magazin.de/pub/listings/magazin/2010/02/Perl*

[2] Pidgin: *http://www.pidgin.im*

[3] "500+ Sound Effects" in MP3 format: *http://www.amazon.com/gp/product/B002OVD5FK*

THE AUTHOR

Michael Schilli works as a software engineer with Yahoo! in Sunnyvale, California. He is the author of *Goto Perl 5* (German) and *Perl Power* (English), both published by Addison-Wesley, and he can be contacted at *mschilli@perlmeister.com*. Michael's homepage is at *http://perlmeister.com*.