Serve a daily dose of information

APPETIZERS


catsi, Fotolia

When tackling something complex, such as a foreign language or Vim commands, digesting small bits of knowledge might be more effective than consuming a super-sized meal of information. This month's Perl column gives you a method of serving up knowledge snippets by email. **BY MICHAEL SCHILLI**

No matter how good your command of English might be, there is always room for improvement. Or do you already know what "cynosure" or "exonym" means? Because I subscribe to "A.Word.A .Day" (AWAD) [2], I receive a new word every day by email. Figure 1 shows how the service describes the daily word in simpler terms and provides examples of the word in action.

A Tip a Day

Because this approach is so effective, I thought of extending it to other fields. If there are daily tips for the Vim editor, why not have them for Perl? Or for the Java pitfall of the day? All it would take would be a script that stores the tips, and a cronjob that sends the tips to a list of subscribers every morning.

Figure 1: Expanding vocabulary with A.Word.A.Day.

The xaday script (Listing 1) uses an SQLite database [3] to store the tips but supports editing of tips with a normal editor. When called with the *-m* parameter and an email address, *xaday* will send off one tip, remember the date of publication in the database, and find an unpublished tip when called upon again. The script works its way through the queue, day by day, entry by entry, until it runs out of entries. To add new tips to the queue, call *xaday -e* and add the tips to the database (Figure 2).

As with Perl's POD format, the tips are separated by *=head1* headers in the edited file. The headline becomes the subject line, and the body text becomes the body of the outgoing email tip. When you store the modified file, *xaday* pushes the content back into the database. You can add new tips and modify enqueued tips or tips that have already been published.

Tip on the Table

Figure 3 shows where the information snippets end up in the *tips* table of the database. The SQLite client might not have a smart ASCII art display like MySQL, but playing with the *.width*, *.mode*, and *.headers* formatting commands improves output readability.

The application just needs a simple table with the columns *head*, *text*, and *published* to store the heading, the body text, and the date of publication as your tip archive. The SQL commands shown in *xaday.sql* (Figure 4) set up the table. The SQLite client, *sqlite3*, runs the commands as follows:

```
sqlite3 dbname.dat <xaday.sql
```

The database ends up in a regular file named *dbname.dat*. The Perl client later will use SQL commands to access this unusual database.

A Bed of Roses

Querying a database usually involves writing SQL and embedding it in Perl; *xaday* uses *DB::Rose* from CPAN, instead. The Rose loader takes a quick sniff of the database table when launched; the *make_classes()* method automatically creates the mappings in your Perl code.

Users can call *xaday* with the *-e* option to add more tips. Line 39 in Listing 1 creates a temporary file that feeds the editor for this purpose.

The user needs to append the new tip with a *=head1* header. The *EDITOR* en-

vironment variable tells you which editor is currently set; *vim*, *emacs*, or even *pico* are common settings.

File Changes

After calling the editor, the script checks to see whether the edited file has changed or whether the user has decided to discard the changes.

In the latter case, the program exits with a message. If the file has changed, then the program has to push the changes into the database. To help the program decide which entries are from the database and which are new input from the user, *xaday* appends an ID in curly brackets to every header in the database before displaying the file in the editor. If the file to be edited has a line that reads "*=head1 Title {13}*", the entry must be from the line with ID 13 in the database.

Modifying the numbers will change the order of the entries because the email client simply follows those IDs in ascending order. When the program sees a new entry without an ID, it assigns a new number and adds the entry to the database. The *AUTOINCREMENT* mechanism in the SQL definition makes sure that new entries are assigned unused IDs in ascending order.

Digging Down the Stack

To dispatch mail, the script needs to locate the tip with the lowest ID that does

Listing 1: xaday

```
001 #!/usr/bin/perl -w
002 ################################
003 # xaday - A tip every day
004 # m@perlmeister.com, 2007
005 ################################
006 use strict;
007 use Rose::DB::Object::
    Loader;
008 use Getopt::Std;
009 use File::Temp qw(tempfile);
010 use Sysadm::Install qw(:all);
011 use Mail::Mailer;
012
013 my $RECSEP    = qr/^=head1/;
014 my $HEAD      = "=head1";
015 my $MAIL_FROM =
016   'me@_foo.com';
017
018 getopts( "d:lepm:f:",
019   \my %opts );
020
021 die "usage: $0 -d dbfile
    ..."
022   unless $opts{d};
023
024 my $loader =
025   Rose::DB::Object::Loader
026   ->new(
027   db_dsn =>
028 "dbi:SQLite:
    dbname=$opts{d}",
029   db_options => {
030     AutoCommit => 1,
031     RaiseError => 1
032   },
033   );
034
035 $loader->make_classes();
036
037 if ( $opts{e} or $opts{l} ) {
038   my ( $fh, $tmpf ) =
039     tempfile( UNLINK => 1 );
040   my $tips =
041     Tip::Manager
042     ->get_tips_iterator();
043   my $data_before = "";
044
045   while ( my $tip =
046     $tips->next() )
047   {
048     $data_before .= "$HEAD "
049       . $tip->head() . " {"
050       . $tip->id() . "}"
051       . "\n\n"
052       . $tip->text()
053       . "\n\n";
054   }
055   if ( $opts{l} ) {
056     print $data_before;
057     exit 0;
058   }
059   blurt( $data_before,
060     $tmpf );
061   system(
062     "$ENV{EDITOR} $tmpf");
063   my $data_after =
064     slurp($tmpf);
065   die "No change"
066     if $data_before eq
067     $data_after;
068
069   db_update( \$data_after );
070 }
071
072 if ( $opts{f} ) {
073   db_update( $opts{f} );
074 }
075
076 if ( $opts{m} ) {
077   my $tips =
078     Tip::Manager->get_tips(
079     query => [
080       "published" => undef
081     ],
082     sort_by => 'id',
083     limit   => 1,
084     );
085   if (@$tips) {
086     $tips->[0]->published(
087       DateTime->today() );
088     $tips->[0]->update();
089     mail(
090       $opts{m},
091       $tips->[0]->head(),
092       $tips->[0]->text()
093     );
094   }
095   else {
096     die
097 "Nothing left to publish";
098   }
099 }
100
101 ################################
102 sub text2db {
103 ################################
104   my ($text) = @_;
105   $text = ""
106     unless defined $text;
107
108   my @fields = ();
109
110   while ( $text =~
111     /^($RECSEP.*?)
112       (?=$RECSEP|\s*\Z)/smgx
```

not have a date in its *published* column, but a *NULL* value instead.

In line 78, *get_tips()* fires off an SQL query under the hood to search for all records with a *published* entry of *NULL* and sorts the resulting list in ascending order of IDs.

The limit is set to *1* to make sure that only the first result is returned from the database. The *published* method passes *DateTime->today()* as a parameter to assign today's date to the returned entry before *update()* stores the change in the database.

After using all the tips, the script dies with an error message in line 96, which causes the cronjob to notify the user of this issue by email.

Looking to the Future

The *text2db()* function uses a regular expression with a positive look ahead to separate individual tip entries. The construct that starts with *(?=* does not consume matching expressions, but simply takes a sneak preview.

After finding a paragraph that starts with *=head1*, *text2db()* passes it on to the *rec_parse()* function defined in line 126. The function attempts to extract the column values for *head*, *id*, and *text*. If it is successful, the function returns all three values; otherwise, it returns *undef*. The *text2db()* function also performs some cosmetic surgery and removes white space at the top and tail.

Listing 1: xaday

```
113     )
114   {
115     my ( $head, $info, $tip )
116       = rec_parse($1);
117     $tip =~ s/\s+\Z//;
118     $tip =~ s/\A\s+//;
119     push @fields,
120       [ $head, $info, $tip ];
121   }
122   return \@fields;
123 }
124
125 ##############################
126 sub rec_parse {
127 ##############################
128   my ($text) = @_;
129
130   if ( $text =~
131     /$RECSEP\s+(.*?)
132      (?:\s+\{(.*?)\})?
133      $
134      (.*)
135     /smgx
136     )
137   {
138     return ( $1, $2, $3 );
139   }
140
141   return undef;
142 }
143
144 ##############################
145 sub db_update {
146 ##############################
147   my ($in) = @_;
148
149   my $data;
150
151   if ( ref($in) ) {
152     $data = $$in;
153   }
154   else {
155     $data = slurp($in);
156   }
157
158   my $fields =
159     text2db($data);
160
161   my @keep_ids =
162     map { $_->[1] } @$fields;
163   my $gone;
164   if (@keep_ids) {
165     $gone =
166       Tip::Manager
167       ->delete_tips(
168       where => [
169         "!id" => \@keep_ids
170       ]
171       );
172   }
173   else {
174     $gone =
175       Tip::Manager
176       ->delete_tips(
177       all => 1 );
178   }
179   print
180     "$gone rows deleted\n"
181     if $gone;
182
183   for (@$fields) {
184     my ( $head, $info, $tip )
185       = @$_;
186
187     my $rec;
188
189     if ( defined $info ) {
190       $rec =
191         Tip->new(
192         id => $info );
193       $rec->load();
194       $rec->head($head);
195       $rec->text($tip);
196       $rec->update();
197     }
198     else {
199       $rec = Tip->new(
200         text => $tip,
201         head => $head,
202       );
203       $rec->save();
204     }
205   }
206 }
207
208 ##############################
209 sub mail {
210 ##############################
211   my ( $to, $head, $body ) =
212     @_;
213
214   my $mailer =
215     Mail::Mailer->new();
216
217   $mailer->open(
218     {
219       'From' => $MAIL_FROM,
220       'To'   => $to,
221       'Subject' => $head,
222     }
223   );
224   print $mailer $body;
225   close $mailer;
226 }
```

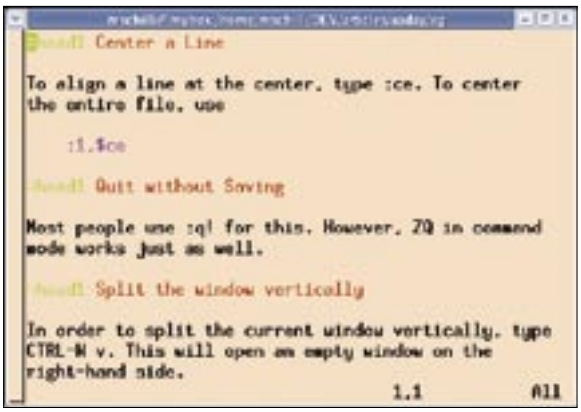

Figure 2: The sender can add new tips manually with a normal editor, like the three Vim tips here.

The *db_update()* function in line 145 expects either a filename (as a scalar) or a string (as a reference) from which to take input data.

The function calls *text2db()*, extracts the headlines and bodies of all tips, and discovers which entries the user has deleted from the original database. The function stores all IDs of preserved and new records in the *@keep_ids* array.

Refreshing Rose

The *delete_tips()* method uses the "*!id*" = > *\@keep_ids* condition to delete records from the database for those ID fields that do not match any of the values in the *@keep_ids* array.

If the array is empty, *DB::Rose* refuses to delete all the entries. In this case, line 176 of the *else* branch issues a *delete_tips()* call with the *all* flag set. The script displays the number of rows deleted on standard output in both cases.

Line 183 iterates over all the tips defined in the editor. The script updates the corresponding database entry for those tips that have a predefined ID, as indicated by a defined scalar *$info*. To do this, Rose first runs *load()* to load the content of a newly created *Tip* class object from the database; updates the individual fields with the *head()*, *text()*, and other methods; and then calls *update()* to write the data from local memory into the database.

If *$info* is undefined, lines 199 through 201 simply create a new object and set the values via the *head* and *text* methods. The *save()* method inserts the record into the database, and a new unused ID is assigned via the database's *AUTOINCREMENT* mechanism.

Outgoing mail is dispatched by the *Mail::Mailer* module from CPAN. The *mail()* function in line 209 only expects the receiving address, the header, and the content of the tip and then talks to the local Sendmail daemon to get it delivered to the recipient.

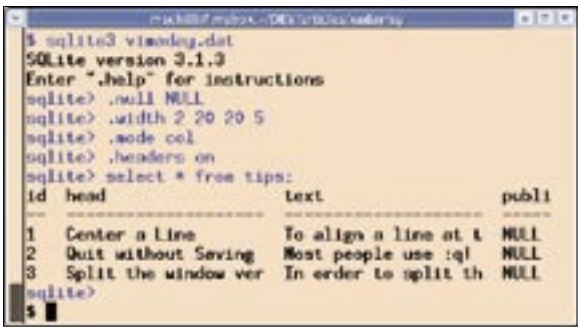

Figure 3: The tips are stored in an SQLite database. With just a couple of formatting instructions, the client will display the table in a readable way.

Installation

All the modules I have used here are available from CPAN, and a CPAN shell will quickly resolve the dependencies. The *MAIL_FROM* variable in line 15 must be modified to reflect the mailing list used for the tips. To allow the script to send mail at 7:30am every morning, the line

```
30 07 * * * /path_to/xaday
-d /path_to/dbfile.dat -m
mlist@somewhere.com
```

creates a cronjob. *dbfile.dat* is the SQLite file, and the mailing list specified with *-m* is the target address from which subscribers will anxiously await new tips every day. To fill the database with tips, just call

```
xaday -d
/path_to/dbfile.dat -e
```

add your tips in the editor that automatically launches, and save your changes. As an alternative to the editor, you can pass in a pre-filled text file to the script using the *-f* option. The *-l* option lets you display the data fed to the database thus far to check whether your tips have reached the database correctly.

Extensions

If you prefer to dispatch tips on weekdays only, you can either modify the cronjob or change the script to query the day of the week and quit if it happens to be a Saturday or Sunday.

The Perl *(localtime(time))[6]* construct will give you the weekday as a number from 0 (Sunday) to 6 (Saturday). To have a break on national holidays, you can add an extra table *holidays*, which stores national holidays in its *date* column. The script could offer a command-line option, such as *-D*, to add a specified date to the table.

Figure 4: The xaday.sql file is passed in to the SQLite client. The SQL commands create a database table.

Conclusion

Of course, the tips themselves really are your most important commodity. It makes sense to compose tips for a week in advance, for example, to avoid a couple of vacation days from interrupting the tip service.

And, for the record, "cynosure" is an object that serves as a focal point of attention; an "exonym" is a place name not used by the local inhabitants. ■

INFO

[1] Listings for this article: *http://www.linux-magazine.com/Magazine/Downloads/84*

[2] A.Word.A.Day (AWAD): *http://wordsmith.org/awad/*

[3] SQLite: *http://www.sqlite.org*

THE AUTHOR

Michael Schilli works as a Software Developer at Yahoo!, Sunnyvale, California. He wrote "Perl Power" for Addison-Wesley and can be contacted at *mschilli@perlmeister.com*. His homepage is at *http://perlmeister.com*.