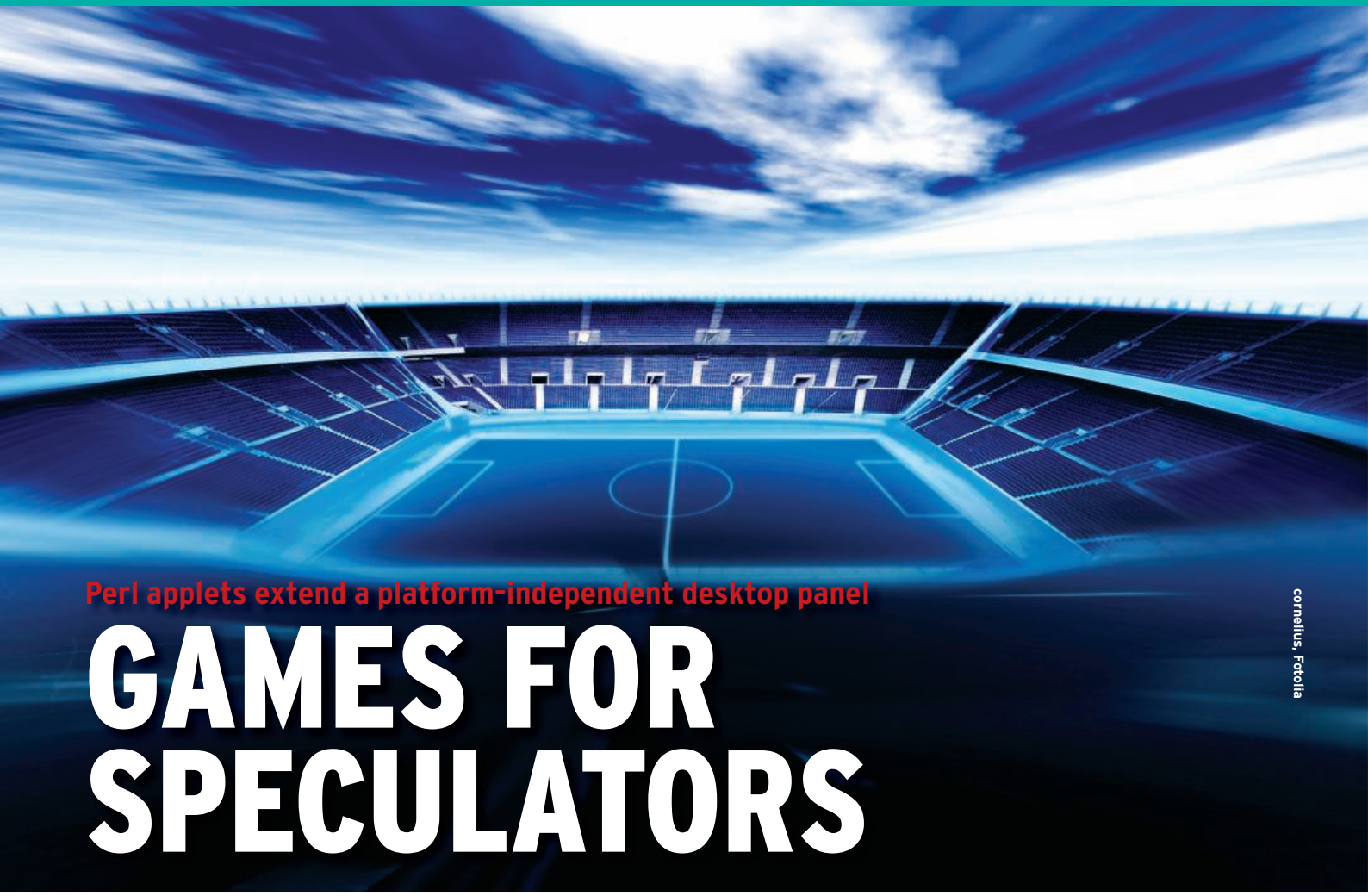

Perl applets extend a platform-independent desktop panel

GAMES FOR SPECULATORS

cornelius, Fotolia

One panel has a neat collection of applets and another has spectacular looks – but a combination of the two is rare. Now help draws nigh for the desktop: PerlPanel is extensible with do-it-yourself widgets.

BY MICHAEL SCHILLI

Regardless of whether you use Gnome or KDE, every desktop offers panels. They dock at the bottom or top of your screen, giving a home to menus and icons for launching programs, showing active applications in taskbars, or helping you switch between virtual desktops. The aim of the PerlPanel project is to use Perl to provide a platform-independent panel. At the same time, PerlPanel aims to let users add their own applets, simply by hashing up a couple of scripts.

Trial Run

On Ubuntu, you can install PerlPanel and the Perl modules on which it depends by typing *sudo apt-get install perlpanel* at the command line. To take it for a trial run, type */usr/bin/perlpanel*. Figure 1 shows the panel GUI at the bottom of the desktop.

If the space at the bottom of the screen is already occupied by another panel, you can move it to the right or left border, or just ditch it if you feel brave enough to burn some bridges behind you.

"Ticker" Widget

If you would like to keep track of share prices, a standard application might not be the best option for you as share price ticker windows tend to hide other applications. Instead, a panel applet might be the ideal choice for the budding investor as it lets you keep a constant eye on the latest share price developments, no matter which application you are currently working with. To allow this to happen, the applet checks the share prices you are interested in by querying Yahoo Fi-

Listing 1: getquote

```
01 #!/usr/local/bin/perl -w
02 use strict;
03 use Finance::YahooQuote;
04
05 $Finance::YahooQuote::TIMEOUT
06   = 60;
07
08 exit 0 unless @ARGV;
09
10 my @quotes = getcustomquote(
11  [@ARGV],
12  [
13   "Symbol",
14   "Last Trade (Price Only)"
15  ]
16 );
17
18 if (!exists $quotes[0][1]) {
19  die
20    "Fetching quote failed\n";
21 }
22
23 for my $quote (@quotes) {
24  print "@$quote\n";
25 }
```

Figure 1: PerlPanel lets you add do-it-yourself applets.

nance every 5 minutes and displays the results on screen (the ticker symbols to watch are stored in a local configuration file, *~/.ticker-rc*, in your home directory). The applet ignores comment and empty lines in the configuration file and expects a ticker symbol in every line. Figure 1 shows the applet display for selected prices in the panel on the desktop.

The *getquote* script in Listing 1 queries the Yahoo server to reel in share prices; it expects a list of share symbols on the command line, and outputs the latest share prices line by line. The applet code proper is shown in Listing 2, *Ticker.pm*. The applet simply parses the configuration file, calls the *getquote* script at five minute intervals, and refreshes the display in the PerlPanel with the returned values. If an impatient user clicks the applet instead, it assumes they can't wait until the next update; in this case, the applet rushes off to ask the Yahoo server and updates the panel display straight away.

Get Those Quotes!

The share symbols passed in to *getquote* at the command line are stored in the *@ARGV* array in typical Perl style. The *getcustomquote* function in the Finance::YahooQuote module retrieves the share prices from the Yahoo server, specifying that it is only interested in the fields *Symbol* (the share name symbol specified at the command line), and *Last Trade (Price Only)*. Without this restriction, Yahoo would return a whole bunch of data the applet doesn't need at every *getquote* call, so it's better to say up front what you need and what you don't.

If a transmission error occurs, a single value with an error message is returned; if the query is successful, Yahoo returns an array whose entries are pointers to arrays with the symbol and last trade values. Line 18 checks if the returned results really are two-column entries and bails out immediately if not. Line 5 sets a timeout of 60 seconds; if a network error occurs, this will prevent the script from waiting for ever. Share prices that reach the script are sent to its standard output by the *for* loop in Line 23. and formatted as *Symbol LastTrade* on every line of output.

Strictly by the Rules

Listing 2 *Ticker.pm* contains the applet code and has to follow the PerlPanel's rules. This includes a module in the PerlPanel::Applet::Name name space, a *new()* constructor, and an initial *configure()* function, which the panel runs first. The *expand()* and *fill()* functions stipulate how the widget changes if more space becomes available in the panel. *widget()* must return a pointer to the topmost Gtk widget in the applet, and *get_default_config()* normally returns a structure that configures the applet. However, as the configuration is stored in an external file in our case, to avoid any changes by the user requiring a restart of the applet, the function only returns *undef* here.

Enough said about the PerlPanel API – what we need now is the application code. The applet uses *configure()* to build

its GUI, which comprises a label with share price data and a button that handles user clicks. Line 42 defines this using the widget's *signal_connect()* method, which assigns an anonymous subroutine to the *clicked* event; the subroutine calls the applet's *stocks_update()* method. After all the widgets have been defined, *show_all()* draws them in the panel. Line 56 calls *stocks_update()* for the first time before the program uses *add_timeout()* to define an event that reoccurs every five minutes (5 * 60 * 1000 milliseconds), which also calls *stocks_update()*. The function in turn calls *symbols()* to parse the *~/.ticker-rc* file, thus immediately catching short-term symbol updates by the user.

Comment lines, blanks and anything that does not look like a ticker symbol is simply ditched by the function; later on,

Listing 2: Ticker.pm

```
001 package
002   PerlPanel::Applet::Ticker;
003 use strict;
004 use Log::Log4perl qw(:easy);
005
006 my $REFRESH = 5 * 30_000;
007 my ($CFG_FILE) =
008   glob "~/.ticker-rc";
009 my $GETQUOTE =
010   "/usr/bin/getquote";
011
012 Log::Log4perl->easy_init(
013  {
014   level => $DEBUG,
015   file  =>
016     ">>/tmp/ticker.log",
017  }
018 );
019
020 ############################
021 sub new {
022 ############################
023  my ($package) = @_;
024
025  my $self = {};
026  bless($self, $package);
027  return $self;
028 }
029
030 ############################
031 sub configure {
032 ############################
033  my ($self) = @_;
034
035  $self->{label} =
036    Gtk2::Label->new(
037   "Ticker");
038
039  $self->{widget} =
040    Gtk2::Button->new();
041  $self->{widget}
042    ->signal_connect(
043   'clicked',
044   sub {
045    $self->stocks_update();
046   }
047    );
048
049  $self->{widget}
050    ->set_relief('none');
051  $self->{widget}
052    ->add($self->{label});
053
054  $self->{widget}->show_all;
055
056  $self->stocks_update();
057
058  PerlPanel::add_timeout(
059   5 * 60_000,
060   sub {
061    $self->stocks_update();
062    return 1;
063   }
064  );
065
066  return 1;
067 }
068
069 ############################
070 sub symbols {
071 ############################
072  my @symbols = ();
073
074  if (
075   !open(FILE, "<$CFG_FILE"))
076  {
077   ERROR
078     "Cannot open $CFG_FILE";
079   return ();
080  }
081
082  while (<FILE>) {
083   s/#.*//g;
084   s/[^\w\.]//g;
085   next if /^\s*$/;
086   chomp;
087   push @symbols, $_;
088  }
089
090  return @symbols;
091 }
092
093 ############################
094 sub stocks_update {
095 ############################
096  my ($self) = @_;
097
098  my ($tag, $buffer);
099  my $symbols = join " ",
100    symbols();
101
102  DEBUG "Updating '$symbols'";
103
104  if ($symbols eq "") {
105   $self->{label}->set_markup(
106    "No symbols defined");
107   return undef;
108  }
109
110  if ( !open(COMMAND,
111        "$GETQUOTE $symbols |"
112   )) {
113   $self->{label}->set_markup(
114    "Fetch failed ($!)");
115   ERROR "Fetch failed ($!)";
116   return undef;
117  }
118
119  $tag =
120    Gtk2::Helper->add_watch(
121   fileno(COMMAND),
122   'in', sub {
123    if (eof(COMMAND)) {
124     DEBUG
125     "Received data: $buffer";
126     close(COMMAND);
127     Gtk2::Helper
128       ->remove_watch($tag);
129     $buffer =~ s/\n/ /g;
130     $self->{label}
131       ->set_markup($buffer);
132    } else {
133     $buffer .= <COMMAND>;
134    }
135   }
136    );
137
138  return 1;
139 }
140
141 ############################
142 sub expand { return 0; }
143 sub fill   { return 0; }
144
145 sub widget {
146  return $_[0]->{widget};
147 }
148
149 sub get_default_config {
150  return undef;
151 }
152
153 1;
```

Anzeige
wird
separat
angeliefert

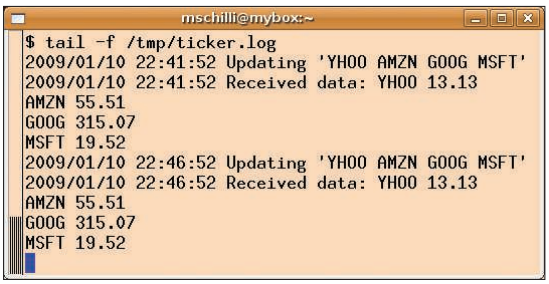

Figure 2: The Perl applet logs its current activities to the /tmp/ticker.log file.

these symbols will be sent to the *getquotes* script, and it's essential to avoid unpleasant surprises with unchecked command-line parameters.

Background

When Perl's *open()* function is called by *stocks_update()* in line 110, it creates a pipe to allow *getquote* to run in the background. Be careful here: If the code were to grab the output of the externally called process right after, it might have to wait a couple of seconds for the results to trickle in from the Internet, which is not a good idea for a GUI that must react swiftly to user input that can happen at any time. Instead, *stocks_update()* uses the Gtk2::Helper *add_watch()* function, which accepts a file descriptor (*fileno()* generates a descriptor from a Perl file handle) and jumps to a callback function when data arrives on it. This means that the script keeps running in the meantime, terminating *stocks_update()* and jumping to the applet's main event loop, where it can process user input and other external events without any delays.

If *get_quote* has finally sent some data, but still has more to come, the call to *eof(COMMAND)* is false and the Else loop in Line 132 appends the data to the existing results. When *get_quote* finishes, it's the IF branch's turn; the file handle is cleaned up by *close()*, and *remove_watch()* stops watching the corresponding descriptor. Line 129 then transforms the column format to a single *Symbol Price Symbol Price ...* line and then calls the *set_markup()* method to send it to the label widget, which displays the text in the panel.

To make sure that the developer knows what the applet is up to, *Ticker.pm* first initializes Log4perl to redirect the log statements embedded in the code to the */tmp/ticker.log* file. Figure 2 shows a couple of lines from the log. If you do not need this additional output, simply comment out the call to *easy_init()* in the code.

Figure 3: The newly written Ticker Perl applet is available for selection.

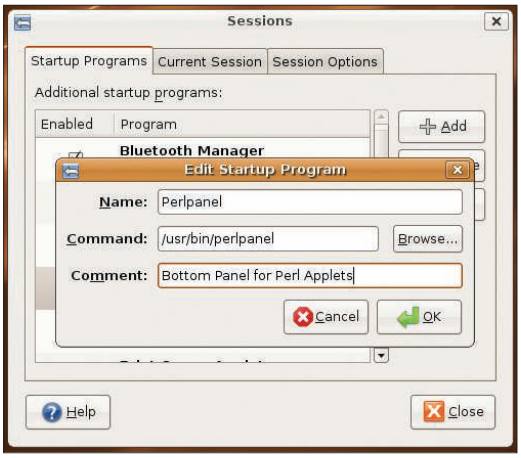

Figure 4 This configuration tells the Gnome Session Manager to launch the PerlPanel when you log in.

Installation

On Ubuntu, you can type *sudo apt-get install perlpanel* at the command line to install PerlPanel along with the Perl modules on which it depends. The files are available from the Ubuntu Package repository, so there is no need for a CPAN Shell. The *getquote* script then must be installed in */usr/bin* and requires the Finance::-YahooQuote module, which is also available as a Ubuntu package (*libfinance-yahooquote-perl*).

To tell PerlPanel about the new ticker applet, you must copy the *Ticker.pm* file to the */usr/share/perlpanel/PerlPanel* directory, where you will find some other applets that are installed as part of the PerlPanel distribution. Then you need to add the line

```
Ticker:Stock Ticker:Utilities
```

to the applet registry file, */usr/share/perlpanel/applet.registry*. This specifies that the panel will find the "Ticker" widget in the applet directory under the name of *Ticker.pm*, that it is a stock ticker, and that it should be located in the Utiltities section.

After it restarts, PerlPanel knows that the applet exists, but still does not display it. To allow this to happen, the user first has to add the applet to the panel. To do so, you need to press the *Actions* button in the panel, and select the *Configure* menu item. In the dialog box that follows, click the *+Add* button. This gives you the selection box (Figure 3), which shows a selection of ready-to-run applets. One of them is your ticker, which you can then select and install by pressing the *+Install Applet* button. The applet then appears in the container, where you can move it up or down to change its corresponding horizontal display position in the panel.

To launch the panel automatically when you log in to a session with your window manager, you must add the */usr/bin/perlpanel* program to your session start dialog; on Gnome, this looks something like the dialog box shown in Figure 4, which appears when you click *System* | *Preferences* | *Sessions* in the main menu.

For more information on building your own Perl applets, you can read the *perlpanel-applet-howto.pod* file, which is not part of the Ubuntu package, but is available with the source code from the PerlPanel CVS repository [2].

All of the panel's functions, including the desktop pager, the taskbar, and the dialog boxes for adding new applets and their configurations, were written in Perl and give you a taste of what's possible with PerlPanel. ■

INFO

[1] PerlPanel project: *http://savannah.nongnu.org/projects/perlpanel*

[2] Listings for this article: *http://www.linux-magazine.com/resources/article_code*