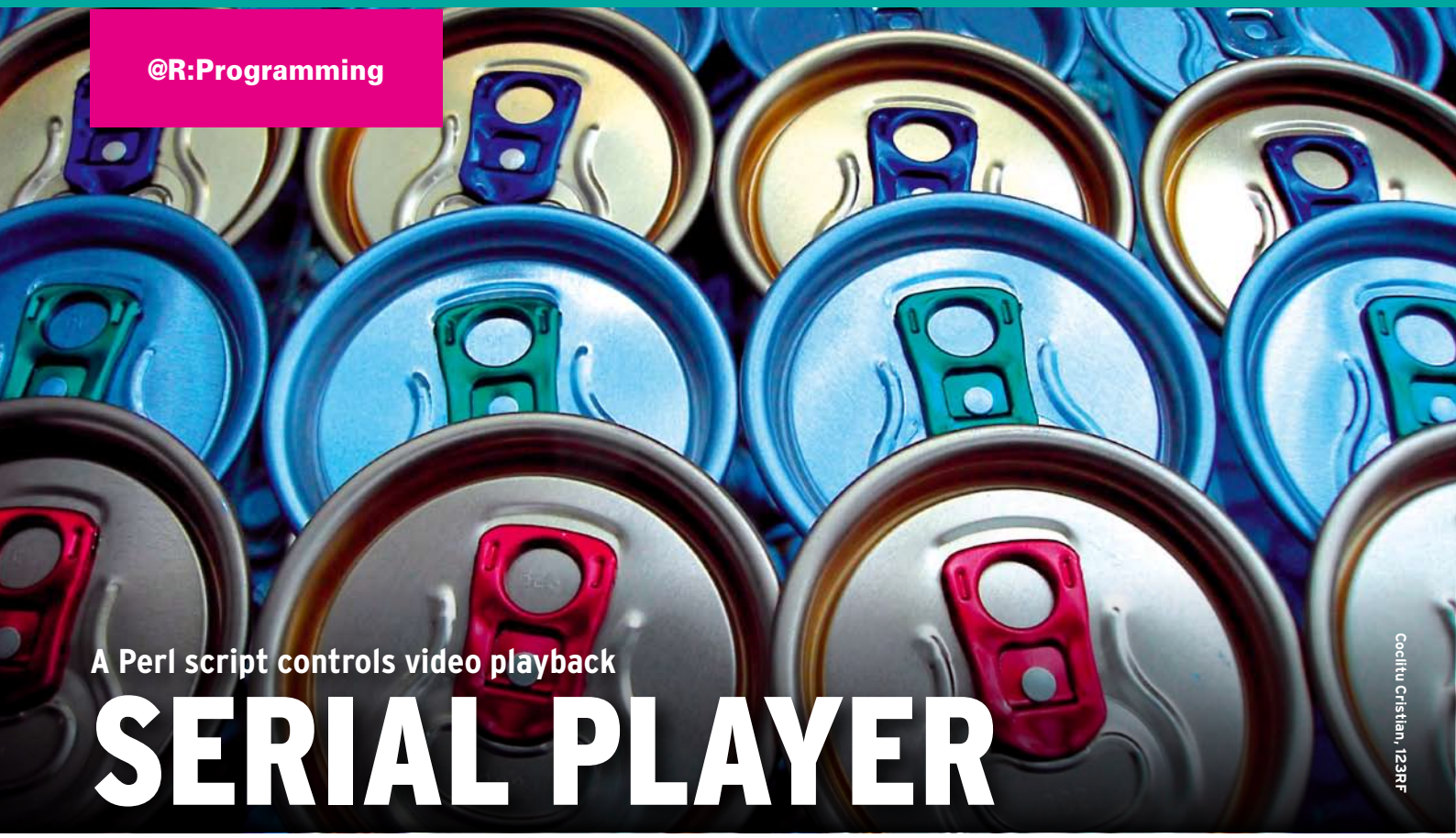
@R:Programming



Coclitu Cristian, 123RF

A Perl script controls video playback

SERIAL PLAYER

A Perl script with a Gtk2 interface remembers how far videos have been viewed and, if needed, continues playing where the last interruption occurred. **BY MICHAEL SCHILLI**

While riding the bus to work, I enjoy watching recorded TV shows on my netbook. However, quite often, I have to interrupt an interesting program because the bus arrives at its destination. If the next stage of the trip is only a quarter of an hour, it's probably not worth trying to watch the next scene. Instead, I might decide to watch a brief news show, rather than getting immersed in a complicated plot again. This leaves me with a number of half-watched shows, and I require a tool that lets me carry on watching from the point at which I left off.

Chaos Prevention

What I need is a controlling GUI for the video player that remembers the timestamp of the last interruption for a whole list of TV programs: a time machine for video files that does things exactly the way my trusty Tivo [1] does at home. The digital video recorder (DVR) records a number of TV shows and remembers the timestamp of the last interruption. If you then open a show or movie in the list some time later, the recorder starts to play exactly where you left off. iTunes or podcast programs work in a similar fashion. So, how difficult could it be to write a program in Perl that does this, too? Listing 1 (ttv) shows the results in fewer than 150 lines, and Figure 1 presents the GUI in action.

It would be downright absurd to reinvent a miracle of video technology like MPlayer. Fortunately, this versatile piece of software has everything it takes to create a movie time machine. As you can see in Figure 2, MPlayer counts the number of seconds the movie has been playing; therefore, it is a fairly easy task for a control program to find out how far along the player is in a movie. In fact, users can even press the Page up and Page down keys while viewing to skip backward and forward in the video, as MPlayer adapts the output accordingly.

A second prerequisite for a GUI like *ttv* is MPlayer's *-ss n* option, which tells the player to skip *n* seconds into the video instead of playing it from the beginning.

The workings of the *ttv* Perl script should be fairly clear. The Gtk2-based user interface waits for the user to double-click a video from the displayed list. If this is the first time the user has picked this particular show, the GUI launches MPlayer and lets it start playing from the beginning. While the player is working, and the user is enjoying the movie, the script accesses MPlayer's standard output to grab the number of elapsed seconds and caches the number.

If the user interrupts the playback (e.g., by pressing *Q* in the MPlayer window), the GUI pops back up, and the script stores the elapsed playback time under the video file's name in a YAML file called *~/.ttv.dat* below the user's home directory (Figure 3).

Foreign Territory

Regular readers of this column can probably guess that controlling a GUI component in combination with an external program like MPlayer, and without affecting the viewing experience, will once again involve the event-based POE Perl framework from CPAN. Just as with Curses-based terminal control, presented here in a previous issue [2], POE keeps pace with all kinds of GUI event loops and is the perfect choice for controlling quasi-parallel processes. In this case, the GUI script launches the video player as a separate process behind the scenes, but POE grabs its output data in a robust and elegant single-process/single-thread ap-

Figure 1: At the click of a mouse, the Gtk2 interface launches selected videos from the point at which they were interrupted – or plays them right from the start if you have not viewed them previously.

proach. When data are available, it triggers a callback handler and checks that the player is running or the user has quit the program.

Because line 3 of Listing 1 requests the Gtk2 module before the POE modules, POE knows that, rather than its own event loop, Gtk2 will be controlling the process using the CPAN POE::Loop::Glib module as an invisible bridge. Line 12 defines the path to the YAML file, in which the script will later store the hash content of the *$OFFSETS* reference. The data structure adds a floating point number to the video file name, thus specifying the number of seconds of playback elapsed per video.

The global *$REWIND* variable tells the script to rewind 10 seconds before jumping into the middle of a video that was interrupted earlier, so the viewer has a chance to get back into the plot. Line 18 searches the current directory for MP4 and AVI video files. If you use different formats, you might need to modify this.

State Machine and States

In typical POE style, the session constructor in lines 26-34 defines a total of five different states between which the state machine defined by the script switches. After launching the POE kernel in line 36, the GUI accepts and processes user input until the user finally quits the program by clicking the close button in the main window.

The *_start* state in line 28 defines the initial state. The *ui_start()* function assigned to it creates the graphical interface and is defined starting in line 72.

As explained in previous columns [2] on the subject of POE, the *KERNEL*, *SESSION*, and *HEAP* macros pick up state machine variables passed as function arguments from Perl's *@_* array. *HEAP* is a hash for a state machine session and is used to store a plethora of global variables that the state machine passes to each callback, while keeping them apart from any other sessions that might be happening at the same time.

Save or Fold?

Line 80 calls the constructor for the main GUI window, *Gtk2::Window*; its frame will later include a listbox with videos available for play. The *toplevel* parameter defines this as the application's main window. The script stores a reference to it on the heap – not to access it in callbacks later but to ensure that Perl stores a reference to the main window in a variable that will not disappear in a puff of smoke when the *ui_start()* function terminates. If this were to happen, the application window would just collapse, even though I want the application to go on running.

A call to the *signal_ui_destroy()* method in line 83 tells the POE kernel to terminate all active sessions when the main window collapses (e.g., because the user has clicked the *Close* button). The *Gtk2::SimpleList* widget created in line 87 accepts the data for the two-column display.

As shown in Figure 1, each row of the video list comprises a timestamp on the left and the name of the matching video file on the right. Both columns are text data types; that is, they only contain simple strings without color highlighting or any other fancy styles.

The script also stores a reference to the widget as *slist* on the session heap, to make sure it isn't cleaned up prematurely. The *add()* method places the listbox into the main window, and the subsequent call to *show_all()* then draws the complete GUI on screen.

Black Magic Widget

The *list_box_redraw()* function defined in lines 109-116 refreshes the listbox by passing in new values in the form of an array containing arrays with two elements each: timestamps and video names. The refresh happens just by assigning the updated data structure to the {*data*} entry of the listbox object. Black magic within the widget (the entry is bound by *tie*) immediately triggers a redraw of the graphical display. The *timer()* function in lines 119-133 converts the timestamp for a video file, which is basically a number of seconds, into the *hh:mm:ss* format.

```
Forced audio codec: mad
Opening audio decoder: [faad] AAC (MPEG2/4 Advanced Audio Coding)
AUDIO: 48000 Hz, 2 ch, s16le, ~0.0 kbit/279620.27% (ratio: 536870911->192000)
Selected audio codec: [faad] afm: faad (FAAD AAC (MPEG-2/MPEG-4 Audio) decoder)
AO: [alsa] 48000Hz 2ch s16le (2 bytes per sample)
Starting playback...
VDec: vo config request - 960 x 544 (preferred colorspace: Planar YV12)
VDec: using Planar YV12 as output csp (no 0)
Movie-Aspect is 1.76:1 - prescaling to correct movie aspect.
VO: [xv] 960x544 => 960x544 Planar YV12
A:  30.2 V:  30.2 A-V:  0.003 ct:  0.022 757/757 18%  1%  1.1% 0 0
```

Figure 2: When running, MPlayer sends the number of elapsed seconds to standard output, where the ttv script picks up the data.

If the user double-clicks a row in the listbox, a call to *signal_connect* in line 96 tells the POE state machine to enter the *click* state and call the *click()* function (lines 52-69). The only argument passed in with *ARG1* is a reference to the listbox state data from which the *get_row_data_from_path()* method extracts the row the user clicked. The second element in the array reference returned is the file name of the selected video. A call to *yield()* in line 65 tells the POE kernel to jump to the *play_video* machine state by passing in the file name for the video to play.

This launches the *play_video()* function (lines 136-180), which tries to discover whether the global *$OFFSETS* variable has a value in seconds for the video before then using the POE::Wheel::Run module to launch the external player. This wheel from the POE kernel gearbox expects the external program and its arguments in *Program* and *ProgramArgs*, respectively. The *-fs* option launches MPlayer in full-screen mode for enhanced viewing, and *-ss* supplies the number of seconds to fast forward.

In Search of Lost Time

Because MPlayer does not use newlines while counting seconds of played video material on its standard output, the normal line-based filter in POE::Wheel::Run won't capture it. Therefore, line 166 pulls in POE::Filter::Stream, which does not wait for a line to complete but tells the wheel to jump to the *output* state whenever a new snippet of text appears.

The *stdout_handler()* in lines 183-196 is called to handle the *output* state on any text printed by MPlayer. It always receives a snippet of the latest MPlayer diagnostic output in *ARG0* and uses the regular expression in line 190 to extract the number of video seconds played from it (highlighted in red in Figure 2).

For this, it looks for the *V:* string, either at the start of a line or after a blank,

Listing 1: ttv - Part 1

```
001 #!/usr/local/bin/perl -w
002 use strict;
003 use Gtk2 '-init';
004 use Gtk2::SimpleList;
005 use POE;
006 use POE::Wheel::Run;
007 use POE::Filter::Stream;
008 use YAML
009   qw(LoadFile DumpFile);
010
011 my ($home) = glob "~";
012 my $YAML_FILE =
013   "$home/.ttv.dat";
014 my $OFFSETS = {};
015 my $REWIND  = 10;
016
017 my @VIDEOS =
018   sort { -M $a <=> -M $b }
019   (<*.mp4>, <*.avi>);
020
021 if (-f $YAML_FILE) {
022  $OFFSETS =
023    LoadFile($YAML_FILE);
024 }
025
026 POE::Session->create(
027  inline_states => {
028   _start     => \&ui_start,
029   play_video => \&play_video,
030   click      => \&click,
031   output => \&stdout_handler,
032   play_ended => \&play_ended,
033  }
034 );
035
036 $poe_kernel->run();
037 exit 0;
038
039 ############################
040 sub play_ended {
041 ############################
042  my ($kernel, $heap) =
043    @_[ KERNEL, HEAP ];
044
045  DumpFile($YAML_FILE,
046   $OFFSETS);
047  listbox_redraw(
048   $heap->{slist});
049 }
050
051 ############################
052 sub click {
053 ############################
054  my ($kernel, $session,
055      $gtk_list_data)
056    = @_[ KERNEL,
057          SESSION, ARG1 ];
058
059  my ($sl, $path) =
060    @$gtk_list_data;
061  my $row_ref = $sl
062    ->get_row_data_from_path(
063   $path);
064
065  $kernel->yield(
066   "play_video",
067   $row_ref->[1]
068  );
069 }
070
071 ############################
072 sub ui_start {
073 ############################
074  my ($kernel, $session,
075   $heap) =
076    @_[ KERNEL, SESSION,
077    HEAP ];
078
079  $heap->{main_window} =
080    Gtk2::Window->new(
081   'toplevel');
082
083  $kernel->signal_ui_destroy(
084   $heap->{main_window});
085
086  $heap->{slist} =
087    Gtk2::SimpleList->new(
088   'Timer' => 'text',
089   'Video' => 'text',
090    );
091
092  listbox_redraw(
093   $heap->{slist});
094
095  $heap->{slist}
096    ->signal_connect(
097   row_activated =>
098     $session->callback(
099    "click")
100    );
101
102  $heap->{main_window}
103    ->add($heap->{slist});
104  $heap->{main_window}
105    ->show_all;
106 }
107
108 ############################
109 sub listbox_redraw {
110 ############################
111  my ($slist) = @_;
112
113  @{ $slist->{data} } =
114    (map { [ timer($_), $_ ] }
115     @VIDEOS);
116 }
117
118 ############################
119 sub timer {
120 ############################
121  my ($video) = @_;
122
123  my $sec = 0;
```

and then encloses the floating point number that follows it – *[\d.]+)* – with a capturing parenthesis. If the expression matches, it grabs the value and dumps it into the *$1* variable, which stores the match for the first capture in the regular expression. The other set of parentheses at the beginning of the regular expression does not have a capture function, as the non-capturing notation *?:* indicates.

If the regex finds a usable value, *stdout_handler()* stores it under the video name in a global hash that the *$OFFSETS* reference points to. The script stores this data after playback is interrupted by the user permanently in the YAML file below the user's home directory. This happens in the *play_ended()* function in lines 40-49, which the state machine triggers after the wheel detects that the external MPlayer program is no longer running. The *sig_child()* kernel method call in line 174 makes sure that the POE kernel takes care of dead, externally launched processes and prevents them from hanging around forever as zombies.

Figure 3: The script uses a YAML file (~/.ttv.dat) to store the elapsed viewing time for a video. Videos not stored here are played right from the beginning when the user selects them.

Patch for POE::Loop::Glib

The CPAN POE::Loop::Glib module Version 0.037 had a serious bug when I finished writing this article that caused the GUI to crash after running the video for a couple of seconds.

If Version 0.038 is available from CPAN when this article goes to press, the module author must have applied my patch, fixing the bug. If this is not the case, you can download the patch along with the other listings from the *Linux Magazine* server [3].

The following short sequence of commands quickly updates the module distribution after downloading its tarball from *search.cpan.org*:

```
$ tar zxfv POE-Loop-Glib-0.037.tgz
$ cd POE-Loop-Glib-0.037
$ patch -p1 <
   ../poe-loop-glib-0.037.patch
patching file Changes
patching file Makefile.PL
patching file lib/POE/Loop/Glib.pm
```

The normal procedure of *perl Makefile.PL; make; sudo make install* installs the patched module in your Perl directory tree.

To install the other modules, use either a CPAN shell or your Linux distribution's package manager. This assumes that your package manager has packages for the Perl modules required. Also, you need to ensure that the CPAN POE::Loop::Glib module is installed as an invisible bridge. Although it is not explicitly quoted in the program listing, you will need it.

It just remains to be said that you should avoid exploiting the script's functional scope. I wouldn't advise running more than three different movies at the same time because you might confuse plots, especially if Matt Damon and Leonardo DiCaprio are starring in very similar features. ■

Listing 1: ttv - Part 2

```
124  $sec = $OFFSETS->{$video}
125    if
126    exists $OFFSETS->{$video};
127
128  return
129    sprintf("%02d:%02d:%02d",
130   int($sec / (60 * 60)),
131   ($sec / 60) % 60,
132   $sec % 60);
133 }
134
135 ############################
136 sub play_video {
137 ############################
138  my (
139   $kernel, $session,
140   $heap,   $video
141    )
142    = @_[
143    KERNEL, SESSION,
144    HEAP,   ARG0
145    ];
146
147  my $offset = 0;
148
149  $offset =
150    $OFFSETS->{$video} -
151    $REWIND
152    if
153    exists $OFFSETS->{$video}
154    and $OFFSETS->{$video} >
155    $REWIND;
156
157  my $wheel =
158    POE::Wheel::Run->new(
159   Program =>
160     "/usr/bin/mplayer",
161    ProgramArgs => [
162     "-fs",   "-ss",
163     $offset, $video
164    ],
165    StdoutFilter =>
166      POE::Filter::Stream->new(
167      ),
168    StdoutEvent => 'output',
169    CloseEvent => 'play_ended',
170     );
171
172  $heap->{video} = $video;
173
174  $kernel->sig_child(
175   $wheel->PID(),
176   'sig_child'
177  );
178
179  $heap->{player} = $wheel;
180 }
181
182 ############################
183 sub stdout_handler {
184 ############################
185
186  my ($heap, $input) =
187    @_[ HEAP, ARG0 ];
188
189  if ($input =~
190   /(?:^| )V:\s*([\d.]+)/m)
191  {
192   $OFFSETS ->{
193       $heap->{video} } = $1;
194  }
195
196 }
```

INFO

[1] Tivo, the digital video recorder: *http://tivo.com*

[2] "Perl Flip It" by Michael Schilli, *Linux Magazine*, April 2010, pg. 68

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