Internet video organizer in Perl

COUCH POTATO

Video files are ready for download, but your storage space is limited, and, if you're like most people, you may have a hard time letting go of past favorites. You need a higher authority to decide when it's time to trim down your collection. **BY MICHAEL SCHILLI**

After filling up my disk with dozens of freely available videos in the course of a couple of weeks, I started to look around for a management software. Ideally, it would let me choose between available recordings, and, if space got tight, automatically and gracefully remove the old stuff that I hadn't watched for weeks. In other words: I needed a Tivo.

This digital video recorder (DVR) by the manufacturer of the same name and its clones are icons of US TV recording culture – ask any kid. The Tivo boxes have an easy to use interface that lets users record TV programs on disk for viewing some time later and skipping unwanted commercials. And instead of randomly turning on the TV and surfing channels, you select from a stack of programs previously recorded by the Tivo. With this "time-shifting" technology, viewers cut themselves loose from the broadcasting schedule and watch TV when they have the time.

Figure 1 shows a selection of programs that my five-year-old (but pimped) Tivo recorded over the course of several days. Because the box might record more than you have time to watch and due to disk space restrictions, the Tivo deletes older programs automatically after a couple of days, unless you say *Save until I delete*. The Tivo distinguishes between recordings that are due for deletion shortly (exclamation mark), have a couple of days to go (no highlighting), or just have one day (yellow dot), as well as recordings that it will keep forever (green dot).

Figure 1: A selection of TV programs recorded on the Tivo digital video recorder.

The *tv* script that we will be looking at today simulates a simple version of this user interface. Instead of using well-known graphical toolkits such as Perl/Tk, GTK, or Wx widgets, the script uses the Curses::UI widget collection, which is based on the Curses library. Curses::UI creates typical GUI elements, such as dialogs, menus, or list boxes, simply in an ASCII terminal. The 80s look is back – pure nostalgia!

The script expects to find the video files in a pre-configured directory, which it scans every 60 seconds. If it discovers a change, the script refreshes its interface. It also continually checks if the total size of the video files is above a certain threshold – the default is 20 GB. If so, and if the files are not tagged for keeping, it keeps on removing the oldest files from disk without prompting the user to confirm until the total size drops below the threshold.

On Keyboards: Perl

To navigate the list box that *tv* displays on the screen, users can either press the arrow keys (including [Page-Up]/[Down]), or the keyboard shortcuts familiar to Vi users: [K] (to go up), and [J] (to go down). Users can press the [D] (for delete) key to manually remove a file. When the confirm prompt appears (Figure 3), pressing *Y* or [Enter] while the cursor is in the *OK* box tells *tv* to delete the file from disk, and refreshes the list box.

To tag a file with an asterisk, that is to protect the file against automatic deletion by the hard disk janitor, users simply select a list box entry, and press the [*] key.

To play a program using Mplayer, users simply select a file and press [Enter]. The versatile Mplayer, which is available from from the Mplayer homepage at[2], will play any popular video format.

Pressing [Q] quits Mplayer. Mplayer also has keyboard shortcuts for actions such as fast-forward and rewind. To

Listing 1: tv

```
001 #!/usr/bin/perl -w
002 ################################
003 # tv - manage video files
004 ################################
005 use strict;
006 use Videodir;
007 use Curses::UI::POE;
008 use Curses;
009
010 my $MPLAYER =
011   "/usr/bin/mplayer";
012
013 my $V = Videodir->new();
014
015 my $CUI =
016   Curses::UI::POE->new(
017  -color_support => 1,
018  inline_states  => {
019   _start => sub {
020    $poe_kernel->delay(
021     'wake_up', 60);
022   },
023   wake_up =>
024     \&wake_up_handler,
025  }
026   );
027
028 my $WIN =
029   $CUI->add(
030  qw( win_id Window ));
031
032 my $TOP = $WIN->add(
033  qw( top Label
034    -y 0 -width -1
035    -paddingspaces 1
036    -fg white -bg blue
037    ), -text => top_text()
038 );
039
040 my $LBOX = $WIN->add(
041  qw( lb Listbox
042    -padtop 1 -padbottom 1
043    -border 1 ),
044  -onchange    => \&selected,
045  -onselchange => \&changed,
046 );
047
048 my $BOTTOM = $WIN->add(
049  qw( bottom Label
050    -y -1 -width -1
051    -paddingspaces 1
052    -fg white -bg blue
053    ), -text => bottom_text(),
054 );
055
056 $CUI->set_binding(
057  sub { selected($LBOX); },
058  KEY_ENTER()
059 );
060 $CUI->set_binding(
061  sub { exit 0; }, "q");
062 $CUI->set_binding(
063  \&delete_confirm, "d");
064 $CUI->set_binding(\&keep,
065  "*");
066
067 redraw(); # initial draw
068 $CUI->mainloop;
069
070 ################################
071 sub ttl_icon {
072 ################################
073  my ($ttl) = @_;
074  return $ttl < 0 ? "!"
075    : $ttl <= 5   ? " "
076    : "*";
077 }
078
079 ################################
080 sub changed {
081 ################################
082  $BOTTOM->text(
083   bottom_text());
084 }
085
086 ################################
087 sub selected {
088 ################################
089  my $cmd = "$MPLAYER "
090    . active_item()->{path}
091    . ">/dev/null 2>&1";
092  `$cmd &`;
093 }
094
095 ################################
096 sub bottom_text {
097 ################################
098  my $item = active_item();
099
100  # Work around PGdown bug
101  return unless defined $item;
102
```

quit the *tv* program, again just press the [Q] key.

Multitasking for Multimedia

Listing 1 first includes the Curses::UI:: POE and Curses modules, both of which are available from CPAN. Curses::UI includes a selection of handy Curses widgets to help with all sorts of different UI needs.

To make the script multitasking-capable, which it needs to be in order to perform periodic refreshing for example, Curses::UI::POE defines a derivative class which integrates the GUI in the POE frameworks event loop. You may recall me using POE previously in our regular Perl column, mainly to add cooperative multitasking support to allow GUIs to run smoothly, although the controlling program is doing something strenuous.

The constructor called in Line 16 uses the *color_support* option to specify that the new terminal GUI supports ANSI colors. The *inline_states* parameter defines the start status, *_start*; the POE kernel automatically enters this state, shortly after launching.

When it gets there, the *delay()* method ensures that the POE kernel enters the *wake_up* state after exactly 60 seconds, and runs the *wake_up_handler* function defined in Line 117.

This is where the Videodir (see below) module's *rescan()* method scans the video directory, and remembers the names of all the files, along with their last modification timestamps. A small database resides inside the video directory in form of a YAML file named .meta: its entries specify how long the user intends to keep the video clips. *Videodir::rescan()* reads this information and stores it in an internal data structure which the function *redraw()* then accesses to update the list box in the GUI.

Weeding

The *Videodir::shrink()* method launched in Line 123 shrinks the video directory

Listing 1: tv (continued)

```
103  my $str = sprintf
104    "%d/%d | %.1f days"
105    . " old | %s GB | TTL %s",
106    $LBOX->get_active_id() +
107    1,
108    scalar @{ $V->{items} },
109    $item->{age},
110    $item->{size},
111    $item->{ttl};
112
113  return $str;
114 }
115
116 ################################
117 sub wake_up_handler {
118 ################################
119  $V->rescan()
120    ;  # Get newly added files
121  redraw();
122
123  redraw() if $V->shrink();
124
125  # Re-enable timer
126  $poe_kernel->delay(
127   'wake_up', 60);
128 }
129
130 ################################
131 sub top_text {
132 ################################
133  return "tv1.0 | "
134    . $V->{total_size}
135    . " GB total | "
136    . "$V->{max_gigs} GB max";
137 }
138
139 ################################
140 sub delete_confirm {
141 ################################
142  my $item = active_item();
143
144  my $yes = $CUI->dialog(
145   -title =>
146     "Confirmation required",
147   -buttons =>
148     [ 'yes', 'no' ],
149   -message =>
150     "Are you sure you want "
151     . "to delete "
152     . "$item->{file}?",
153   qw( -tbg white -tfg red
154       -bg white -fg red
155       -bbg white -bfg red )
156  );
157
158  if ($yes) {
159   $V->remove($item->{file});
160   redraw();
161  }
162 }
163
164 ################################
165 sub redraw {
166 ################################
167  $LBOX->{ -values } =
168    [ map { $_->{file} }
169     @{ $V->{items} } ];
170
171  $LBOX->{-labels} = {
172   map {
173    $_->{file} =>
174      ttl_icon($_->{ttl})
175      . " $_->{file}"
176    } @{ $V->{items} }
177  };
178
179  $LBOX->draw(1);
180  $TOP->text(top_text());
181  $BOTTOM->text(
182   bottom_text());
183 }
184
185 ################################
186 sub keep {
187 ################################
188  my $it = active_item();
189  $V->{meta}->{ $it->{file} }
190    ->{keep} = 10000;
191  $V->meta_save();
192  $V->rescan();
193  redraw();
194 }
195
196 ################################
197 sub active_item {
198 ################################
199  return $V->{items}
200    ->[ $LBOX->get_active_id()
201    ];
202 }
```

by deleting older videos if their total size exceeds a certain threshold. All the *wake_up_handler* has to do then is to call *delay()* to tell the POE kernel to wake it up again in 60 seconds. The function then quits and hands control back to the POE kernel which then goes back to handling user input and refreshing the GUI.

Letter Boxes

tv starts building the ASCII GUI in Line 28. The *add()* method adds a new *Window* type widget which takes up the whole of the current terminal window. Then three widgets are added to the Window object. Working from the top downward, *add()* inserts the top info bar *$TOP*, the list box *$LBOX*, and the lower bar *$BOTTOM* into the GUI. (See the script in action in Figure 2.)

The first two *add()* parameters set an alias for the new widget and specify the widget type. The two bars are of the *Curses::UI::Label* type; the code for the list box with the video entries is defined in *Curses::UI::Listbox*. The *add()* method's *-y* option specifies the vertical position of the widget with *1* representing the topmost row, and *-1* the bottom row. *-bg* specifies the background color, and *-fg* the font color.

-width -1 spreads the info bar over the full width of the terminal. *-paddingspaces* pads out the blue bars to the end of the line, even if the label entry is shorter.

Normally, these parameters would be passed in as *Key => Value* pairs, but to avoid bloating the listings, I opted for a space-saving notation that uses *qw(...)* to separate options in the string at word boundaries and pass these options on as a list.

-border 1 draws a thin blue frame around the list box. Rather than having the frame overwrite the top and bottom bars, the list box honors the *-padtop 1* and *-padbottom 1* options to leave the necessary space.

The script processes two types of list box events: *-onselchange* and *-onchange*. The first type of list box event occurs when a user presses an arrow key to move the list box cursor up or down. In this case, the list box event calls the *changed* function defined in Line 81, which in turn outputs the metadata for the selected video file into the GUI footer.

The footer gives the user information such as which element this is, and how many files there are in total (for example 1/74), how old the file is, how much storage space the file occupies, and how long it has to live, unless the user does something about it. For example, *TTL 4.3* tells the user that the time to live is 4.3 days. The file can be deleted any

Listing 2: Videodir.pm

```
001 ################################
002 package Videodir;
003 ################################
004 use strict;
005 use warnings;
006 use YAML
007   qw(LoadFile DumpFile);
008 use File::Basename;
009
010 ################################
011 sub new {
012 ################################
013  my ($class, %options) = @_;
014
015  my $self = {
016   dir => "$ENV{HOME}/tv",
017   meta_file    => ".meta",
018   keep_default => 5,
019   meta         => {},
020   max_gigs     => 20,
021   %options
022  };
023
024  $self->{meta_path} =
025    $self->{dir} . "/" .
026    $self->{meta_file};
027
028  bless $self, $class;
029  $self->rescan();
030  return $self;
031 }
032
033 ################################
034 sub rescan {
035 ################################
036  my ($self) = @_;
037
038  if (-f $self->{meta_path}) {
039   $self->{meta} =
040     LoadFile(
041    $self->{meta_path});
042  }
043
044  $self->{total_size} = 0;
045  my @items = ();
046
047  my $dir = $self->{dir};
048  for my $path (<$dir/*>) {
049
050   next unless -f $path;
051   my $file = basename $path;
052
053   $self->{meta}->{$file}
054     ->{keep} =
055     $self->{keep_default}
056     unless
057     defined $self->{meta}
058     ->{$file}->{keep};
059
060   my $size = -s $path;
061   $self->{total_size} +=
062     $size;
063
064   my $age =
065     age_in_days($path);
066
067   push @items,
068     {
069    file => $file,
070    path => $path,
071    age  => $age,
072    size => gb($size),
073    ttl  =>
074      $self->{meta}->{$file}
075      ->{keep} - $age,
076     };
077  }
078
079 $self->{total_size} =
080   gb($self->{total_size});
081
082 # Delete outdated entries
083 for my $k (
084  keys %{ $self->{meta} })
085 {
086  delete $self->{meta}->{$k}
087    unless
```

time after this if storage space is getting tight.

Indexed List Box

The *$LBOX* list box object's *get_active_id()* method detects the entry currently selected in the list box; it returns the index of the corresponding list element. The *Videodir.pm* module has a data structure that contains video metadata in the same order that the list box will display them later.

The second event the list box processes is *-onchange*. The event is triggered when a user presses [Enter] for a selected entry, or when the user clicks an entry with the mouse. This tells *tv* that the user wants to view the video. Line 92 calls Mplayer in the background using backticks and &. This is important because we want the GUI to keep on accepting keyboard input rather than freezing.

In addition to the callback definition for the list box, Line 56 specifies that pressing [Enter] calls the *selected()* function. The *KEY_ENTER()* macro is defined in the *Curses* module, and references the [Return] or [Enter] key. Thanks to the onchange event handler we defined for the list box earlier on, this would happen without the explicit *set_binding* instruction – because an onchange event occurs whenever a list box entry is selected. But this event would fail if a user selected the same entry again after viewing the video.

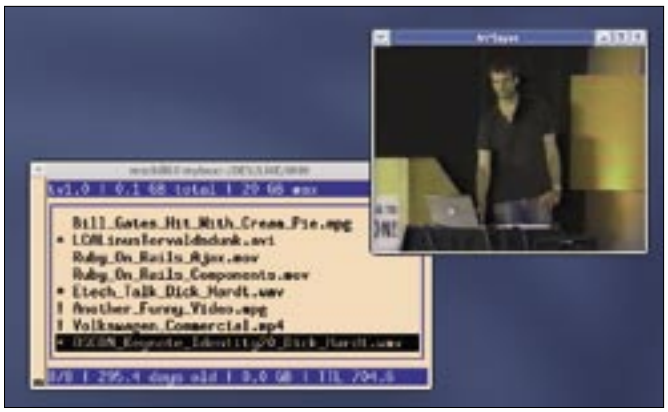

Figure 2: The Perl script tv uses Curses::UI to create a Tivo-look-alike GUI. Dick Hardt's OSCON 2005 talk on "Identity 2.0" is certainly worth keeping around forever, so I marked this item with an asterisk *.

Lines 60 through 65 map more keys. Users can press [Q] to quit watching; this tells *tv* to quit via *exit 0*. The [D] key

Listing 2: Videodir.pm (continued)

```
088     -f "$self->{dir}/$k";
089   }
090
091   $self->meta_save();
092
093   # Sort by descending by age
094   $self->{items} = [
095    sort {
096     $a->{age} <=> $b->{age}
097      } @items
098   ];
099
100   return $self->{items};
101 }
102
103 ###################################
104 sub gb {
105 ###################################
106   my ($val) = @_;
107   return sprintf "%.1f",
108     $val / (1024**3);
109 }
110
111 ###################################
112 sub remove {
113 ###################################
114   my ($self, $file) = @_;
115
116   my $path =
117     "$self->{dir}/$file";
118
119   if (-f $path) {
120    unlink $path
121      or die
122      "Cannot unlink $path";
123   }
124   $self->rescan();
125 }
126
127 ###################################
128 sub age_in_days {
129 ###################################
130   my ($file) = @_;
131
132   return (
133    sprintf "%.1f",
134    (time() - (stat $file)[9])
135      / 24 / 3600
136   );
137 }
138
139 ###################################
140 sub shrink {
141 ###################################
142   my ($self) = @_;
143
144   my $deleted = 0;
145
146   my @doomed = reverse
147     grep { $_->{ttl} < 0 }
148     @{ $self->{items} };
149
150   while ($self->{total_size} >
151    $self->{max_gigs})
152   {
153    last unless @doomed;
154    my $item = shift @doomed;
155    $deleted++;
156    $self->remove(
157     $item->{file});
158   }
159   return $deleted;
160 }
161
162 ###################################
163 sub meta_save {
164 ###################################
165   my ($self) = @_;
166   DumpFile($self->{meta_path},
167    $self->{meta});
168 }
169
170 1;
```

Figure 3: The delete dialog for a program pops up when a user presses [D], prompting the user to confirm their choice.

calls *delete_confirm()* in Line 140 to delete the selected video file, prompting the user to confirm before doing so. When a user types an asterisk (*), the *keep* function is called and sets the TTL for the file to 10000 days in the meta-database to prevent it from being deleted.

The *ttl_icon* method in Line 71 helps the GUI modify the display to reflect the various TTLs for the videos. If a TTL is less than zero, that is if the file is due for deletion, an exclamation mark is displayed. Nothing is displayed for a TTL of less than five days, and an asterisk in all other cases. After all this preparatory work, the GUI is now complete.

Line 68 triggers the the Curses::UI:: POE module's *mainloop* which in turn launches the POE kernel with its associated multitasking activities. A POE-only application should never perform synchronous hard disk access. However, the fact that *tv* occasionally reads the Inode data for the video files, is just about acceptable. The GUI might stutter from time to time, but it won't freeze.

If something changes in the video directory, the *wake_up_handler()* will find out what within 60 seconds, when the *Videodir.pm* module's *rescan()* method is called in Line 119. It then refreshes the module's internal data structure. The data is then handed on to the list box by the *redraw()* function in Line 165. The list box's *draw()* method redraws the graphical elements. As the number of files, the space they occupy on disk, or even the selected entry can change, *redraw()* also redraws the header and footer bars.

Video Fans of the World Unite

The *Videodir.pm* module in Listing 2 abstracts access to video files. The video directory, which defaults to *~/tv*, not only contains all the video files, but also a *.meta* file that stores the TTL data in YAML format (Figure 4). The *keep* key in *.meta* specifies the number of whole days a file should be kept in the directory after creation.

The modify date stored for the file is used as a timestamp. To set the time to live for a file, the *Videodir.pm* module's *age_in_days* function first calculates the difference between the current time and the Unix mtime for the file in days. The TTL is then the vector between the *keep* value (in days) set in the metafile, and the file age (Line 75). In Line 16 the *new* constructor defines a few default values for constants that can be overwritten when called. For example, if you create a Videodir object by saying *new(max_gigs => 50)*, the disk space threshold is 50 GB rather than 20.

The *rescan* method in Line 34 reads both the video directory (using the *$dir/** glob, which will not find *.meta*) and the metafile, which *rescan* parses using the YAML module's *LoadFile()* method. The method refreshes the internal data structure stored in the *items* key to reflect the current state. Each element in the *items* array is a pointer to a hash that contains the values for the keys:

- *file*: filename
- *path*: absolute path
- *age*: age in days
- *size*: filesize in GBytes
- *ttl*: time to live in days before deletion protection is revoked

The metafile automatically assigns a *keep* value of five days to any new files it finds (*keep_default* parameter). At the end of the *rescan*, *Videodir.pm* calls *meta_save()* to write the new *keep* values to the metafile. Before doing so, Lines 83 through 89 remove the entries for any files that have disappeared from the disk since the last scan, in order to update the metafile.

When the disk space threshold is reached, the *shrink()* method keeps on removing files that are due for deletion until the total size drops below the threshold value. To do this, *grep* filters any entries with a *ttl* below zero. As the entries in the array previously referenced by *$self->{items}* are sorted by date in descending order (the newest files come first), *reverse* reverses the order of the resulting list to sort the files in order of due date. If the total file size is below the threshold value, *shrink()* returns 0 without doing anything. The caller in the *tv* script checks the return value, as it will only need to refresh the list box display if some files have been removed.

Figure 4: The metadata in ~/tv/.meta are stored in YAML format and contain the TTL data for the video files.

Installation Trick

With version 0.95 of the CPAN Curses:: UI module, the event loop chokes on keyboard input, so you might like to download the patched version [3]. You need to install *Videodir.pm* in a directory where *tv* will find it. Happy viewing. ■

INFO

[1] Listings for this article: *ftp://www.linux-magazin.de/pub/listings/magazin/2006/08/Perl*

[2] Mplayer homepage: *http://mplayerhq.hu*

[3] Patched version of the Curses::UI module: *http://perlmeister.com/errata/Curses-UI-0.95-patch-ms1.tar.gz*