Gtk2-Perl: An Introduction

Or How I Learned to Stop Worrying and Make Things *Pretty*

Scott Paul Robertson
http://spr.mahonri5.net
spr@mahonri5.net

May 9, 2007

What is Gtk2-Perl?

Gtk2-Perl is the set of Perl bindings for Gtk2, and related libraries[1]. These allow you to code Gtk2 and Gnome applications in Perl. Gtk2-Perl is fairly easy to get started using, but has a few hang ups that can be difficult to discover, or sometimes the documentation is just not there. The developers are aware of the issue, and encourage users to provide documentation, reference guides, or tutorials.

[1] Glade, Glib, etc

Online Documentation

There are a few major online sources for Gtk2-Perl documentation:

- Gtk2-Perl Tutorial - The offical tutorial.
- Gtk2-Perl Study Guide - Another excellent tutorial/reference
- Introduction to GUI Programming with Gtk2-Perl - As it says, it's a good introduction to gui programming in general.
- Gtk2-Perl Pod Documentation - You might have these as man pages as well, very helpful.

A Simple Window

We'll start with an example of a simple single button application. First we have to initalize Gtk2:

```
#!/usr/bin/perl -w
use strict;

use Gtk2 '-init';
use readonly TRUE => 1;
use readonly FALSE => 0;
```

- The `'-init'` prepares the Gtk2 environment, otherwise we would have to do it ourselves later.
- Defining the constants is very helpful for code readability later.

A Simple Window

Now we set up the window object and a button.

```
my $window = Gtk2::Window->new;
$window->set_title ('Testing a Button');
$window->signal_connect (delete_event => sub { Gtk2->main_quit; TRUE});

my $button = Gtk2::Button->new ('_Press Me');
$button->signal_connect(clicked => sub {print "Pressed!\n"; });

$window->add ($button);
$window->show_all;
Gtk2->main;
```

Explination

In this code we make two objects, a Window and Button. Both are created in a similar method, with `->new`. Every object has a set of unique methods, and methods it has inherited. For example, button's ancestry is: GLib::Object → Gtk2::Object → Gtk2::Widget → Gtk2::Container → Gtk2::Bin. So we use any function found in these classes as well. `signal_connect` is from the GLib::Object class, so everything should have it. Buttons also have the signals: activate, pressed, released, enter, and leave. We can connect signals to these events as well.

Signals and Events

Gui programming is *event driven*. In other words, code is run based on events caused by the user. Once you call `Gtk2->main`, your code will only respond on events, which are connected with `signal_connect`.

- `signal_connect(signal_name => &function_to_call)`

Connecting the signals can be tedious, but it is the only way. We could have a button that implements all of the events from its class:

```
my $button2 = Gtk2::Button->new ('Try Me');
$button2->signal_connect(clicked => sub { print "clicked\n"; });
$button2->signal_connect(activate => sub { print "activated\n"; });
$button2->signal_connect(pressed => sub { print "presed\n"; });
$button2->signal_connect(released => sub { print "released\n"; });
$button2->signal_connect(enter => sub { print "entered\n"; });
$button2->signal_connect(leave => sub { print "left\n"; });
```

Arranging Widgets: VBox and Hbox

A Window is a container that can hold only one item. Of course, if you want multiple buttons or menubars and the like, you'll need to use a different container. There are three basic types of containers: vboxes, hboxes, and tables.

Vboxes and Hboxes have the same interface, and provide *packing* in either the vertical or horizontal direction, respectively. As an example of hbox and vbox:

w	w	w
i	i	i
d	d	d
g	g	g
e	e	e
t	t	t

widget
widget
widget
widget

Arranging Widgets: VBox and Hbox

There are a number of different methods for packing widgets.

- `add` - Adds the widget to the end of the box (right or bottom)
- `pack_start` - Adds the widget to the end of the box, with options.
- `pack_end` - Adds the widget to the start of the box, with options.

For example:

```
my $vbox1 = Gtk2::VBox->new;
$vbox1->add($button);
$vbox1->add($button2);
my $vbox2 = Gtk2::VBox->new;
$vbox2->pack_start_defaults($button3);
# Widget, Expand to fill space, Fill all space it can, padding
$vbox2->pack_start($button4, FALSE, TRUE, 10);
my $hbox = Gtk2::HBox->new;
$hbox->pack_end_defaults($button5);
$hbox->pack_end($button6, TRUE, TRUE, 5);
# Homogeneous, spacing
my $bighbox = Gtk2::HBox->new (FALSE, 20);
```

Arranging Widgets: Tables

Tables are a very flexible and powerful way to do complex packing easily. They layout with columns and rows, and widgets can expand beyond one column and row. Tables are very useful for Text Entry boxes, aligned check and radio buttons, and other things without builtin labels.

```
# Rows, Columns, homogeneous
my $table = Gtk2::Table->new (3, 4, TRUE);
$table->set_col_spacings(5);
# Widget, left, right, top, bottom
$table->attach_defaults($label1, 0,1, 0,1);
$table->attach_defaults($entry1, 1,3, 0,1);
$table->attach_defaults($label2, 0,1, 1,2);
$table->attach_defaults($entry2, 1,2, 1,2);
$table->attach_defaults($button, 3,4, 0,3);
$table->attach_defaults($button2, 0,1, 2,3);
```

Dialog Boxes

Not many gui programs just have one window. You'll see programs pop up errors in a window, ask for additional information, among other things. You'll at some point want this functionality for yourself. Gtk2-Perl offers two methods for doing this, Gtk2::Dialog, and its simplier child Gtk2::MessageDialog.

Dialog Boxes: MessageDialog

MessageDialog provides an easy interface for making error messages and other simple dialogs. More complex windows should use Dialog instead, as the flexibility is usually needed there. A Message Dialog is created by:

```
my $dialog = Gtk2::MessageDialog->new ($window, 'destroy-with-parent',
                                       'question', 'yes-no',
                                       'Really Do It?');
$response = $dialog->run;
print "$response\n";
$dialog->destroy;
```

MessageDialog needs to know who its parent window is, after that are three values filled by Gtk2::DialogFlags, Gtk2::MessageType, and Gtk2::ButtonsType enumerations. Now is a good time to point out that Gtk2-Perl often uses enumerations for value passing. The enumeration values are in the pod documentation, and you always specify them as strings. For example, MessageDialog offers a number of MessageType's: info, warning, question, or error.

Dialog Boxes: Dialog

Dialog boxes allow for much more flexibility, and becaue of this we will use this to look into making a simple preferences dialog. We want to be able to set variables that are various types through a couple common methods. We'll begin by going over the constructor.

```
my $dialog = Gtk2::Dialog->new ('Title', $window,
    'destroy-with-parent', 'gtk-cancel' => 'cancel',
    'gtk-save' => 'accept');
```

Dialog lets you specify a title of the window, asks for the parent window, and for Gtk2::DialogFlags. After that it accepts a list of of `'button_text' => 'response-id'` pairs. Response-ids are found in the Gtk2::ResponseType enum. This allows you to have many response options. You can combine DialogFlags by doing `[qw/modal destroy-with-parent/]` for the argument.

Dialog Boxes: Dialog

Dialog boxes come with a built in vbox that is already holding a box with the buttons you've specified in the constructor. You'll want to configure other containers and have the outer-most added to the default vbox.

```
$dialog->vbox->pack_start ($main_hbox,FALSE,FALSE,0);
```

We'll be using a two column approach to a preferences dialog, so We'll have two vboxes held by a single hbox.

```
$main_hbox->pack_start ($left_vbox,TRUE,FALSE,0);
$main_hbox->pack_start (Gtk2::VSeperator->new,FALSE,FALSE,0);
$main_hbox->pack_start ($right_vbox,TRUE,FALSE,0);
```

Entries

Entries are the standard text entry field. Best used when you want a user to input a string, whatever that string may be. Entries usually have a label widget, and we can connect the label's mnemonic with the entry. Some useful functions of Gtk2::Entry:

- `set_text ('text here')` - Sets the text in the box
- `get_text` - Returns the string in the box

A few useful Gtk2::Label functions:

- `new_with_mnemonic ('_string')`
- `set_mnemonic_widget ($widget)` - sets the mnemonic to point to the widget
- `set_alignment (double x, double y)` - Changes how the text is aligned, 1.0 being right or bottom, 0.5 being middle, 0.0 being left or top.

Entries

For our use we'll have just two entries, a directory and a name entry. These will be packed in with a table.

```
my $dir_entry = Gtk2::Entry->new;
my $dir_entry_label = Gtk2::Label->new_with_mnemonic ("_Directory");
$dir_entry_label->set_alignment (1,0.5);
$dir_entry_label->set_mnemonic_widget ($dir_entry);
$dir_entry->set_text ("/");


my $name_entry = Gtk2::Entry->new;
my $name_entry_label = Gtk2::Label->new_with_mnemonic ("_Name");
$name_entry_label->set_alignment (1,0.5);
$name_entry_label->set_mnemonic_widget ($name_entry);
$name_entry->set_text ("Name Goes Here");
```

Radio Buttons

Radio buttons can be useful in some situations, and they look cool. We'll take three and pack them into a table. Take note with the signal handling for radio buttons, it can be a bit tricky. Also you have to keep them grouped as you construct them.

```
my $radio1 = Gtk2::RadioButton->new (undef,'First');
my @r_group = $radio1->get_group;
my $radio2 = Gtk2::RadioButton->new (@r_group,'Second');
@r_group = $radio1->get_group;
my $radio3 = Gtk2::RadioButton->new (@r_group,'Third');
@r_group = $radio1->get_group;
$radio1->signal_connect (clicked => sub {
            ($radio1->get_active) and ($radios = 1); });
$radio2->signal_connect (clicked => sub {
            ($radio2->get_active) and ($radios = 2); });
$radio3->signal_connect (clicked => sub {
            ($radio3->get_active) and ($radios = 3); });
# Initial Values:
if ($radios == 1) { $radio1->set_active (TRUE); }
elsif ($radios == 2) { $radio2->set_active (TRUE); }
else { $radio3->set_active (TRUE); }
```

Check Boxes

Check boxes are very simple, with this we'll show off a cool feature, the sensitivity. If a widget is sensitive, you can interact with it (ie: send events to it). This check box will enable/disable the radio buttons.

```
my $checkbox = Gtk2::CheckButton->new ("_Radio Buttons");
$checkbox->signal_connect (clicked => sub {
            if ($checkbox->get_active) {
                $checked = 1;
                $radio1->set_sensitive (TRUE);
                $radio2->set_sensitive (TRUE);
                $radio3->set_sensitive (TRUE);
            } else {
                $checked = 0;
                $radio1->set_sensitive (FALSE);
                $radio2->set_sensitive (FALSE);
                $radio3->set_sensitive (FALSE);
            } });
if ($checked) { $checkbox->set_active (TRUE); }
else { $checkbox->set_active (FALSE); }

# Sensitivity on the Radios
if (!$checked) {
    $radio1->set_sensitive (FALSE);
    $radio2->set_sensitive (FALSE);
    $radio3->set_sensitive (FALSE);
}
```

Running It

After filling the various containers with our widgets, we call the command `$dialog->show_all` to ensure all the widgets will appear. We can then run the dialog in a modal fashion (blocking) by `$dialog->run`, or we can let it run and capture the 'response' signal from the dialog and discover the response via `$_[1];`

```
$dialog->signal_connect (response => sub {$_[0]->destroy});
$dialog->show_all;
```

File Dialogs

There are two standard file dialogs in Gtk2, Gtk2::FileSelection (this is what xmms uses), and Gtk2::FileChooser. FileChooser is the now standard gnome dialog that looks a lot nicer than the old FileSelection. The Gtk2-Perl Study Guide has a good example of it, and I've implemented in in the example preferences dialog, in the choose folder mode.
Avoid using FileSelection at all costs, it is *ugly*.

Progress Bars

If you're writting a gui that has to execute and number of jobs in the background, and you care about your users[2], you might want to let them know how things are going. The Gtk2::ProgressBar class provides this functionality. It is fairly easy to use, but ensuring a good user experience[3] can be a bit tricky.

[2] I know, you don't, but lets pretend for now
[3] I know, I know

Progress Bars

After creating the progress bar widget, we'll have to set the orientation, and then work on a way to increment the bar in a proper fashion. Also note that the function `set_text` can update the text shown across the progress bar.

```
my $progressbar = Gtk2::ProgressBar->new;
$progressbar->set_orientation ('left-to-right');
```

We then start the dialog in a non-modal fashion (as in the previous example), and begin the loop that will go through the jobs to be done.

```
my $total_ops = 10;
my $increment = 1 / $total_ops;
my $fraction = $progressbar->get_fraction;
for (my $x=0; $x < $total_ops; $x++) {
```

Progress Bars

Now for the for loop:

```
if (!$run) {
    $run_dialog->destroy;
    return;
}
if ($fraction <= 1.0) {
    $progressbar->set_fraction($fraction);
    $fraction += $increment;
}

while (Gtk2->events_pending) {
    Gtk2->main_iteration;
}
Gtk2::Gdk->flush;
```

Progress Bars

That last while loop is very important. It tells the gui to update and refresh. Without this your progress bar will not work correctly. Any time you have a part of your gui that doesn't draw properly, throw one of these in.

```
while (Gtk2->events_pending) {
    Gtk2->main_iteration;
}
Gtk2::Gdk->flush;
```

Now we finish the for loop, using a sleep command to simulate the job occurring.

```
    sleep 2;
}
```

Progress Bars

You'll notice that if you have a task that blocks, that until your code reaches the refreshing while loop, the gui is completely unresponsive. This is pretty crappy. My solution to this problem is to use the default perl threading method, ithreads. ithreads are very well documented. To apply it to this situation, you simply send a message to the other thread and put the thread with the gui into a loop with the refreshing while loop, waiting to get a message back that it's done with the task. You can think of it as having a thread for the gui, and a worker thread that will do one task at a time.
Be aware that Gtk2-Perl is not thread safe, so you need to start your threads before the use Gtk2 statement.

Pod Documentation

The provided documentation for Gtk2-Perl is a treasure trove of information, spend a while with a few of the well documented items (Gtk2::Dialog, Gtk2::SimpleList) to get a feel for the format and layout, and then with a bit of trial and error you should be able to take advantage of the classes that only have listings of methods.

Working With Lists

TreeView is famous for producing good results, but being a pain to code for. Gtk2-Perl has a special class, Gtk2::SimpleList, that allows you to create a TreeView without all the pain and sufferring. It's well documented, and once you've created the widget, you can use TreeView functions to have a fine tuned TreeView, without all the pain.

GLib Classes

If you happen to use GLib in your code, there is a hang up. The GLib classes expect special values for TRUE and FALSE. These still work as expected, but if you don't use the GLib provided definitions, things will break. So simply instead of defining the readonly's do this:

```
use Glib qw/TRUE FALSE/;
```

Online Resources

- Gtk2-Perl Tutorial - `http://gkt2-perl.sf.net/doc/gtk2-perl-tut/`
- Gtk2-Perl Study Guide - `http://forgeftp.novell.com/gtk2-perl-study/homepage/`
- Gtk2-Perl Pod Documentation - `http://gtk2-perl.sourceforge.net/doc/pod/`
- My Tutorial for Gtk2-Perl - `http://spr.mahonri5.net/wordpress/gtk2-perl-tutorial`
  This covers a few things not mentioned tonight: menubars, toolbars, statusbars, and greater detail on ithreads.

Online Resources

Perl ithreads references

- man (3perl) threads - `http://perldoc.perl.org/threads.html`
- Perl Thread Tutorial (man perlthrtut) - `http://perldoc.perl.org/perlthrtut.html`
- Things you need to know before programming Perl ithreads (perl mo - `http://www.perlmonks.org/?node=288022`

Where to find this

This document can be found online at
`http://spr.mahonri5.net/wordpress/2006/03/08/gtk2-perl-nupm-presentation/`
along with the example code.
If you'd like to see a fully working Gtk2-Perl application that takes advantage of all the things we've mentioned in this presentation, please have a look at goggify.pl, found at
`http://mahonri5.net/svn/public/oggify/goggify.pl`.
Goggify is a program designed to take a directory tree of flac audio files and encode them into a matching directory tree of files that are more space friendly.