Ruby-EFL tutorial

Introduction

A few days ago, I was looking on the Internet for a new programming language to learn. I used to program with Java but I wanted something at least as clean (not like C nor C++), which can use native libraries easily and which implements “new” programming concepts (the kind of thing I already discovered in Lisp for instance). That's why I was happy to find out RUBY. To be honest, I hesitated with Python, but after reading some user comments I decided to give RUBY a try.

Ruby is a really interesting language and after having tried some basics programs, I wanted to add a decent UI (User Interface). Bindings are provided for many toolkits such as QT, GTK, TK... But I wanted a graphical interface easy to program and more game-oriented. There is also many choices (bindings) available such as OpenGl, SDL or EFL. The two first are well-known but the latter obviously is not. However, this tutorial will help you understanding and using Ruby with EFL.

EFL stands for Enlightenment Foundation Libraries (see http://www.enlightenment.org). It is a set of components used by enlightenment desktop (e17) which can be used totally independently. For more information on the role of each component, you may read the documentation at the address mentioned above. However it is not necessary to understand this tutorial.

I chose EFL because it is designed to be efficient and easy to program. Moreover, one of my friend takes part in EFL development and can answer to every question I have about it.

First steps with Ruby-EFL

Step 1 – Installing Ruby-EFL

Depending on your operating system, this operation may be more or less difficult. For instance, with Mandriva you just have to install the following packages:

- ruby-ecore
- ruby-edje
- ruby-eet
- ruby-esmart
- ruby-evas

This can be achieve by `urpmi ruby-ecore ruby-edje ruby-eet ruby-esmart ruby-evas` as root in a shell or in rpmdrake. Other distributions may have the same kind of packages.

Otherwise, you have to download the source code by yourself. More information on Tilman Sauerbeck's website: http://code-monkey.de/pages/ruby-efl

Step 2 – First test program

Now you have to check your Ruby-EFL installation. Copy-paste the following program in your favorite text-editor:

```
(1) #!/usr/bin/ruby
(2) require "ecore"
(3) require "ecore_x"
(4) require "evas"
(5) require "ecore_evas"
(6)
(7) w,h = 800,600
(8) ee = Ecore::Evas::SoftwareX11.new
(9) ee.title = "Hello World Ruby-EFL"
(10)
(11) bg = Evas::Rectangle.new(ee.evas)
(12) bg.move 0, 0
(13) bg.resize w, h
(14) bg.set_color 255, 0, 0, 255
(15) bg.show
(16)
(17) ee.resize w, h
(18) ee.show
(19)
(20) Ecore::main_loop_begin
```

Save it in “test1.rb” then execute “ruby test1.rb”. If your installation is correct, you should have a new window filled in red on your screen.

Step 3 – Understanding basic concepts

In the program above, the header is easily understandable. First we declare that our file has to be executed with “ruby” and then we include some header in order to use EFL. Next, we ask Ecore to initialize a new window managed by Evas. We choose to use the “SoftwareX11” renderer but you could try to replace it by “GlX11” (OpenGl renderer), “XRenderX11” or “Fb”. SoftwareX11 should work on every computer but with worse performances than the others.

Line 9 is obvious: we set the window title (and in the same time, what is displayed in the task bar). Starting with line 11, we create a rectangle filled in red (<255,0,0,255> in RGB-alpha) on the previously created Evas context associated with the window (we get it with ee.evas). We put the top left corner of this rectangle on the upper left position in the window and we resize it to (w,h) in order to totally fit the window. Finally we display it with the “show” command. By the way, nothing is displayed until we display the whole window. That's what is done with the two lines 17 and 18: firstly resizing the window to (w,h) which were previously set to (800,600), then displaying the window.

The last line is used to put the application in an idle state, waiting for events to occur.

As you can see, compared to some others rendering toolkit, there is no need with EFL to redraw everything each time the window is covered by another window for instance. In fact, Evas takes care of everything for us.

Congratulations, you have made your first Ruby-EFL application.

Expending Ruby-EFL

Purposes

Maybe it could seem odd to you that the second chapter of this tutorial is about expending Ruby-EFL, however I wanted to show you how to improve our first example in order to make it in a better ruby-ish way.

Have a look at lines 11 to 15 of the above program. On the first line, we have to pass “ee.evas” to the constructor of Rectangle class, then we repeat on each line the three characters “bg.”. This is obviously too much, especially accounting for the fact that perhaps we are going to create many rectangles.

I will show you how to solve this “problem”. This is inspired from another Ruby binding for Evas that can be found here: http://neugierig.org/software/ruby/evas/evas.html

Improving Evas class

Thanks to Ruby, we don't have to create a new class which inherits from the Evas class, we just have to modify it in place to add some new methods.

```
(1)  module Evas
(2)    class Evas
(3)      def rectangle(options=nil)
(4)        r = Rectangle.new(self)
(5)        unless options.nil?
(6)          r.resize(*options['size']) unless
     options['size'].nil?
(7)          r.move(*options['position']) unless
     options['position'].nil?
(8)          r.set_color(*options['color']) unless
     options['color'].nil?
(9)          r.show unless options['show'].nil? if
     options['show']
(10)          r.name=options['name'] unless options['name'].nil?
(11)       end
(12)       r
(13)     end
(14)   end
(15) end
```

In this example, I just add a new factory method to the Evas class (in module Evas). Now, to create a new rectangle, we just have to do:

```
(1)  bg = ee.evas.rectangle(
(2)      'position' => [0,0],
(3)      'size' => [w,h],
(4)      'color' => [0, 255, 0, 255],
(5)      'show' => true)
```

It is a little bit nicer and we can do the same kind of thing for each kind of shape.

Managing events

Purposes

In this chapter, I will show you how to capture events. If you have tried to resize the window of the previous programs, you may be aware that it messes our nice rectangle. We have to resize our background whenever the main window is resized. I will show you how to capture the “resize” event. Moreover, I will show you how to capture mouse events.

Window resizing event

Catching events with EFL is done with callbacks methods. However, Ruby allows us to pass block of code to a function and it is what is used. So, window resize event will be catched this way:

```
(1) ee.on_resize {
(2)   top, left, width, height = ee.geometry
(3)   bg.resize width, height
(4) }
```

As you can guess ee.geometry returns the position of the left upper corner of the Evas canvas and its dimensions. Really easy isn't it?

Capturing mouse events

Now what if we want to add a new rectangle each time an user click? The method is quite similar excepted that this time we capture the on_mouse_up:

```
(1) bg.on_mouse_up {|event|
(2)   x, y = [event.position.canvas_x,event.position.canvas_y]
(3)   r = ee.evas.rectangle(
(4)     'position' => [x, y],
(5)     'size' => [50,60],
(6)     'color' => [100, 0, 100, 255],
(7)     'show' => true)
(8) }
```

That's all. A new rectangle is drawn wherever and whenever a user click (in fact, when he/she releases the mouse button). If you want to test which button has been pressed, you need to read event.buttons. For instance, you can do that kind of thing:

```
(1) bg.on_mouse_up {|event|
(2)   x, y = event.position.canvas_x,event.position.canvas_y
(3)   ee.evas.rectangle(
(4)     'position' => [x, y],
(5)     'size' => [50,60],
(6)     'color' => [100, 0, 100, 255],
(7)     'show' => true) if event.button==1
(8)   ee.evas.rectangle(
(9)     'position' => [x, y],
(10)    'size' => [70,30],
(11)    'color' => [255, 50, 50, 255],
(12)    'show' => true) if event.button==2
(13)  ee.evas.rectangle(
(14)    'position' => [x, y],
(15)    'size' => [50,50],
(16)    'color' => [200, 200, 50, 255],
(17)    'show' => true) if event.button==3
(18) }
```

This will draw a different rectangle (different dimensions and different colors) whether you use the left button (1), the middle one (2) or the right one (3).

There is still one problem: if you click on previously drawn rectangle, no new rectangle is added. This is because “bg” doesn't receive any event, only the clicked rectangle receives it. However, if you set the property “pass_events” of each rectangle to “true”, they won't react to any event and will pass them to the shape below them (and so on until they reach our background rectangle).

To add the “pass_events” property to the rectangle constructor, add this line at the good place:

```
(1)  r.pass_events=options['pass_events'] unless
     options['pass_events'].nil?
```

Then add “'pass_events' => true” into the constructor calls.

If you want that rectangles react to some events (setting an event handling accordingly) and that rectangles pass these events, you can set the “repeat_events” property to “true”.