Ruby (http://www.sitepoint.com/ruby/)

An Introduction to FXRuby

(http://www.sitepoint.com/author/jhibbard/)

James Hibbard (http://www.sitepoint.com/author/jhibbard/)

Published September 13, 2012

FXRuby is a powerful library for developing cross-platform graphical user interfaces (GUIs). It is based on the FOX toolkit (an open source, highly optimized library written in C++) and offers Ruby developers the possibility of coding applications in the language they love, whilst at the same time taking advantage of FOX's underlying performance and functionality.

In this article, I'm going to show you how to get up and running with FXRuby, introduce you to some of the more commonly used widgets, and demonstrate how to build a simple application with some real world value.

The code from this tutorial is available from our GitHub repo (https://github.com/sitepoint-editors/password-generator).

Installation

Presuming you have Ruby 1.9 installed on your machine.

Windows 7:

- `gem install fxruby`

Ubuntu 12.04:

- `sudo apt-get install ruby1.9.1-dev g++ libxrandr-dev libfox-1.6-dev`
- `sudo gem install fxruby`

Mac OS X:

- `sudo gem install fxruby`

If you run into any trouble, more detailed instructions can be found here: https://github.com/lylejohnson/fxruby/wiki (https://github.com/lylejohnson/fxruby/wiki)

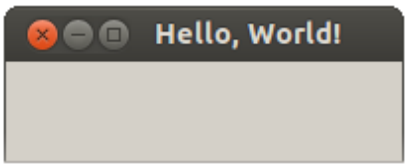
Hello, World!

So, let's start off with the customary "Hello, World!" application.

To do this create a new file on your computer, name it `hello_world.rb` and enter the following code:

```
1 require 'fox16'
2 include Fox
3 app = FXApp.new
4 main = FXMainWindow.new(app, "Hello, World!" , :width => 200, :height => 50)
5 app.create
6 main.show(PLACEMENT_SCREEN)
7 app.run
```

Run this file with the command `ruby hello_world.rb` and you should see something like this:



Let's examine what the above code is doing:

- We start by requiring the fox16 library.
- All of FXRuby's classes are defined within the Fox module, so including Fox in our program's global namespace removes the need to precede these classes with a `Fox::` prefix.
- We then create an instance of the `FXApp` class (where App stands for Application Object). The `FXApp` instance is central to an FXRuby program and has many important tasks, such as managing the event queue and handling signals.
- Next we create an instance of `FXMainWindow`, passing it the previously constructed `FXApp` object as the first argument. This associates the window we're creating, with our application. We also pass it three further arguments: window title, window width and window height.
- A call to `FXApp#create` ensures that the application's window gets created.
- Windows are however invisible by default in FXRuby, so we need to call `FXMainWindow#show` for it to be displayed. The argument `PLACEMENT_SCREEN` ensures that it is centred on the screen.
- Finally, we start the program's main loop by calling the `FXApp#run`. This method will not return until the program exits.

Time for a Little Refactoring

Although the above code works just fine, it isn't very Ruby-like and as you start to add widgets to your application, things quickly become cluttered. Therefore, a common idiom in FXRuby is to create your application's window as a subclass of `FXMainWindow`, like so:

```ruby
require 'fox16'
include Fox

class HelloWorld < FXMainWindow
  def initialize(app)
    super(app, "Hello, World!" , :width => 200, :height => 50)
  end
  def create
    super
    show(PLACEMENT_SCREEN)
  end
end

app = FXApp.new
HelloWorld.new(app)
app.create
app.run
```

You notice, that we have defined a `create` method within our `HelloWorld` class. We need this as when we call `app.create` , our `FXApp` instance will in turn call the `create` method of all of the windows with which it is associated. We can also use this method to have our new `HelloWorld` object call `show` on itself after it has been created.

Another popular FXRuby idiom is to move the `FXApp` and `HelloWorld` construction into a start-up block, like so:

```ruby
if __FILE__ == $0
  FXApp.new do |app|
    HelloWorld.new(app)
    app.create
    app.run
  end
end
```

Here, `__FILE__` is the name of the current file and `$0` is the name of file where execution started. By comparing the two, we can ensure that our file is the main file being run, rather than it having been required or loaded by another file. This is definitely overkill for such a small app, but serves to demonstrate the typical style of an FXRuby program.

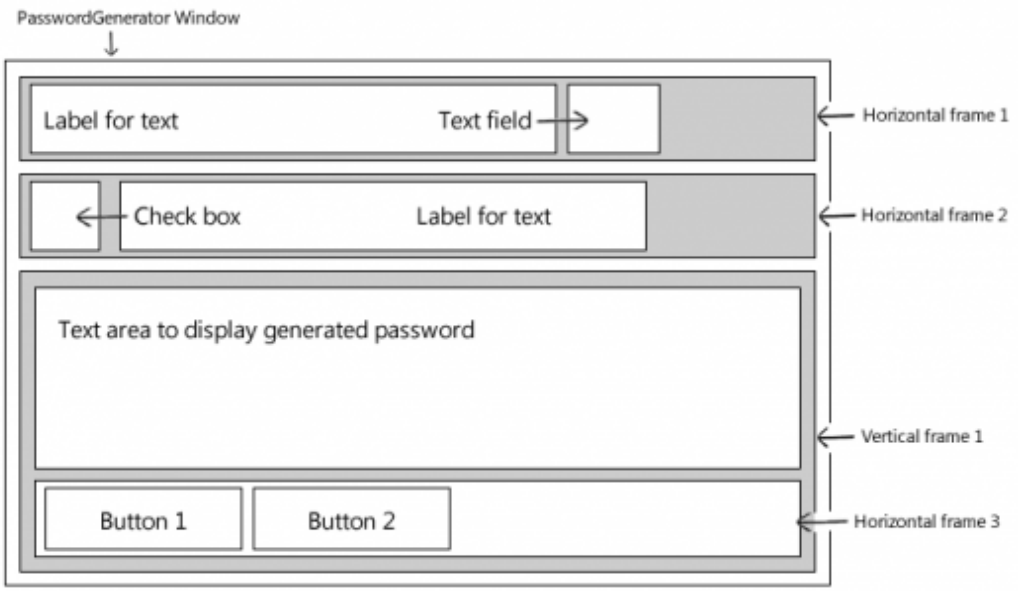
Something a bit more exciting

"Hello, World!" apps are great, but let's move on to something with a little more practical value. In this next section I'm going to show you how to make a simple password generator, which, at the push of a button, will output a random password of an arbitrary length.

The layout

Before we start coding, let's take a moment to consider which elements should be present in our GUI. We'll need a text field into which the user can type the length of the desired password. We'll also need a check box, so that the user can opt to include special characters in the password. The password itself should be displayed on some sort of text area and finally we're going to need two buttons: one to generate the password and one to copy it to the clipboard.

Here is a simple mock-up of what our GUI should look like:



In FXRuby we use layout managers to control the position and size of the widgets. In this case I am going to use objects of the class `FXHorizontalFrame` (which arranges its children horizontally) and `FXVerticalFrame` (which arranges its children vertically). The first argument for each of these layout managers is its parent window. I also pass a layout hint to the vertical frame (`LAYOUT_FILL`), which tells it to take up as much space as is available to it, both horizontally and vertically.

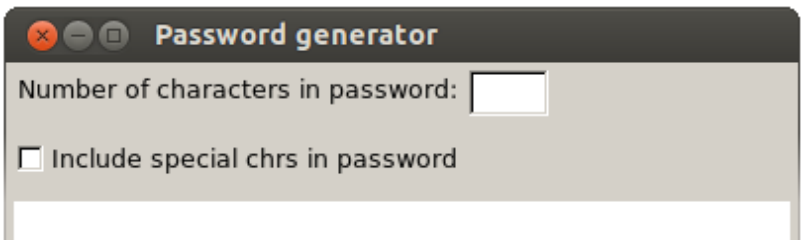
To create the widgets themselves I will use instances of the following FXRuby classes:

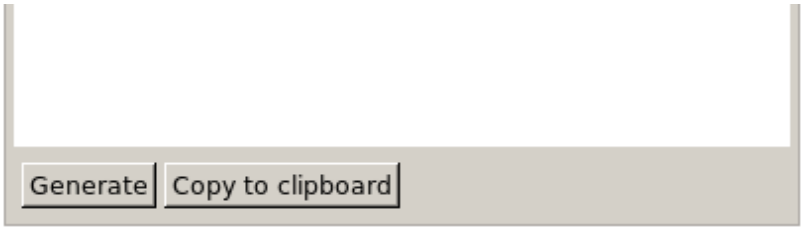
- `FXLabel` – to create a label on which we can display some text
- `FXTextField` – to create a text field into which the user can type a single line of input
- `FXCheckButton` – to allow the user to select or deselect an option

- `FXText` – to create a text area to display the output
- `FXButton` – to create a pushable button to execute a command

```
1  require 'fox16'
2  include Fox
3
4  class PasswordGenerator < FXMainWindow
5    def initialize(app)
6      super(app, "Password generator", :width => 400, :height => 200)
7      hFrame1 = FXHorizontalFrame.new(self)
8      chrLabel = FXLabel.new(hFrame1, "Number of characters in password:")
9      chrTextField = FXTextField.new(hFrame1, 4)
10     hFrame2 = FXHorizontalFrame.new(self)
11     specialChrsCheck = FXCheckButton.new(hFrame2, "Include special characters in password")
12     vFrame1 = FXVerticalFrame.new(self, :opts => LAYOUT_FILL)
13     textArea = FXText.new(vFrame1, :opts => LAYOUT_FILL | TEXT_READONLY | TEXT_WORDWRAP)
14     hFrame3 = FXHorizontalFrame.new(vFrame1)
15     generateButton = FXButton.new(hFrame3, "Generate")
16     copyButton = FXButton.new(hFrame3, "Copy to clipboard")
17   end
18   def create
19     super
20     show(PLACEMENT_SCREEN)
21   end
22 end
23
24 if __FILE__ == $0
25   FXApp.new do |app|
26     PasswordGenerator.new(app)
27     app.create
28     app.run
29   end
30 end
```

If you run this code on your computer you should see the skeleton of our application, looking something like this:





Generate a Random String Based on User Input

To create our password we can use Ruby's `Integer#chr` method, which returns a string containing the character represented by the receiver's value. If the user wants to include special characters in their password, then we can use any of the 93 characters from 33-126 on the ASCII chart. Otherwise we stick to values 48-57, 65-90 and 97-122 which represent numbers 1-9, uppercase A-Z and lowercase a-z respectively.

```
1  def generatePassword(pwLength, charArray)
2    len = charArray.length
3    (1..pwLength).map do
4      charArray[rand(len)]
5    end.join
6  end
7
8  numbers = (1..9).to_a
9  alphabetLowerCase = ("a".."z").to_a
10 alphabetUpperCase = ("A".."Z").to_a
11 allPossibleChars = (33..126).map{|a| a.chr}
```

```
1 p generatePassword(25, numbers + alphabetLowerCase + alphabetUpperCase)
2 => "fO0470a7tfdAM80u8jZ2aA0SG"
3 p generatePassword(25, allPossibleChars)
4 => "o>0]bl{6._l;s%MFCYz1Gl;hV"
```

Of course, to remember such a long and random password you will need to be using a password manager such as KeePass (http://keepass.info/), but everyone does that anyway, right? :-)

Connecting the Two

So, we've got the skeleton of our GUI up and running and our `generatePassword` method is doing what it should. It's time to connect the two.

In FXRuby we use the `connect` method to associate a user actions, such as mouse clicks, with blocks of code. In the case of `FXButton` it sends a `SEL_COMMAND` message to its target when it is clicked. The syntax is as follows:

```
FXButton.connect(SEL_COMMAND)do
  # This code fires when the button is clicked
  p "Yay! I was clicked!"
end
```

Let's apply this to our code.

You will notice that in the `generateButton.connect` block I have done the following:

- I have added a line to clear the `FXText` widget in which we want to display our output. If we didn't do this, then every time we generated a new password, it would be appended to the old one.
- I then append the result of calling `generatePassword` to the now blank `FXText` widget.
- I call `generatePassword` with the argument `chrTextField.text.to_i` . This is the integer value of whatever the user has entered into the text field.

```ruby
require 'fox16'
include Fox

NUMBERS = (1..9).to_a
ALPHABET_LOWER = ("a".."z").to_a
ALPHABET_UPPER = ("A".."Z").to_a
ALL_POSSIBLE_CHARS = (33..126).map{|a| a.chr}

class PasswordGenerator < FXMainWindow
  def initialize(app)
    super(app, "Password generator", :width => 400, :height => 200)

    hFrame1 = FXHorizontalFrame.new(self)
    chrLabel = FXLabel.new(hFrame1, "Number of characters in password:")
    chrTextField = FXTextField.new(hFrame1, 4)

    hFrame2 = FXHorizontalFrame.new(self)
    specialChrsCheck = FXCheckButton.new(hFrame2, "Include special characters in password")

    vFrame1 = FXVerticalFrame.new(self, :opts => LAYOUT_FILL)
    textArea = FXText.new(vFrame1, :opts => LAYOUT_FILL | TEXT_READONLY | TEXT_WORDWRAP)

    hFrame3 = FXHorizontalFrame.new(vFrame1)
    generateButton = FXButton.new(hFrame3, "Generate")
    copyButton = FXButton.new(hFrame3, "Copy to clipboard")

    generateButton.connect(SEL_COMMAND) do
      textArea.removeText(0, textArea.length)
      textArea.appendText(generatePassword(chrTextField.text.to_i, ALL_POSSIBLE_CHARS))
    end
  end

  def generatePassword(pwLength, charArray)
    len = charArray.length
    (1..pwLength).map do
      charArray[rand(len)]
    end.join
  end

  def create
    super
    show(PLACEMENT_SCREEN)
```

```
43    end
44  end
45
46  if __FILE__ == $0
47    FXApp.new do |app|
48      PasswordGenerator.new(app)
49      app.create
50      app.run
51    end
52  end
```

Special Characters

Now let's give the user the ability to select if they want special characters to be included in the password, or not. The most straightforward way to do this is to declare an instance variable `@includeSpecialCharacters`, which we can initialize to `false`. Then we can connect our check box to a block of code that will update the value of this instance variable (by xoring its value with `true`) whenever the box is selected or deselected.

```
1  @includeSpecialCharacters = false
2  specialChrsCheck = FXCheckButton.new(hFrame2, "Include special characters in password")
3  specialChrsCheck.connect(SEL_COMMAND) { @includeSpecialCharacters ^= true }
```

I have also included a second method called `chooseCharset`, which recieves `@includeSpecialCharacters` as an argument and returns an array containing the set of characters from which the password is to be constructed.

```
1  def chooseCharset(includeSpecialCharacters)
2    if includeSpecialCharacters
3      @charSets.first
4    else
5      @charSets.last
6    end
7  end
```

The character sets themselves are passed to the `PasswordGenerator` object upon initialization and are available in the instance variable `@charSets`.

```
charSets = [ALL_POSSIBLE_CHARS, NUMBERS + ALPHABET_LOWER + ALPHABET_UPPER]
PasswordGenerator.new(app, charSets)
```

Our `generatePassword` method will, in turn, take the array returned by `chooseCharset` as an argument and generate the password accordingly.

The Finishing Touches

It would be nice if the user could copy the generated password to the clipboard at the push of a button. Luckily the `FXText` widget provides clipboard support out of the box (i.e. you can copy its text to the clipboard using Ctrl + C) and it doesn't take much additional code for us to interact with the clipboard programmatically.

The first thing to do is to call `FXWindow#acquireClipboard` when the 'Copy to clipboard' button is pressed:

```
copyButton.connect(SEL_COMMAND) do
  acquireClipboard([FXWindow.stringType])
end
```

We pass the method an array containing `FXWindow.stringType` (one of FOX's pre-registered drag types), to indicate that we have some string data to place on the clipboard. If successful the `acquireClipboard` method will return `true` .

Now, whenever another window requests the clipboard's contents FOX will send a `SEL_CLIPBOARD_REQUEST` message to the current clipboard owner. As we called `acquireClipboard` on the main window, the main window is now the owner of the clipboard and needs to respond this message type:

```
self.connect(SEL_CLIPBOARD_REQUEST) do
  setDNDData(FROM_CLIPBOARD, FXWindow.stringType, Fox.fxencodeStringData(textArea.text))
end
```

The `setDNDData` method takes three arguments. The first tells FOX which kind of data transfer we're trying to accomplish, the second is the data type and the last is the data itself.

Aesthetics

With this done, I'm going to make two small changes to the layout of the GUI. Firstly, I'm going to place the `FXTextField` and the `FXCheckButton` in a group box and secondly I'm going to give the two buttons a uniform width.

The groupbox will be an object of the class `FXGroupBox`. It is a layout manager and, as is the case with the other layout managers, its first argument specifies its parent window. It also accepts various layout hints as additional arguments. Here I have used `FRAME_RIDGE` and `LAYOUT_FILL_X` which give it a ridged frame and tell it to occupy as much space as is available to it horizontally. To add some outer padding to the groupbox, I have introduced an object of the class `FXPacker` which will encapsulate all of the other layout managers.

```
1  packer = FXPacker.new(self, :opts => LAYOUT_FILL)
2  groupBox = FXGroupBox.new(packer, nil, :opts => FRAME_RIDGE | LAYOUT_FILL_X)
```

Giving our two buttons a uniform width is slightly easier. We just include the layout hint `PACK_UNIFORM_WIDTH` when creating the `FXHorizontalFrame` which is their direct parent.

```
1  hFrame3 = FXHorizontalFrame.new(vFrame1, :opts => PACK_UNIFORM_WIDTH)
```

A Final Bug Fix

In Ruby 1.87 entering a negative number into `chrTextField` caused the interpreter to enter an endless loop. To avoid this problem we can pass `[0, chrTextField.text.to_i].max` as a first argument to `generatePassword` which will then take the value of 0 or whatever the user entered, depending on which is higher.

Here's the final code

```ruby
require 'fox16'
include Fox

NUMBERS = (1..9).to_a
ALPHABET_LOWER = ("a".."z").to_a
ALPHABET_UPPER = ("A".."Z").to_a
ALL_POSSIBLE_CHARS = (33..126).map{|a| a.chr}

class PasswordGenerator < FXMainWindow
  def initialize(app, charSets)
    super(app, "Password generator", :width => 400, :height => 200)
    @charSets = charSets

    packer = FXPacker.new(self, :opts => LAYOUT_FILL)
    groupBox = FXGroupBox.new(packer, nil, :opts => FRAME_RIDGE | LAYOUT_FILL_X)

    hFrame1 = FXHorizontalFrame.new(groupBox)
    chrLabel = FXLabel.new(hFrame1, "Number of characters in password:")
    chrTextField = FXTextField.new(hFrame1, 4)

    hFrame2 = FXHorizontalFrame.new(groupBox)

    @includeSpecialCharacters = false
    specialChrsCheck = FXCheckButton.new(hFrame2, "Include special characters in password")
    specialChrsCheck.connect(SEL_COMMAND){ @includeSpecialCharacters ^= true }

    vFrame1 = FXVerticalFrame.new(packer, :opts => LAYOUT_FILL)
    textArea = FXText.new(vFrame1, :opts => LAYOUT_FILL | TEXT_READONLY | TEXT_WORDWRAP)

    hFrame3 = FXHorizontalFrame.new(vFrame1, :opts => PACK_UNIFORM_WIDTH)
    generateButton = FXButton.new(hFrame3, "Generate")
    copyButton = FXButton.new(hFrame3, "Copy to clipboard")

    generateButton.connect(SEL_COMMAND) do
      textArea.removeText(0, textArea.length)
      pwLength = [0, chrTextField.text.to_i].max
      charSet = chooseCharset(@includeSpecialCharacters)
      textArea.appendText(generatePassword(pwLength, charSet))
    end

    copyButton.connect(SEL_COMMAND) do
      acquireClipboard([FXWindow.stringType])
```

```
43      end
44
45      self.connect(SEL_CLIPBOARD_REQUEST) do
46        setDNDData(FROM_CLIPBOARD, FXWindow.stringType, Fox.fxencodeStringData(textArea.text))
47      end
48    end
49
50    def generatePassword(pwLength, charArray)
51      len = charArray.length
52      (1..pwLength).map do
53        charArray[rand(len)]
54      end.join
55    end
56
57    def chooseCharset(includeSpecialCharacters)
58      if includeSpecialCharacters
59        @charSets.first
60      else
61        @charSets.last
62      end
63    end
64
65    def create
66      super
67      show(PLACEMENT_SCREEN)
68    end
69  end
70
71  if __FILE__ == $0
72    FXApp.new do |app|
73      charSets = [ALL_POSSIBLE_CHARS, NUMBERS + ALPHABET_LOWER + ALPHABET_UPPER]
74      PasswordGenerator.new(app, charSets)
75      app.create
76      app.run
77    end
78  end
```

You can also obtain this code from from our GitHub repo (https://github.com/sitepoint-editors/password-generator).

Conclusion

I hope that in this article I've been able to give you a comprehensive overview of how FXRuby works and demonstrate the ease with which you can create a cross-platform graphical user interface in the language you love. I'd like to finish by presenting several resources which have helped me enormously when using this library:

- FXRuby – Create Lean and Mean GUIs with Ruby, by Lyle Johnson (creator of FXRuby) – http://pragprog.com/book/fxruby/fxruby (http://pragprog.com/book/fxruby/fxruby)
- FXRuby on github – https://github.com/larskanis/fxruby (https://github.com/larskanis/fxruby) (Lars Kanis is the current maintainer)
- FXRuby's documentation – http://rubydoc.info/github/larskanis/fxruby/1.6/frames (http://rubydoc.info/github/larskanis/fxruby/1.6/frames)
- The FXRuby mailing list (low traffic, but a good place to ask questions) – http://rubyforge.org/mail/?group_id=300 (http://rubyforge.org/mail/?group_id=300)

(http://www.sitepoint.com/author/jhibbard/)

James Hibbard (http://www.sitepoint.com/author/jhibbard/)

I'm a web developer currently living in the sunny north of Germany. I enjoy coding in both JavaScript and Ruby and can often be found in SitePoint's JavaScript forum. When I'm not coding, I enjoy running.

(https://twitter.com/jchibbard)

You might also like:

Ru: Ruby in Your Shell (http://www.sitepoint.com/ru-ruby-shell/)

Book: Jump Start Rails (https://learnable.com/books/jump-start-

rails/?utm_source=sitepoint&utm_medium=related-items&utm_content=getting-started-ruby)

RethinkDB in Ruby: Map Reduce and Joins (http://www.sitepoint.com/rethinkdb-ruby-map-reduce-joins/)

No Reader comments