Chapter 3. Hello, World!

Table of Contents

There are a few things common to all programs that use FXRuby, and the purpose of this chapter is to help you get familiar with those. We'll do this by developing a short program that simply displays a button containing the text, "Hello, World!". For reference, this program is included in the *examples* subdirectory of the standard FXRuby source code distribution.

First Things First

All of the code associated with the FXRuby extension is provided by the `fox16` gem, so we need to start by requiring this gem:

```
require 'fox16'
```

Since all of the FXRuby classes are defined under the `Fox` module, you'd normally need to refer to them using their "fully qualified" names (i.e. names that begin with the `Fox::` prefix). Because this can get a little tedious, and because there's not really much chance of name conflicts between FXRuby and other Ruby extensions, I usually like to add an `include Fox` statement so that all of the names in the `Fox` module are "included" into the global namespace:

```
require 'fox16'

include Fox
```

Every FXRuby program begins by creating an `FXApp` instance:

```
require 'fox16'

include Fox

theApp = FXApp.new
```

The `FXApp` instance has a lot of responsibilities in an FXRuby application. One of the most

frequent ways you'll use it is to kick off the application's main event loop (which you'll see later in this tutorial). The application is also the object responsible for managing "global" resources like timers and the FOX registry. It is a different approach from some other GUI toolkits for Ruby, where these kinds of global resources are accessed using module-level methods. As you continue to develop programs using FXRuby, you'll learn about other ways that the `FXApp` object is used.

The next step is to create an `FXMainWindow` instance to serve as the application's main window. If you've used Ruby/Tk before, this is similar to its `TkRoot` window:

```
require 'fox16'

include Fox

theApp = FXApp.new

theMainWindow = FXMainWindow.new(theApp, "Hello")
```

Here, we pass *`theApp`* as the first argument to `FXMainWindow.new` to associate the new `FXMainWindow` instance with this `FXApp`. The second argument to `FXMainWindow.new` is a string that will be used for the main window's title.

So far, all we've done is instantiate the client-side objects. Unlike most other toolkits, FOX makes a distinction between client-side data (such as an `FXMainWindow` object) and server-side data (such as the X window associated with that Ruby object). To create the server-side objects associated with the already-constructed client-side objects, we call `FXApp#create`:

```
require 'fox16'

include Fox

theApp = FXApp.new

theMainWindow = FXMainWindow.new(theApp, "Hello")

theApp.create
```

By default, all windows in FXRuby programs are invisible, so we need to call our main window's `show` instance method:

```
require 'fox16'

include Fox

theApp = FXApp.new

theMainWindow = FXMainWindow.new(theApp, "Hello")
theApp.create

theMainWindow.show
```

The last step is to start the program's main loop by calling *theApp*'s `run` instance method:

```
require 'fox16'

include Fox

theApp = FXApp.new

theMainWindow = FXMainWindow.new(theApp, "Hello")
theApp.create

theMainWindow.show

theApp.run
```

The `FXApp#run` method doesn't return until the program exits. It is similar to the `mainloop` method used for Ruby/Tk and Ruby/GTK, and you can in fact use `FXApp#mainloop` as an alias for `run` if you prefer.

At this point, we have a working (if not very interesting) program that uses FXRuby. If you run it, you'll see something like this: