Foxing the Ruby

Brett S Hallett
Dragon City Systems

30 May 2003
reprint 30 Nov 2013

1 Disclaimer

The following information has been extracted from reading various FOX and FXRuby documents, reading example program listings & experiments with my own FXRuby code and correspondence with FXRuby users. Conclusions drawn from this study may be a quite incorrect observations of actually how FXRuby & FOX function. In other words , if you want clarification then you must study the actual source code of those two products.

2 Whats a GUI Interface ?

This might seem to be a stupid question in todays computer terms. Most computers are using different versions of Microsoft Windows, Gnome or KDE on Linux, or Apples Graphic Interface, and users are therefore already *using* a GUI (Graphic User Interface). However these users are generally not trying to *program* a GUI. This is where FXRuby comes in. FXRuby allows us to create GUI interface to our programs for the comfort of the user, and hopefully reduce input complexities for ourselves. Most users would not understand how to use a command line program, indeed many would not know about DOS Windows or Linux Consoles – and why should they !

So its up to us to develop a GUI for even simple programs - FXRuby does this very well.

3 So whats FXRuby?

Coming from a MS-Windows Delphi/Paradox development environment I naturally expected the availability of a *GUI interface tool* for Ruby. Soon realising this was not the case, I looked around for possible options. QT, Tk , GTK+ & FOX (FXRuby) appeared as useful candidates. After some simple evaluation I have chosen FXRuby.

Put simply : Our Ruby Program (uses) FXRuby classes (which calls) FOX Library (which) maintains our GUI Form (which) accepts user inputs & and actions.

While FXRuby is a very powerful tool to build FOX GUI 'forms' it is quite difficult to understand just how to define a form and control the layout (placement) of *widgets* (*controls* in MS terms)on that form. This small paper is an attempt to offer some assistance in this matter.

I have read in the various documents on FOX that the *Layout Managers* within FOX are a superior technique to the more typical MS Windows *drag & drop* widget placement, so this is my attempt make that mental conversion.

At the time of writing this, May 2003, there is no FOX GUI building tool for the rapid creation of Ruby programs although there is a project to fill this need, check www.sourceforge.net.[1]

4 Approaching a Layout Manager

I was having a problem defining a FXRuby Form so that I could test my Ruby programs, because I could not *visualise* the way I was supposed to

- define a form
- place widgets

[1]It will be interesting to see if they adopt the Layout Manager or X/Y drag & drop coordinate technique to achieve rapid form development

- access those widgets (and their values) from Ruby

Looking at the numerous example programs supplied with FXRuby [2], it was not all clear *why* certain FXRuby commands were being used.

Hopefully, I have discovered enough information to make sense of how to approach designing your (and mine) forms without too much difficulty. I am trying to develop a *technique of thought*[3] and not offering a pedantic do this or do that!, and certainly not designing lots of boiler plate designs – *your* GUI needs will be quite different to mine.

[2]which are quite useful by the way

[3]"Thinking Forth" by Leo Brodie, is an excellent example of thinking about the development process

5 Thinking about your Form

Before attempting to create a form it is advised to do a quick 'pen & paper'[4] sketch of the general layout. As FXRuby's Layout Managers do the actual positioning on the final form, you dont' need to be highly accurate, its only a simple sketch of your desired form.

Having done that, look at your sketch and draw rectangles around obvious groups, eg buttons, clusters of related fields (eg: name & address), summary fields, etc.

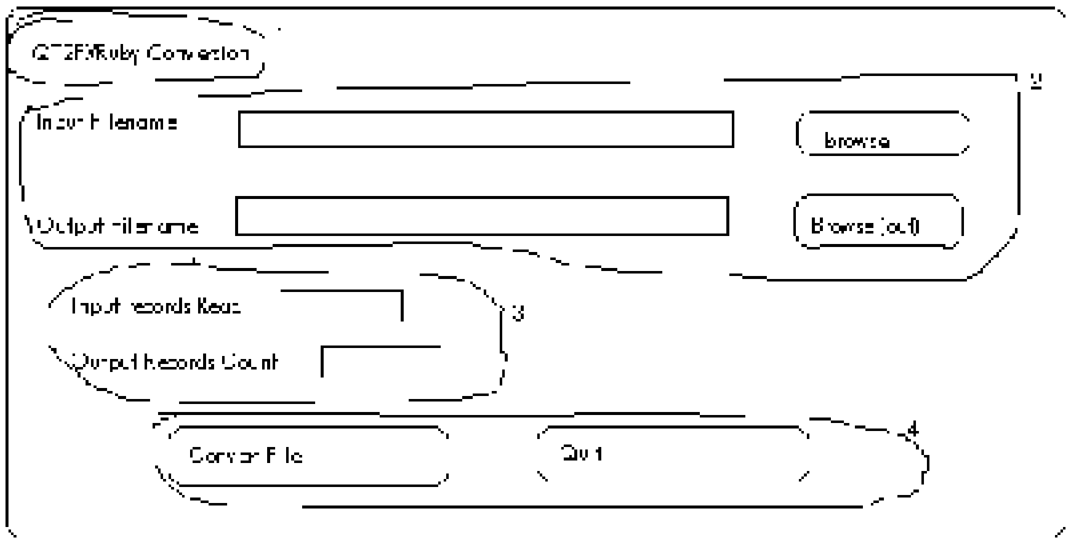

Figure 1: Sample sketch of form

By drawing these *rough frames* around related groups, we can start to make the mental connection to *FXRuby frames*. In FXRuby, a frame is a *container* for related widgets which are placed inside that frame. All the Layout Managers are actually FXRuby frames, although only FXHorizontalFrame and FXVerticalFrame use the word. Indeed, the 'outside' frame is the form window itself!

In the example sketch above, I have drawn 4 rectangles to group related widgets.

1. a text heading
2. the input/output file selection functions, with text, data entry fields & buttons
3. a results group with text and data display fields
4. action buttons

One way to think about frames is to relate them to the individual pieces in those sliding pieces puzzles – you are given pieces of different sizes & shapes you are to make a specific new shape. In our case we are to create frames (and put widgets into them), and then, arrange them into a complete FXRuby Window. FXRuby frames (puzzle pieces) are all basically rectangular boxes : square, taller than wide, wider than tall[5]

[4]you know that funny flat material that pencils can make marks on :-)
[5]never round :-)

Widgets are placed inside these frames by linking widgets to frames using *references* . If you check any widgets **new** command you will notice the first parameter is usually *'parent'*, this is the *connecting* reference to a higher level defined *frame*.

eg:

```
mainframe = FXVerticalFrame.new(self, opts=LAYOUT_FILL_X|LAYOUT_FILL_Y)

  FXLabel.new(mainframe, 'QT2FXRuby Conversion', icon= nil,
                                opts=JUSTIFY_LEFT|LAYOUT_FILL_X)
  matrix = FXMatrix.new(mainframe, 3, MATRIX_BY_COLUMNS|LAYOUT_FILL_X)

  FXLabel.new(matrix, 'Input File name :', nil,
                                opts=LABEL_NORMAL|JUSTIFY_LEFT|LAYOUT_FILL_X)
```

In the above code extract, notice how the *first* FXLabel is connecting to *mainframe*, while the *second* FXLabel is connected to *matrix*, which in turn is also connected to *mainframe*. If we change matrix's connection to another frame then the second FXLabel will automatically 'follow' matrix's new position because it belongs to matrix.

6 FXRubys Layout Managers

Available layout managers are :

Name
FXPacker
FXTopWindow
FXHorizontalFrame
FXVerticalFrame
FXMatrix
FXSwitcher
FXGroupBox
FXSplitter

Check out the FOX-TOOLKIT documentation for a full description of these layout managers. There is no point repeating that documents excellent information. Note that most FOX/FXRuby documentation is mostly reference material, as you would expect.

7 FXRuby commands

When confronted with a typical FXRuby command – what's all this ? , you might well you ask?[6]

To set the scene, I will firstly 'dissect' one command 'out of context', with no reference to any program or problem, so that hopefully we will feel more comfortable when looking at other commands.

Be warned - I dont know all the parameters, and we will often simply accept the *default* value assigned by FOX. because they are probably correct.

7.1 A typical FXRuby (FOX) command

Important note: although the FX classe descriptions use the name=? *style* for parameter description, you should only use the the *value* required in your code ie: ,x=12, should be written ,12, etc.
FXRuby & Ruby do not use parameter naming,**all paramters are positional**.

The following **new** command is from the Fox::FXMainWindow class and defines the main window, into which we are going to place other widgets.

```
new(app, title, icon=nil, miniIcon=nil, opts=DECOR_ALL,
 x=0, y=0, width=0, height=0,
 padLeft=0, padRight=0, padTop=0, padBottom=0, hSpacing=4, vSpacing=4) {|theMainWindow| ...}
```

and all this is simply a bunch of *possible* parameters we *might* need to direct FXRuby to achieve our desire effect on the form.

Taking each parameter in turn:

```
(app, title,
```

Parameter	Purpose
app	the variable name (supplied by you) of the *application*
	to which *this* FXRuby command is attached to, see later
title	the title, usually in quotes, that will appear in the top information bar of the form

```
, icon=nil, miniIcon=nil
```

Parameter	Purpose
icon	have not used this , leave as **nil**
miniIcon	have not used this , leave as **nil**

```
 opts=DECOR_ALL,
```

opts= is the most powerful feature of the FXRuby interface, and the most problematic, to understand.

You will often see opts= declarations like :

```
opts = LAYOUT_CENTER_X|PACK_UNIFORM_WIDTH|FRAME_RAISED)
```

If you browse around in the documentation you will see many possible combinations of opts= depending upon which FXRuby command you are using.[7] In the above case FXRuby is being directed to force any related widgets to be centered in the form (in the 'X' direction, left to right), forced to be uniform (ie: the same) width, and placed on a raised frame (raised border around all those widgets)

Every opts= parameter , eg PACK_UNIFORM_WIDTH, are defined (in FOX) with a unique integer value which will set/unset BOOLEAN values for testing by FOX at runtime. The above example is combining just three possible options. We dont need to be bothered with the actual value of the parameters, just use their defined names.

```
 x=0, y=0, width=0, height=0,
```

x & y	declare coordinate positions of the top left corner of your form,
	enter the required values at this position (ie: ,0,0)
width & height	define the actual size of your form (in pixels)
	enter the required values at this position (ie: ,120,20)

This is where you define your forms displayed size. If you dont define these parameters, the Layout Manager will simply create a form to fit and display all the widgets you have defined.[8]

[6]As I most definitely did!
[7]Its quite difficult to figure out which options you actually need or can use in a given situation !
[8]This feature of Layout Managers is really clever!

`padLeft=0, padRight=0, padTop=0, padBottom=0, hSpacing=4, vSpacing=4)`

More or less self explainatry, these set the padding (white space) you wish to have *around* any widgets you place on the form, basically you are overriding FXRuby default values. Mostly you can leave them alone. Again, you dont use the parameters *name*, just enter the *value* you require.

`{|theMainWindow| ...}`

This is Ruby code that *you* wish to be executed when the **new** command is executed.

8 Developing a Form

The following program is based upon Lyle Johnston's first example program in his document *Developing Graphical User Interfaces with FXRuby.* Check out the WEB and search for FXRuby which should lead you to Lyles' FXRuby download site (www.sourceforge.net).

```
#!/usr/bin/env ruby
# form1.rbw
require "fox"
include Fox

application = FXApp.new("FXRuby", "FoxTest")

main = FXMainWindow.new(application, "Hello FXRuby", nil, nil, DECOR_ALL,
       0, 0, 300, 150)

application.create()

main.show(PLACEMENT_SCREEN)
application.run()
```

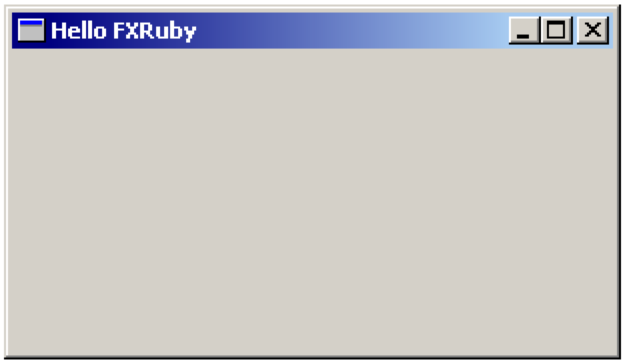

Figure 2: A simple FORM, user defined dimensions

So what have we achieved? , well we now have a simple form, that we have decided how big it is (300x150), with the standard windows title bar, with our title "Hello FXRuby". And it did not take much effort at all! Lyles' report explains all the Ruby code above, however all we need to notice is that by calling FXApp we *connected* our Ruby variable *application* to the FXRuby GUI Library which in turn is our entry point to the FOX GUI library[9]. Our Ruby variable*main* causes FOX to create a form to our specifications and connect this form to *application*. You will soon get used to connecting objects to other objects in the building of your desired forms. Basically, whats happens here is that a FXApp was created, inside that we have created a form called *main*, displayed on the center of your screen, by the Ruby command.

main.show(PLACEMENT_SCREEN)

Of course this form does not actually do anything - yet!

[9]Written in C++ as it happens

8.1 Adding a Button

A typical useful widget is a Button. Buttons are often used as a control center for a users interaction to your form. The below code will place 2 buttons on our form.

```
#!/usr/bin/env ruby
# form2.rbw
require "fox"
include Fox

application = FXApp.new("FXRuby", "FoxTest")


main = FXMainWindow.new(application, "FXRuby Form2", nil, nil, DECOR_ALL,
       0, 0, 300, 150)

FXButton.new(main, "Button 1", nil, application, FXApp::ID_QUIT)
FXButton.new(main, "Button 2", nil, application, FXApp::ID_QUIT)
#
application.create()
main.show(PLACEMENT_SCREEN)
application.run()
```

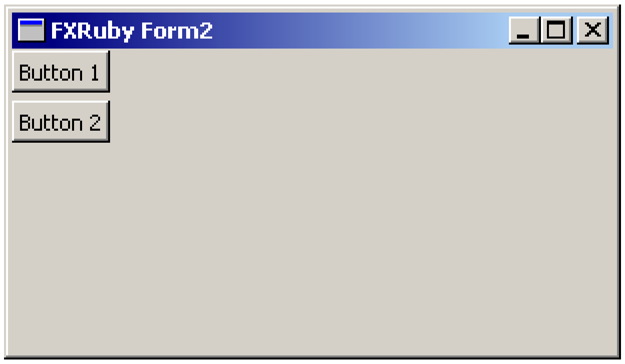

Figure 3: Adding a Button

So what happened? , well we now have two buttons on our form, placed in the top left hand corner of our Form, because we did not insist on any particular position, FXRuby 's default Layout Manager used this default position.

Each FXButton class accepts numerous parameters, however I have only used enough to get a functioning Button.

```
FXButton.new(main, "Button 2", nil, application, FXApp::ID_QUIT)
```

this displays a Button inside our Form *main*, with caption 'Button 2', no ICONs (nil), its message target is *application*, and its selector (message) is FXApp::ID_QUIT.

When the user clicks on this button, the FXApp is called (via our *application* reference) passing 'FXApp::ID_QUIT' which will activate the QUIT function within FXApp. The Form will disappear.

8.2 Placing the Button where we want

```
#!/usr/bin/env ruby
# form3.rbw

require "fox"
include Fox

application = FXApp.new("FXRuby", "FoxTest")

main = FXMainWindow.new(application, "FXRuby Form 3", nil, nil, DECOR_ALL,
         0, 0, 300, 150)
# =========================
frame1 = FXHorizontalFrame.new(main,
            LAYOUT_CENTER_X|PACK_UNIFORM_WIDTH|FRAME_RAISED)
FXButton.new(frame1, "Button 1", nil, application, FXApp::ID_QUIT)
FXButton.new(frame1, "Button 2", nil, application, FXApp::ID_QUIT)
# ========================

application.create()
main.show(PLACEMENT_SCREEN)
application.run()
```

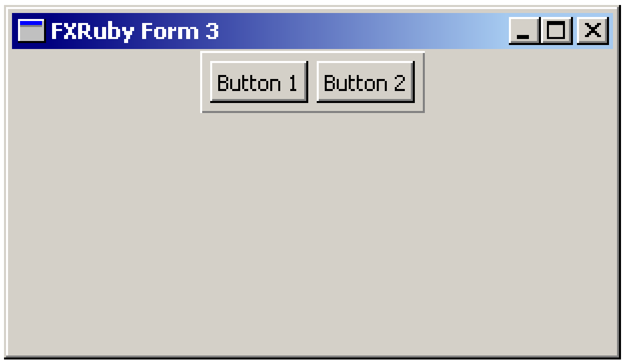

Figure 4: Adding a Button

The previous program simply accepted the FXRuby defaults, however as we wish to control the placement of our widgets, we need to give FXRuby more information.

```
frame1 = FXHorizontalFrame.new(main,
            LAYOUT_CENTER_X|PACK_UNIFORM_WIDTH|FRAME_RAISED)
```

By adding a *FRAME* and setting its opts= parameters we can control placement, size, etc, of all the widgets *within* that frame. Remember we control placement of widgets by *naming* which other widget we want to belong to.

In this case I used a FXHorizontalFrame, which gives us a manageable space on the FXMainWindow, into which we can place other widgets, including more FRAMES !

The opts= used above, place any of the frames contents (widgets), to be centered on the form(across), forced to be equal in width, and placed on a raised frame.

Note that the FXButtons have been modified to belong to *frame1*, which in turn belongs to *main*.

8.3 Adding Extra Buttons

```
#!/usr/bin/env ruby
# form4.rbw

require "fox"
include Fox

application = FXApp.new("FXRuby", "FoxTest")

main = FXMainWindow.new(application, "FXRuby Form 4",
                        nil, nil, DECOR_ALL,
                        0, 0, 300, 150)
# ========================
frame1 = FXHorizontalFrame.new(main,
            LAYOUT_CENTER_X|PACK_UNIFORM_WIDTH|FRAME_RAISED)
FXButton.new(frame1, "Button 1", nil, application, FXApp::ID_QUIT)
FXButton.new(frame1, "Button 2", nil, application, FXApp::ID_QUIT)
FXButton.new(frame1, "Button 3", nil, application, FXApp::ID_QUIT)
FXButton.new(frame1, "Button 4", nil, application, FXApp::ID_QUIT)
# ========================
application.create()
main.show(PLACEMENT_SCREEN)
application.run()
```

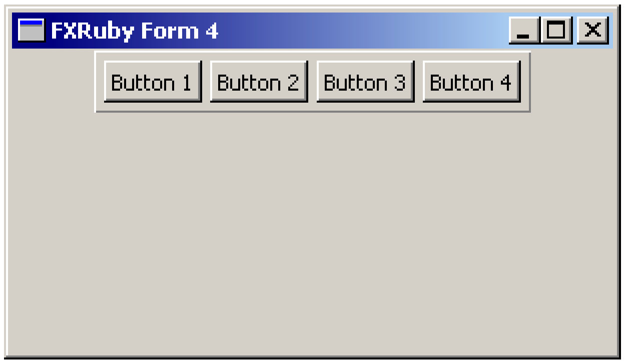

Figure 5: Adding a Button

Because I did not supply any FXHorizontalFrame *size parameters* the Layout Manager simply created a frame that could contain all the widgets defined to be placed inside that frame. It only took 2 lines of code to place the new button widgets. They all QUIT if pressed!

8.4 Centering the Frame

```
#!/usr/bin/env ruby
# form5.rbw

require "fox"
include Fox

application = FXApp.new("FXRuby", "FoxTest")

main = FXMainWindow.new(application, "FXRuby Form 5",
                        nil, nil, DECOR_ALL,
                        0, 0, 300, 150)
# ========================
frame1 = FXHorizontalFrame.new(main,
            LAYOUT_CENTER_X|LAYOUT_CENTER_Y|PACK_UNIFORM_WIDTH|FRAME_RAISED)
FXButton.new(frame1, "Button 1", nil, application, FXApp::ID_QUIT)
FXButton.new(frame1, "Button 2", nil, application, FXApp::ID_QUIT)
FXButton.new(frame1, "Button 3", nil, application, FXApp::ID_QUIT)
FXButton.new(frame1, "Button 4", nil, application, FXApp::ID_QUIT)
# ========================
application.create()
main.show(PLACEMENT_SCREEN)
application.run()
```

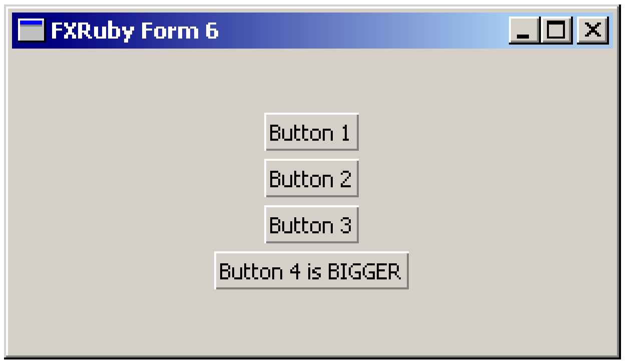

Figure 6: Adding a Button

By adding one extra parameter, LAYOUT_CENTER_Y, to the FXHorizontalFrame command the complete set of buttons are now relocated to the center of the form. This illustrates a very powerful feature of the Layout Manager approach to form creation, whenever the user resizes the form all the widgets will be re positioned into the same relative position on the form, ie: these buttons will remain in the center of the form. Whereas, with a X/Y coordinate layout system, the buttons would remain at position X/Y regardless of how the user resized the form, and would therefore appear X/Y coordinate positions from the top left corner of the newly sized form, not centered even if your original 'drag & drop' editing had placed them there.

8.5 Automatic Button Width Setting

```
#!/usr/bin/env ruby
# form6.rbw

require "fox"
include Fox

application = FXApp.new("FXRuby", "FoxTest")

main = FXMainWindow.new(application, "FXRuby Form 6",
                        nil, nil, DECOR_ALL,
                        0, 0, 300, 150)
# ========================
frame1 = FXVerticalFrame.new(main,
            opts =LAYOUT_CENTER_X|LAYOUT_CENTER_Y)

FXButton.new(frame1, "Button 1", nil, application, FXApp::ID_QUIT,
    LAYOUT_CENTER_X|LAYOUT_CENTER_Y|FRAME_RAISED)
FXButton.new(frame1, "Button 2", nil, application, FXApp::ID_QUIT,
    LAYOUT_CENTER_X|LAYOUT_CENTER_Y|FRAME_RAISED)
FXButton.new(frame1, "Button 3", nil, application, FXApp::ID_QUIT,
    LAYOUT_CENTER_X|LAYOUT_CENTER_Y|FRAME_RAISED)
FXButton.new(frame1, "Button 4 is BIGGER", nil, application, FXApp::ID_QUIT,
    LAYOUT_CENTER_X|LAYOUT_CENTER_Y|FRAME_RAISED)
# ========================
application.create()
main.show(PLACEMENT_SCREEN)
application.run()
```

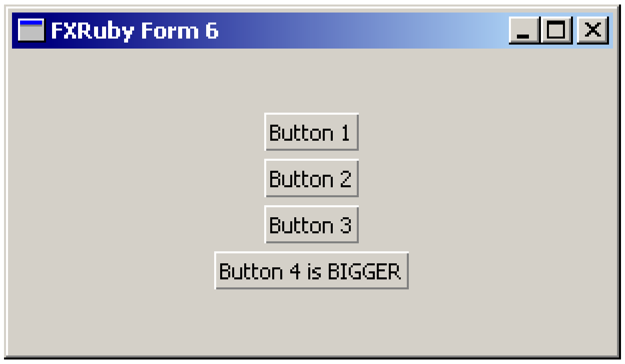

Figure 7: Button width adjusts to fit text

In this example , form6, I have made a button bigger by simply entering more text into its *text* field, FOX adjusts the button size automatically to fit, the other buttons are unaltered. Note however, that I removed the FRAME_RAISED from the FX_VERTICAL_FRAME command and therefore had to add this option to *each* FXButton to retain the buttons raised 'look'.

This illustrates nicely the way in which you can manipulate the look of the widgets, either by setting individual widgets opts= parameters or setting opts= parameters at a higher level, previous examples were using the settings from the container frame to affect the buttons look.

But be warned , Buttons are funny widgets and do not always react the way you expect, I've noted that different usage of FRAME_RAISED & FRAME_SUNKEN , and other parameters, depending upon which container you are setting opts= at , do not always result in the effect you desire!

9 A complete FXRuby Program & its Form

The following program was developed from the above sketch outline, its slightly different from the original sketch, but its similarities should be obvious.

```
require 'fox'
include Fox

class MainWindow < FXMainWindow
  def initialize(app)
    super(app, 'Form2', nil, nil,  DECOR_ALL, 0, 0, 604, 223)

    mainframe = FXVerticalFrame.new(self, LAYOUT_FILL_X|LAYOUT_FILL_Y)
```

```
# (1) place a  text heading in the top left corner of mainframe

    FXLabel.new(mainframe, 'QT2FXRuby Conversion',  nil,
                           JUSTIFY_LEFT|LAYOUT_FILL_X)
```

```
#
# (2) the input/output file selection functions are placed into a matrix layout manager.
#
    matrix = FXMatrix.new(mainframe, 3, MATRIX_BY_COLUMNS|LAYOUT_FILL_X)

# FXMatrix's divide space into ROWS or COLUMNS, here we have 3 COLUMNS

    FXLabel.new(matrix, 'Input File name :', nil,
                        LABEL_NORMAL|JUSTIFY_LEFT|LAYOUT_FILL_X)
# Place a text label left justified in column 1

    lineedit1= FXDataTarget.new("")
    textfield1 = FXTextField.new(matrix, 16, lineedit1, FXDataTarget::ID_VALUE,
                         TEXTFIELD_NORMAL|LAYOUT_FILL_X|LAYOUT_FILL_COLUMN)
# followed by a textfield in column 2

    pushbutton1= FXButton.new(matrix, 'Browse..(in)', nil, nil, 0,
                         BUTTON_NORMAL|LAYOUT_FILL_X)
#  followed by a Button in column 3

    FXLabel.new(matrix, 'Output File name :', nil,
                         LABEL_NORMAL|JUSTIFY_LEFT|LAYOUT_FILL_X)
#   as the three (3) columns are now used , a new ROW is automatically started,
#   place the text label, left justified in column 1 ( of row 2)

    lineedit2= FXDataTarget.new("")
    textfield2 = FXTextField.new(matrix, 16, lineedit2, FXDataTarget::ID_VALUE,
                         TEXTFIELD_NORMAL|LAYOUT_FILL_X|LAYOUT_FILL_COLUMN)
# followed by a textfield in column 2  ( of row 2)

    pushbutton2= FXButton.new(matrix, 'Browse..(out)', nil, nil, 0,
                         BUTTON_NORMAL|LAYOUT_FILL_X)
#  followed by a Button in column 3  ( of row 2)
```

Continues ↦

```
#
# (3) The results group of the form is handled with a horizontal frame
#

    form2c = FXHorizontalFrame.new(mainframe, LAYOUT_FILL_X|PACK_UNIFORM_WIDTH)

    FXLabel.new(form2c, 'Input Records Read', nil, LABEL_NORMAL|LAYOUT_FILL_X)

    lineedit3= FXDataTarget.new(0)
    FXTextField.new(form2c, 5, lineedit3, FXDataTarget::ID_VALUE,
                                TEXTFIELD_NORMAL|LAYOUT_FILL_X)

    FXLabel.new(form2c, 'Output Records Count', nil, LABEL_NORMAL|LAYOUT_FILL_X)

    lineedit4= FXDataTarget.new(2)
    FXTextField.new(form2c, 5, lineedit4, FXDataTarget::ID_VALUE,
                                 TEXTFIELD_NORMAL|LAYOUT_FILL_X)
```

```
#
# (4) The action buttons are placed last
#

     form2d = FXHorizontalFrame.new(mainframe, LAYOUT_CENTER_X|PACK_UNIFORM_WIDTH)

      pushConvert= FXButton.new(form2d, 'Convert', nil, nil, 0,
                                              BUTTON_NORMAL)
      pushExit= FXButton.new(form2d,    'Exit', nil, nil, 0,
                                              BUTTON_NORMAL)
  end # def initialize
```

```
  def create
    super
    show(PLACEMENT_SCREEN)
  end # create

end  # class MainWindow

# ============= main prog ============
 # Construct an application
 theApp = FXApp.new('Smithy','Max')

 # Construct the main window
 MainWindow.new(theApp)

# Create and show the application windows
 theApp.create

 # Run the application
 theApp.run
# ============= end ============
```

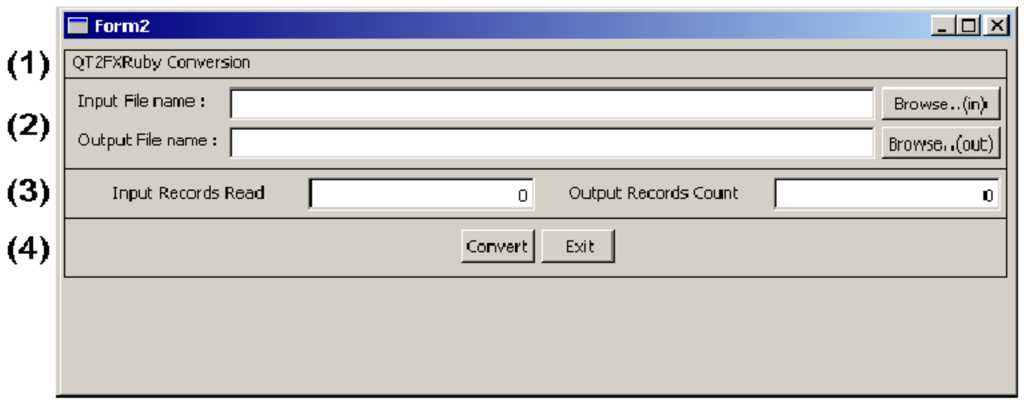

Figure 8: The complete form

The numbers in brackets eg: (1),(2), etc are references to the numbers we used on the sketch in section 5 when designing this form and also referred to in comment lines in the example FXRuby code.[10]

Inspecting the program code above and referring to the Figure 8 :

(1) As the heading is simple text, FXHorizontalFrame was used

(2) notice how a choice was made to use a FXMatrix Layout Manager rather than a FXHorizontalFrame. Using FXMatrix allowed the definition of opts=MATRIX_BY_COLUMNS which forced FXMatrix to position the three fields (a title, a text field & a button) in their own column positions, automatically adjusting widths an alignment.

(3) FXHorizontalFrame was used allowing the four fields to position themselves. opts= LAYOUT_FILL_X automatically pads the widgets across the frame space

(4) FXHorizontalFrame was used, but forcing the two buttons to be centered by opts=LAYOUT_CENTER_X

[10]The thin frame lines will not appear on the actual form

10 Useful opts= parameters

Important note: although the FX classe descriptions use the name=? *style* for parameter description, you should only use the the *value* required in your code ie: ,x=12, should be written ,12, etc.
FXRuby & Ruby do not use parameter naming,**all paramters are positional**.

Parameter	**Meaning**	**FXRuby Class**
BUTTON_NORMAL	display a button as a raised widget, centered text	FXButton
FRAME_SUNKEN	makes the frame appear sunken	FXButton All Frames
JUSTIFY_RIGHT	positions widget from the right side of the frame positions contents to right	FXButton FXTextField All Frames
JUSTIFY_LEFT	positions widget from the left side of the frame positions contents to left	FXButton FXTextField All Frames
LAYOUT_EXPLICIT	use X/Y coordinates to position a widget in a frame	FXButton FXTextField All Frames
LAYOUT_CENTER_X	positions widget in center of frame (left/right)	All Frames
LAYOUT_CENTER_y	positions widget in center of frame (up/down)	All Frames
PACK_UNIFORM_WIDTH	Forces all the widgets to be of uniform (same) width	FXTextField FXLabel FXButton
LAYOUT_FILL_X	pad (fill) between placed widgets to space widgets across a frame	FXTextField FXLabel FXButton All Frames
LAYOUT_FILL_Y	pad (fill) between placed widgets to space widgets down a frame	FXTextField FXLabel FXButton All Frames
MATRIX_BY_COLUMNS	position widgets by columns in the frame	FXMatrix frames
MATRIX_BY_ROWS	position widgets by rows in the frame	FXMatrix frames
TEXTFIELD_NORMAL	same as FRAME_SUNKEN|FRAME_THICK	FXTextField

11 Attaching code to Widgets

The main reason for using FXRuby is to offer the user a pleasant GUI interface to your Ruby program. Therefore, we need to be able to accept values from the Forms Widgets and output results to the Form.

In the code extract below, the variable *reccount* is a Ruby variable *connected* to the FXTextField on the FXRuby Form, this connection allows for easy access to the *values* entered by the user.

If the user enters a value into the displayed FXTextField, the code will automatically add 10 and redisplay the result. Interesting, after clicking the cursor on the field, simply pressing the keyboard *Enter* key will add 10 and update the display with the new value - why?[11].

```
    reccount= FXDataTarget.new(0)
    FXTextField.new(form2c, 5, reccount, FXDataTarget::ID_VALUE,
                    TEXTFIELD_NORMAL|LAYOUT_FILL_X|JUSTIFY_RIGHT)
    reccount.connect(SEL_COMMAND) do |sender, sel, checked|
               reccount.value = reccount.value + 10
end
```

The Ruby fieldname.*connect* line is the standard technique for attaching a *action. connect* is part of the FXRuby Responder2 standard module

Remember that you may put as much Ruby code between the **do** and the **end** as is required, however, it is preferable to place larger code segments into their own file and simply call that module from the FXRuby form, or place them as methods separately in the same program file. It all depends upon your prefered coding style and how much code is involved.

In the example extract code below, when the user presses the 'Start Conversion' Button, *pushbutton4*, the method *qt2fxruby*, located in file *qt2fxrubyxy* is executed, passing two values from the Form. The file *qt2fxrubyxy* is some 900 lines long, clearly not to be simply inserted between **do & end**! [12]

```
require 'qt2fxrubyxy'

   pushbutton4= FXButton.new(mainframe,'Start Conversion' ,
                             nil, nil, 0,
                             BUTTON_NORMAL)

   pushbutton4.connect(SEL_COMMAND) do |sender, sel, checked|
              qt2fxruby( lineedit1.value, lineedit2.value ) # call conversion program
              exit()
end # pushbutton4
```

[11]Because we have caused a *action* (pressing Return) and FXRuby is reacting to the that *action*

[12]qt2fxruby is actually a *complete* Ruby program, the Form is simply a GUI interface