Chapter 5. Drag and Drop

Table of Contents

One of the more powerful features available to FOX applications is drag-and-drop. It's also one of the more complicated to understand. For more background, see the standard FOX documentation on Drag and Drop.

Drop Sites

We're going to start by presenting a skeleton application consisting of a main window widget (a `DropSite` instance) that parents an `FXCanvas` widget:

```
require 'fox16'

include Fox

class DropSite < FXMainWindow
  def initialize(anApp)
    # Initialize base class
    super(anApp, "Drop Site", :opts => DECOR_ALL, :width => 400, :height => 300)

    # Fill main window with canvas
    @canvas = FXCanvas.new(self, :opts => LAYOUT_FILL_X|LAYOUT_FILL_Y)
  end

  def create
    # Create the main window and canvas
    super

    # Show the main window
    show(PLACEMENT_SCREEN)
  end
end

if __FILE__ == $0
  FXApp.new("DropSite", "FXRuby") do |theApp|
    DropSite.new(theApp)
    theApp.create
    theApp.run
  end
```

```
end
```

Since the main program (i.e. the part at the end) won't change for the rest of the tutorial, I won't show that code anymore. Since an `FXCanvas` widget relies on some other object (its message target) to draw its contents, we need to handle `SEL_PAINT` messages generated by the canvas. We'll do that by adding a handler that clears the canvas to its current background color:

```
require 'fox16'

include Fox

class DropSite < FXMainWindow
  def initialize(anApp)
    # Initialize base class
    super(anApp, "Drop Site", :opts => DECOR_ALL, :width => 400, :height => 300)

    # Fill main window with canvas
    @canvas = FXCanvas.new(self, :opts => LAYOUT_FILL_X|LAYOUT_FILL_Y)

    # Handle expose events on the canvas
    @canvas.connect(SEL_PAINT) do |sender, sel, event|
      FXDCWindow.new(@canvas, event) do |dc|
        dc.foreground = @canvas.backColor
        dc.fillRectangle(event.rect.x, event.rect.y, event.rect.w, event.rect.h)
      end
    end
  end

  def create
    # Create the main window and canvas
    super

    # Show the main window
    show(PLACEMENT_SCREEN)
  end
end
```

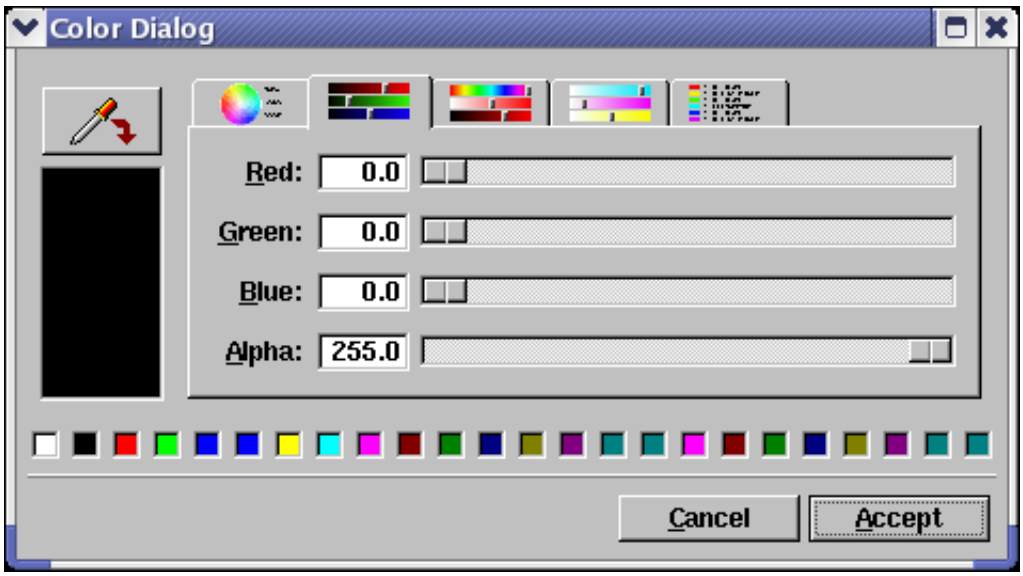
Run this basic version of the program to be sure that it's working properly so far. You should simply see an empty window with a white background.

Now, on to the fun stuff. Our goal is to be able to drag color data from some other window, such as an `FXColorWell` widget, and drop it onto the canvas in order to change the canvas' background color. In order for a FOX widget to be able to accept drops at all, we need to first call its `dropEnable` method:

```
def initialize(anApp)
  # Initialize base class
  super(anApp, "Drop Site", :opts => DECOR_ALL, :width => 400, :height => 300)

  # Fill main window with canvas
  @canvas = FXCanvas.new(self, :opts => LAYOUT_FILL_X|LAYOUT_FILL_Y)

  # Handle expose events on the canvas
```

```
  @canvas.connect(SEL_PAINT) do |sender, sel, event|
    FXDCWindow.new(@canvas, event) do |dc|
      dc.foreground = @canvas.backColor
      dc.fillRectangle(event.rect.x, event.rect.y, event.rect.w, event.rect.h)
    end
  end

  # Enable canvas for drag-and-drop messages
  @canvas.dropEnable
end
```

At this point, let's try a little test to see if the program does anything interesting yet. Start by running some other FOX or FXRuby program to use as a drag source for the color data. You should be able to use any program that displays an FXColorWell widget, and this includes the standard color dialog box shown here:



Each of the small colored boxes near the bottom of the color dialog box are color wells, and the large box on the left-hand side of the color dialog box is also a color well. Now start your drag-and-drop test program and try to drag a color from one of these color wells onto this window. At this point, the mouse pointer should turn into a stop sign, indicating that the canvas isn't accepting drops of color data yet.

To correct this problem, we need to use the canvas' `acceptDrop` method to indicate whether or not we'll accept certain kinds of drops. You can call `acceptDrop` any time after receiving the initial `SEL_DND_ENTER` message, but it's usually done in response to a `SEL_DND_MOTION` message. Let's add a handler for `SEL_DND_MOTION` messages from the canvas in DropSite's `initialize` method. For now, we'll unconditionally accept drops from any drag source, regardless of what kind of data they're offering:

```
def initialize(anApp)
  # Initialize base class
  super(anApp, "Drop Site", :opts => DECOR_ALL, :width => 400, :height => 300)

  # Fill main window with canvas
  @canvas = FXCanvas.new(self, :opts => LAYOUT_FILL_X|LAYOUT_FILL_Y)

  # Handle expose events on the canvas
```

```
  @canvas.connect(SEL_PAINT) do |sender, sel, event|
    FXDCWindow.new(@canvas, event) do |dc|
      dc.foreground = @canvas.backColor
      dc.fillRectangle(event.rect.x, event.rect.y, event.rect.w, event.rect.h)
    end
  end

  # Enable canvas for drag-and-drop messages
  @canvas.dropEnable

  # Handle SEL_DND_MOTION messages from the canvas
  @canvas.connect(SEL_DND_MOTION) do
    # Accept drops unconditionally (for now)
    @canvas.acceptDrop
  end
end
```

Now try the previous test again. This time, when you try to drag from a color well to the drop-enabled canvas, you should see the mouse pointer turn into a small filled square. This is a visual cue to the user indicating that the canvas will accept a drop of the drag-and-drop data.

Now it's time to get more specific about what kind of data is being dragged between these applications, and how to process that data. So far, our drop-enabled canvas merely knows that you're dragging some kind of data from a drag source, but it doesn't know what kind of data it is. We really need to be more exclusive about what kinds of data are acceptable. FOX uses unique drag types to distinguish between different kinds of "draggable" data. As you'll see later, you have the freedom to define drag types for any kind of application-specific data that you need; but for now, we're going to use FOX's built-in drag type for color data.

Drag types (even the standard ones) must be registered before they can be used, and so we'll start by adding a few lines to `DropSite`'s `create` method to register the drag type for color data:

```
def create
  # Create the main window and canvas
  super

  # Register the drag type for colors
  FXWindow.colorType = getApp().registerDragType(FXWindow.colorTypeName)

  # Show the main window
  show(PLACEMENT_SCREEN)
end
```

Note that the first time that `registerDragType` is called for a particular drag type name (such as `FXWindow.colorTypeName`) it will generate a unique identifier for that drag type. Subsequent calls to `registerDragType` for the same drag type name will just return the previously-generated drag type. Now, we want to modify our `SEL_DND_MOTION` handler so that it's a little more picky about which kinds of drops it will accept:

```
# Handle SEL_DND_MOTION messages from the canvas
@canvas.connect(SEL_DND_MOTION) do
  if @canvas.offeredDNDType?(FROM_DRAGNDROP, FXWindow.colorType)
    @canvas.acceptDrop
```

```
  end
end
```

Here, we call the canvas' `offeredDNDType?` method to ask if the drag source can provide its data in the requested format. Only if `offeredDNDType?` returns true will we call `acceptDrop` as before.

The last step is to actually handle the drop, and for that we add a handler for the `SEL_DND_DROP` message:

```
# Handle SEL_DND_DROP message from the canvas
@canvas.connect(SEL_DND_DROP) do
  # Try to obtain the data as color values first
  data = @canvas.getDNDData(FROM_DRAGNDROP, FXWindow.colorType)
  unless data.nil?
    # Update canvas background color
    @canvas.backColor = Fox.fxdecodeColorData(data)
  end
end
```

Assuming that the drag source is able to provide its data in the requested format, the `getDNDData` method will return a string (which for our purposes is just a byte buffer). If you've defined your own application-specific drag types, this data can of course be anything, and we'll see examples of this in a later tutorial. But the data for standard drag types like `FXWindow.colorType` can be decoded using the appropriate built-in library functions. In this case, we use the `fxdecodeColorData` method to convert the bytes into a color value that we can use.

Now comes the moment of truth. Try running your test program again (one that displays a color well). Now, when you drag a color from a color well and drop it onto the `DropSite` window, the canvas should change its background color accordingly.

The complete program is listed below, and is included in the *examples* directory under the file name *dropsite.rb*.

```
require 'fox16'

include Fox

class DropSite < FXMainWindow
  def initialize(anApp)
    # Initialize base class
    super(anApp, "Drop Site", :opts => DECOR_ALL, :width => 400, :height => 300)

    # Fill main window with canvas
    @canvas = FXCanvas.new(self, :opts => LAYOUT_FILL_X|LAYOUT_FILL_Y)

    # Handle expose events on the canvas
    @canvas.connect(SEL_PAINT) do |sender, sel, event|
      FXDCWindow.new(@canvas, event) do |dc|
        dc.foreground = @canvas.backColor
        dc.fillRectangle(event.rect.x, event.rect.y, event.rect.w, event.rect.h)
      end
    end
```

```
    # Enable canvas for drag-and-drop messages
    @canvas.dropEnable

    # Handle SEL_DND_MOTION messages from the canvas
    @canvas.connect(SEL_DND_MOTION) do
      if @canvas.offeredDNDType?(FROM_DRAGNDROP, FXWindow.colorType)
        @canvas.acceptDrop
      end
    end

    # Handle SEL_DND_DROP message from the canvas
    @canvas.connect(SEL_DND_DROP) do
      # Try to obtain the data as color values first
      data = @canvas.getDNDData(FROM_DRAGNDROP, FXWindow.colorType)
      unless data.nil?
        # Update canvas background color
        @canvas.backColor = Fox.fxdecodeColorData(data)
      end
    end
  end

  def create
    # Create the main window and canvas
    super

    # Register the drag type for colors
    FXWindow.colorType = getApp().registerDragType(FXWindow.colorTypeName)

    # Show the main window
    show(PLACEMENT_SCREEN)
  end
end

if __FILE__ == $0
  FXApp.new("DropSite", "FXRuby") do |theApp|
    DropSite.new(theApp)
    theApp.create
    theApp.run
  end
end
```