Drag Sources

As before, we're going to start by presenting a skeleton application consisting of a main window widget (a `DragSource` instance) that parents an `FXCanvas` widget:

```
require 'fox16'

include Fox

class DragSource < FXMainWindow
  def initialize(anApp)
    # Initialize base class
    super(anApp, "Drag Source", :opts => DECOR_ALL, :width => 400, :height => 300)

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

    # 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

    # 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("DragSource", "FXRuby") do |theApp|
    DragSource.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. Furthermore, the `DragSource` class has a few things in common with the `DropSite` class from the previous example, namely:

- We've defined a `SEL_PAINT` handler for the canvas widget that just clears the canvas to its current background color; and,
- The drag type for colors (`FXWindow.colorType`) is registered in the `DragSource`'s `create` method.

As before, you may want to 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 red background.

Now for this application, we want a drag operation to begin when the user presses the left mouse button inside the canvas and starts "dragging" away from the canvas. So the first change we need to make is to add a handler for the `SEL_LEFTBUTTONPRESS` message from the canvas:

```
@canvas.connect(SEL_LEFTBUTTONPRESS) do
  #
  # Capture (grab) the mouse when the button goes down, so that all future
  # mouse events will be reported to this widget, even if those events occur
  # outside of this widget.
  #
  @canvas.grab

  # Advertise which drag types we can offer
  dragTypes = [FXWindow.colorType]
  @canvas.beginDrag(dragTypes)
end
```

Note that there are usually two things you're going to want to do in the `SEL_LEFTBUTTONPRESS` handler for a drag source. The first is to call `grab` on the window that acts as the drag source. Calling `grab` "captures" the mouse in the sense that subsequent mouse motion events will be reported as if they occurred inside the grab window. This is important, since in our case we're going to be dragging from this window to some other window, possibly a window in a different application altogether.

The second thing you'll want to do in the `SEL_LEFTBUTTONPRESS` handler for a drag source is to call `beginDrag`. This not only kicks the application into drag-and-drop mode, but also provides a way for you to inform the system about the formats of data (i.e. the drag types) that this drag source is able to provide. For this example, the drag source is just going to offer one drag type.

Since releasing the left mouse button should end the drag operation, it's important to also add a handler for the `SEL_LEFTBUTTONRELEASE` message:

```
@canvas.connect(SEL_LEFTBUTTONRELEASE) do
  @canvas.ungrab
  @canvas.endDrag
end
```

This is pretty much the mirror image of our `SEL_LEFTBUTTONPRESS` handler. We call `ungrab` to release the mouse capture, and `endDrag` to clean up the drag-and-drop state.

The next change is to add a `SEL_MOTION` handler, to handle mouse motion events during the drag operation:

```
@canvas.connect(SEL_MOTION) do |sender, sel, event|
  if @canvas.dragging?
    @canvas.handleDrag(event.root_x, event.root_y)
    unless @canvas.didAccept == DRAG_REJECT
      @canvas.dragCursor = getApp().getDefaultCursor(DEF_SWATCH_CURSOR)
    else
      @canvas.dragCursor = getApp().getDefaultCursor(DEF_DNDSTOP_CURSOR)
    end
  end
end
```

The `dragging?` method returns true if we're in the middle of a drag-and-drop operation for the drag source window, i.e. after the call to `beginDrag` but before the call to `endDrag`. If we're not currently processing a drag operation, we're not really interested in mouse motion events for the canvas.

If we *are* processing a drag operation, however, we need to call `handleDrag` on the drag source window. FOX uses this information to send drag-and-drop messages (such as `SEL_DND_ENTER`, `SEL_DND_MOTION` and `SEL_DND_LEAVE`) to the

window that the mouse is currently over. Note that the coordinates passed to `handleDrag` are root window coordinates, and not window-local coordinates.

Another good thing to consider doing here is to change the shape of the cursor depending on the drop site's response to the drag-and-drop messages from the drag source. For this example, we change the drag cursor to one of FOX's built-in cursor shapes (`DEF_SWATCH_CURSOR`) if we get any response other than `DRAG_REJECT` from the drop site. If the drop site rejects this drag source's overtures, we instead change the drag cursor to a cursor that resembles a stop sign (`DEF_DNDSTOP_CURSOR`).

I've purposely avoided suggesting that you run the program for the last couple of changes, since until now there wasn't going to be any visual indication that anything interesting was happening. But now you should be able to run the application in its current state and see a few things.

A first test is to confirm that the cursor shape changes to the "stop" sign when you attempt to drag over some window that isn't willing to accept a drop of the `FXWindow.colorType` drag type. Start up one of the other FXRuby example programs (such as the *scribble.rb* example) alongside your drag source program, and then try "dragging" from the drag source's canvas onto the Scribble program's canvas. During the drag attempt, the cursor should maintain its shape as a "stop" sign since the canvas in the *scribble.rb* example isn't drop-enabled.

Your next test can confirm that the cursor shape changes to the "swatch" cursor (a small square) when you attempt to drag over a window that is drop-enabled, and which is willing to accepts drops of this drag type. The obvious choice is the example from the previous section (the dropsite.rb program), but you should have success with any FXRuby example program that displays color wells. Start up one of these drop-enabled programs alongside your drag source program, and then try "dragging" from the drag source's canvas onto the drop site. While the mouse pointer is over a drop-enabled window, the cursor should change its shape.

The last (and most important) step is to actually complete the data transfer. At any time during a drag-and-drop operation, a drop site may request a copy of the drag-and-drop data by calling the `getDNDData` method (as described in the previous section). When this happens, FOX sends a `SEL_DND_REQUEST` message to the current drag source window, and so we now add a handler for that message:

```
@canvas.connect(SEL_DND_REQUEST) do |sender, sel, event|
  if event.target == FXWindow.colorType
    @canvas.setDNDData(FROM_DRAGNDROP, FXWindow.colorType, Fox.fxencodeColorData(@canvas.backColor))
  end
end
```

The first important thing to note here is that the `target` field of the `FXEvent` instance will contain the drag type requested by the drop site. If your drag source offers its data in multiple formats, you definitely need to check this in order to know how to package-up the data for transfer. However, even if you're only offering the data in a single format (as in this example) it's still a good idea to check the requested type, since a rogue drop site could ask for the data in some unexpected format.

Assuming that the drag type is as expected, the last step is send the data to the drop site by calling `setDNDData`. As with the call to `getDNDData` for our previous drop site program, you want to be sure to specify the origin of the data (`FROM_DRAGNDROP`), the drag type (`FXWindow.colorType`) and the data itself as a binary string.

As a final test of the completed program, try dragging from this window to some drop-enabled window (as described earlier). Now, when you release the left mouse button to complete the "drop", the drop site should change its color to red.

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

```
require 'fox16'

include Fox

class DragSource < FXMainWindow
  def initialize(anApp)
    # Initialize base class
    super(anApp, "Drag Source", :opts => DECOR_ALL, :width => 400, :height => 300)
```

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

    # 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

    # Handle left button press
    @canvas.connect(SEL_LEFTBUTTONPRESS) do
      #
      # Capture (grab) the mouse when the button goes down, so that all future
      # mouse events will be reported to this widget, even if those events occur
      # outside of this widget.
      #
      @canvas.grab

      # Advertise which drag types we can offer
      dragTypes = [FXWindow.colorType]
      @canvas.beginDrag(dragTypes)
    end

    # Handle mouse motion events
    @canvas.connect(SEL_MOTION) do |sender, sel, event|
      if @canvas.dragging?
        @canvas.handleDrag(event.root_x, event.root_y)
        unless @canvas.didAccept == DRAG_REJECT
          @canvas.dragCursor = getApp().getDefaultCursor(DEF_SWATCH_CURSOR)
        else
          @canvas.dragCursor = getApp().getDefaultCursor(DEF_DNDSTOP_CURSOR)
        end
      end
    end

    # Handle left button release
    @canvas.connect(SEL_LEFTBUTTONRELEASE) do
      @canvas.ungrab
      @canvas.endDrag
    end

    # Handle request for DND data
    @canvas.connect(SEL_DND_REQUEST) do |sender, sel, event|
      if event.target == FXWindow.colorType
        @canvas.setDNDData(FROM_DRAGNDROP, FXWindow.colorType, Fox.fxencodeColorData(@canvas.backColor))
      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("DragSource", "FXRuby") do |theApp|
    DragSource.new(theApp)
    theApp.create
    theApp.run
  end
end
```