The Paradox Paint Application (PDXPAINT.FSL)

Use the Paradox Paint application to make simple drawings. You can use any of the following components repeatedly while you draw.

- **Brush Color:** Click a color to choose the color of the square to draw.
- **Brush Size:** Click a size box to choose the size of the square to draw.
- **Erase button:** Click the Erase button to clear the canvas.

To draw, select a pen color and size, then drag the mouse on the canvas to create a succession of squares.

Reference

For information on the programming techniques used to create the Paradox Paint application, see Paradox Paint Programmer's Help.

Paradox Paint Programmer's Help

This small sample application, called Paradox Paint, demonstrates techniques for working with UIObjects at run time. The application demonstrates how you can create modular applications and portable objects by localizing variables and custom procedures rather than attaching as much as possible to the form.

Most of the code in this application is attached to three objects: *colorBox*, *sizeBox*, and *thePaper*. Variables global to these three objects are contained in *paintBox* and are declared in the Var window of *paintBox*. If you want to use these objects and the attached code in another application, you can copy *paintBox* (and all the objects *paintBox* contains), paste it into another form, and it will run without modification. Similarly, if you want a color palette, you can copy and paste *colorBox*. After you declare the necessary variables in the destination form, the palette is created.

Reference

Select one of the following objects for a description of the code attached to it:

Paint Form
paintBox
colorBox
sizeBox
thePaper
helpButton
eraseButton

Form Methods

Code attached to the Paradox Paint form performs two tasks: it makes sure you're running the application from the working directory, and it invokes the Help application when you press F1.

Select one of the following methods for a description of the code attached to the form:

open method

keyPhysical method

Select one of the following objects for a description of the code attached to it:

paintBox

colorBox

sizeBox

thePaper

helpButton

eraseButton

Form open Method

Code attached to the form's built-in **open** method makes sure you're running this application from the working directory. The call to **isFile** does the checking: when you give a file name without a path or an alias, Paradox looks in the working directory by default.

```
method open(var eventInfo Event)
   if not eventInfo.isPreFilter() then
      if not isFile("pdxpaint.fsl") then
         msgInfo("Startup Error!", "The ObjectPAL example files must
                 be in the working directory.")
         close()
      endIf
   endIf
endmethod
```

Form keyPhysical Method

Code attached to the form's built-in **keyPhysical** method invokes the Windows Help application to display context-sensitive help when you press F1. The call to **vChar** identifies each key as it's pressed. The call to **disableDefault** blocks Paradox's built-in response.

```
method keyPhysical(var eventInfo KeyEvent)
if eventInfo.isFirstTime() then
  if eventInfo.vChar() = "VK_F1" then
     disableDefault
     helpShowIndex(":WORK:pdxpaint.hlp")
  endIf
endIf
endmethod
```

Code Attached to *paintBox*

The box that frames this application is named *paintBox*. Variables declared in *paintBox's* Var window are global to all objects *paintBox* contains.

The *paintBox* Var window

```
var
  brushColor LongInt ; color of paint brush
  brushSize Point    ; size of paint brush
endvar
```

Select one of the following objects for a description of the code attached to it:

Paint Form

colorBox

sizeBox

thePaper

helpButton

eraseButton

Code Attached to *colorBox*

colorBox is a palette of colors. With one exception, the code attached to *colorBox* is generalized and self-contained: You can add or delete colored boxes within *colorBox* as often as you like, and the code will run. The exception is the returned value *brushColor*, which is declared in the Var window of *paintBox*. If you copy *colorBox* to another form, remember to declare *brushColor* in the Var window of an object that contains *colorBox*.

Select one of the following windows or methods for a description of the code attached to *colorBox:*

The Var window

open method

close method

mouseDown method

setColor procedure

Select one of the following objects for a description of the code attached to it:

Paint Form

paintBox

sizeBox

thePaper

helpButton

eraseButton

The *colorBox* Var Window

Variables declared in the Var window are visible to all built-in methods, custom methods, and custom procedures attached to *colorBox* as well as to all the objects *colorBox* contains.

```
var
  newColor, oldColor UIObject
  colorArray Array[] String
endVar
```

The *colorBox* open Method

The code attached to *colorBox*'s built-in **open** method takes inventory of the colored boxes it contains. It uses the UIObject type method **enumObjectNames**, which creates an array of object names, beginning with the name of the object that called the method, and continuing through all objects the calling object contains.

Next, the UIObject type **attach** method reads the object name stored in item 2 of the array, and assigns that object to the variable *oldColor*. Then, the custom procedure *setColor* uses *oldColor* to specify a default color.

```
method open(var eventInfo Event)
   self.enumObjectNames(colorArray) ; item 1 is self,
                                    ; item 2 is first item contained
   oldColor.attach(colorArray[2])   ; attach to first contained item
   setColor(oldColor)               ; specify a default color
endmethod
```

The *colorBox* close Method

The code attached to *colorBox*'s built-in **close** method uses the array created by the **open** method to make all colored boxes appear popped out. It gets this effect by setting the Frame.Style property to Outside3DFrame. (Inside3DFrame makes a box look pushed in.)

```
method close(var eventInfo Event)
   var
      i SmallInt ; declare the variable to make code execute faster
   endVar

   for i from 2 to colorArray.size() ; start at 2 because item 1 is self
      oldColor.attach(colorArray[i])
      if oldColor.frame.style <> Outside3DFrame then ; makes boxes pop out
         oldColor.frame.style = Outside3DFrame
      endIf
   endFor
endmethod
```

The *colorBox* mouseDown Method

The built-in **mouseDown** method executes for each mouse click *colorBox* receives. It also executes for each mouse click received by the colored boxes it contains. By default, a box passes a mouse click to its container. There's no code attached to the colored boxes, so the mouse click is passed to *colorBox*.

The event packet for the mouse click contains information about the target object (that is, the object that was actually clicked). If the target was *colorBox*, this code ignores the click. But, if the target was one of the colored boxes, this method gets information about the target and passes it to the custom procedure *setColor*.

```
method mouseDown(var eventInfo MouseEvent)
   if not eventInfo.isTargetSelf() then ; if self is not the target
      eventInfo.getTarget(newColor)     ; then a colored box is the target
      setColor(newColor)                ; pass colored box info to setColor
  proc
   endIf
endmethod
```

The *colorBox* setColor Procedure

The custom procedure *setColor* is declared in *colorBox*'s Proc window. It gets the Color property of the colored box specified in the **mouseDown** or **open** methods. It also sets the pushed-in and popped-out appearance of the boxes, as appropriate.

You could use a custom method here but there are tradeoffs: a custom method is public, and can be called by objects not in the containership path. However, a custom method isn't needed here. The only object that calls this procedure is *colorBox*. A custom procedure is private, but Paradox for Windows can call a custom procedure faster than a custom method, though the code inside executes at the same speed.

```
proc setColor(newColor UIObject)
  brushColor = newColor.color ; get Color property of chosen color box
  oldColor.frame.style = Outside3DFrame ; make old color box pop out
  newColor.frame.style = Inside3DFrame  ; make new color box push in
  oldColor.attach(newColor)   ; assign oldColor for next time through
endproc
```

Code Attached to *sizeBox*

Code attached to *sizeBox* works similarly to the code attached to *colorBox*: the **open** method takes inventory of contained objects, the **close** method sets display attributes, the **mouseDown** method processes a mouse click, and a custom procedure (**setSize**) assigns a value to a variable based on the object that got the mouse click. Like *colorBox*, *sizeBox* is self-contained and portable.

Select one of the following for a description of the code attached to *sizeBox:*

The Var window

open method

close method

mouseDown method

setSize procedure

Select one of the following objects for a description of the code attached to it:

Paint Form

paintBox

colorBox

thePaper

helpButton

eraseButton

The *sizeBox* Var Window

Variables declared in the Var window are visible to all built-in methods, custom methods, and custom procedures attached to *sizeBox* as well as to all the objects *sizeBox* contains.

```
var
   oldSize, newSize UIObject
   sizeArray Array[] String
endVar
```

The *sizeBox* open Method

The code attached to *sizeBox*'s built-in **open** method takes inventory of the colored boxes it contains. It uses the UIObject type method **enumObjectNames**, which creates an array of object names, beginning with the name of the object that called the method, and continuing through all objects the calling object contains.

Next, the UIObject type **attach** method reads the object name stored in item 2 of the array, and assigns that object to the variable *oldSize*. Then, the custom procedure *setSize* uses *oldSize* to specify a default color.

```
method open(var eventInfo Event)
   self.enumObjectNames(sizeArray) ; item 1 is self,
                                   ; item 2 is first item contained
   oldSize.attach(sizeArray[2])    ; pass UIObject var oldSize
                                   ; to the setSize proc
   setSize(oldSize)
endmethod
```

The *sizeBox* close Method

The code attached to *sizeBox*'s built-in **close** method uses the array created by the **open** method to make all brush size boxes appear popped out. It gets this effect by setting the Frame.Style property to Outside3DFrame. (Inside3DFrame makes a box look pushed in.)

```
method close(var eventInfo Event)
   var
      i SmallInt
   endVar

   for i from 2 to sizeArray.size() ; start at 2 because item 1 is self
      oldSize.attach(sizeArray[i])
      if oldSize.frame.style <> Outside3DFrame then
         oldSize.frame.style = Outside3DFrame
      endIf
   endFor
endmethod
```

The *sizeBox* mouseDown Method

The built-in **mouseDown** method executes for each mouse click *sizeBox* receives. It also executes for each mouse click received by the brush size boxes it contains. By default, a box passes a mouse click to its container. There's no code attached to the boxes, so the mouse click is passed to *sizeBox*.

The event packet for the mouse click contains information about the target object (that is, the object that was actually clicked). The call to **isTargetSelf** tests the target of the event. If the target was *sizeBox*, this code ignores the click. But, if the target was one of the brush size boxes, this method calls **getTarget** to get information about the target. This information is then passed to the custom procedure *setSize*.

```
method mouseDown(var eventInfo MouseEvent)
   if not eventInfo.isTargetSelf() then
      eventInfo.getTarget(newSize)
      setSize(newSize)
   endIf
endmethod
```

The *sizeBox* setSize Procedure

The custom procedure *setSize* is declared in *sizeBox*'s Proc window. It gets the Size property of the brush size box specified in the **mouseDown** or **open** methods. It also sets the pushed-in and popped-out appearance of the boxes, as appropriate.

You could use a custom method here but there are tradeoffs: a custom method is public, and can be called by objects not in the containership path. However, a custom method isn't needed here. The only object that calls this procedure is *sizeBox*. A custom procedure is private, but Paradox for Windows can call a custom procedure faster than a custom method, though the code inside executes at the same speed.

```
proc setSize(newSize UIObject)
  brushSize = newSize.size
  oldSize.frame.style = Outside3DFrame
  newSize.frame.style = Inside3DFrame
  if oldSize.name <> newSize.name then
     oldSize.attach(newSize)
  endIf
endproc
```

Code Attached to *thePaper*

Code attached to the built-in **mouseMove** method gets data about the mouse position and uses it to create paint objects. This method is triggered when the mouse moves over *thePaper*. Three MouseEvent type methods get information from the event packet:

- **isLeftDown** reports whether the left mouse button is pressed as the mouse is moving.
- **isInside** reports whether the mouse pointer is still inside the object.
- **getMousePosition** returns the position of the mouse pointer at the time of the event.

thePaper mouseMove method

This statement uses the UIObject type method create to create an object onscreen. The argument BoxTool says to create a box; arguments eventInfo.x() and eventInfo.y() specify the coordinates of the upper left corner of the box, and arguments brushSize.x() and brushSize.y() specify the coordinates of the lower right corner of the box. The argument Self specifies a coordinate system relative to the calling object; in this case, it's thePaper. See the ObjectPAL Reference Guide for details about using create.

```
method mouseMove(var eventInfo MouseEvent)
   var
      thePaint,
      targetUI  UIObject
      thePoint,
      newPoint  Point
   endVar

   if eventInfo.isLeftDown() then
      If eventInfo.isTargetSelf() and eventInfo.isInside() then
          thePaint.create(boxTool, eventInfo.x(), eventInfo.y(),
                          brushSize.x(), brushSize.y(), self)
         thePaint.color = brushColor
         thePaint.frame.color = brushColor
         thePaint.visible = Yes
      else
         eventInfo.getMousePosition(thePoint)
         eventInfo.getTarget(targetUI)
         if targetUI.containerName = "#Page2.paintBox.thePaper" then
            thePaint.attach(targetUI.ContainerName)
            targetUI.convertPointWithRespectTo(thePaint, thePoint,
                                                          newPoint)
            thePaint.create(boxTool, newPoint.x(), newPoint.y(),
                            brushSize.x(), brushSize.y(), thePaint)
            thePaint.color = brushColor
            thePaint.frame.color = brushColor
            thePaint.visible = Yes
         endIf
      endIf
   endIf
endmethod
```

Select one of the following objects for a description of the code attached to it:

Paint Form

paintBox

colorBox

sizeBox

helpButton
eraseButton

Help Button Methods

Code attached to the built-in **pushButton** method of the button named *helpButton* invokes the Help application to display help for the Paradox Paint form.

```
method pushButton(var eventInfo Event)
   message("Loading HELP; one moment, please...")
   helpShowIndex(":WORK:pdxpaint.hlp")
   message("")
endmethod
```

Select one of the following objects for a description of the code attached to it:

Paint Form
paintBox
colorBox
sizeBox
thePaper
eraseButton

Erase Button Methods

Code attached to the built-in **pushButton** method of the button named *eraseButton* erases the paint; that is, it deletes the colored boxes. The call to **enumObjectNames** fills an array with the names of the objects contained by *thePage*. The first item in this array is the name of *thePage*, and that's why the **for** loop sets the value of *arrayCtr* to 2.

```
method pushButton(var eventInfo Event)
var

   boxArray Array[] String
   pageName UIObject
   arrayCtr SmallInt
endvar
   doDefault
   message("One moment while I erase your drawing...")
   delayScreenUpdates(True)
   pageName.attach("thePaper")
   pageName.enumObjectNames(boxArray)
   for arrayCtr from 2 to boxArray.Size()
      pageName.attach(boxArray[arrayCtr])
      pageName.delete()
   endFor
   delayScreenUpdates(False)
   message("")
endmethod
```

Select one of the following objects for a description of the code attached to it:

Paint Form
paintBox
colorBox
sizeBox
thePaper
helpButton