T U T O R I A L

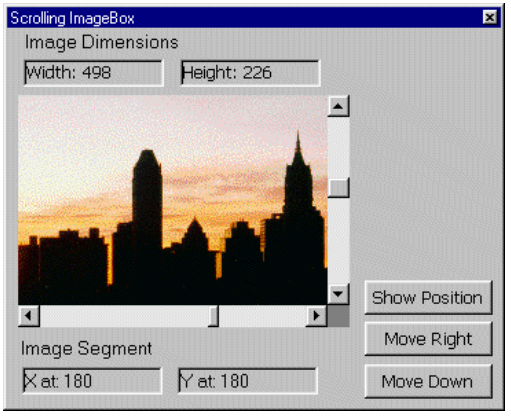
A Scrolling ImageBox Control

Visual Basic doesn't provide a control that can display large images and allow the user to select an area to view with the help of scroll bars, so the ScrollingImageBox control is one that you will use in all kinds of projects. You can also use this control on Web pages. It provides a Picture property, similar to the regular ImageBox control. The Picture property can be assigned an image file, which will be displayed on the control. The difference between the custom scrolling ImageBox control we are going to develop here and the regular ImageBox control is that the custom control doesn't resize the image of its viewing area. It attaches two scroll bars that let the user view any portion of the image. We are also going to add a PictureFromURL property that will download and display an image from a Web server.

The Form shown in Figure 1 is the test Form for the control, which can be found in the ScrlImg folder on the CD. This control was designed to host large images; if the image is smaller than the control's viewing area, the control won't be resized automatically. Instead, the scroll bars will be disabled, but not removed. You might want to tweak the code to add a feature that will autosize the control if the image is smaller than its viewing area. In addition to allowing the user to scroll the image with the help of the scroll bars, the control provides properties for reporting the image's size and position on the control, as well as methods for scrolling the image under program control.

FIGURE 1:

The ScrollImageBox control displaying a large image

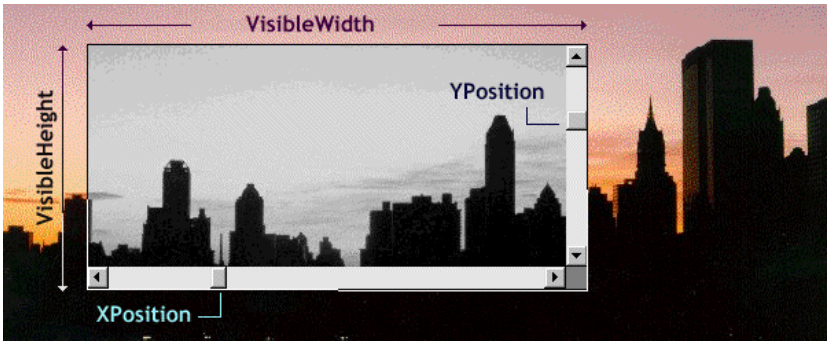


The ScrollImageBox Control's Custom Members

The ScrollImageBox control has the following properties, methods, and events. Figure 2 shows the properties.

FIGURE 2:

The properties of the ScrollImageBox custom control that report the location of the image's visible area on the control



The Xposition and YPosition Properties These two properties report the coordinates of the top left visible pixel of the image (the origin of the image's visible section).

The VisibleWidth and VisibleHeight Properties These two properties return the dimensions of the visible part of the image (which are the same as the control's dimensions). The rectangle of the image displayed in the control's viewing area starts at coordinates (XPosition, YPosition) and has dimensions (VisibleWidth, VisibleHeight). All properties are expressed in pixels.

The PictureWidth and PictureHeight Properties These two properties return the actual size of the image, including the part that's not visible.

The ScrollTo(X, Y) Method This method allows you to scroll the image to pixel (X, Y) from within your code. After calling this method, the control's XPosition property becomes equal to X, and the YPosition property becomes equal to Y.

The ScrollX and ScrollY Events These events are triggered every time the user scrolls the thumb on the vertical or horizontal bar. They are raised from within the scroll bars' Scroll event, and they can be used to monitor any changes in the values of the scroll bars continuously as the user slides them.

The ChangeX and ChangeY Events These events are triggered every time the user changes the value of the vertical or horizontal scroll bar. They are raised from within the scroll bars' Change event, and they can be used to monitor any changes in the scroll bars after the user has moved the thumb to a new value.

Two more useful properties you can easily add to the control are the SmallChange and LargeChange properties, which would allow developers to set the equivalent properties of the ScrollBar controls. Instead, the custom control scrolls the image by 1 pixel when the scroll arrows are clicked and by 10 pixels when the area of the scroll bar between the arrows and the thumb is clicked. You can easily implement these

properties by adding the corresponding property procedures with the ActiveX Control Interface Wizard and mapping them to the corresponding properties of the two ScrollBar controls.

Testing the ScrollImgBox Control

The test project of the Scrolling Image custom control has two Forms: The TestForm (shown in Figure 3) and the VideoForm (shown in Figure 4). The TestForm contains an instance of the ScrollImgBox control, a few Command Buttons that let you scroll the image under program control, and a few labels that display the dimensions and the current location of the image.

FIGURE 3:

The TestForm Form of the ScrollImgBox control's test project

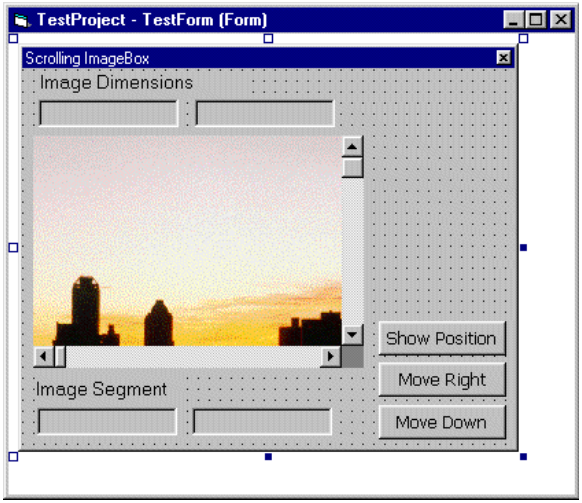
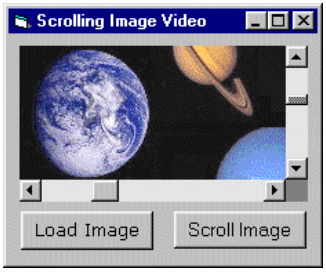


FIGURE 4:

The VideoForm Form of the ScrollImgBox control's test project



The Move Right and Move Down buttons lets you scroll the image in the corresponding direction by five pixels at a time. As the image is repositioned on the control (either with the buttons or manually), the coordinates of its upper-left visible pixel are displayed in the two Label controls at the bottom of the Form. The two Labels at the top display the dimensions of the image, even if part of it is not visible. Finally, the Show Position button displays the image's dimensions and the origin of its first visible pixel on a message box.

The VideoForm downloads an image from a Web server and displays it on the ScrollImgBox control. Notice that you must first download an image and then scroll it. For those of you who are not interested in using the control on Web pages, I have placed an image on the control already, which is displayed when you open the project (it doesn't have to be downloaded from a Web server). If you click the Scroll Image button, the image code will start panning the image smoothly, giving the impression of video playback. I'll discuss the code of the VideoForm later in this tutorial. Let's start with code of the TestForm.

The TestForm maintains two variables that correspond to the current position of the image on the custom control, which are declared as:

```
Dim X As Integer, Y As Integer
```

As soon as the Form is loaded, the dimensions of the bitmap are displayed on two Label controls at the top of the Form:

```
Private Sub Form_Load()
    Label1.Caption = "Width: " & ScrollImgBox1.PictureWidth
    Label2.Caption = "Height: " & ScrollImgBox1.PictureHeight
End Sub
```

When you click the Move Right and Move Down buttons , the X and Y variables increase by five pixels, causing the image to scroll in the corresponding direction by that amount. The ScrollTo method is then called with the updated values of the X and Y controls to scroll the image. Here are the Click event handlers of the two Command Buttons:

```
Private Sub Command1_Click()
    X = X + 5
    ScrollImgBox1.ScrollTo (X), (Y)
End Sub

Private Sub Command2_Click()
    Y = Y + 5
    ScrollImgBox1.ScrollTo (X), (Y)
End Sub
```

Finally, each time the image is scrolled on the control, the coordinates on the two Labels at the bottom of the Form are updated from within the ScrollX/ScrollY and ChangeX/ChangeY events of the control:

```
Private Sub ScrollImgbox1_ChangeX(X As Integer)
    Label4.Caption = "X at: " & X
End Sub

Private Sub ScrollImgbox1_ChangeY(Y As Integer)
    Label5.Caption = "Y at: " & Y
End Sub

Private Sub ScrollImgBox1_Click()
    MsgBox "I was clicked"
End Sub

Private Sub ScrollImgbox1_ScrollX(X As Integer)
    Label4.Caption = "X at: " & X
End Sub

Private Sub ScrollImgbox1_ScrollY(Y As Integer)
    Label5.Caption = "Y at: " & Y
End Sub
```

No matter how the image is scrolled (manually or under program control), these events will be triggered, and the Form will be updated. Now that you have seen how the control is used on a Form, we can look at the details of its implementation. I'll discuss the VideoForm test Form toward the end of this tutorial, and you'll see how the custom control can download an image over the Internet.

Implementing the ScrollImgBox Control

The ScrollImgBox control contains two PictureBox controls. One of them (the Picture2 control) is hidden (Visible property = False), and its AutoSize property is set to True. This control stores the entire image. The other PictureBox control (the Picture1 control) is visible and holds only part of the image.

The two scroll bars attached to the control allow the user to select any part of the image to view, and they are totally independent of the PictureBox controls. They are placed to the right and at the bottom of the Picture1 control from within the control's code. Their Max values match the size of the image displayed. The vertical scroll bar's Max property, for example, is set to the image's height minus the height of the

control's viewing area (see Figure 2, earlier in this tutorial). Each time one of the scroll bars changes value, another part of the image is copied on the visible PictureBox control with the PaintPicture method.

Adding the Standard Members

The ScrollImgBox control was designed with the help of the ActiveX Control Interface Wizard, but it's a fairly complicated control. In this tutorial I will discuss the more interesting parts of the control's implementation. For details on using the ActiveX Control Interface Wizard, see Chapter 16 of *Mastering Visual Basic 6*.

Let's start with the standard members of the custom control. If you want to design the control from scratch, start the ActiveX Control Interface Wizard and follow these steps (or open the ScrlImg project, start the ActiveX Control Interface Wizard, and examine its screens):

1. In the Select Interface Members screen, add the following standard members to the control:
   - BorderStyle, Enabled, and Picture properties
   - Refresh method
   - Click, DblClick, KeyDown, KeyPress, KeyUp, MouseDown, MouseMove, and MouseUp events
2. In the Create Custom Interface Members screen, add the following members:

Properties	Methods	Events
PictureWidth	ScrollTo	ScrollX
PictureHeight		ScrollY
VisibleWidth		ChangeX
VisibleHeight		ChangeY
PictureFromURL		DownloadFailed
Xposition		DownloadCompleted
Yposition		

3. In the Set Mapping screen, set the following mappings:
   - Map the ScrollX and ScrollY events to the Scroll events of the horizontal and vertical scroll bar.

- Map the ChangeX and ChangeY events to the Change events of the horizontal and vertical scroll bar.

Every time the scroll bars change value, the custom events will be triggered. Of course, the Wizard will not add the code for scrolling the image on the control. We'll have to edit the code and add the statements for scrolling the images later.

- Map the Xposition and Yposition properties to the Value property of the horizontal and vertical scroll bars so that they always reflect the current position of the image on the control.
- Map the Picture property to the Picture property of the Picture1 control.
- Map all standard events (mouse and keyboard events) to the corresponding events of the Picture1 control, which is the largest visible element on the control.

4. In the Set Attributes screen, set the following member attributes:

- Add an Integer argument to the ChangeX and ChangeY events:

```
ChangeX(X As Integer)
ChangeY(Y As Integer)
```

(X and Y are the values of the image's origin when it's repositioned on the control by the user.)

- Add the same arguments to the ScrollX and ScrollY methods.
- Set the PictureFromURL property's data type to String.
- Make all other properties Integer. (If you plan to use the control with images that exceed 32,000 pixels in either direction, make all properties and arguments Long instead of Integer.) An image of dimensions 32K ×32K pixels is probably too large for most Windows applications).
- Assign the following arguments to the ScrollTo method:

```
ScrollTo(X As Integer, Y As Integer)
```

Just enter the string "X As Integer, Y As Integer" (without the quotes) in the Arguments box.

5. Click the Finish button, and the Wizard will generate the skeleton for the control.

Programming the Custom Members

Now you must step in and add the code that will make the control work. The code added by the Wizard is trivial and takes care of the basic properties, but it

doesn't scroll the image. You must supply the code that makes the control unique. Let's examine the custom properties of the control.

The XPosition and YPosition Properties

These properties are implemented with the following Property Get procedures:

```
Public Property Get XPosition() As Variant
    XPosition = HScroll1.Value
End Function

Public Property Get YPosition() As Variant
    YPosition = VScroll1.Value
End Function
```

You can make these two properties read-only by omitting the matching Property Set procedures. In this case, the developer will have to use the ScrollTo method to scroll the image. The values of these properties are not stored in local variables either; they are read directly from the scroll bars' Value properties. If you have mapped the Xposition and Yposition properties to the Value property of the corresponding scroll bar, the Wizard will automatically insert the Procedure properties.

The VisibleWidth and VisibleHeight Properties

These properties are not stored in local variables either. They are calculated from within the corresponding Property Get procedures, and they are read-only:

```
Public Property Get VisibleWidth() As Integer
    VisibleWidth = (UserControl.Width - VScroll1.Width) / _
                   Screen.TwipsPerPixelX
End Property

Public Property Get VisibleHeight() As Integer
    VisibleHeight = (UserControl.Height - HScroll1.Height) / _
                    Screen.TwipsPerPixelY
End Property
```

The Picture Property

The custom control's Picture property is mapped to the Picture property of the Picture1 control. The Property procedures for the Picture property are shown next:

```
Public Property Get Picture() As Picture
    Set Picture = Picture1.Picture
End Property

Public Property Set Picture(ByVal New_Picture As Picture)
    Set Picture1.Picture = New_Picture
```

```
    ArrangeControl
    UserControl_Paint
End Property
```

When a new picture is assigned to the control's Picture property, the ArrangeControl subroutine is called, which arranges the various controls on the UserControl object. It then calls the SetScrollbars function, which sets the Max properties of the two scroll bars. Here's the ArrangeControl subroutine:

```
Private Sub ArrangeControl()
    Picture2.Top = 0
    Picture2.Left = 0
    Picture2.Width = UserControl.Width - VScroll1.Width
    Picture2.Height = UserControl.Height - HScroll1.Height
    HScroll1.Left = 0
    HScroll1.Width = UserControl.Width - VScroll1.Width
    HScroll1.Top = UserControl.Height - HScroll1.Height
    VScroll1.Left = UserControl.Width - VScroll1.Width
    VScroll1.Top = 0
    VScroll1.Height = UserControl.Height - HScroll1.Height
    SetScrollbars
End Sub
```

After sizing and positioning the scroll bars' controls on the Form, the ArrangeControl subroutine calls the SetScrollbars subroutine, which sets up the properties of the two scroll bars (sets their Max properties according to the size of the control's visible area and the image size, disables them if required, and so on). The SetScrollbars subroutine is shown next:

```
Private Sub SetScrollbars()
    m_PictureWidth = ScaleX(Picture1.Picture.Width, 8, vbPixels)
    m_PictureHeight = ScaleY(Picture1.Picture.Height, 8, vbPixels)
    HScroll1.Enabled = True
    VScroll1.Enabled = True
    HScroll1.Visible = True
    VScroll1.Visible = True
' should we disable horizontal scrollbar?
    If m_PictureWidth <= Picture2.ScaleWidth Then
        Picture2.Width = m_PictureWidth * Screen.TwipsPerPixelX
        HScroll1.Enabled = False
        HScroll1.Width = Picture2.Width
        HScroll1.Visible = False
        VScroll1.Left = Picture2.Left + Picture2.Width
    End If
' should we disable vertical scrollbar?
    If m_PictureHeight <= Picture2.ScaleHeight Then
        Picture2.Height = m_PictureHeight * Screen.TwipsPerPixelY
```

```
            VScroll1.Enabled = False
            VScroll1.Height = Picture2.Height
            VScroll1.Visible = False
            HScroll1.Top = Picture2.Top + Picture2.Height
        End If
        HScroll1.Min = 0
        HScroll1.Max = m_PictureWidth - Picture2.ScaleWidth
        HScroll1.LargeChange = 10
        HScroll1.SmallChange = 1
        VScroll1.Min = 0
        VScroll1.Max = m_PictureHeight - Picture2.ScaleHeight
        VScroll1.LargeChange = 10
        VScroll1.SmallChange = 1
    End Sub
```

The *m_PictureWidth* and *m_PictureHeight* arguments are two local variables that store the actual dimensions of the bitmap displayed on the custom control. The expressions Picture1.Picture.Width and Picture1.Picture.Height return the dimensions of the bitmap displayed on the Picture1 control in HiMetric units. The ScaleX and ScaleY methods convert the dimensions of the bitmap from HiMetric units to pixels. These values won't change in the course of the control's life unless a new image is loaded.

The two If clauses disable the scroll bar(s) if the image is smaller than the custom control's visible area. This control was meant to be used with large images and hasn't been optimized for small images. Even if the entire image can fit in the control's viewing area, the scroll bars will be disabled, but they'll remain visible. The last few statements in the subroutine set the properties of the two scroll bars. When either scroll bar is set to its maximum value, the rightmost and bottom part of the image should appear on the control.

After the controls have been arranged on the UserControl object and the ScrollBar controls have been set up, the Picture Property Set Procedure's code calls the UserControl_Paint event. This is where the we copy part of the image from the invisible onto the visible PictureBox control:

```
Private Sub UserControl_Paint()
On Error Resume Next
    Picture2.PaintPicture Picture1.Picture, 0, 0, _
    Picture2.Width, Picture2.Height, _
    HScroll1.Value, VScroll1.Value, _
    Picture2.Width, Picture2.Height
End Sub
```

The PaintPicture method copies a bitmap from a PictureBox control or Form object to another and is described in detail in Chapter 6. The code presented so

far takes care of the placement of the image on the control. To scroll the visible part of the image every time the user changes the scroll bars, we must simply invoke the UserControl_Paint event from within the Change and Scroll events of the two ScrollBar controls. In addition, we must also raise the ChangeX, ScrollX and the ChangeY, ScrollY events. Here are the handlers of the Change and Scroll events of the horizontal scroll bar:

```
Private Sub HScroll1_Change()
    UserControl_Paint
    RaiseEvent ChangeX(HScroll1.Value)
End Sub

Private Sub HScroll1_Scroll()
    UserControl_Paint
    RaiseEvent ScrollX(HScroll1.Value)
End Sub
```

The ScrollTo method scrolls the bitmap to the specified position by manipulating the Value property of the two scroll bars. The two If structures in the following listing make sure that the bitmap isn't scrolled to an invalid position.

```
Public Sub ScrollTo(X As Integer, Y As Integer)
    If HScroll1.Enabled Then
        If X > HScroll1.Max Then X = HScroll1.Max
        HScroll1.Value = X
    End If
    If VScroll1.Enabled Then
        If Y > VScroll1.Max Then Y = VScroll1.Max
        VScroll1.Value = Y
    End If
End Sub
```

The rest of the code was generated by the Wizard. It's straightforward, and I need not repeat it here. You can open the ScrollingImage project in the Visual Basic IDE and examine the Property procedures and the methods of the custom control. Notice that most standard members are mapped to the equivalent members of the UserControl object and the visible PictureBox (Picture2).

Downloading Images over the Internet

In order to use the ScrollImgBox control on a Web page (or even in a desktop application that can contact a Web server and download images), we must add a custom property, which is similar to the Picture property, but it can download a bitmap from a Web server. Let's call this property PictureFromURL. This property need not be set at design time. Instead, the host application (a VB application or a Web page)

can set this property at runtime. When the PictureFromURL property is set, the control will download the specified image from a Web server and display it. All other properties and methods remain the same.

When this control is used on a Web page, there's no use for the Picture property. The page to which it belongs is going to be viewed on a client computer, and the image must be downloaded anyway (the page can't expect that the picture will be stored in the host computer's disk). The PictureFromURL property is implemented with the following Property procedures:

```
Public Property Get PictureFromURL() As String
    PictureFromURL = m_PictureFromURL
End Property

Public Property Let PictureFromURL(ByVal newValue As String)
    m_PictureFromURL = newValue
    If Ambient.UserMode And newValue <> "" Then
        AsyncRead newValue, vbAsyncTypePicture, "PictureFromURL"
    End If
End Property
```

The local variable *m_PictureFromURL* is a string variable that stores the URL of the image to be downloaded. The value of this property is passed to the AsyncRead method, which then downloads it. The PictureFromURL property can be assigned a value such as a file name on the local disk:

```
File://c:/Images/NYork.GIF
```

or a file's URL on the Web:

```
http://www.sybex.com/Images/NYork.gif
```

The UserMode property of the UserControl object is True at runtime only, and the control will not attempt to download the image at design time.

To download the image, the code calls the AsyncRead method specifying the *vbAsyncTypePicture* constant as the AsyncType argument. The specified resource will be downloaded asynchronously and will be stored in a local file on the host computer. When the download completes, the AsyncReadComplete event is raised. In this event's handler, we must display the image on the control. As with the Picture property, we are going to call the ArrangeControl subroutine to arrange the controls on the UserControl object and then invoke the UserControl's Paint event to actually display the visible portion of the image. Here's the AsyncReadComplete event's handler:

```
Private Sub UserControl_AsyncReadComplete(AsyncProp As AsyncProperty)
On Error GoTo NoImage
    If AsyncProp.PropertyName = "PictureFromURL" Then
        Set Picture1.Picture = AsyncProp.Value
```

```
            ArrangeControl
            UserControl_Paint
            RaiseEvent DownloadCompleted
        End If
        Exit Sub

NoImage:
    RaiseEvent DownloadFailed
    Debug.Print "Download failed"
End Sub
```

The Value property of the AsyncProp object is the name of a file (a string) in which the downloaded bitmap was stored. This value can be assigned directly to the Picture property of the Picture1 control. The actual file where the bitmap is stored is of no interest to the developer. It's a temporary file, which is located in the Temporary Internet Files folder, and will eventually be removed.

Testing the PictureFromURL Property

The PictureFromURL property can be used with Web pages, as well as desktop applications, as long as the user has established a connection to the Internet. The VideoForm test Form of the ScrolImg project (shown in Figure 4, earlier in this tutorial) downloads an image from a Web server when the Load Image button is clicked. If you have the Personal Web server installed on your computer, place the image to be downloaded in the root folder of the Web server and enter the following lines in the Load Image button's Click event handler:

```
Private Sub Command2_Click()
        ScrollImgBox1.PictureFromURL = "http://127.0.0.1/nysky.gif"
End Sub
```

The image is expected to reside in the server's root folder. This test Form will also work if you are using Internet Information Server. If you are not running a Web server on your computer, you can post the image on your ISP's server. Most ISPs provide a folder on their server for each subscriber, and you can post your files there.

NOTE

The process of posting files to an ISP server is described in Chapter 21 of *Mastering Visual Basic 6*. For those of you who might be reading this tutorial on the Sybex Web site and who don't have the book handy, I've included a summary of the process at the end of this tutorial. If you're going to post the image on an ISP's server, you must change the URL of the image in the previous code segment accordingly.

The code behind the Scroll Image button is quite simple. It moves the image around by calling its ScrollTo method repeatedly. Here's the code that scrolls the image around:

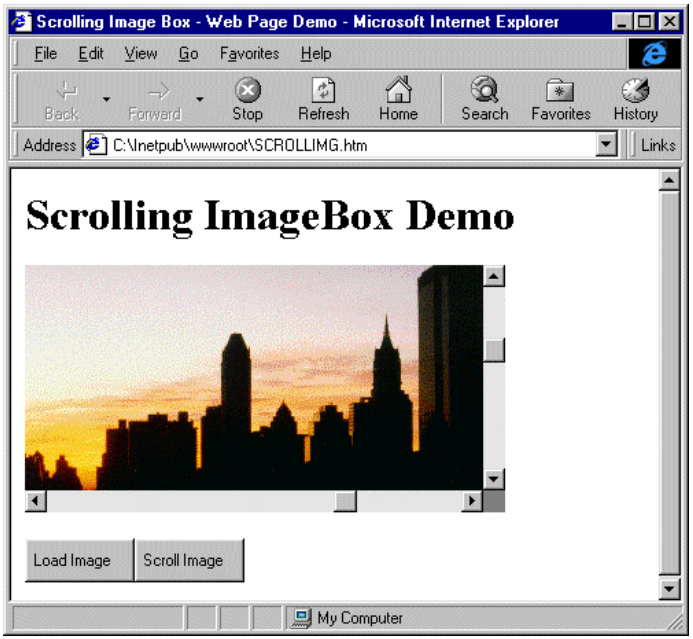
```
Private Sub Command1_Click()
Dim i As Integer
Dim j As Integer
Dim VWidth As Integer

    VWidth = ScrollImgBox1.PictureWidth - ScrollImgBox1.VisibleWidth
    For i = 0 To 90
        ScrollImgBox1.ScrollTo 0, i
        DoEvents
    Next
    For i = 0 To VWidth
        ScrollImgBox1.ScrollTo i, 90
        DoEvents
    Next
    For i = 90 To 40 Step -1
        ScrollImgBox1.ScrollTo VWidth, i
        DoEvents
    Next
    For i = VWidth To 0 Step -1
        ScrollImgBox1.ScrollTo i, 40
        DoEvents
    Next
End Sub
```

Using the ScrollImgBox on a Web Page

To test the custom control's ability to download images over the Web, we are going to build a Web page that uses the ScrollImgBox control. After the page has been loaded on the client computer, it downloads the NYSKY.GIF image file from the Web server and scrolls it, just like the VIDEOFORM test Form. The page shown in Figure 5 was created with FrontPage Express, a WYISYG HTML editor that comes with Internet Explorer 4 (full installation). It's just as easy to create the HTML page with a simple editor such as Notepad, but FrontPage Express can locate the Class ID of the various ActiveX controls in the Registry and create the appropriate <OBJECT> tags for you.

FIGURE 5:

The SCROLLIMG.HTM page



To insert an instance of an ActiveX control on a Web page, follow these steps:

1. Start Visual Basic and open the project ScrlImg (in the AXImage folder of the CD).
2. Select the UserControl object in the Project Explorer window.
3. Choose File ➢ Make ScrollImageBox.ocx.
4. After the OCX file of the control is created, open the DOS window, switch to the folder that contains the control, and enter the following command:

   ```
   C:\WINDOW\SYSTEM\REGSVR32 SCROLLIMGBOX.OCX
   ```

   This command will register the Scrolling ImageBox control with your system, and any application can use it.
5. Switch to FrontPage Express and start a new page.
6. Choose Insert ➢ Other Components ➢ ActiveX Control, and in the list of ActiveX controls, select ScrollingImage.ScrolImgBox. (This item will appear only if you haven't changed the names of the project and the UserControl

object. If you did, select the ActiveX control by its new name). In the ActiveX Control Properties window you can also set the dimensions of the control.

7. Place the two Command Buttons below the ScrollImgBox control by selecting the Microsoft Forms 2.0 Command Button in the list of ActiveX controls.

FrontPage Express will generate the HTML code for the document. All you really need are the definitions of the ActiveX controls. Choose View ➢ HTML to view the source code. It's not easy to read because FrontPage doesn't use uppercase for the HTML tags. Here's the listing of the SCRLIMG.HTM Web page after some beautification and the addition of the SCRIPT section:

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type"
content="text/html; charset=iso-8859-1">
<meta name="GENERATOR" content="Microsoft FrontPage Express 2.0">
<TITLE>Scrolling Image Box - Web Page Demo</TITLE>
<SCRIPT LANGUAGE=VBScript>
Sub LoadButton_Click()
    ScrollImageBox.PictureFromURL="nysky.gif"
End Sub
Sub ScrollButton_Click()
    Width=ScrollImageBox.PictureWidth-ScrollImageBox.VisibleWidth
    For i=1 to 70
        ScrollImageBox.ScrollTo CInt(0), CInt(i)
        ScrollImageBox.Refresh
    Next
    For i=0 to Width
        ScrollImageBox.ScrollTo CInt(i), 70
        ScrollImageBox.Refresh
    Next
    For i=70 to 20 Step -1
        ScrollImageBox.ScrollTo CInt(Width), CInt(i)
        ScrollImageBox.Refresh
    Next
    For i=Width to 0 Step -1
        ScrollImageBox.ScrollTo CInt(i), 20
        ScrollImageBox.Refresh
    Next
End Sub
</SCRIPT>
```

```
</HEAD>
<BODY bgcolor="#FFFFFF">
<H1>Scrplling ImageBox Demo</H1>
<OBJECT CLASSID="clsid:9C6BC6AF-0A8B-11D2-94D1-0000E8CE04E8"
NAME="ScrollImageBox" BORDER="0" WIDTH="350" HEIGHT="180">
</OBJECT>
<P>
<OBJECT ID="LoadButton" classid="clsid:D7053240-CE69-11CD-A777-
00DD01143C57"
BORDER="0" WIDTH="80" HEIGHT="32">
    <PARAM NAME="Caption" VALUE="Load Image">
</OBJECT>
<OBJECT ID="ScrollButton" CLASSID="clsid:D7053240-CE69-11CD-A777-
00DD01143C57"
BORDER="0" WIDTH="80" HEIGHT="32">
    <PARAM NAME="Caption" VALUE="Scroll Image">
</OBJECT>
</BODY>
</HTML>
```

The CLASSID for the AXImage control on your computer will be different. *Use the SCROLLIMG.HTM page as a sample, but it will not work as is.* You must change the CLASSID attribute's value and then open it with Internet Explorer.

The script is similar to the Visual Basic code we used in the VideoForm test Form to scroll the image around. The SCROLLIMG.HTM page expects that the image to be downloaded (NYSKY.GIF) resides in the root folder of the Web server. To open the page, start Internet Explorer and enter the following URL in the Address box:

```
HTTP://127.0.0.1/SCROLLIMG.HTM
```

The SCROLLIMG.HTM page will work as expected when opened and viewed on a computer on which the ScrollImgBox control has been registered. If you attempt to access this file on your server from another computer, you won't see the custom control on the page because the control hasn't been registered on the host computer. Use the same computer on which the custom control was developed to test the SCROLLIMG.HTM page.

In the last section of this tutorial, you'll find information on posting the images to be downloaded on your ISP's server. The information in the next section applies to most ISPs, but you may have to contact your ISP for more information on posting to user sites.

Safety and Security

The default security settings of Internet Explorer will prevent the control from initializing and executing on the computer. This control is not safe for scripting, and Internet Explorer will ignore it. (To make a control safe for scripting, use the Package and Deployment Wizard that comes with Visual Basic 6.)

To view the SCROLLIMG page, follow these steps:

1. In Internet Explorer, choose View ➢ Internet Options to open the Internet Options dialog box.
2. Select the Security tab, and click the Low option button.
3. Close the dialog box, connect to the server, and open the page.

This time the control will load. When you click the Load Image button, you'll get a warning. Internet Explorer will tell you that the AcitveX control you're about to view may be unsafe. Click Yes anyway (you've seen the control's code, and it doesn't do anything to your file system). After the image downloads, click the Scroll Image button to scroll the image in its box.

Posting a File on a Web Server

Testing a control that downloads property values from a Web server requires that you connect to the server. If you have your own Web server (Internet Information Server or the Personal Web Server), you can just copy an image file to one of the Web server's virtual directories. You can just copy it to the root directory and then connect to your own server using the following URL:

```
http://127.0.0.1/NYSky.gif
```

The address 127.0.0.1 is the address of your own computer on the network. Actually, you don't even need a network. As long as you have a Web server installed on your computer, use this URL to connect to it. If the Web server is on another computer in the same network, substitute the actual computer name for the string "127.0.0.1".

Most of you, however, don't have you own Web server to test your controls. You can still use your Internet service provider's server, as long as you have permission to post files on it (most ISPs provide a few Mbytes to subscribers, where they

can post user pages). To test the ScollImgBox control with your ISP's server, you must post the NYSKY.GIF file to your directory on the server. First, create a new folder on your hard disk, and place the NYSKY.GIF file there (if you have any HTML files you would like to post, place them there as well). Now, start the Web Publishing Wizard (a utility that comes with Windows and FrontPage), and follow these steps:

1. The first page of the Web Publishing Wizard is a welcome screen. Click the Next button to open the Select a File or Folder window.
2. In the File or Folder Name box, enter the name of the folder you just created. Click Next.
3. In the next window, specify a friendly name for the Web server, and click the Advanced button.
4. In the next window, select the protocol to be used for transferring the file. Select FTP and click Next.
5. In the next window, you specify the URL of the service provider. You must provide the name of the server on which the ISP places user pages (don't change the name of the local directory). For most ISPs, this URL is something like:

   ```
   users.ISPname.com/yourname
   ```

   The *yourname* variable is your login name. If your address is `jdoe@usa1.com`, your login name is jdoe, and the URL of your directory on the ISP's server would be:

   ```
   users.usa1.com/jdoe
   ```

TIP

Many ISPs use the tilde character in front of the user name (`users.usa1.com/~jdoe`). If you search for "user pages" in your ISP's Web site, you'll find all the information you need.

6. Click Next again to open the last window of the Wizard, and click on Finish to publish your files.
7. To test your directory on the server, start Internet Explorer and enter the URL of the Rates.txt file on the server:

   ```
   http://users.usa1.com/jdoe/nysky.gif
   ```

   You should see the contents of the Rates.txt file in the browser's window. This means that you have successfully posted the file, and your Rates control will also be able to see it on the server and download it.

8. Copy the URL from the browser's Address box and paste it in your code (pass it as argument to the DownloadRates method of the Rates control).

The process I describe here will work with many ISPs, but the exact steps may not apply to all ISPs. If you have problems posting your files to the server, just find out from your ISP what it takes to publish your own user or personal Web site. Instead of an entire Web site, just place the Rates.txt there. Or, if you already have a Web site, place the Rates.txt file in the same directory. It's not going to affect your Web, since none of the pages contain hyperlinks to this file.