TUTORIAL

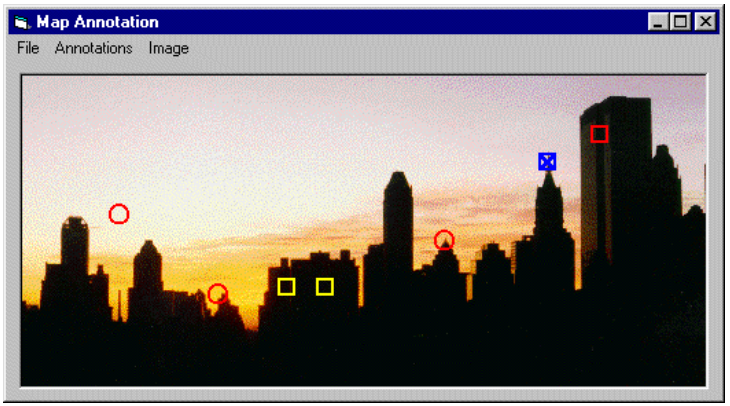
An Image Annotation Application

This tutorial puts together many of the graphics methods and techniques we discussed in Chapters 6 and 7 of the Mastering VB6 book. Annotate is an image annotation application that works best with maps.

The Annotate application, shown in Figure 1, lets you load an image and place annotation marks on it. Annotate is not a "draw-over-image" application; it can't draw lines and other shapes over the image. Instead, it allows you to place special marks on an image, drag them around to different locations, change their shape and color, add comments to each of them, and finally, delete unwanted marks. The application was developed for annotating maps with simple marks to indicate wells and other installations. For the examples in this tutorial, I'm using a simple image of the New York City skyline.

FIGURE 1:

The Annotate application's main Form



Using the Annotate Application

The Annotate application operates on images that must be opened with the File ➢ New command. The annotations are stored in a different file, and they have the extension ANT. The ANT file contains a reference to the image file (which can reside in any folder on the hard drive and can be of any type Visual Basic can handle, e.g., GIF, BMP, JPEG) and the information about the annotation marks (shape, color, and comments).

Open the File menu, select New, and you'll be prompted for the name of an image file. Once the image is loaded, you can annotate it with the commands of the Annotations menu. The Annotations menu has the following structure:

Color Sets the color of the selected symbol; new marks will also be drawn in this color.

Blue

Red

Green

Yellow

White

Black

Shape Sets the shape of the selected mark; new marks will also have the same shape.

Circle

Dot

Box

XBox

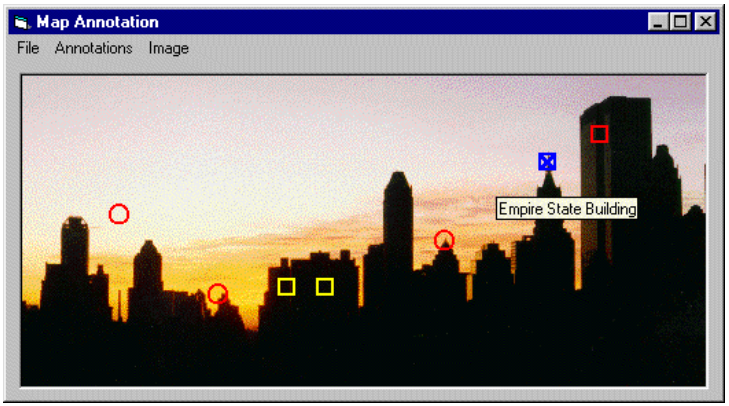
Comments Assigns a string of text to the selected mark; the comments are displayed in a ToolTip box when the pointer rests over the mark for a second or so.

Delete This command deletes the selected mark.

To place a new annotation mark on the bitmap, select its shape and color from the Annotations menu, then right-click the image to add the annotation mark. To move it to another location, drag it with the mouse and drop it at the new location. To change its characteristics, right-click the annotation mark and the Annotations menu will pop up. On this menu, you can select a new color or shape for the mark, delete the mark, or add comments to it. When you select the Comments command, a small Form will appear on which you can enter your comments. To see the comments, you can either right-click a mark and select Comments or rest the pointer over the mark for a second or so, and the comments will appear in a ToolTip box, as shown in Figure 2.

FIGURE 2:

A simple image with different types of annotation marks and the comments entered for one of the marks in a ToolTip box



To save an image and its annotations, select the File ➢ Save As command and the application will prompt you to select a filename for the annotations. You can easily add a plain Save command, which stores the annotations in the currently open file. The annotation files have the extension ANT, and they store the name of the image that was annotated along with the information about each mark (location, shape, and color) and the comments. For more information on the structure of ANT files see the section "Saving the Annotations," at the end of this tutorial.

Coding the Annotate Application

If you use the Annotate application for a while, you'll understand that it doesn't draw directly on the PictureBox control on which the annotation marks are placed. If you drew the annotation marks directly on the bitmap, you wouldn't be able to move them around. (Actually, you'd have to perform quite a number of calculations from within the MouseDown event to figure out which mark was clicked, then drag it around.)

To simplify the logic of the application and make sure that annotation marks blend with their backgrounds, I used a little trick. I placed the annotation marks on small PictureBox controls without borders. Keeping the marks on separate, individual controls allows the code to quickly detect which mark is clicked and then drag it around following the movement of the mouse.

The application's main Form contains a PictureBox that has its AutoSize property set to True. Each time a new bitmap is loaded on the large PictureBox, the Form is resized accordingly. If the images you want to annotate are too large to fit on your

monitor, you should make the main Form scrollable using the techniques discussed in the section "Implementing Scrolling Forms" of Chapter 10.

You must also place a borderless PictureBox on the large PictureBox control. The small PictureBox is called *Bullet* and it has an Index value of 0. As you have guessed, each time a new annotation mark is placed on the bitmap, a new member will be added to the *Bullet* control array. The Bullet PictureBox's dimensions are 16 × 16 pixels and its ScaleMode property must be set to 3 (Pixels). Its Visible property must also be set to False so that the first member of the array will remain invisible.

Creating New Annotation Marks

New annotation marks are created when the user right-clicks the Picture1 control. When the right mouse button is released over the large PictureBox control, the following code is executed:

```
Private Sub Picture1_MouseUp(Button As Integer, Shift As Integer, _
                X As Single, Y As Single)
    If MAPNAME = "" Then Exit Sub
    If Button = 2 Then
        maxMarks = maxMarks + 1
        Load Bullet(maxMarks)
        Bullet(maxMarks).Move X - Bullet(maxMarks).ScaleWidth / 2, _
                              Y - Bullet(maxMarks).ScaleHeight / 2
        Bullet(maxMarks).Visible = True
        Bullet(maxMarks).ToolTipText = ""
        Bullet(maxMarks).Tag = MARKSHAPE
        DrawBullet maxMarks, MARKCOLOR, MARKSHAPE
    End If
End Sub
```

The Load statement creates a new member in the *Bullet* array and increases the number of annotation marks by one (variable *maxMarks*). The Tag property of the *Bullet* control array is used to store the name of the shape, and the new control's ToolTipText property is reset to an empty string. The DrawBullet() subroutine draws the annotation shape on the corresponding Bullet control. This is a very interesting subroutine because it must draw a shape transparently over the bitmap.

Drawing Transparent Shapes

Drawing a transparent shape directly over the bitmap isn't challenging, except for the fact that the shape must be drawn on another PictureBox control. To create the impression that the shape is drawn over the bitmap, the code must copy the underlying bitmap pixels to the *Bullet* PictureBox control (with the PaintPicture method)

and then draw on top of this bitmap. If you want to add new shapes or replace the existing ones, this is where you should insert the appropriate code.

```
Function DrawBullet(BulletIndex As Integer, FRCOLOR As Long, _
                    FRSHAPE As String)
    Bullet(BulletIndex).PaintPicture Picture1.Image, _
       0, 0, Bullet(BulletIndex).ScaleWidth, _
       Bullet(BulletIndex).ScaleHeight, _
       Bullet(BulletIndex).Left, Bullet(BulletIndex).Top, _
       Bullet(BulletIndex).ScaleWidth, _
       Bullet(BulletIndex).ScaleHeight, 13369376
    Bullet(BulletIndex).FillColor = FRCOLOR
    Bullet(BulletIndex).ForeColor = FRCOLOR
    Bullet(BulletIndex).Tag = FRSHAPE
    If FRSHAPE = "CIRCLE" Then
        X = Bullet(BulletIndex).ScaleWidth / 2
        Y = Bullet(BulletIndex).ScaleHeight / 2
        Bullet(BulletIndex).Circle (X, Y), X * 0.9
    ElseIf FRSHAPE = "DOT" Then
        X = Bullet(BulletIndex).ScaleWidth / 2
        Y = Bullet(BulletIndex).ScaleHeight / 2
        Bullet(BulletIndex).FillStyle = 0
        Bullet(BulletIndex).Circle (X, Y), X * 0.9
        Bullet(BulletIndex).FillStyle = 1
    ElseIf FRSHAPE = "BOX" Or FRSHAPE = "XBOX" Then
        X = Bullet(BulletIndex).ScaleWidth
        Y = Bullet(BulletIndex).ScaleHeight
        Bullet(BulletIndex).DrawWidth = 2
        Bullet(BulletIndex).Line (2, 2)-(X - 2, Y - 2), , B
        If FRSHAPE = "XBOX" Then
            Bullet(BulletIndex).Line (2, 2)-(X - 2, Y - 2)
            Bullet(BulletIndex).Line (2, Y - 2)-(X - 2, 2)
        End If
    Else
'       MsgBox "Application Error: Unknown shape requested!"
        Exit Function
    End If
    Picture1.Refresh
End Function
```

FRSHAPE and *FRCOLOR* arguments specify the annotation mark's shape and color. The application keeps track of the current mark's shape and color with the *MARKSHAPE* and *MARKCOLOR* Form variables. Every time the user selects a new shape and/or color from the Annotations menu, these two variables are set accordingly.

For example, when the Annotations ➢ Color ➢ Blue command is selected, the following code is executed:

```
Private Sub BlueDot_Click()
    MARKCOLOR = RGB(0, 0, 255)
    If selDotIndex = -1 Then
        Exit Sub
    Else
        DrawBullet selDotIndex, MARKCOLOR, MARKSHAPE
    End If
    ClearColorChecks
    BlueDot.Checked = True
End Sub
```

The program sets the *MARKCOLOR* variable to a blue color so that subsequent annotation marks are drawn in this color. If a mark is selected at the time (variable *selDotIndex* is not -1), then it's redrawn in the newly selected color with a call to the DrawBullet() subroutine.

Likewise, when a new shape is selected, the following code is executed:

```
Private Sub ShapeBox_Click()
    MARKSHAPE = "BOX"
    If selDotIndex = -1 Then
        Exit Sub
    Else
        DrawBullet selDotIndex, MARKCOLOR, MARKSHAPE
    End If
    ClearShapeChecks
    ShapeBox.Checked = True
End Sub
```

Dragging the Annotation Marks

When the *Bullet* PictureBox is clicked upon, the code must prepare it for a drag operation. This is taken care by the following lines in the control's MouseDown event:

```
Private Sub Bullet_MouseDown(Index As Integer, Button As Integer,_
               Shift As Integer, X As Single, Y As Single)
    If Button = vbLeftButton Then
        Dragging = True
        XStart = X
        YStart = Y
        Bullet(Index).ZOrder 0
    End If
End Sub
```

The *Dragging* variable is declared as Boolean in the Form's declaration section so that it will be available to all mouse-related events. While the *Bullet* PictureBox is being dragged around, the following code is executed from within the control's MouseMove event:

```
Private Sub Bullet_MouseMove(Index As Integer, Button As Integer, _
                Shift As Integer, X As Single, Y As Single)
    If Not Dragging Or Button <> vbLeftButton Then Exit Sub
    Bullet(Index).Visible = False
    Bullet(Index).Move Bullet(Index).Left + (X - XStart), _
                       Bullet(Index).Top + (Y - YStart)
    Bullet(Index).Visible = True
    DrawBullet Index, Bullet(Index).FillColor, Bullet(Index).Tag
End Sub
```

The *Index* argument lets your code know which of the *Bullet* PictureBox controls is being dragged. If you're not in the middle of a drag operation or the left mouse button has been released, the code exits the MouseMove subroutine. If a drag operation is in progress, the code moves the control around, following the movement of the mouse. It then calls the DrawBullet() subroutine to draw the annotation mark at its new location.

Finally, in the MouseUp event, the dragging operation ends. If the MouseUp event is triggered by the release of the right mouse button, then the Annotations menu (it's called *AnnotationsMenu*) is displayed:

```
Private Sub Bullet_MouseUp(Index As Integer, Button As Integer, _
               Shift As Integer, X As Single, Y As Single)
    If Button = vbRightButton Then
        selDotIndex = Index
        MARKSHAPE = Bullet(selDotIndex).Tag
        PopupMenu AnnotationsMenu
    ElseIf Button = vbLeftButton Then
        Dragging = False
    End If
End Sub
```

TIP

The code would be almost trivial if I had used the DragMode property of the Bullet control and the DragDrop event of the large PictureBox. This technique would simplify the coding of the drag operation, but the marks wouldn't be transparent during dragging. By implementing the drag operation in code, I am able to draw the mark on the bitmap myself and make the underlying pixels show through the Bullet control's background.

Saving the Annotations

The code that saves the annotations to a text file is probably the first part of the code you may wish to revise. The program stores all the information to a text file, which means you can't use double quotes in your comments. Also, when comments are displayed in the ToolTip box, multiple text lines appear as long strings with two vertical lines at the points where they should break (text lines don't break when displayed in a ToolTip box). Apart from these two limitations, the code is functional and stores all the information needed by the application to load a bitmap and its annotations from a disk file.

The ANT file starts with the path name of the image being annotated. Then the annotations follow, and they are stored are follows:

<x coordinate> <y coordinate> <color> <shape>

<comments>

These two lines are repeated for every annotation mark on the image. Here's a typical ANT file with three annotation marks:

```
D:\Maps\Samples\Nysky.bmp
 113.5333      13.46667      255          DOT
"WELL #D104
Capacity 100400"
 265.5333      113.4667      65535        DOT
"WELL #F081
Not pumping"
 109           168           255          DOT
"WELL #E330
Capacity 83000"
```

Here's the code that saves the annotations of the current bitmap:

```
Private Sub FileSaveAs_Click()
CommonDialog1.DialogTitle = "Select Project File to Open"
CommonDialog1.DefaultExt = "ANT"
CommonDialog1.Filter = "Application Files|*.ANT"
CommonDialog1.CancelError = True
On Error GoTo FileSaveError
CommonDialog1.ShowSave
fNumMap = FreeFile()
Open CommonDialog1.FileName For Output As #fNumMap
    Print #fNumMap, MAPNAME
    On Error Resume Next
    For i = 1 To maxMarks
        thisDot = thisDot + 1
```

```
            vtest = Bullet(i).Visible
            If Err.Number = 0 Then
                Print #fNumMap, Bullet(i).Left, Bullet(i).Top, _
                      Bullet(i).FillColor, Bullet(i).Tag
                Print #fNumMap, Chr(34) & Notes(i) & Chr(34)
            End If
            Err.Clear
        Next
        Close #fNumMap
        Exit Sub

    FileSaveError:
        MsgBox "Error in saving annotation map!"
        On Error Resume Next
        Close #fNumMap
    End Sub
```

The code starts by writing the path name of the image to the output file. Then, it stores the information of each annotation mark with a For … Next loop that scans all the elements of the control array *Bullet.*

Notice the line that assigns the Visible property of the Bullet(i) control to the *vtest* variable. This is a dummy line that produces a runtime error if there is no such control. The code of the Annotate application unloads controls from the *Bullet* array at will and doesn't renumber the members that follow. This means that the *Bullet* control array may contain "holes," which correspond to the items that are removed from the array during the course of the program. If the loop attempts to save all the elements of the *Bullet* array, the missing elements will cause runtime errors. One way to detect missing items is to access a property. If the element is missing, a runtime error will be generated. Because of the statement `On Error Resume Next`, the error is caught at the following line. The statements that save the information to the file are executed only if no error occurs, which means that the element Bullet(i) exists and corresponds to an annotation mark.

The code that reads the annotation marks from the disk file and redraws the annotated bitmap is shown next:

```
Private Sub FileOpen_Click()
    CommonDialog1.DialogTitle = "Select Map to Annotate"
    CommonDialog1.Filter = "Application Files|*.ANT"
    CommonDialog1.InitDir = App.Path
    CommonDialog1.CancelError = True
On Error GoTo FileOpenError
    CommonDialog1.ShowOpen
    fNum = FreeFile()
    Open CommonDialog1.FileName For Input As #fNum
```

```
    Input #fNum, MAPNAME
    On Error GoTo ImageNotFound
    Picture1.Picture = LoadPicture(MAPNAME)
    Form1.Palette = Picture1.Picture
    Picture1.Picture = LoadPicture(MAPNAME)
    Form1.Width = Picture1.Width + 3 * Picture1.Left
    Form1.Height = Picture1.Height + 60 * Screen.TwipsPerPixelY
    Form1.Refresh
    ClearDots
    On Error Resume Next
    i = 0
    While Not EOF(fNum)
        i = i + 1
        Load Bullet(i)
        Bullet(i).Visible = True
        Input #fNum, BLeft, BTop, BColor, BShape, BNote
        Bullet(i).Left = BLeft
        Bullet(i).Top = BTop
        DrawBullet (i), (BColor), (BShape)
        Notes(i) = BNote
        Bullet(i).ToolTipText = BNote
    Wend
    maxMarks = i
    Close #fNum
    Exit Sub

FileOpenError:
    Exit Sub
ImageNotFound:
   MsgBox "Image file " & MAPNAME & " not found"
   Exit Sub
End Sub
```