Visual Basics

By Barry Seymour

When I first started working in Visual Basic, drag and drop was a drag, so I dropped it.

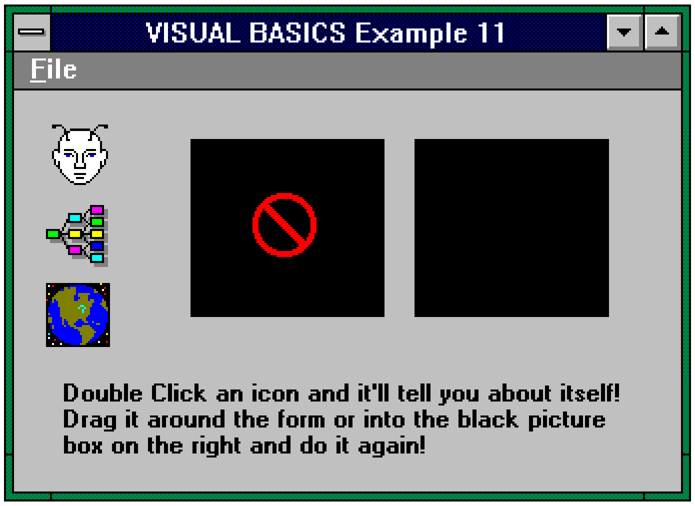
Sorry, I *had* to use that line.

In my opinion the VB manual doesn't cover drag and drop clearly enough, so over the weekend I figured I'd knock together a simple example. After I did I felt like a bit of a con artist. My drag and drop operation *simulated* the dropping of a control into another control by hiding the source control and copying it's picture into the destination control. I wanted to do the *real thing* and put that control *inside* the other.

What I did was use the API call *SetParent* to actually make the dragged control a child of the destination control. When the control is dragged OUT of the destination control, it's made a child of the form again.

The advantages are sublte, but powerful. *All the code for the dragged control can be encapsulated into that control.* Many examples I've seen have code spread out over several controls; with this technique that could be avoided. The non-linearity of Visual Basic can be tough enough to track; encapsulating the code for this control can keep things simpler.

VBEX11.MAK has a control array called SourcePic which consists of three pictures which you can drag around the form. There is another control array of two large pictures (DestPic), one which will allow you to drop a control into it, the other which won't. (I'll leave it to you to guess which is which...)



Double Click an icon and it'll tell you about itself!
Drag it around the form or into the black picture box on the right and do it again!

There are a number of events the system is prepared to respond to: the trick is knowing where they are and how to define them. They are..

1. The user clicks and holds a mouse button on a movable control.
2. The user drags that control over the form, possibly over other controls
3. The user drops the control on a destination control and it responds accordingly.

NOTES TO THE READER: I'm using a new LINE JOIN identifier which isn't found in Visual Basic; this allows a simple global search and replace throughout all sample code. The identifier is **<+>.** Replace all of these with a null string and your code will be OK. I've also given up changing the font to Courier since Helv/Arial is more readable: The code module symbols will also make it easier to see where code begins and ends..

Note also that all source code and instructions for creating this example are placed at the bottom of the form. Just *read* for now!

Let's follow the chronology of a simple drag and drop operation to see what we have to see...

1. The user clicks and holds a mouse button on a movable control. We place code in the Source control to capture this action. Specifically, we put a *Drag* statement in the control's MouseDown event . You want to obtain the handle of the control so you can set it's parent later, and you also want to save to variables the X and Y coordinates of the mouse within the picture, so when the user drops the control later we can drop it in the right place. We'll use the form-level variables MouseWOffset and MouseHOffset variables to do that. The code is like this...

```
Sub SourcePic_MouseDown (Index As Integer, Button As Integer, Shift As
Integer, X As Single, Y As Single)
  DestPic(1).Picture = LoadPicture() ' clear picture
  MouseWOffset = X
  MouseHOffset = Y
  SourcePicHandle = GetFocus()
  SourcePic(Index).Drag
End Sub
```

At this point the user will see an outline of the picture move around the form while he moves the mouse while holding down the button.

2. The user drags the control over the form, possibly over other controls. We use the *DragOver* event from each control to catch this one -- sorry, no encapsulation here! If the user drags the control over a forbidden destination we need to change the MousePointer of the control. In this example, the first member of the DestPic control array has been made a 'forbidden' destination; the second is OK. This is handled in the DestPic_Dragover event.

```
Sub DestPic_DragOver (Index As Integer, Source As Control, X As Single, Y As
Single, State As Integer)
   'if over picture 0 then indicate that a drop isn't allowed.
   If State = 0 And Index = 0 Then
      Source.MousePointer = 12
   Else
      MousePointer = 0' Change pointer to no drop.
   End If
   If State = 1 Then Source.MousePointer = 0  ' Use default mouse pointer.
End Sub
```

The *State* parameter indicates whether the dragged control is *entering* (0) or *leaving* (1) the area bounded by the control. Note that the code always changes the Source.MousePointer back to 0 if the source control is leaving the destination control.

3. The user drops the control on a destination control, which responds accordingly. Once again we look to code in the destination control, specifically the *DragDrop* event.

First we'll look at the response of the destination control DestPic(). We evaluate whether or not we *can* drop a control here; secondly we perform the required action. If the destination is a valid one, use get the handle of the source control to make it a child the destination control using *SetParent()*.

```
Sub DestPic_DragDrop (Index As Integer, Source As Control, X As Single, Y As
Single)
   Source.MousePointer = 0 ' reset mouse pointer
   If Index = 0 Then Exit Sub ' DROP NOT ALLOWED ON #1
   Z% = SetParent(SourcePicHandle, DestPicHandle)
   'note we got DestPicHandle in Form_Load.
   Source.top = Y - MouseHOffset
   Source.Left = X - MouseWOffset
End Sub
```

Notice we've used the saved X and Y offsets of the mousepointer so we can place the control correctly. Without this information, the best we could do is have the control's top left corner snap to the point of the mousepointer, which would be *tres* irritating.

Bear in mind that every control must be ready to respond to a DragDrop event, *including the parent form itself.* Note we set the *form* as the parent -- this handles the circumstance correctly even when the control is being dragged out of a destination control and back onto the form. The same positioning technique also works.

```
Sub Form_DragDrop (Source As Control, X As Single, Y As Single)
  Z% = SetParent(SourcePicHandle, VBEX11.hWnd)
  Source.top = Y - MouseHOffset
  Source.Left = X - MouseWOffset
End Sub
```

I've put a little code into the SourcePic_DblClick event to demonstrate the encapsulation I mentioned earlier. Each control can check to see who it's parent is by using the API call *GetParent()*. Once it does that it can respond to events accordingly. When you paste the following code into your example you'll see what I mean.

I'm now taking a tip from Visual Basic's help file system. To make it easy for your to reproduce this example, create a form named VBEX11 and create the following controls...

SourcePic()	Control array of three picture controls, 0, 1 and 2. (Put whatever images you like into them.
DestPic()	Control array of two picture controls, 0 and 1.

Set AutoRedraw = -1 (TRUE) for all controls and the parent form.

Create a menu with the following elements...

Control Name:	**Caption**
FileMain	&File
FileRepos	&Reposition Pictures
FileExit	E&xit

When you're done, bulk paste the following code into your form. Everything will end up where it's supposed to. (The code has been reduced to point size 6 to eliminate unwanted breaks in lines of code!)

```
Dim MouseHOffset As Integer, MouseWOffset As Integer
Dim PicInBox As Integer

Declare Function SetParent Lib "User" (ByVal hWndChild As Integer, ByVal hWndNewParent As Integer) As Integer
Declare Function GetParent Lib "User" (ByVal hWnd As Integer) As Integer
Declare Function GetFocus Lib "User" () As Integer

Dim DestPicHandle As Integer, SourcePicHandle As Integer

Sub Form_DragDrop (Source As Control, X As Single, Y As Single)
  Z% = SetParent(SourcePicHandle, VBEX11.hWnd)
  Source.top = Y - MouseHOffset
  Source.Left = X - MouseWOffset
End Sub

Sub SourcePic_MouseDown (Index As Integer, Button As Integer, Shift As Integer, X As Single, Y As Single)
  DestPic(1).Picture = LoadPicture() ' clear picture
  MouseWOffset = X
  MouseHOffset = Y
  SourcePicHandle = GetFocus()
  SourcePic(Index).Drag
End Sub

Sub DestPic_DragOver (Index As Integer, Source As Control, X As Single, Y As Single, State As Integer)
  'if over picture 0 then indicate that a drop isn't allowed.
  If State = 0 And Index = 0 Then
                Source.MousePointer = 12
  Else
                MousePointer = 0' Change pointer to no drop.
  End If
  If State = 1 Then Source.MousePointer = 0  ' Use default mouse pointer.
End Sub

Sub DestPic_DragDrop (Index As Integer, Source As Control, X As Single, Y As Single)
  Source.MousePointer = 0 ' reset mouse pointer
  If Index = 0 Then Exit Sub ' DROP NOT ALLOWED ON #1
  Z% = SetParent(SourcePicHandle, DestPicHandle)
  'note we got DestPicHandle in Form_Load.
  Source.top = Y - MouseHOffset
  Source.Left = X - MouseWOffset
End Sub

Sub FileRepos_Click ()
  DestPic(1).Picture = LoadPicture()
  For X% = 0 To 2
                SourcePic(X%).SetFocus
                PicHandle% = GetFocus()
                Z% = SetParent(PicHandle%, VBEX11.hWnd)
                SourcePic(X%).Left = 240
                SourcePic(X%).top = 240 + (600 * X%)
                SourcePic(X%).Visible = -1 'TRUE
  Next X%
End Sub

Sub FileExit_Click ()
  End
End Sub

Sub Form_Load ()
  VBEX11.Show
  DestPic(1).SetFocus
  DestPicHandle = GetFocus()
End Sub

Sub SourcePic_DblClick (Index As Integer)
  ThisHandle% = GetFocus()
  ParentHandle% = GetParent(ThisHandle%)

  Msg$ = "Hi! I'm Source Picture " + LTrim$(RTrim$(Str$(Index))) + ", and "
  Msg$ = Msg$ + "my parent is "
  If ParentHandle% = VBEX11.hWnd Then
                Msg$ = Msg$ + "the form VBEX11!  "
  Else
                Msg$ = Msg$ + "the destination picture!  "
  End If
  Msg$ = Msg$ + "Cool, huh?"

  MsgBox Msg$, 64
  MsgBox "Party on, Garth."
  MsgBox "Party on, Wayne."

End Sub
```

As always, this column plus sample code is available on the Windows Online BBS in Danville, California, phone 1 510 736-8343. This column and source code is in VBEX11.ZIP, and may be distributed as freeware.

Barry Seymour
Marquette Computer Consultants
San Rafael, CA 415/459-0835
for **Windows Online News**