# Implementing Cut, Copy and Paste Operations in your Visual Basic 6 Program

One of the things I often tell my students and readers when they are developing menus for their own Visual Basic applications is the importance of adhering to Windows standards when creating and naming your menu items. Inevitably, one of the questions that is then asked is whether Visual Basic can be used to implement the Cut, Copy and Paste menu operations that just about every Windows program contains. When I answer 'yes', the follow-up question usually is whether there is some kind of automatic method for implementing this feature. Unfortunately, the answer to that question is no---there's no automatic method to provide this functionality, but implementing these features in your own programs is not that difficult.

## The Windows Clipboard Object

The key to implementing Cut, Copy and Paste features in your program is a Visual Basic Object called the Clipboard Object. Those of you who have used Visual Basic for any length of time are probably familiar with other kinds of Visual Basic Objects, such as the Debug Object and the Printer Object.

Actually, it's a misnomer to call the Clipboard Object a Visual Basic Object---it's really a Windows Object. The Clipboard is one of the great features of the Windows Operating System, in that Windows application programs can place data on the Clipboard (text and graphics data for the most part), and this data can then be accessed by another Windows program.

As an example, I'm sure you know that you can select and then copy text from Microsoft Word to the Windows Clipboard, then within Excel, paste that text into a Worksheet cell. It's important to note here that there is only one Clipboard Object in Windows---all Windows programs share that same single Clipboard object. When you implement Cut, Copy and Paste features in your Visual Basic programs, you'll be using the same single Windows Clipboard object to do so. What that means is that if you copy text to the Clipboard from within Word, and then copy text from a TextBox in your Visual Basic program to the Clipboard, the text from the TextBox will overlay the text from Word.

One thing that frequently confuses beginners is the fact that the Windows Clipboard is capable of having more than one type of data on it at the same time. For instance, in the example I just gave where the Visual Basic text overlaid the Word text on the clipboard, if I then copy a graphic in the form of bitmap data from Microsoft Paint to the Clipboard, both the Visual Basic text and the bitmap can co-exist on the Clipboard at the same time. The rule is that you can't have two instances of the same data type on the Clipboard at the same time.

What types of data can be placed on the Clipboard? If you check Visual Basic Help, you'll find that the Clipboard can hold up to nine different types of data, although I'll only be discussing two---text data and bitmap data--in this article.

It's possible to interrogate the Clipboard to determine the type of data that it currently contains. Here's some code taken right out of Visual Basic help that, when placed in the Click Event procedure of a Form, displays a Msgbox telling you the type of data that the Clipboard contains.

```
Private Sub Form_Click ()

' Define bitmap formats.

Dim ClpFmt, Msg ' Declare variables.

On Error Resume Next ' Set up error handling.

If Clipboard.GetFormat(vbCFText) Then ClpFmt = ClpFmt + 1
If Clipboard.GetFormat(vbCFBitmap) Then ClpFmt = ClpFmt + 2
If Clipboard.GetFormat(vbCFDIB) Then ClpFmt = ClpFmt + 4
If Clipboard.GetFormat(vbCFRTF) Then ClpFmt = ClpFmt + 8

Select Case ClpFmt
  Case 1
    Msg = "The Clipboard contains only text."
  Case 2, 4, 6
    Msg = "The Clipboard contains only a bitmap."
  Case 3, 5, 7
    Msg = "The Clipboard contains text and a bitmap.
  Case 8, 9
    Msg = "The Clipboard contains only rich text."
  Case Else
    Msg = "There is nothing on the Clipboard."
End Select

MsgBox Msg ' Display message.

End Sub
```

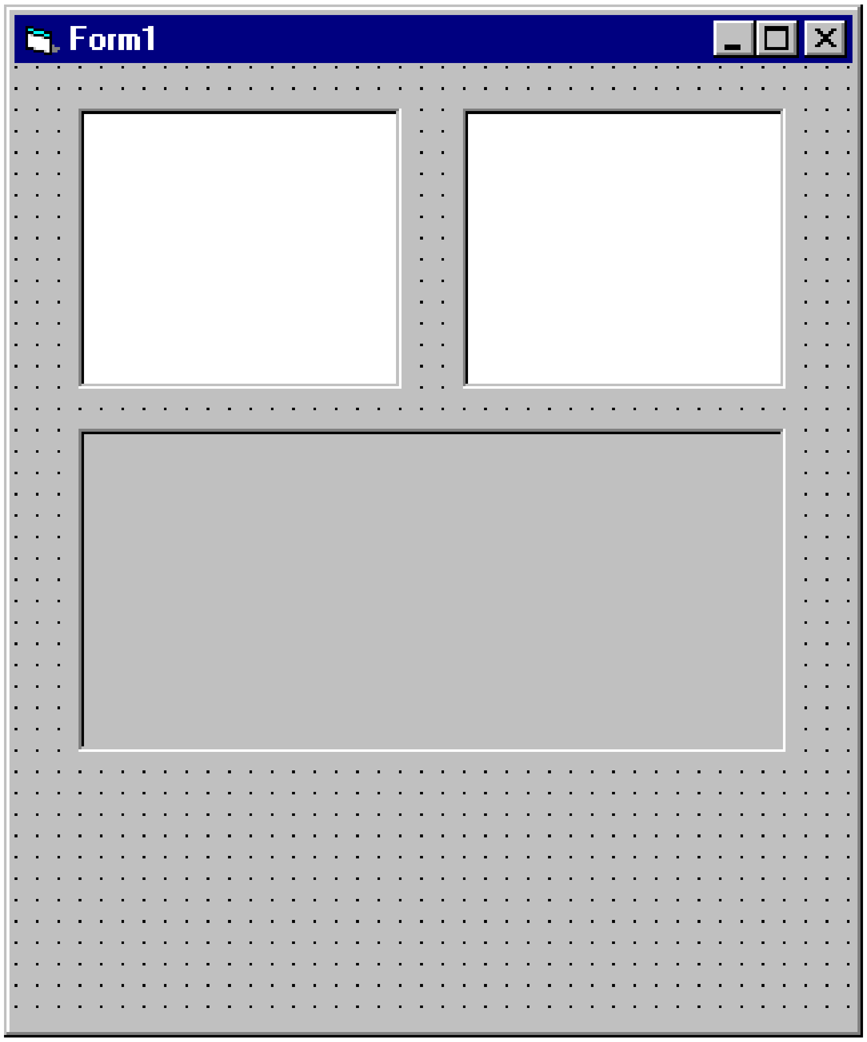
Again, I'll only be discussing two types of Clipboard data in this article---Text and Bitmap.

## Clipboard Methods

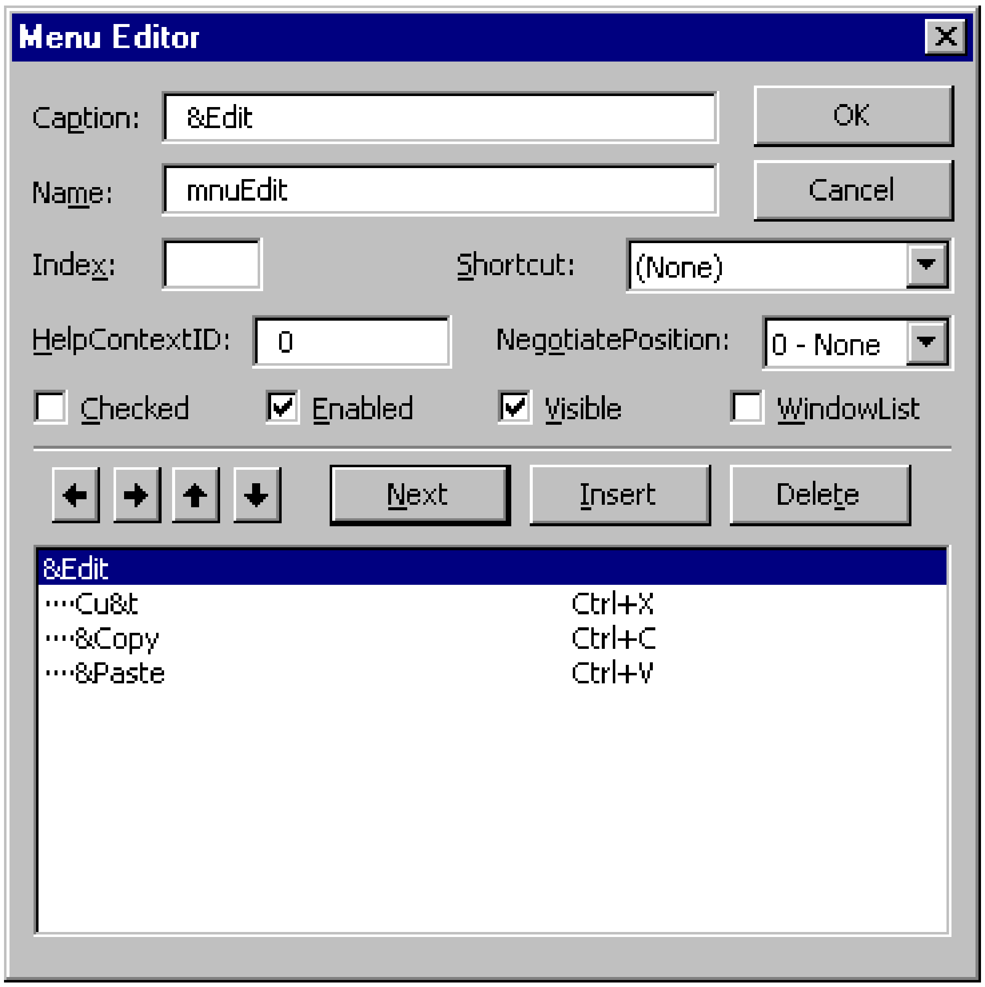
The Clipboard Object has no Properties, but it does have a number of Methods that can be used to interrogate the Clipboard, and to copy data to it and from it. If you'd like to follow along with me as I discuss these methods, it would be a good idea for you to start up Notepad and Windows Paint now.

Have you done that?

Great. Now let's create a new Standard.EXE Visual Basic project, and place two TextBox controls and a PictureBox control on the form. We'll be using the TextBox controls to demonstrate Cut, Copy and Paste operations of Text data, and we'll be using the PictureBox to demonstrate Cut, Copy and Paste operations of Bitmap Data.



Now, let's build a menu, using the Visual Basic Menu Control, consisting of an Edit menu, with three submenu items captioned 'Cut', 'Copy' and 'Paste.'



Menu Editor

Caption: &Edit

Name: mnuEdit

Index:

Shortcut: (None)

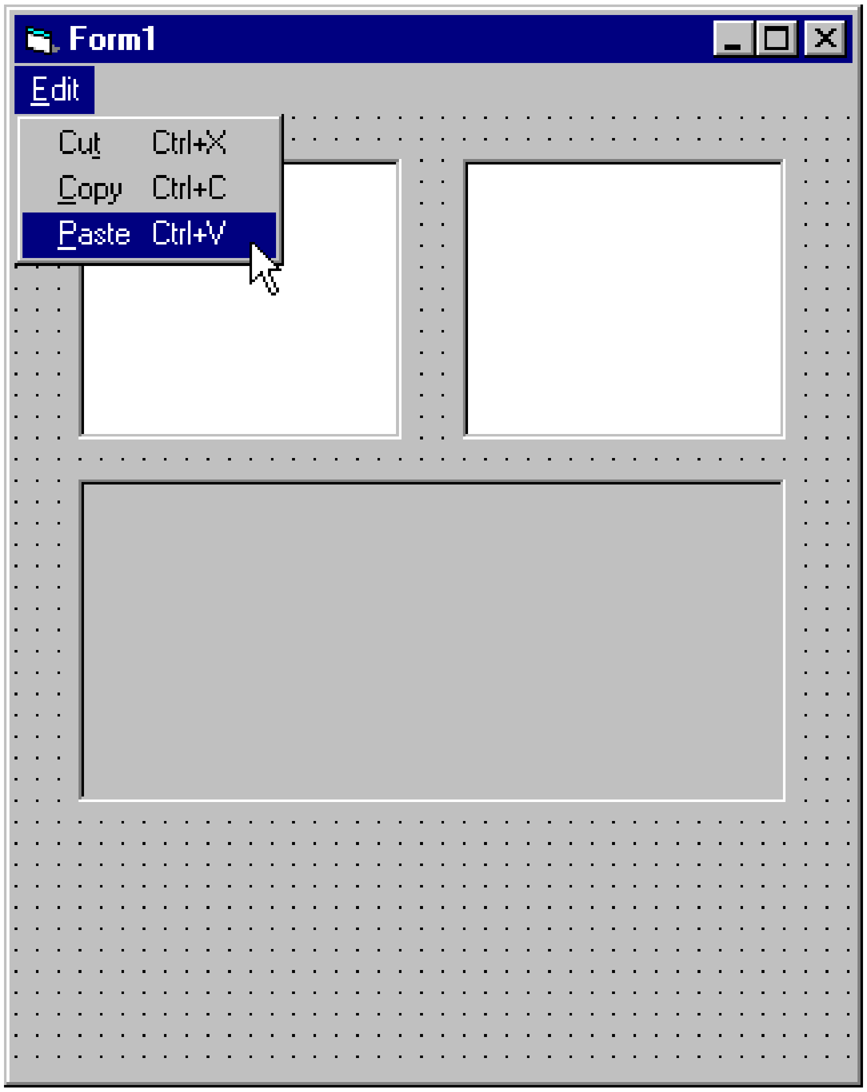
HelpContextID: 0

NegotiatePosition: 0 - None

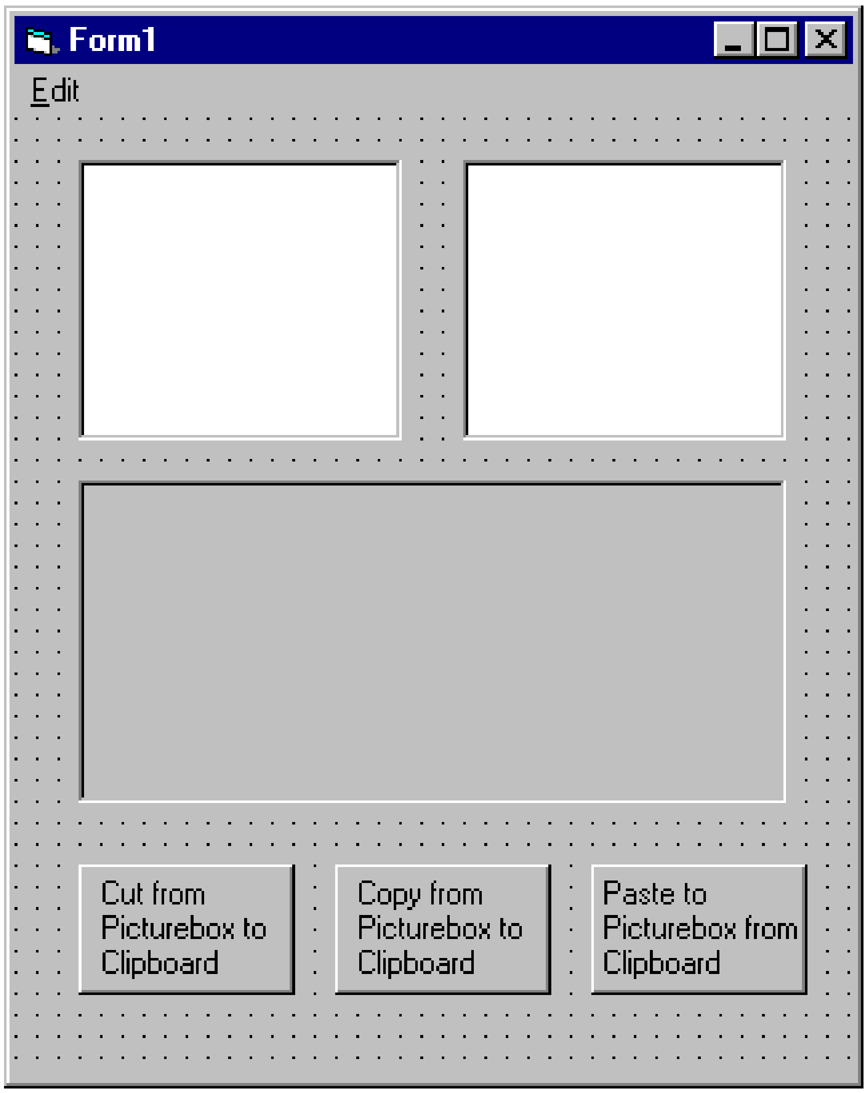
☐ Checked ☑ Enabled ☑ Visible ☐ WindowList

```
&Edit
····Cu&t                      Ctrl+X
····&Copy                     Ctrl+C
····&Paste                    Ctrl+V
```

Notice that I have taken care to follow the Windows design standards for naming the Captions of the Edit menu items, and also in the assignment of access keys and shortcut keys for each menu item.



Finally, let's place three command buttons on the form, and name them cmdCut, cmdCopy and cmdPaste respectively. The command buttons will be used to Cut, Copy and Paste to and from the PictureBox control, and the menu items will be used to Cut, Copy and Paste to and from the TextBox controls.


Form1
Edit
Cut from Picturebox to Clipboard
Copy from Picturebox to Clipboard
Paste to Picturebox from Clipboard

Now that we have the visual portion of this demonstration project in place, I want to briefly discuss the Methods of the Clipboard Object.

## Clear

The Clear Method of the Clipboard will 'clear' the contents of the Clipboard. It's important to note that even though you may have multiple types of data on the Clipboard at the same time, executing the Clipboard's Clear method will clear ***everything*** from the Clipboard. Microsoft recommends that you clear the contents of the Clipboard before placing any data (text or bitmap) on it. Here's the syntax for the Clear method:

**Clipboard.Clear**

Clearing data form the Clipboard is simple enough. Let's take a look at the Clipboard method designed to interrogate the Clipboard.

## GetFormat

The GetFormat method returns a True or False value indicating whether an item on the Clipboard object matches a specified format provided as an argument. The GetFormat method can be used to determine if the Clipboard contains data of a particular type---for instance, text or bitmap data. Here's the syntax for the GetFormat method:

```
GetFormat(vbCFText)
```

will return a True value if the Clipboard contains Text (vbCFText is the Visual Basic Intrinsic Constant for text), and False if the Clipboard contains anything else. Possible values for the argument to GetFormat are:

| **Constant** | **Value** | **Description** |
|---|---|---|
| vbCFLink | &HBF00 | DDE conversation information |
| vbCFText | 1 | Text |
| vbCFBitmap | 2 | Bitmap (.bmp files) |
| vbCFMetafile | 3 | Metafile (.wmf files) |
| vbCFDIB | 8 | Device-independent bitmap (DIB) |
| vbCFPalette | 9 | Color palette |

Let's take a look at the two methods designed to work with Graphical data.

## GetData

The GetData Method is used to retrieve Graphical data from the Clipboard object. This data, once retrieved, can then be assigned to any Visual Basic Property that is expecting a graphics file, such as the Picture Property of the Form or PictureBox. You may optionally supply the GetData method with an argument specifying the particular type of Graphic data you want to retrieve form the Clipboard. If you omit the argument, then Visual Basic assumes the default (bitmap.) Here's the syntax for the GetData method:

```
Clipboard.GetData(format)
```

where format is a constant or value that specifies the Clipboard graphics format, as described below:

The settings for format are:

| **Constant** | **Value** | **Description** |
|---|---|---|
| vbCFBitmap | 2 | Bitmap (. bmp files) |
| vbCFMetafile | 3 | Metafile (. wmf files) |
| vbCFDIB | 8 | Device-independent bitmap (DIB) |
| vbCFPalette | 9 | Color palette |

### SetData

The SetData method is the opposite of the GetData method in that it puts graphical data on the Clipboard object. The syntax for the SetData method is:

**Clipboard.SetData(format)**

where format is a constant or value that specifies the format of the graphic. If format is omitted, SetData automatically determines the graphic format.

The settings for format are:

| **Constant** | **Value** | **Description** |
|---|---|---|
| vbCFBitmap | 2 | Bitmap (. bmp files) |
| vbCFMetafile | 3 | Metafile (. wmf files) |
| vbCFDIB | 8 | Device-independent bitmap (DIB) |
| vbCFPalette | 9 | Color palette |

Now let's take a look at the two methods designed to work with Text data.

## GetText

The GetText Method is used to retrieve Text data from the Clipboard object. Here's the syntax for the GetText method:

**Clipboard.GetText(format)**

where format is a constant or value that specifies the Clipboard text format, as described below:

The settings for format are:

| Constant | Value | Description |
|---|---|---|
| vbCFLink | &HBF00 | DDE conversation information |
| vbCFText | 1 | (Default) Text |
| vbCFRTF | &HBF01 | Rich Text Format (.rtf file) |

## SetText

The SetText method is the opposite of GetText in that it places a Text String on the Clipboard object using a specified Text format. The syntax for the SetText method is:

**Clipboard.SetText(format)**

where format is a constant or value that specifies the format of the text.

The settings for format are:

| Constant | Value | Description |
|---|---|---|
| vbCFLink | &HBF00 | DDE conversation information |
| vbCFText | 1 | (Default) Text |

| | | |
|---|---|---|
| vbCFRTF | &HBF01 | Rich Text Format (.rtf file) |

If the format argument is omitted, Visual Basic assumes the default of Text.

As I always say, a picture is worth a thousand words---lets see how we can use these methods to implement Cut, Copy and Paste operations.

## Cut, Copy and Paste Text Data

### Cutting Text to the Clipboard

The first thing I want to demonstrate is how to 'cut' Text data from a TextBox by placing code into the Click Event Procedure of mnuCut. I could write this code to work with a specific TextBox, but even better, I can use the Screen Object's ActiveControl property to identify the control which currently has focus, and in conjunction with its SelText Property, Cut whatever text is currently selected to the Clipboard. Like this:

```
Private Sub mnuCut_Click()

Clipboard.Clear
Clipboard.SetText Screen.ActiveControl.SelText
Screen.ActiveControl.SelText = ""

End Sub
```

Cutting from the currently selected control to the Clipboard is a three step operation.

First, the Clipboard is cleared using the Clear method

```
Clipboard.Clear
```

Then the text that is currently selected in the control that has focus is placed on the Clipboard using the SetText method

```
Clipboard.SetText Screen.ActiveControl.SelText
```

Finally, because this is a Cut operation that we are implementing, we then erase the selected text in the control that has focus by setting its SelText property to "".

```
Screen.ActiveControl.SelText = ""
```

## Copying Text to the Clipboard

Copying Text to the Clipboard is nearly identical to cutting text with the exception that we don't erase the currently selected text from the control that has focus. Here's the code for mnuCopy.

```
Private Sub mnuCopy_Click()

Clipboard.Clear
Clipboard.SetText Screen.ActiveControl.SelText

End Sub
```

Notice how the code is identical to the click event procedure of mnuCut, except for the line of code that set the SelText property of the control with the focus to "".

## Pasting Text from the Clipboard

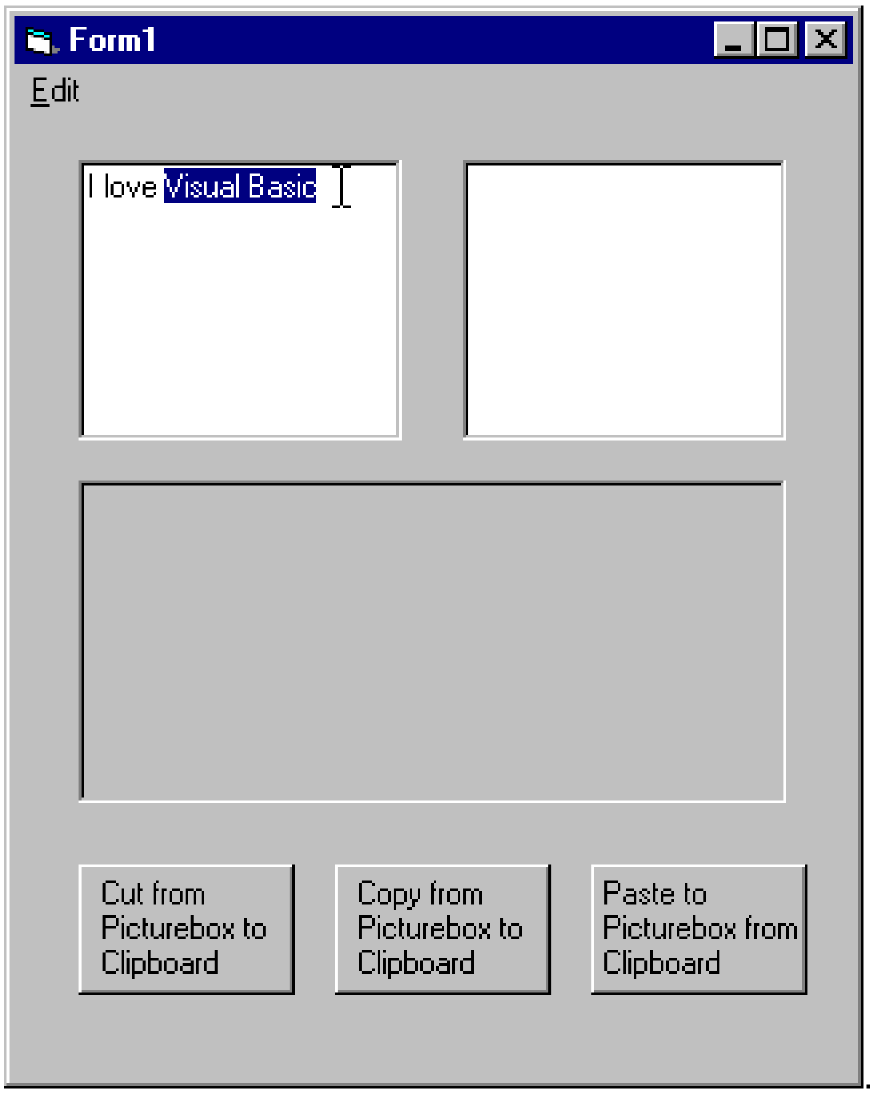
Finally, for the Paste operation, we execute the GetText Method of the Clipboard, which copies the contents of the Windows Clipboard , and assigns it to the SelText property of the ActiveControl.

```
Private Sub mnuPaste_Click()

Screen.ActiveControl.SelText = Clipboard.GetText()

End Sub
```

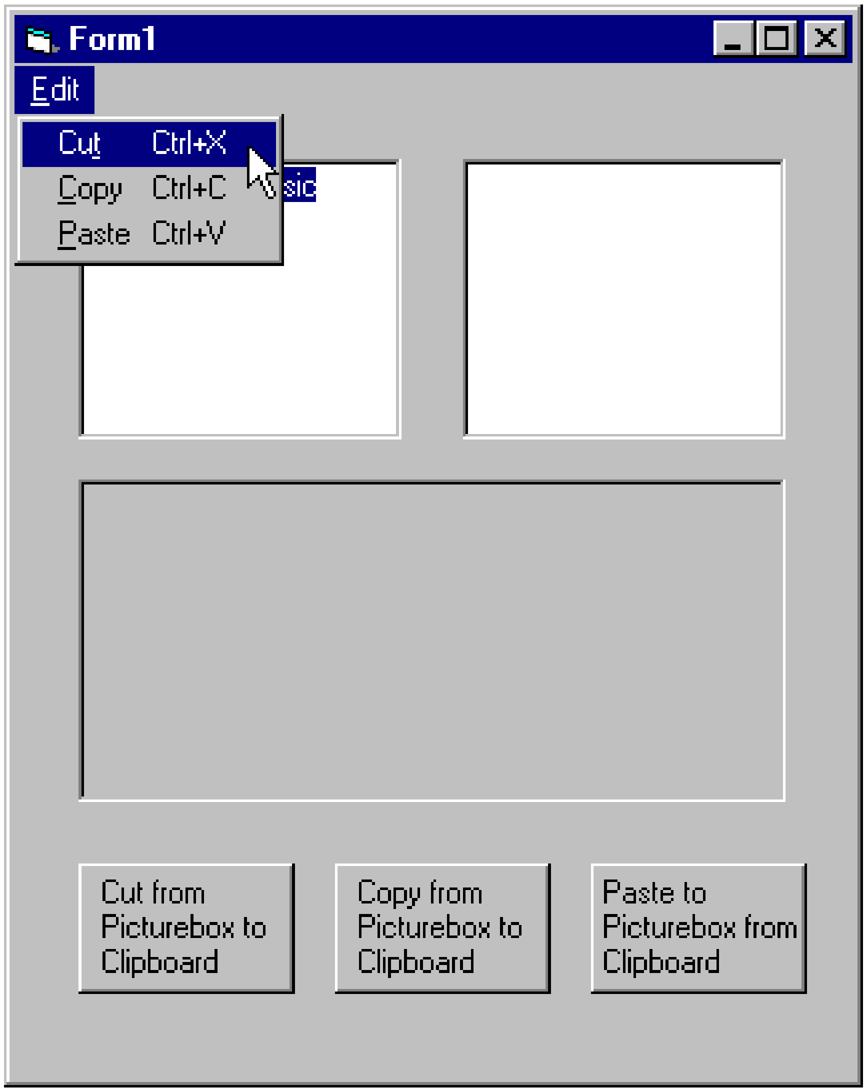
## Putting it altogether

Now let's run the program and see how all this fits together.

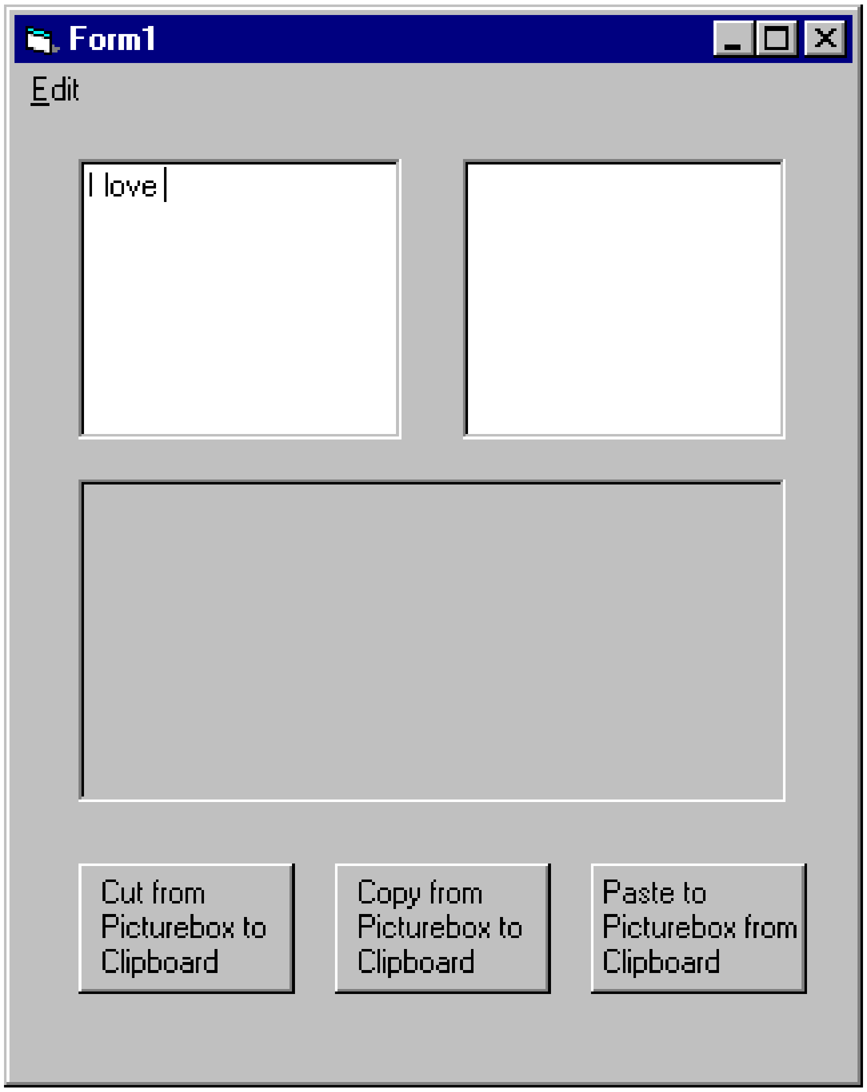
First, let's place some text into the first TextBox, and then select it.


Form1
Edit
I love Visual Basic
Cut from Picturebox to Clipboard
Copy from Picturebox to Clipboard
Paste to Picturebox from Clipboard

If we now select **Edit-Cut** from the menu bar ...

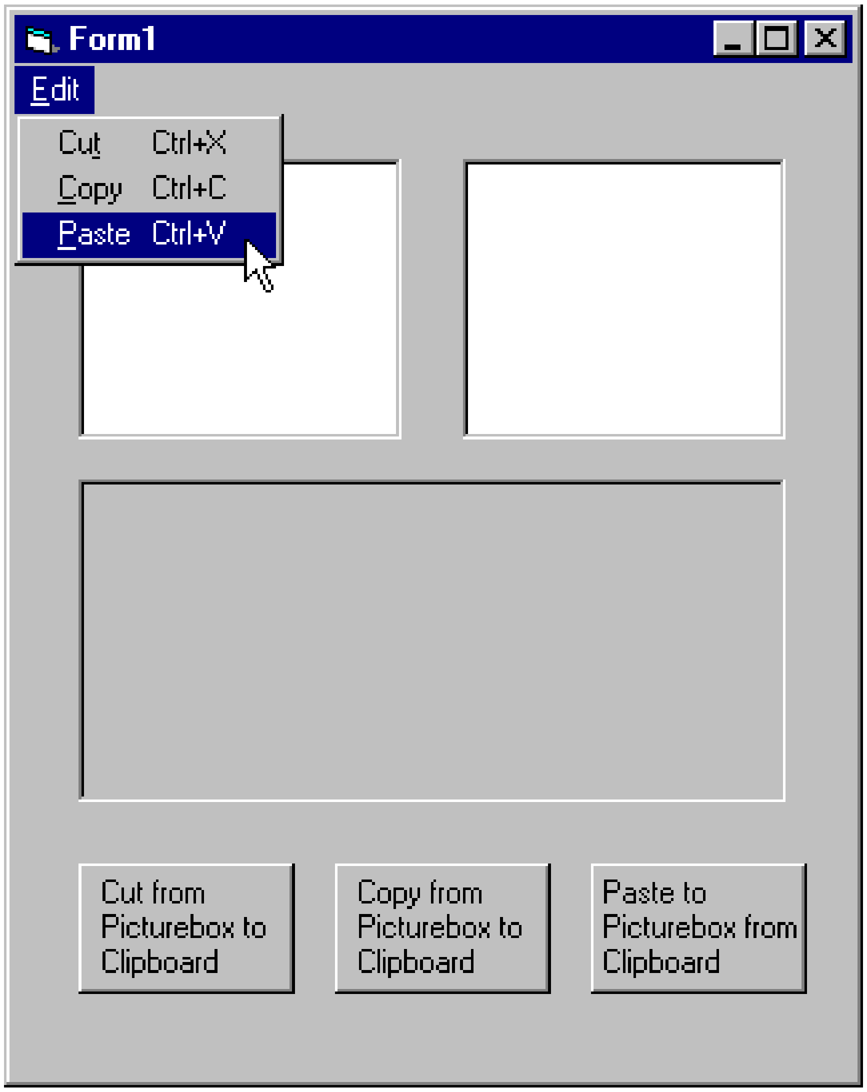

Form1
Edit
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
sic
Cut from Picturebox to Clipboard
Copy from Picturebox to Clipboard
Paste to Picturebox from Clipboard

the selected text in the first TextBox will disappear.

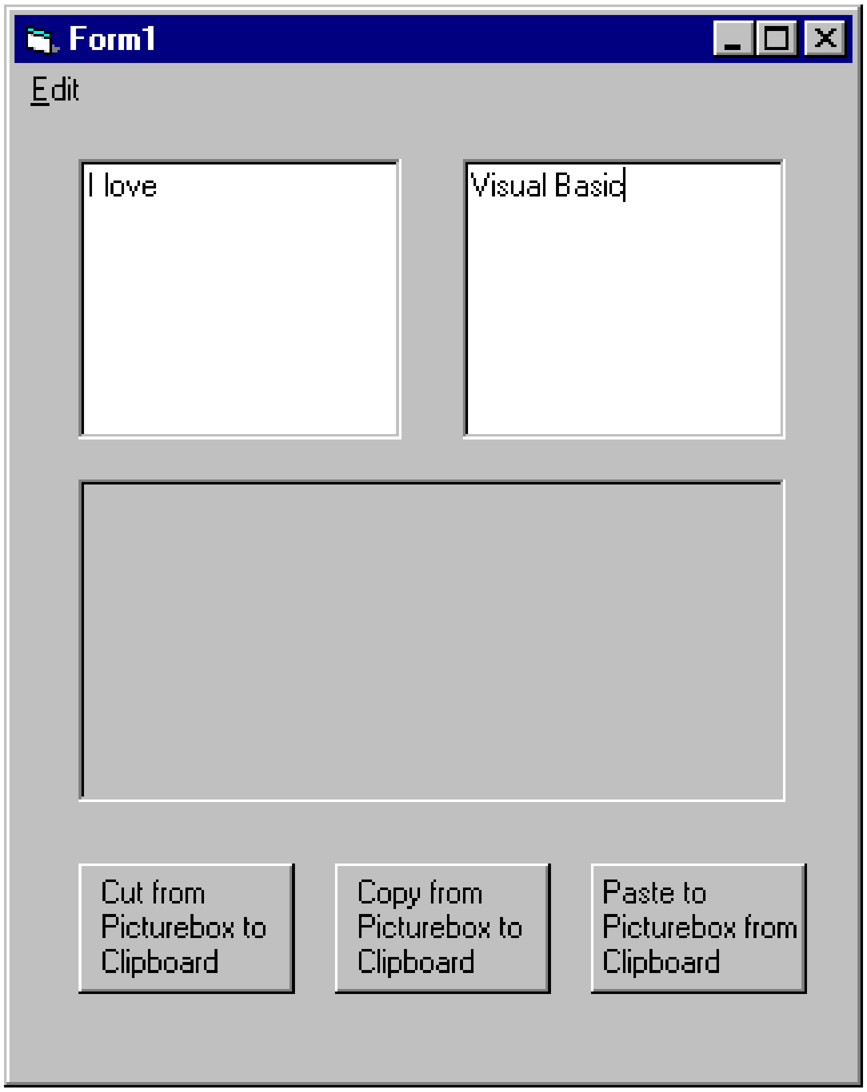

Form1
Edit
I love
Cut from Picturebox to Clipboard
Copy from Picturebox to Clipboard
Paste to Picturebox from Clipboard

Where did it go?

Right now, it's sitting on the Windows Clipboard. If we now click or tab to the second TextBox to give it focus, and then select **Edit-Paste** from the menu bar.

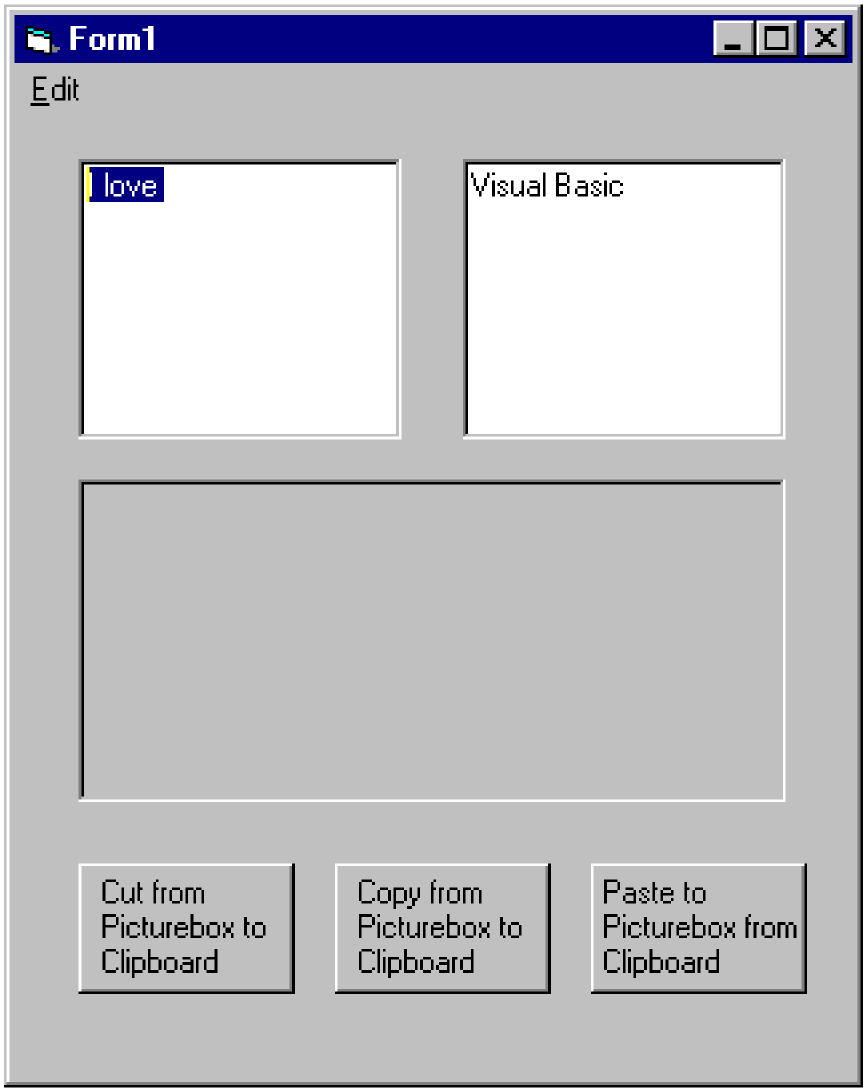

Form1
Edit
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Cut from Picturebox to Clipboard
Copy from Picturebox to Clipboard
Paste to Picturebox from Clipboard

The 'cut' text will then appear in the second TextBox.

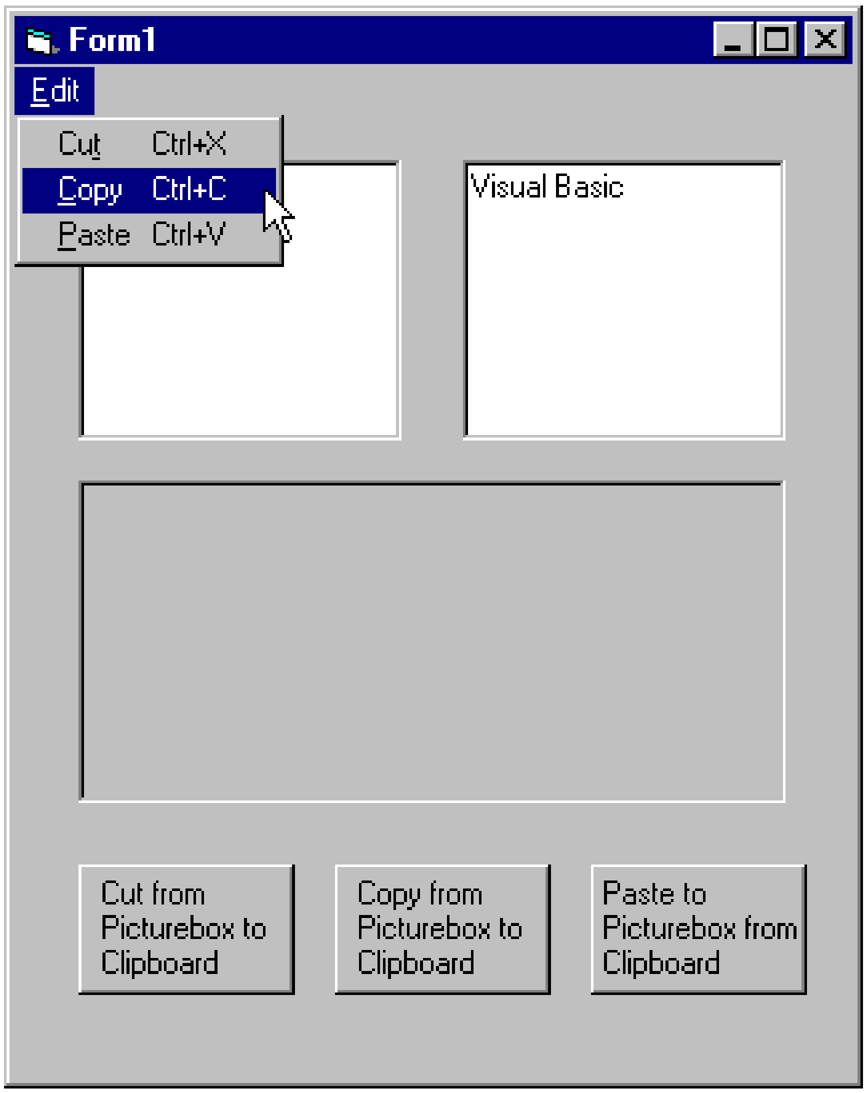

Form1
Edit
I love
Visual Basic
Cut from Picturebox to Clipboard
Copy from Picturebox to Clipboard
Paste to Picturebox from Clipboard

Copy will work in a similar manner, except the selected Text won't be erased from the first TextBox. Let's try that out now, except this time we'll copy the selected text not to another TextBox, but to another application---Notepad.

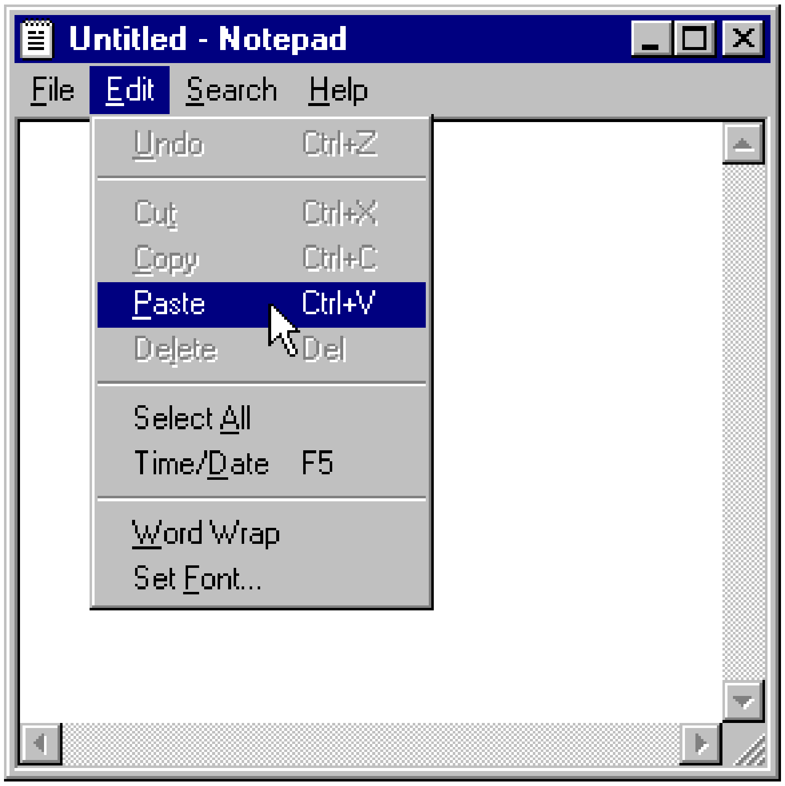
Let's begin as we did before by selecting text in the first TextBox ...


Form1
Edit
I love
Visual Basic
Cut from Picturebox to Clipboard
Copy from Picturebox to Clipboard
Paste to Picturebox from Clipboard

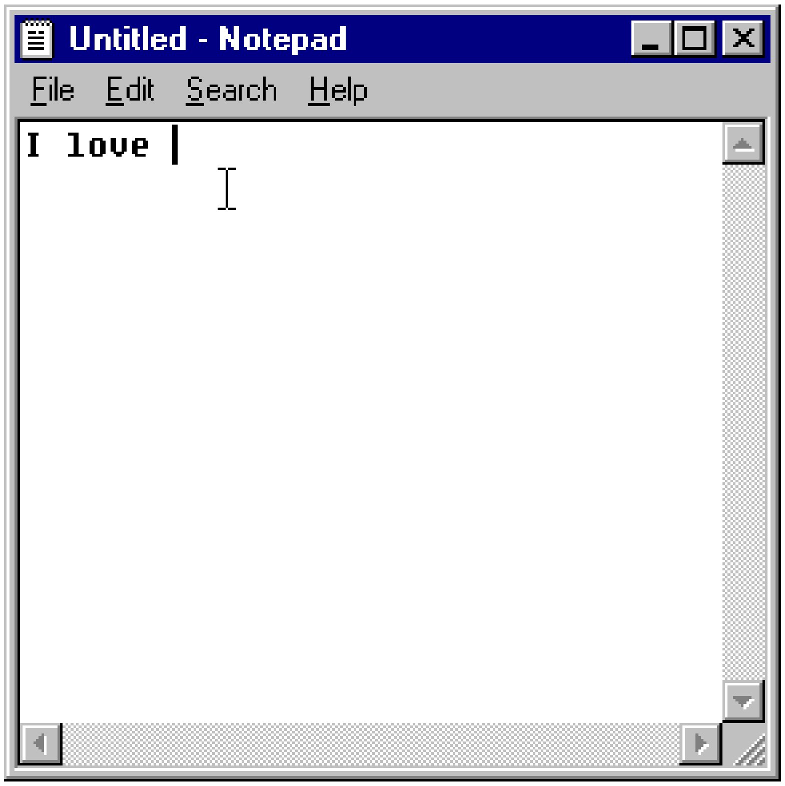
Now we'll select Edit-Copy from the menu bar ...


Form1
Edit
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Visual Basic
Cut from Picturebox to Clipboard
Copy from Picturebox to Clipboard
Paste to Picturebox from Clipboard

and then switch to Notepad, where we'll select Edit-Paste from its menu bar...



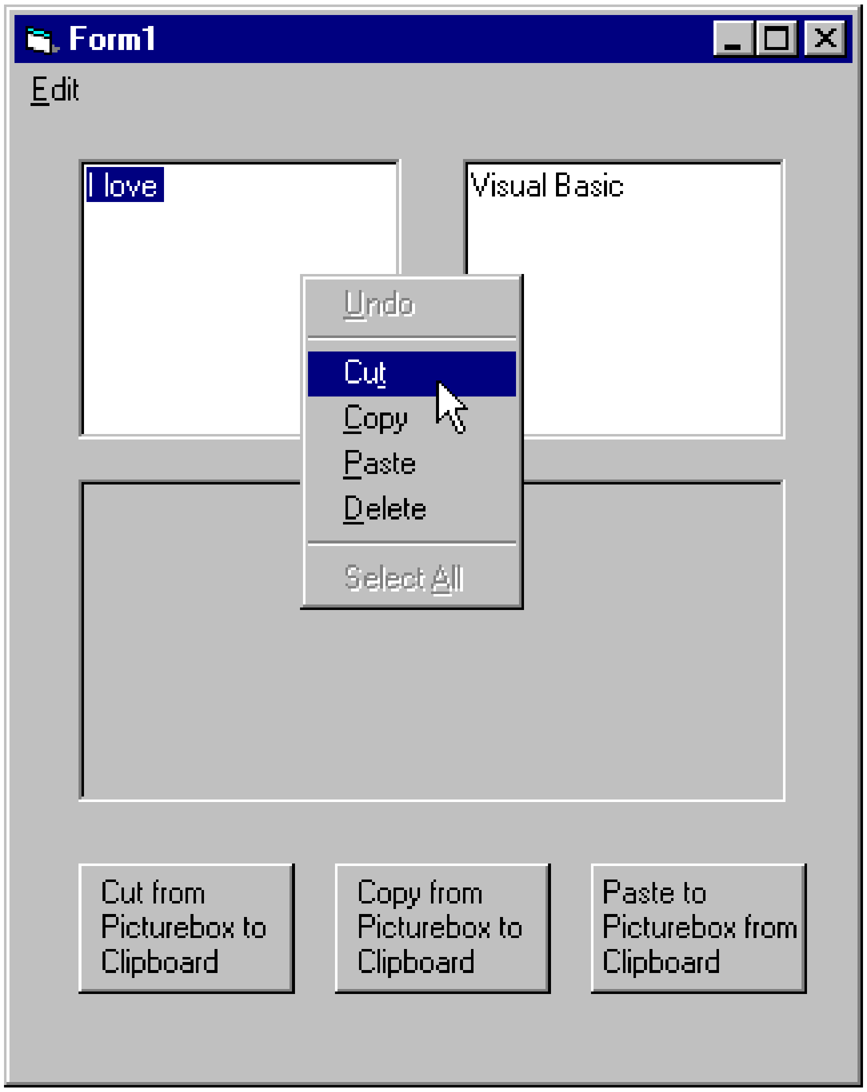
The copied text will then appear in Notepad...



I love

All in all, this isn't difficult at all. With just a few lines of code, you can implement Cut, Copy and Paste operations in your own application. Now, having described and explained how to build this functionality into your own program, I would be remiss if I didn't point out that the TextBox control has this functionality already built in!

That's right---if you right click your mouse in the TextBox, you'll see this PopUp menu...


Form1
Edit
I love
Visual Basic
Undo
Cut
Copy
Paste
Delete
Select All
Cut from Picturebox to Clipboard
Copy from Picturebox to Clipboard
Paste to Picturebox from Clipboard

... and you'll find that you have full Cut, Copy and Paste functionality using the Textbox's PopUp menu as well. But remember---most of your users will be looking for this functionality via the menu bar of your application, so it's a good idea to place it there.

## Cut, Copy and Paste Graphic Data

Cut, Copy and Paste operations on Graphic data follow the same basic process as the operations on Text data.

### Cutting Graphic Data to the Clipboard

As was the case with cutting Text Data from a control, and placing it on the

Clipboard, cutting

Graphic data is a three step operation. Here's the code:

```
Private Sub cmdCut_Click()

Clipboard.Clear
Clipboard.SetData Picture1.Picture
Picture1.Picture = LoadPicture()

End Sub
```

First, the Clipboard is cleared using the Clear method, just as we did with Text Data.

```
Clipboard.Clear
```

Copying Graphic data to the Clipboard requires that we execute the SetData method instead of the SetText method. Unlike the example I showed you where we copied Text by using the SelText Property of the TextBox, here we use the Picturebox's Picture Property as an argument to the SetData method

```
Clipboard.SetData Picture1.Picture
```

Finally, because this is a Cut operation, after we copy the Graphic data to the Clipboard we then 'cut' the graphic from the PictureBox by executing the LoadPicture function with a null argument to erase the graphic from the PictureBox.

```
Picture1.Picture = LoadPicture()
```

## Copying Graphic Data to the Clipboard

Copying Graphic Data to the Clipboard is nearly identical to cutting Graphic Data with the exception that we don't erase the Graphic from the PictureBox control. Here's the code to Copy Graphics data to the Clipboard.

```
Private Sub cmdCopy_Click()

Clipboard.Clear
Clipboard.SetData Picture1.Picture

End Sub
```

## Pasting Graphic Data from the Clipboard

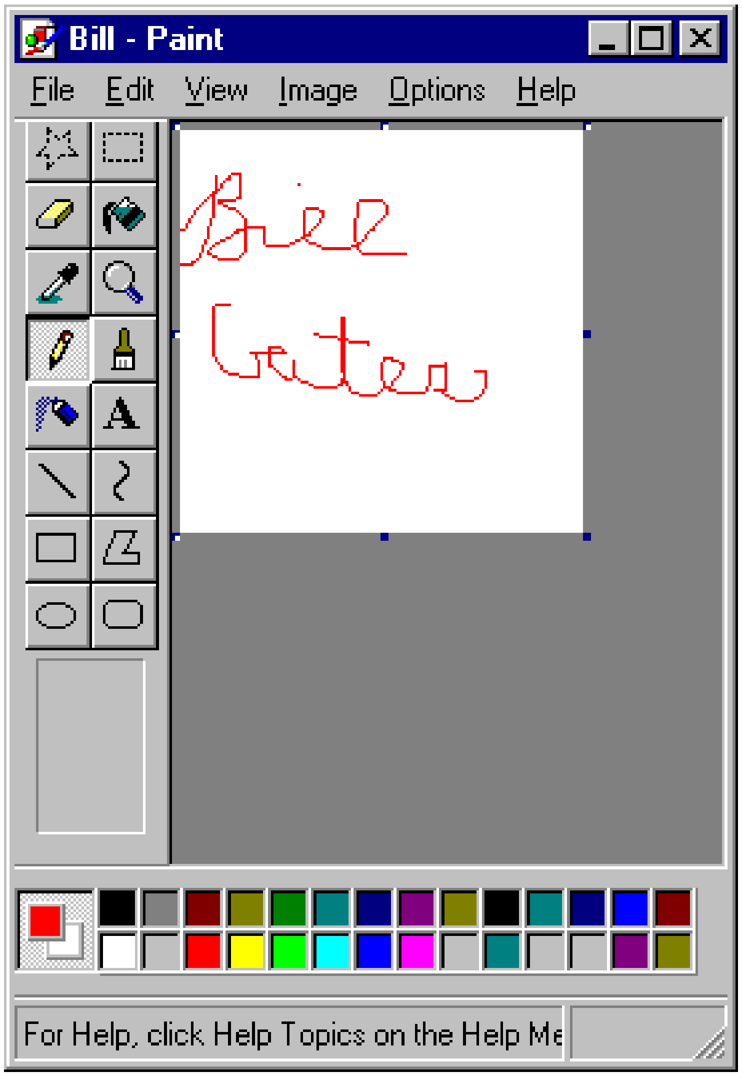
For the Paste operation, we execute the GetData Method of the Clipboard, which copies the contents of the Windows Clipboard, and assigns it to the Picture property of the PictureBox.

```
Private Sub cmdPaste_Click()

Picture1.Picture = Clipboard.GetData()

End Sub
```

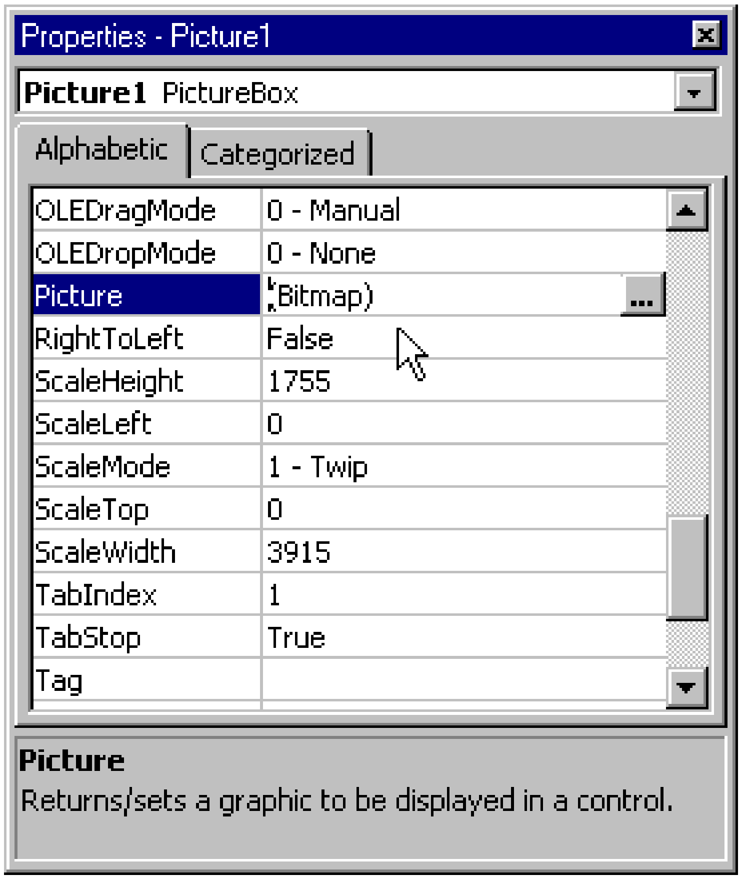
## Putting it altogether

Now let's run this program and see how it all fits together.

First, let's create a Graphic file in bitmap format using Microsoft Paint that we can use for a demonstration, and save it as Bill.bmp...



Now let's set the Picture Property of the PictureBox to point to this file...


Properties - Picture1

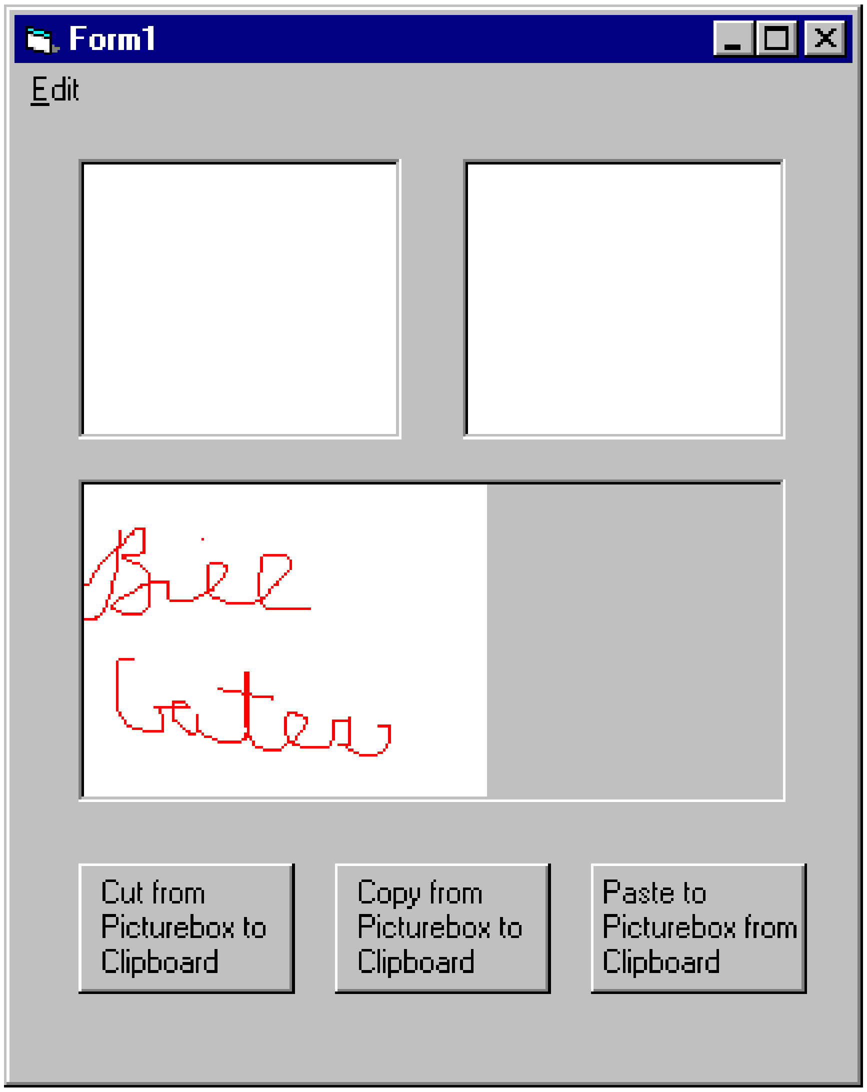
**Picture1** PictureBox

Alphabetic | Categorized

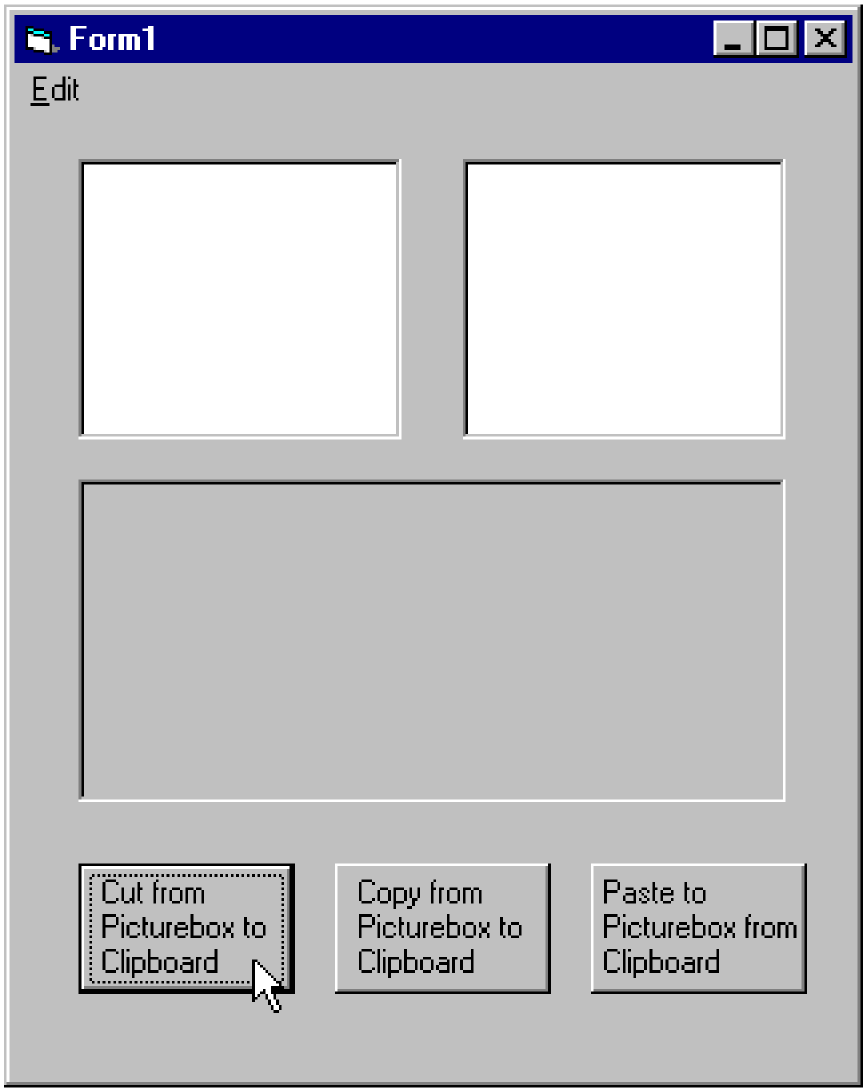
| | |
|---|---|
| OLEDragMode | 0 - Manual |
| OLEDropMode | 0 - None |
| Picture | (Bitmap) |
| RightToLeft | False |
| ScaleHeight | 1755 |
| ScaleLeft | 0 |
| ScaleMode | 1 - Twip |
| ScaleTop | 0 |
| ScaleWidth | 3915 |
| TabIndex | 1 |
| TabStop | True |
| Tag | |

**Picture**
Returns/sets a graphic to be displayed in a control.

Now when we run the program, we'll now see this ...


Form1
Edit
Cut from Picturebox to Clipboard
Copy from Picturebox to Clipboard
Paste to Picturebox from Clipboard

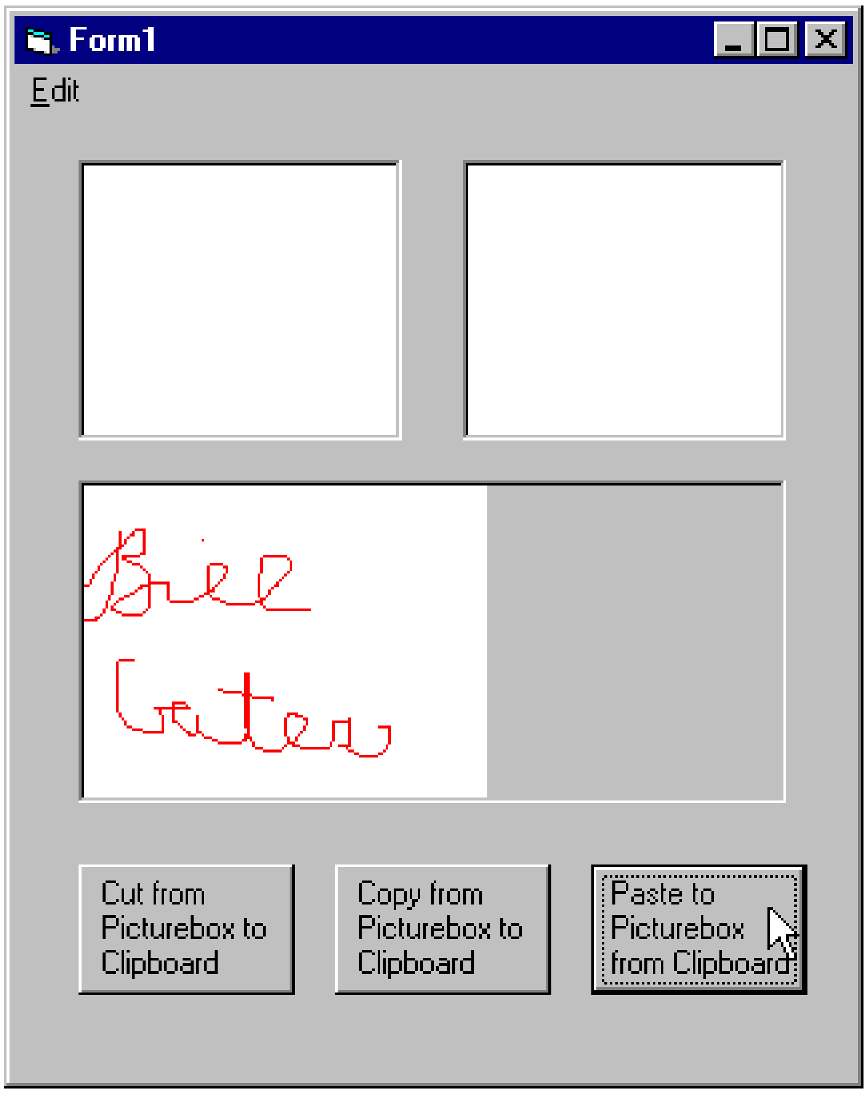
Now, if we click on the 'Cut from PictureBox to Clipboard' command button, the image will disappear from the PictureBox...


Form1
Edit
Cut from Picturebox to Clipboard
Copy from Picturebox to Clipboard
Paste to Picturebox from Clipboard

Where has it gone?

To the Windows Clipboard.

If we now click on the 'Paste to PictureBox from Clipboard' command button, the image will reappear.


Form1
Edit
Cut from Picturebox to Clipboard
Copy from Picturebox to Clipboard
Paste to Picturebox from Clipboard

The 'Copy from PictureBox to Clipboard' will work in a similar manner, except that the image won't be 'erased' from the PictureBox, only copied to the Clipboard.

I think you'll agree that Graphic Data operations, just like Text Data Operations, aren't difficult at all. However, there are some potential problems of which you should be aware.

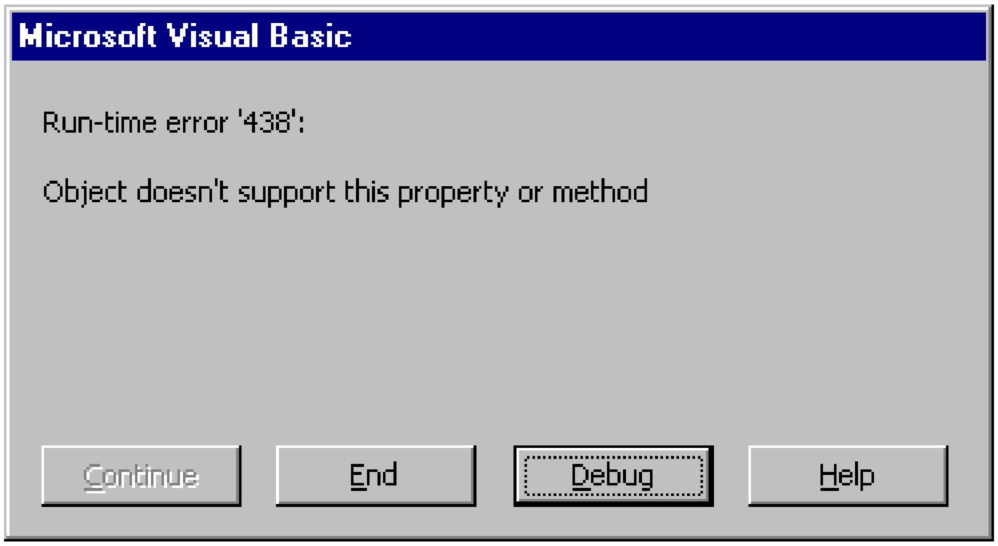
## Problems you might encounter

If you're not careful, there are some things that can trip you up with Cut, Copy and Paste Operations. What are they?

Incompatible Data formats.

Since there are separate methods for working with Text and Graphic formats, you need to be careful to use the correct method for the data (Text or Graphic) with which you are working. You might wonder what happens if you use a method on the wrong data type? Let's see.

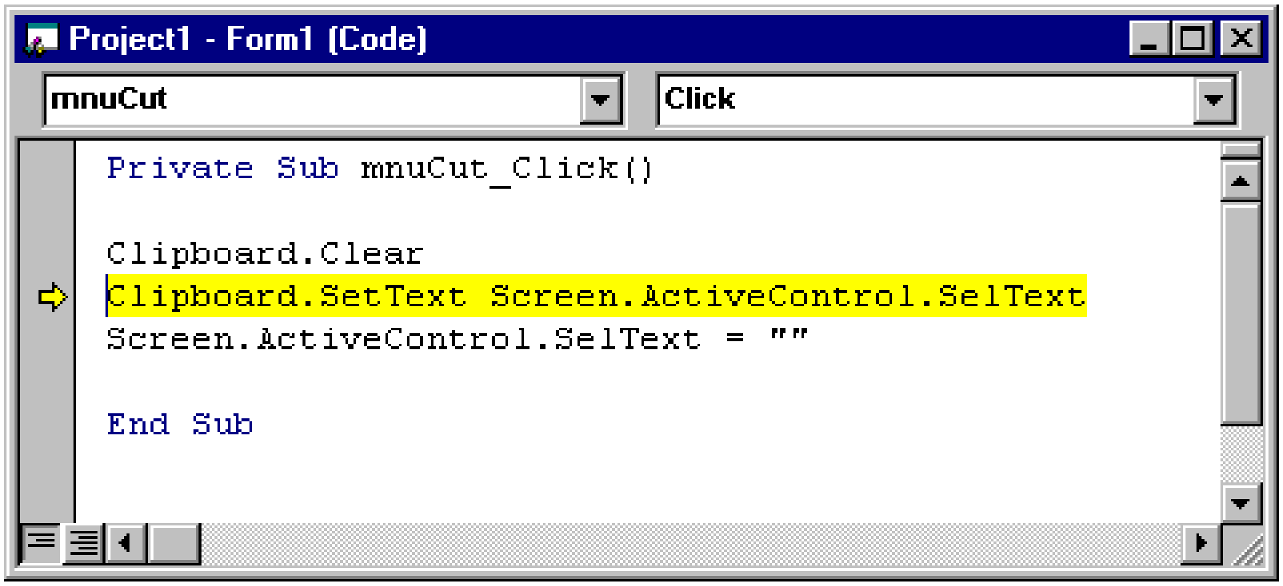
For instance, if we click on the PictureBox, thereby making it the ActiveControl, then select Edit-Cut from the menu bar, we'll receive this error...


Microsoft Visual Basic

Run-time error '438':

Object doesn't support this property or method

The reason for this error is that Visual Basic is trying to execute the SetText method of the Clipboard object on the PictureBox---using the SelText property of the ActiveControl--a property that the PictureBox control does not support.



```
Private Sub mnuCut_Click()

Clipboard.Clear
Clipboard.SetText Screen.ActiveControl.SelText
Screen.ActiveControl.SelText = ""

End Sub
```

How can we avoid this type of error?

We can use an If statement where we first determine the type of control that is the ActiveControl by using the TypeOf keyword will do the trick...

**Private Sub mnuCut_Click()**

**If TypeOf Screen.ActiveControl Is TextBox Then**

```
  Clipboard.Clear
  Clipboard.SetText Screen.ActiveControl.SelText
  Screen.ActiveControl.SelText = ""
End If

End Sub
```

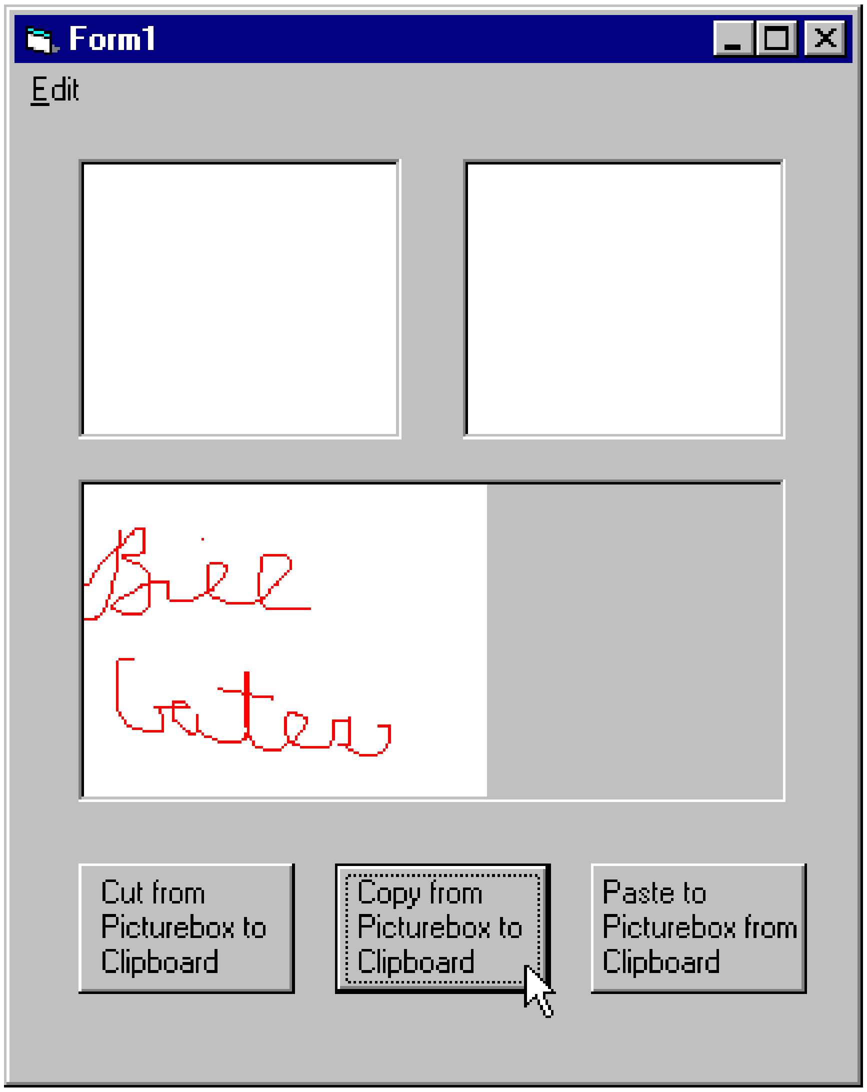
Now the Cut operation will only be executed if the ActiveControl is a TextBox. Likewise, we should change the code in mnuCopy to this...

```
Private Sub mnuCopy_Click()

If TypeOf Screen.ActiveControl Is TextBox Then
  Clipboard.Clear
  Clipboard.SetText Screen.ActiveControl.SelText
End If

End Sub
```

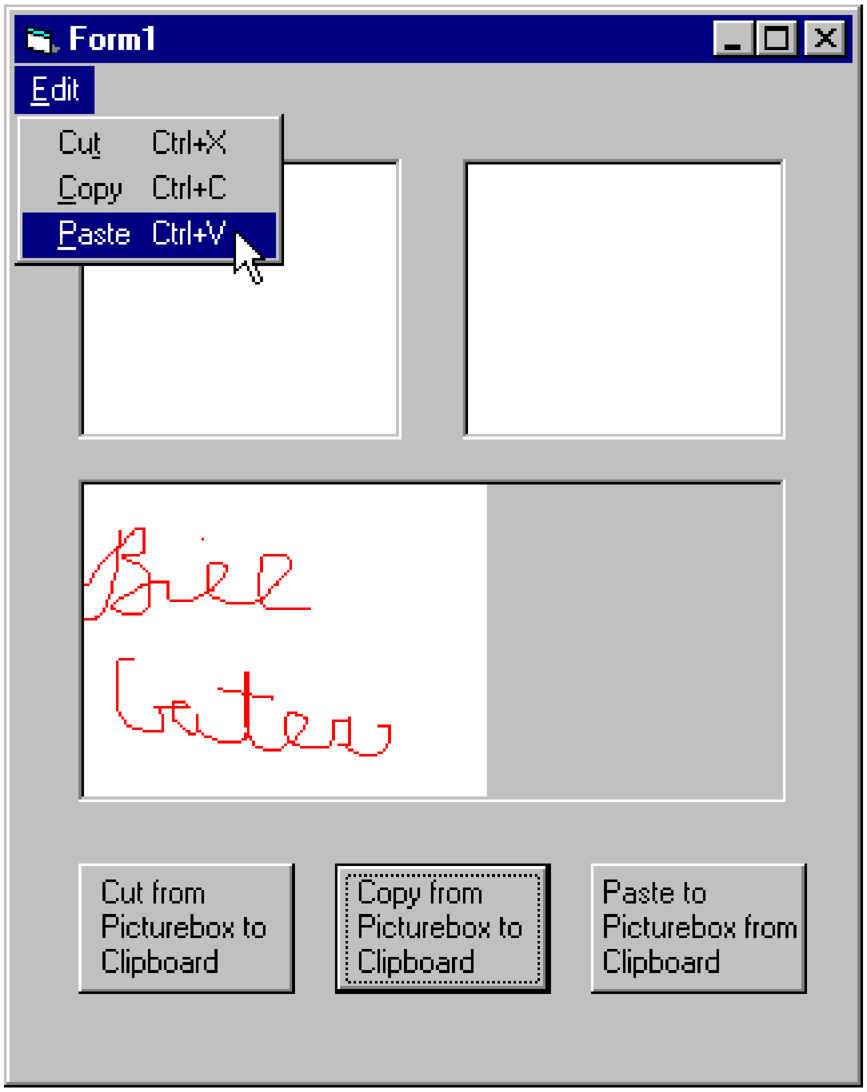
Now if we run the program, both the menu Cut and Copy operations are 'bomb proof', even if the user has not first , even if the user has not first set focus to one of the TextBoxes. But what about the Paste Operation?>

For instance, suppose we try to Paste a graphic on the Clipboard into a TextBox? Let's see.

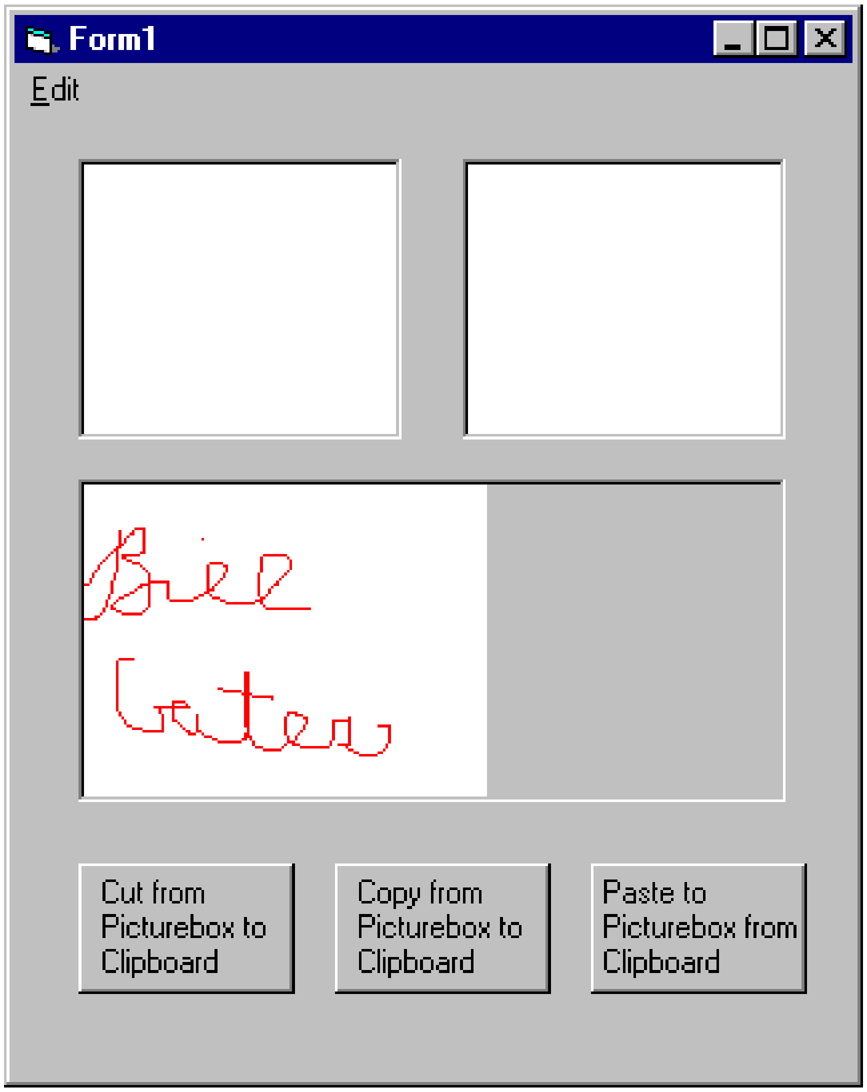
First, click on the 'Copy from PictureBox to Clipboard'...


Form1
Edit
Cut from Picturebox to Clipboard
Copy from Picturebox to Clipboard
Paste to Picturebox from Clipboard

... therefore placing the Bill.bmp data on the Clipboard. Now type a few characters into the first TextBox, setting focus to it, and then select Edit-Paste from the menu bar ...


Form1
Edit
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Bill
Gates
Cut from Picturebox to Clipboard
Copy from Picturebox to Clipboard
Paste to Picturebox from Clipboard

... what happens? Does the program bomb?


Form1
Edit
Cut from Picturebox to Clipboard
Copy from Picturebox to Clipboard
Paste to Picturebox from Clipboard

No, the program doesn't bomb, but it does clear the existing text in the TextBox. Why is that?>

The code in the click event procedure of mnuPaste uses the Clipboard's GetText method ...

```
Private Sub mnuPaste_Click()

Screen.ActiveControl.SelText = Clipboard.GetText()

End Sub
```

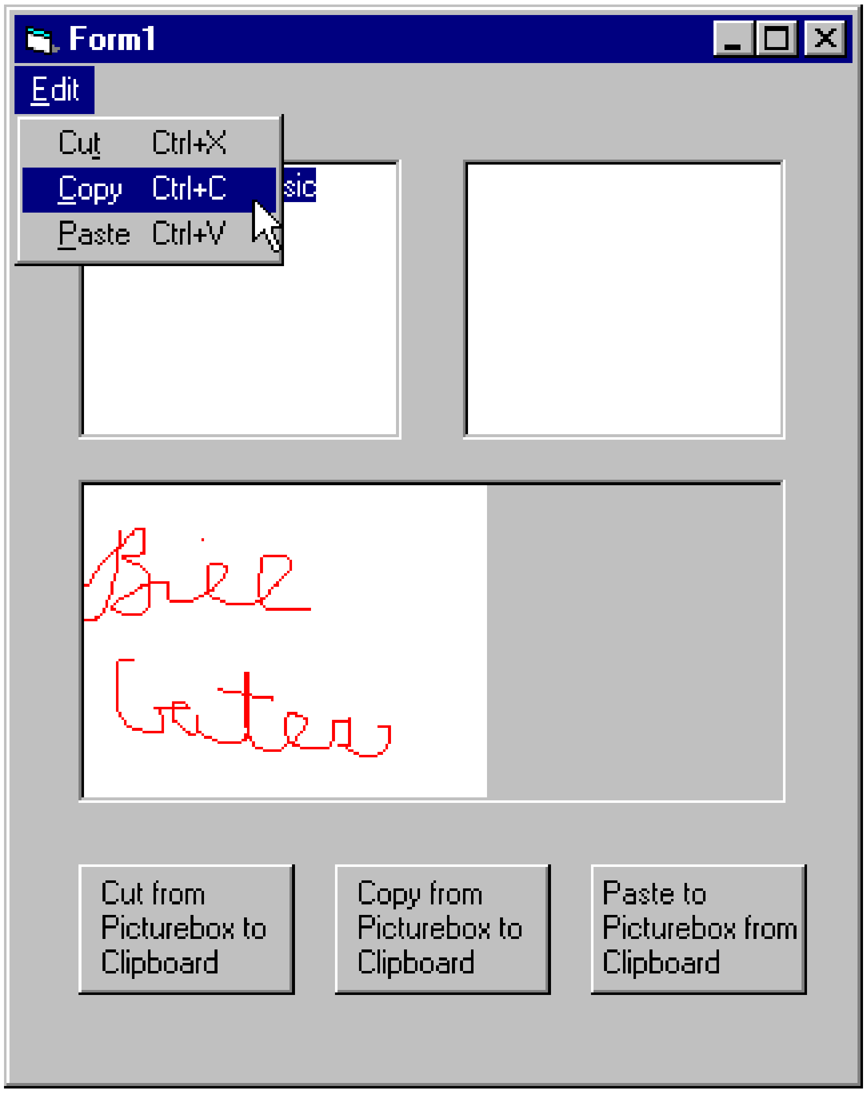
When GetText is executed, if data of the specified data type (in this case Text data) is not found on the Clipboard, the GetText method returns a zero length string. That means any text currently selected in the TextBox control will be replaced with a zero length string---that's why the selected text in the TextBox 'disappeared'. In this case, even though our program didn't bomb, this is hardly an ideal outcome. Better yet would be this code using the GetFormat method to determine if Text data is on the Clipboard.

```
Private Sub mnuPaste_Click()

If Clipboard.GetFormat(vbCFText) Then
  Screen.ActiveControl.SelText = Clipboard.GetText()
End If

End Sub
```

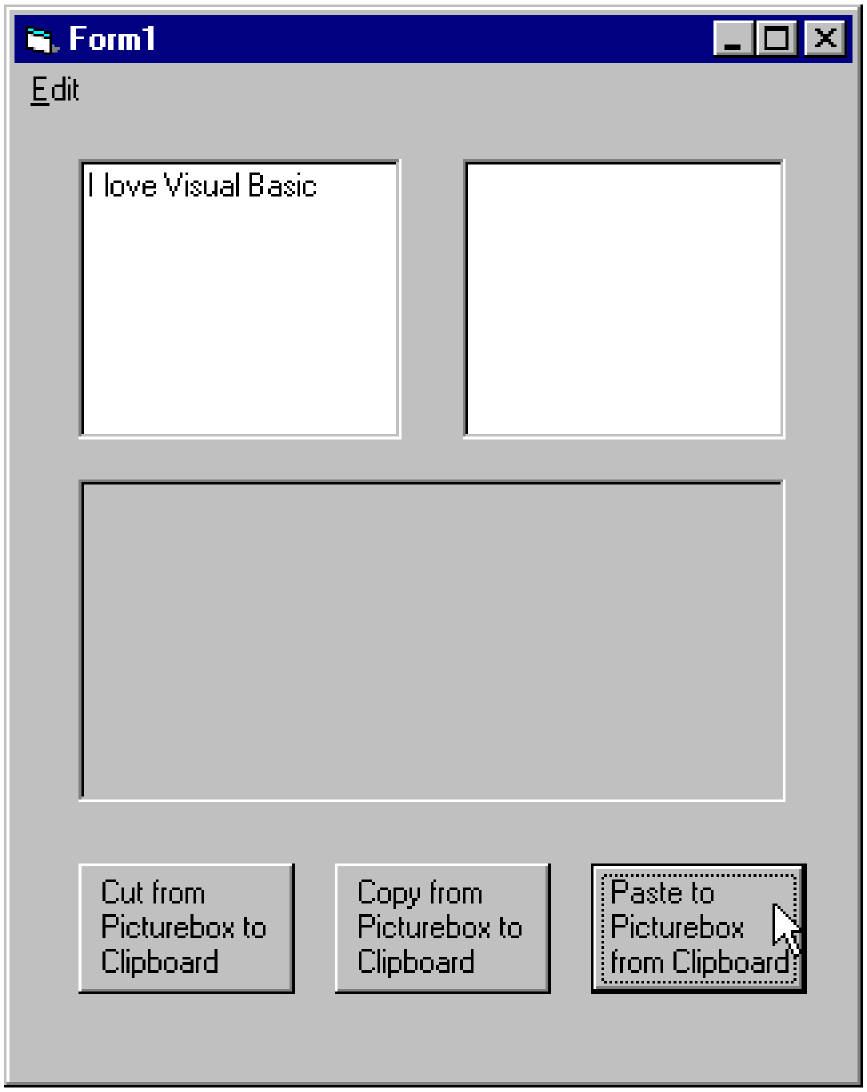
Now if we try to paste non-text data into the TextBox, we won't be able to do so.

What about pasting Text into a PictureBox?

Again, let's place some text in the TextBox, copy it by selecting Edit-Copy...


Form1
Edit
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
sic
Bill
Gates
Cut from Picturebox to Clipboard
Copy from Picturebox to Clipboard
Paste to Picturebox from Clipboard

and then click on the 'Paste to PictureBox from Clipboard' command button...


Form1
Edit
I love Visual Basic
Cut from Picturebox to Clipboard
Copy from Picturebox to Clipboard
Paste to Picturebox from Clipboard

As you can see, the contents of the PictureBox are erased, but no text is placed in the PictureBox.

As we did with the code in the Click Event Procedure of **mnuPaste**, we should modify the code in the Click Event Procedure of **cmdPaste** to first check to see if Graphic data is present on the Clipboard prior to proceeding with the Paste Operation...

```
Private Sub cmdPaste_Click()

If Clipboard.GetFormat(vbCFBitmap) Then
  Picture1.Picture = Clipboard.GetData()
End If

End Sub
```

Now, if Graphic data is not present on the Clipboard, attempting to Paste data into the PictureBox will not occur---and the Image will remain.

## Summary

Cut, Copy and Paste operations can give your program a professional look and feel. I hope that you've seen that implementing these features in your own programs can be amazingly simple---you just need to be careful.