Copy and Paste with RichTextBox

by Karl E. Peterson

Click & Retrieve Source CODE!

CODING A RICHTEXTBOX EDIT MENU

Q It's easy to code Cut, Copy, and Paste routines for a standard text box. For example, you simply write "Text1.SelText = Clipboard.GetText( )" for Paste code. But when I tried this on a RichTextBox, it didn't work correctly. Paste didn't paste anything and Copy copied twice—"Is this an apple?Is this an apple?", for example. Is there a conflict between a TextBox and a RichTextBox, or did I do something wrong?
—Min Kim, Redmond, Washington

A While you don't say specifically, my hunch is that you've included the standard shortcuts such as Ctrl-C and Ctrl-V on your Edit menu. Unlike the standard text box, the RichTextBox automatically supports these shortcuts. This would explain your observation of double cuts, copies, and pastes—your code is doing it once, and the control might also be doing it if you use the shortcut.

You also need to account for another difference: the default format used by the Clipboard's GetText method is vbCFText. If you want to preserve the formatting (why else use a RichTextBox?), you must specify that with vbCFRTF:

```
RichTextBox1.SelRTF = Clipboard.GetText(vbCFRTF)
```

Strangely, Microsoft didn't support this format with the Clipboard's GetData method. You get an "Invalid Clipboard Format" error if you attempt to use this code:

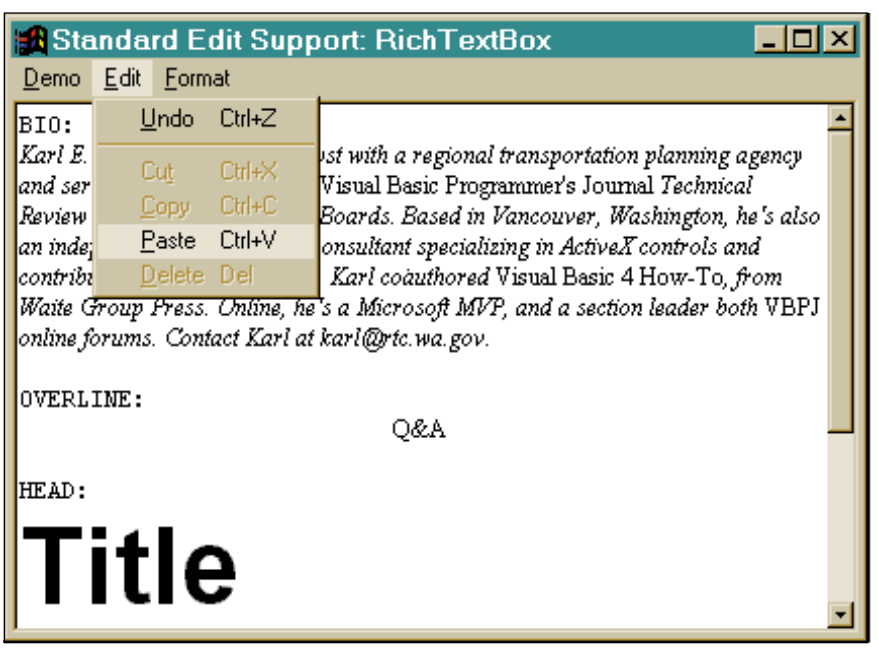

FIGURE 1 ***Code Edit Menus Using SendMessage.*** *Using the SendMessage API allows a single routine to support edit functions for either a standard text box or a rich text box, as well as provide Undo support that VB doesn't offer. You can download this demo from the free, Registered Level of DevX.*

Karl E. Peterson is a GIS analyst with a regional transportation planning agency and serves as a member of the Visual Basic Programmer's Journal Technical Review and Editorial Advisory Boards. Based in Vancouver, Washington, he's also an independent programming consultant specializing in ActiveX controls. Coauthor of Visual Basic 4 How-To from Waite Group Press, Karl's a Microsoft MVP online, and a section leader in both VBPJ online forums. Contact Karl at karl@rtc.wa.gov.

Ask the VB Pro provides you with free advice on programming obstacles, techniques, and ideas. Read more answers from our crack VB pros on the Web at www.inquiry.com/thevbpro. You can submit your questions, tips, or ideas on the site, or mail them to Ask the VB Pro, c/o Fawcette Technical Publications, 209 Hamilton Avenue, Palo Alto, CA 94301-2500, USA.

VB3 VB4 16-bit 32-bit VB5

```
'
' Windows API call used to control textbox
'
#If Win16 Then
  Private Declare Function SendMessage Lib "User" _
     (ByVal hWnd As Integer, ByVal wMsg _
     As Integer, ByVal wParam As Integer, lParam _
     As Any) As Long
#ElseIf Win32 Then
  Private Declare Function SendMessage Lib _
     "user32" Alias "SendMessageA" (ByVal hWnd _
     As Long, ByVal wMsg As Long, ByVal wParam _
     As Long, lParam As Any) As Long
#End If
'
' Edit Control Messages
'
Const WM_CUT = &H300
Const WM_COPY = &H301
Const WM_PASTE = &H302
Const WM_CLEAR = &H303
Const WM_UNDO = &H304
#If Win16 Then
  Const EM_CANUNDO = &H416          'WM_USER + 22
  Const EM_GETMODIFY = &H408        'WM_USER + 8
#ElseIf Win32 Then
  Const EM_CANUNDO = &HC6
  Const EM_GETMODIFY = &HB8
#End If
'
' Flag to track status of Control key
'
Private m_ControlKey As Boolean

Private Sub Form_KeyDown(KeyCode As Integer, _
  Shift As Integer)
  '
  ' Watch for Control key, set flag
  '
  If KeyCode = vbKeyControl Then
     m_ControlKey = True
  End If
End Sub

Private Sub Form_KeyUp(KeyCode As Integer, _
  Shift As Integer)
  '
  ' Watch for Control key, clear flag
  '
  If KeyCode = vbKeyControl Then
     m_ControlKey = False
  End If
End Sub

Private Sub mEdit_Click(Index As Integer)
  '
  ' Call generic routine to perform
  ' requested action. Same routine could
  ' be called from a toolbar event.
  '
  Select Case Index
     Case mUndo
        EditPerform WM_UNDO
     Case mCut
        EditPerform WM_CUT
     Case mCopy
        EditPerform WM_COPY
     Case mPaste
        EditPerform WM_PASTE
     Case mDelete
        EditPerform WM_CLEAR
  End Select
End Sub

Private Sub EditPerform(EditFunction As Integer)
  '
  ' A "wrapper" function for SendMessage
  ' Requests function passed in EditFunction
  ' Beeps if active control is not a textbox
  '
  If TypeOf Me.ActiveControl Is TextBox Then
     Call SendMessage(Me.ActiveControl.hWnd, _
        EditFunction, 0, 0&)
  ElseIf TypeOf Me.ActiveControl Is RichTextBox _
     Then
     If m_ControlKey = False Then
        Call SendMessage(Me.ActiveControl.hWnd, _
           EditFunction, 0, 0&)
     End If
  Else
     Beep
  End If
End Sub
```

LISTING 1 ***Providing Generic Edit Functionality.*** *The SendMessage API call is wrapped into a standalone routine, EditPerform, so you can call it from menus, toolbars, or anywhere else in the code.*

```
RichTextBox1.SelRTF = Clipboard.GetData(vbCFRTF)
```

Inconsistencies such as this have trained me to use the API for general clipboard support. I use a standard set of routines to code an Edit menu for text boxes, and it is a cinch to convert them to also support rich text boxes. The only extra consideration is the added support RichTextBoxes offer for edit shortcuts. To avoid this double support, I set the form's KeyPreview property to True and then toggle a form-level flag variable in the form's KeyDown and FormUp events whenever the user presses or releases the Control key.

Members of the free, Registered Level of The Development Exchange can download a complete demo that supports a standard Edit menu for both types of text boxes (see Figure 1, and the Code Online box at the end of the column for details on how to download the demo). In fact, the core idea is equally valid for toolbar buttons, because you can call the routines from anywhere in your code. You call a generic routine, EditPerform, with a single parameter specifying what type of edit to perform. EditPerform checks the ActiveControl to make sure it's either a TextBox or RichTextBox, then calls the SendMessage API to request the proper edit (see Listing 1). EditPerform skips the call to SendMessage if the ActiveControl is a RichTextBox and the Control key is depressed.

A major advantage of using SendMessage is that it provides the Undo support Visual Basic still lacks. Edit controls maintain a CanUndo flag internally, which you can also query with SendMessage. Because it's considered poor form to offer menu choices that aren't actually possible, it's always a good idea to check this flag before dropping an Edit menu.

You can use another routine in the demo to toggle the Enabled state of each of the standard menu items (see Listing 2). The EditMenuToggle routine enables Cut, Copy, and Delete if the SelLength property is greater than zero; these options would make no sense if no text were selected. Similarly, if there's text on the clipboard, the routine enables the Paste option. Only Undo availability requires resorting to an API.

Q HOW VB5 HANDLES DEFAULT PROJECTS

I recently upgraded to VB5 and would like to set the default new project to automatically include several files that our team uses as the base for our applications. We have written a lot of our own classes to facilitate our work, and it

would be helpful to include these at startup. We had this ability in VB4, but have not been able to locate it for VB5. In VB4/32, the default project name was auto32ld.vbp. I've found the templates for VBADDINs and all the ActiveX projects, but have not found one for a standard EXE. Does this default project exist for VB5, and if so, where can I find it?
—*Jay W. Smith, Bentonville, Arkansas*

A VB5 changed the way default projects are handled, and I think you'll like the change. You can now have any number of default projects, stored as templates under the VB5 folder. Start by putting together all the elements of what you consider a standard project. These can include elements such as forms, modules, classes, and controls. Now save each element in the appropriate folder beneath the \VB5\Template folder. In other words, save the VBP file in the \VB5\Template\Projects folder, your forms in \VB5\Template\Forms, classes in \VB5\Template\Classes, modules in \VB5\Template\Modules, and so on. Now whenever you select New Project under the File menu, the dialog that displays includes your custom project among the types to choose from. As a bonus, whenever you select Add-Form, Add-Class, or Add-Module, the forms, classes, and modules you saved in these special folders appear as choices in the New dialog. Pretty cool, huh?

VB3 VB4 16-bit 32-bit VB5

```
Private Sub EditMenuToggle()
  If TypeOf Me.ActiveControl Is TextBox Or _
     TypeOf Me.ActiveControl Is RichTextBox Then
     '
     ' Determine if last edit can be undone
     '
     Me.mEdit(mUndo).Enabled = SendMessage_
        (Me.ActiveControl.hWnd, EM_CANUNDO, 0, 0&)
     '
     ' See if there's anything to cut, copy,
     ' or delete
     '
     Me.mEdit(mCut).Enabled = _
        Me.ActiveControl.SelLength
     Me.mEdit(mCopy).Enabled = _
        Me.ActiveControl.SelLength
     Me.mEdit(mDelete).Enabled = _
        Me.ActiveControl.SelLength
     '
     ' See if there's anything to paste
     '
     Me.mEdit(mPaste) = _
        Clipboard.GetFormat(vbCFText)
  Else
     '
     ' If active control is not a textbox
     ' then disable all
     '
     Me.mEdit(mUndo).Enabled = False
     Me.mEdit(mCut).Enabled = False
     Me.mEdit(mCopy).Enabled = False
     Me.mEdit(mPaste).Enabled = False
     Me.mEdit(mDelete).Enabled = False
  End If
End Sub
```

LISTING 2 ***Don't Offer What's Not Available.*** *You can write variations on this routine to ensure that when your Edit menu drops, it accurately reflects what's actually possible. The routine uses SendMessage to check the CanUndo flag, and then checks native VB properties for the other options.*

HERE'S A QUICKER ROUNDING METHOD

In my column that appeared in the special Windows NT Enterprise Development issue [*VBPJ* Fall 1997], I presented one method to round up to the next higher multiple of 5. Well, the day after my issue arrived, the mail started coming in. Several of you, Shaun Morris of Pixel Translations being the first, sent me a more efficient algorithm. I'd like to share it with everyone:

```
Function RoundUpToFives (ByVal n As Long) As Long
   RoundUpToFives = ((n + 4) \ 5) * 5
End Function
```

Here, Shaun uses 4, as it's 5 minus 1. I agree that this one's much simpler to read, and as Shaun put it, "No obscure functions and only one division." Speedwise, Shaun's algorithm smoked—if tested inline. When I timed the difference of calling standalone functions, the new approach was only about three times faster. To put that in perspective, five million calls saved one second on my machine. Hey, we don't put our e-mail addresses on these columns for nothing! Thanks, and keep that feedback coming.

Code Online

You can find all the code published in this issue of* VBPJ *on The Development Exchange (DevX) at http://www.windx.com. For details, please see "Get Extra Code in DevX's Premier Club" in Letters to the Editor.

Copy and Paste with RichTextBox
Locator+ Codes

Listings ZIP file, including a complete demo that supports a standard Edit menu for both types of text boxes (free Registered Level): VBPJ1297

Listings for this article, the demo described above, plus the complete Clipboard Viewer application and source code (subscriber Premier Level): AP1297P