Issue Date: FoxTalk October 1997

The Rich Text Format (RTF) ActiveX Control

John V. Petersen

This month, John focuses on a feature in Visual FoxPro that's been eagerly awaited by many -- the ability to work with and display formatted text. The key to this functionality lies with the Rich Text Format ActiveX control. However, despite its powerful features, the RTF ActiveX control does need some improvement, particularly in the area of printing. John has subclassed the RTF control to make it easier for you to include it in your Visual FoxPro applications.

Like text and edit boxes, the RichTextBox control allows the user to enter and edit text, but it provides more advanced formatting features than the conventional TextBox control. For example, you can make text bold or italic, change its color, and create super- and subscripts. You can even adjust paragraph formatting by setting indents on both sides. Files can be opened and saved in both RTF and regular ASCII format.

If you don't already have it installed as an ActiveX control, you'll need to go into the Tools, Options menu and select the Microsoft Rich Text Control, Version 5.0, in the ActiveX list box in the Controls tab. (The file name is RICHTX32.OCX and can be found in the Windows\System directory.)

Key PEMS

Unlike last month's topic in this feature -- the Progress Meter ActiveX control, which had only three significant properties -- the RTF control has considerably more.

Properties

- AutoVerbMenu: If set to .T., the native right-click menu will be available. In regards to the RTF control, the options include the standard clipboard functions of Cut, Copy, and Paste.
- TextRtf: This property returns the RTF code of the currently loaded document.
- Text: This property returns the text-only portions of the currently loaded document, stripping out any RTF control codes.
- ControlSource: This property references the memo field to which the RTF control is bound.

Methods

- LoadFile(cFileName,nFlag): This method will load the referenced file into the RTF control. If nFlag is set to 0, an RTF file will be loaded; if nFlag is set to 1, a plain text file will be loaded.
- SaveFile(cFileName,nFlag): This method will save the currently loaded document with the name specified by cFileName. If nFlag is set to 0, an RTF document will be saved; if nFlag is set to 1, a plain text file will be saved.

Improving the interface

To improve the usefulness of the RTF control, I've created three classes. The first class, OLERTF, is a subclass of the base RTF control. The second class, CntFont, is a container devoted to specifying font attributes, such as name, size, bold, and color. The third class, OLERTFWordProcessor, brings the first two classes together to create a Visual FoxPro-based mini word processor. Details for each of these classes follow. (These classes are contained in the OLECLASSES class library, which is available in the accompanying **Download file**.)

OLERTF

This class is a direct subclass from the RTF control that ships with Visual FoxPro. To complement the features of the RTF control, I've added several custom properties and methods:

- cFileName: This property holds the name and path of the currently loaded file.
- lChanged: This logical property specifies whether pending changes exist.
- lNotify: This logical property specifies whether certain user messages are displayed to the user.
- oWord: This object property holds a reference to Microsoft Word.
- PrintDocument(): This method takes care of printing the currently loaded document.

The decision to use the Word.Basic class -- rather than Word.Application -- stems from the standpoint of backward

compatibility. Although Word 97 implements VBA and has an object model much like Excel, the Word.Basic class still works. As it turns out, the tasks of printing are far easier with the Word.Basic class!

Although the RTF control has a SelPrint() Method, it's not applicable to VFP. In fact, when Help is invoked, a Visual Basic Help File appears! Furthermore, the code samples are all VB-based. Methods like SelPrint() require a handle to the device context to which the contents of the RTF control will be sent. In this case, it's the printer. Visual Basic has a Printer Object and a Printers Collection; one of the properties of the Printer Object is Hdc, which contains the device handle. Therefore, the following is the VB code that would be used to implement the SelPrint() Method:

```
Printer.Print ""
RichTextBox1.SelPrint (Printer.hDC)
```

Unfortunately, VFP has no such object reference to installed printers. The alternative would be to resort to a series of API calls. In lieu of API calls, the custom PrintDocument() Method gets the job done nicely.

The following is the code listing for this class:

```
DEFINE CLASS olertf AS olecontrol
Height = 193
Width = 289
*-- Object Reference to MS Word.
oword = .NULL.
*-- This property holds the file name of the
*-- contained document.
cfilename = (Space(0))
Name = "olertf"
*-- If .T., various user notifcations will be displayed.
lnotify = .F.
*-- This property tracks when unsaved changes exist.
lchanged = .F.
*-- This method will print the RTF document to the printer.

PROCEDURE printdocument
Local lcTempFile
If !IsNull(This.oWord)
   lcTempFile = Sys(3)+".RTF"
   This.SaveFile(lcTempFile,0)
   With This.oWord
     .FileOpen(lcTempFile)
     .FilePrint()
     .FileClose()
   EndWith
   Erase (lcTempFile)
Else
   If This.lNotify
      MessageBox("A word processor to print this document;
                  is not currently available.", ;
                 "Cannot print document",16)
   Endif
Endif
Return
ENDPROC

PROCEDURE KeyPress
*** OLE Control Event ***
LPARAMETERS keyascii
This.lChanged = .T.
ENDPROC

PROCEDURE Init
Local lcOldErr,llErr
lcOldErr = On("error")
On Error llErr = .T.
This.oWord = CreateObject("word.basic")
If llErr
   This.oWord = .NULL.
Endif
On Error &lcOldErr
Return
ENDPROC

PROCEDURE Destroy
  If !IsNull(This.oWord)
     This.oWord.FileQuit()
     This.oWord = .NULL.
```

```
  Endif
ENDPROC
ENDDEFINE
```

CntFont
The purpose of this class is to select a set of font attributes. This class isn`t meant to be used on its own. Rather, an object that has the ability to display fonts interrogates the CntFont class to update its property settings. **Figure 1** illustrates how the CntFont class appears.

***Figure 1:* The CntFont class makes it very easy to select various font attibutes for the custom RTF subclass.**

The key method of the CntFont class is called SetAttribute(). This method maps the values of the individual controls to properties of the CntFont container. These property values, in turn, will update the associated font properties of the OLERTF class.

OLERTFWordProcessor
This class is where everything comes together -- this container class joins the OLERTF and CntFont classes. In addition, I've added a few new methods to the OLERTFWordProcessor class. **Figure 2** illustrates how the OLERTFWordProcessor class appears.

***Figure 2:* RTF.SCX illustrates how the OLERTFWordProcessor class works.**

The OLERTFWordProcessor class includes CommandButtons to open, save, and print. While the OLERTF class provides the printing services, the print CommandButton actually provides the access to this functionality. In addition, the OLERTFWordProcessor also has methods named LoadFile() and SaveFile(), which in turn call the native methods of the contained RTF class. These methods incorporate the GetFile() and PutFile() functions. To remain consistent with many Windows 95 applications, the OLERTFWordProcessor class comes complete with a right-click menu.

A word on the Clipboard functions
It appears that keyboard commands used to simulate keystrokes, such as Ctrl-V for Paste, are not recognized by the RTF control. Invoking the keystrokes manually does work. However, it would be nice to programmatically perform these tasks, such as through the use of a pop-up menu. While this can't be done with native VFP code, this task can be accomplished with the AutoVerbMenu Property. Setting this property to .T. will make the RTF control's pop-up menu available when the right mouse button is clicked. The functions on this menu include the Clipboard functions of Cut, Copy, and Paste.

But the RTF control has a ControlSource property...
Although the RTF ActiveX control has a ControlSource property, it doesn't work as expected. Every time you attempt to bind a memo field directly to the RTF control, you'll receive the following error message: "Function, argument, type, or count is invalid." Once this error occurs, an unbinding process takes place. Interestingly, no error occurs if an attempt is made to bind the RTF control to a character field; unfortunately, a character field is not feasible for storing RTF documents. The solution is clear -- the existing RTF class must be equipped with additional properties and methods for storing RTF documents in a memo field.

Here's what I've added:

BoundField Property
This property mimics the ControlSource property. Simply enter the memo field to which the RTF class should reference.

RefreshDisplayFromBoundField() Method
This method takes the RTF code stored in the bound memo field for the current record and refreshes the TextRTF property of the RTF class. Here's the code for this method:

```
This.TextRtf = Eval(This.BoundField)

Return
```

UpdateBoundField() Method
This method takes the contents of the TextRTF property and stores the code in the bound memo field for the current record. Here's the code for this method:

```
Replace (This.BoundField) With This.TextRtf

Return
```

Summary
Necessary functionality, such as the ability to print, can be obtained by subclassing the RTF control. By augmenting the functionality with additional methods and combining the control with other classes, a powerful word processing tool is available to any VFP application.