November, 2005

Advisor Answers

Using textbox for email address

VFP 9/8

Q: I need to have a textbox configured so that when the user types in an email address, he can click on the email address and go to the default email program. I know you can set the textbox property EnableHyperlinks to True and the put "mailto:" in front of the email address and it will work that way. But I don't want users to have to type
"mailto:" in front of the email address.

–James Lansden (via Advisor forums)

A: This is an intriguing little problem. While setting up the Click to start a new email is fairly simple, getting the textbox to behave as users expect is a little trickier.

There are a number of prefixes you can add to the front of a string to indicate that the string is a hyperlink. In addition to "mailto:", you can use "http://" to indicate a web address, "ftp://" to indicate a file transfer address and "news:" to point to a newsgroup.

Let me start with a quick look at the EnableHyperlinks property. Added in VFP 8, this property of textboxes and editboxes determines whether strings that look like links are treated as links. When EnableHyperlinks is .T., any string beginning "www." or starting with one of the other link prefixes is underlined and clickable. Depending on the _VFP.EditorOptions setting, you can execute the link with either Ctrl+Click or just a click.

So why not use this property? As you note, the problem is that it requires the prefix for anything other than a web address beginning with "www." Most users aren't familiar with the "mailto:" prefix, so you rightly don't want them to have to type it. Fortunately, as is so often the case with FoxPro, there are alternatives.

My immediate reaction to your problem was that you could handle it in just a few steps. First, make sure you actually have an email address. If so, add the "mailto:" tag at the front. Then, use ShellExecute to call on the default email client. In fact, this is more or less the outline of the solution, but as I worked on it, I realized that it could be far more generic.

ShellExecute is an API function that lets you run the default application for a file name or string. Rather than using it directly, you can take advantage of the _ShellExecute class in the FoxPro Foundation Classes. This class handles all the details of calling ShellExecute. All you have to do is instantiate it and call its ShellExecute method, passing the appropriate string.

The basic strategy for turning a textbox value into a link is the same no matter which type of link it is. So, I created an abstract textbox subclass that contains almost all the code for dealing with links. Then, I subclassed for different types of links.

The abstract class is called txtLink. It has four custom properties:

- cPrefix—the prefix to add for this type of link. Empty in txtLink and set in the subclasses.
- cUnderline—the name of a line object created to serve as the underline for this textbox.
- lIsLink—indicates whether the current value of the textbox is a link of the appropriate type.
- oShellExecute—object reference to a _ShellExecute object.

Two custom methods combine to execute the appropriate action when the user clicks on the textbox value. FollowLink adds the prefix, if necessary, and calls the _ShellExecute object's ShellExecute method.

```
LOCAL cLink, nPrefixLen

nPrefixLen = LEN(This.cPrefix)

IF UPPER(LEFT(This.Value, nPrefixLen))== ;
   UPPER(This.cPrefix)
   cLink = This.Value
ELSE
   cLink = ALLTRIM(This.cPrefix) + ALLTRIM(This.Value)
ENDIF

IF This.GetShellExecute()
   This.oShellExecute.ShellExecute(cLink)
ENDIF
```

GetShellExecute makes sure there's a reference to a _ShellExecute object, instantiating one if necessary. Once it's instantiated, the reference is saved, so that future clicks can use the same _ShellExecute object.

```
IF VARTYPE(This.oShellExecute) <> "O"
   This.oShellExecute = NEWOBJECT("_ShellExecute", ;
                        HOME() + "Ffc\_Environ.vcx")
ENDIF
```

The Click method calls this code:

```
IF This.lIsLink
   This.FollowLink()
ENDIF
```

Another custom method, CheckFormat, examines the textbox's value to see whether it has the appropriate format for a link of the specified type. This method is abstract at this level; it needs to be filled in with the appropriate test in each subclass.

The remaining code in txtLink deals with visual issues. To match user expectations, when the textbox contains a link, the value should be underlined. In addition, when clicking the value will fire the link, the mouse pointer should indicate that.

MouseEnter and MouseLeave handle the mouse pointer. MouseEnter checks whether the current value is a link and if so, switches to the arrow pointer:

```
IF This.lIsLink
   This.MousePointer = 15
ENDIF
```

MouseLeave resets the pointer to its default value:

```
This.MousePointer = 0
```

Underlining the value to indicate a link is a little more complicated. My first approach used FontUnderline and InputMask to make the textbox underline itself. It turned out that getting it exactly right was difficult.

Instead, when the value of the textbox constitutes a link, I display a line object beneath it. InteractiveChange handles adding and resizing the line, as well as hiding it when the value no longer constitutes a link. Here’s the code:

```
LOCAL oUnderline, lChanged, cStyle

This.lIsLink = This.CheckFormat()
IF This.lIsLink
   cStyle = ""
   IF This.FontBold
      cStyle = cStyle + "B"
   ENDIF
```

```
IF This.FontItalic
   cStyle = cStyle + "I"
ENDIF

IF EMPTY(This.cUnderline)
   This.cUnderline = "linUnderline" + ;
     ALLTRIM(This.Name)
   ThisForm.AddObject(This.cUnderline,"Line")
ENDIF

oUnderline = EVALUATE("ThisForm." + This.cUnderline)
WITH oUnderline
   .Left = This.Left + 5
   .Top = This.Top + FONTMETRIC(1, ;
          This.FontName,This.FontSize,cStyle)+ 1
   .Height = 0
   .BorderColor = RGB(0,0,255)
   .Visible = .T.

   * Adjust form font, if necessary
   IF ThisForm.FontName <> This.FontName OR ;
      ThisForm.FontSize <> This.FontSize OR ;
      ThisForm.FontBold <> This.FontBold OR ;
      ThisForm.FontItalic <> This.FontItalic

         * Save current settings
         lChanged = .T.
         WITH ThisForm
            .LockScreen = .T.
            cFontName = .FontName
            nFontSize = .FontSize
            lFontBold = .FontBold
            lFontItalic = .FontItalic

            .FontName = This.FontName
            .FontSize = This.FontSize
            .FontBold = This.FontBold
            .FontItalic = This.FontItalic
         ENDWITH
   ENDIF

   .Width = ThisForm.TextWidth(ALLTRIM(This.Value))
   IF oUnderline.Width > This.Width - 10
      * Max line out at end of textbox
      oUnderline.Width = This.Width - 10
   ENDIF

   * Now reset if necessary
   IF lChanged
      WITH ThisForm
         .FontName = cFontName
         .FontSize = nFontSize
         .FontBold = lFontBold
         .FontItalic = lFontItalic
         .LockScreen = .F.
         ENDWITH
      ENDIF
   ENDWITH

ELSE
   IF NOT EMPTY(This.cUnderline)
      oUnderline = EVALUATE("ThisForm." + ;
         This.cUnderline)
      oUnderline.Visible = .F.
   ENDIF
ENDIF
```

Keeping the line displayed is a little tricky. A number of things can make it disappear. To ensure it’s visible when it should be, I added a method called RepaintLine and call it from the Click and KeyPress methods. RepaintLine is simple:

```
IF NOT EMPTY(This.cUnderline)
   oUnderline = EVALUATE("ThisForm." + This.cUnderline)
   oUnderline.Visible = .t.
ENDIF
```

KeyPress requires a little special handling. You need to call RepaintLine after the keystroke has been processed. This code does the trick:

```
LPARAMETERS nKeyCode, nShiftAltCtrl

DODEFAULT(nKeyCode, nShiftAltCtrl)
IF (m.nKeyCode==9 and m.nShiftAltCtrl==0) OR ;
   (m.nKeyCode==15 and m.nShiftAltCtrl==1)
   * Do nothing to make TAB and SHIFT+TAB work
ELSE
    NODEFAULT
ENDIF

This.RepaintLine()
```

Even with all this, there's still one situation where the line disappears. When the value is long enough to scroll the textbox, the line disappears until the textbox loses focus. I've tried a couple of tricks to get it back, but nothing works reliably, so make sure to size the textbox large enough for the expected contents. (My thanks to Christof for suggesting the line object and helping me work out some of the visual issues.)

With all this code in place, the subclasses of txtLink don't require much work. Set cPrefix to the appropriate string, and put code in CheckFormat to check for a valid link of the specified type.

I created two subclasses, txtMailtoLink and txtHyperlink. In txtMailtoLink, cPrefix is set to "Mailto:" and CheckFormat contains very basic code to check for an email address. A string is considered a valid email link if it contains the character "@", followed at some point by a period. (For a more rigorous approach, see Pamela's answer in the July, 2005 issue.)

```
LOCAL lReturn, cValue

lReturn = .F.
cValue = This.Value

IF "@"$cValue AND "."$SUBSTR(cValue, AT("@", cValue))
   lReturn = .T.
ENDIF

RETURN lReturn
```

In txtHyperlink, cPrefix is set to "http://" and CheckFormat contains code that checks that the string either contains a period or begins "http". Again, you may want to apply a more rigorous test. It doesn't check for "www" because some web addresses don't require that string.

```
* Check whether current value is a web address.

LOCAL lReturn, cValue

cValue = This.Value
lReturn = .F.

IF "."$cValue OR UPPER(LEFT(cValue, 4))="HTTP"
   lReturn = .T.
ENDIF

RETURN lReturn
```

All three classes (txtLink, txtMailtoLink and txtHyperlink in WebControls.VCX) are included on this month's Professional Resource CD, along with a very simple form that demonstrates their use. I'll leave it to you as exercise to create the ftp and news subclasses.

–Tamar