A New IDE Add-on: FoxTabs

Doug Hennig

FoxTabs provides easy access to all open windows in your VFP IDE. However, not only is it a great tool, it uses some very cool techniques to do its magic, including Windows message binding and event-driven object messaging.

I don't know about you, but my VFP Interactive Development Environment (IDE) never seems big enough, regardless of the size of my monitor or the resolution I run at. I always have a half-dozen or more windows open at a time: at least one project window, a PRG or two, a couple of classes and some code editing windows for their methods in the Class Designer, and so on. Trying to find the window I want either means poking around the open windows on the desktop or a trip to the Window menu.

A new VFP add-on called FoxTabs provides a great solution to this universal problem. FoxTabs is the first component of a cool community project called VFP Solution Explorer, one of the most eagerly anticipated tools for VFP in a long time. FoxTabs was created by Scott Scovell of Talman Pty Ltd. in Australia. For more information about VFP Solution Explorer, or to download FoxTabs, please visit http://www.vfpsolutionexplorer.com/.

To run FoxTabs, simply DO FOXTABS.APP. **Figure 1** shows it in action. A docked toolbar at the bottom of the VFP window has one tab for each open window, plus a Desktop tab. Opening a window, such as editing a program, adds a tab for the window to the toolbar. When you close a window, it's removed from the toolbar. Clicking a tab brings that window to the top of any other windows on your VFP desktop. Clicking the Desktop tab hides all open windows; restoring a single window is as easy as clicking its tab. Right-clicking a tab invokes a shortcut menu with Save, Save All, Close, and Close All functions. Another menu is available by clicking the up arrow to the left of the Desktop tab. This menu lists each open window (selecting one from the menu is like clicking its tab) and provides Close All, Save All, Recent Files, Options, and Exit functions. Recent Files has submenus by file type—Project, Class, Program, and so on—each providing a list of the most recently opened files of that type. The Options function opens the Options dialog, which provides configuration settings such as tab font and color and what type of recent files to list.

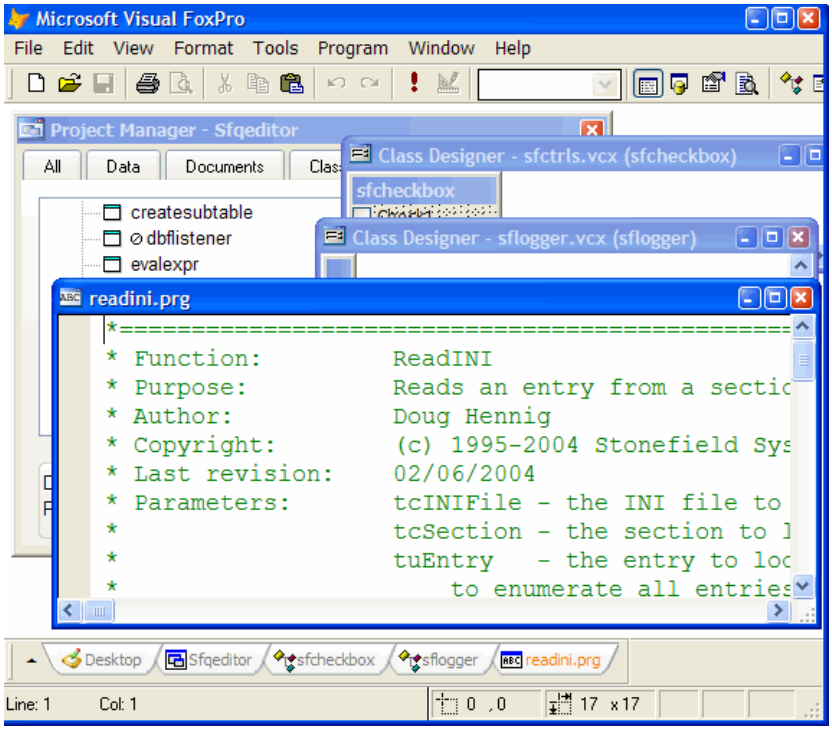

Figure 1. FoxTabs manages your FoxPro IDE, providing fast access to any open window.

Being a hacker (in the good sense), I actually think the coolest part of FoxTabs is not how it works, but how it was built. Let's go under to hood to check out the design.

The secret is in the binding

One of the first questions you might ask about FoxTabs is how it does its magic. It's easy to figure out how it brings a window to the top when you click its tab: ACTIVATE WINDOW TOP takes care of that. But how does it know when a window is opened or closed so it can update the toolbar? That happens through a new VFP 9 feature: Windows event binding. I discussed this topic in my January 2005 FoxTalk article titled "Windows Event Binding Made Easy." Please review that article or the VFP help for the BINDEVENT() function if necessary, as the rest of this article assumes you know how this works.

FoxTabsApplication

FoxTabsApplication.PRG is the main program in FoxTabs. It instantiates the FoxTabsApplication class into a public variable named oFoxTabs, calls its Main method, and then exits.

FoxTabsApplication, defined in FoxTabsApplication.PRG, is the topmost class in the FoxTabs hierarchy. It really just serves one purpose: providing the glue between the user interface and the Windows event handling engine. Its Main method instantiates a configuration settings class so saved settings can be restored, then instantiates the FoxTabsToolbar class, which provides the FoxTabs user interface, and the FoxTabsManager class, which handles Windows events. Finally, it sets up Windows event handling, loads all open windows into tabs in the toolbar, and makes the toolbar visible.

FoxTabsManager

The FoxTabsManager class, based on Custom and contained in FoxTabsManager.PRG, is responsible for setting up and managing Windows event handling for FoxTabs. It's instantiated in the Main method of FoxTabsApplication into that class' FoxTabsManager member. FoxTabsManager has two members: FoxTabs, a collection of objects containing information about each managed window, and WindowsEvents, a instance of the FoxTabsEventHandler class we'll see later, that provides Windows event handling.

The SetBindings method, called by FoxTabsApplication shortly after instantiation, binds the WM_CREATE Windows event to the WMEventHandler method of its WindowsEvents member, so whenever a window is created in the VFP IDE (for example, opening a project, creating a program, or modifying a class), that method fires. It also binds to the four events the WMEventHandler method raises when various Windows events occur. Thus, it delegates Windows event handling to the FoxTabsEventHandler instance in its WindowsEvents member, but receives notification when certain events it's interested in occur.

```
Function SetBindings()

* Bind to all WM_CREATE messages
BindEvent(0, WM_CREATE, This.WindowsEvents, "WMEventHandler")

* Setup event bindings to our Windows message event
* handler
BindEvent(This.WindowsEvents, "WindowShowEvent", This, "WindowShow")
BindEvent(This.WindowsEvents, "WindowSetTextEvent", This, "WindowSetText")
BindEvent(This.WindowsEvents, "WindowSetFocusEvent", This, "WindowSetFocus")
BindEvent(This.WindowsEvents, "WindowDestroyEvent", This, "WindowDestroy")
```

The LoadWindows method, also called by FoxTabsApplication shortly after instantiation, ensures any existing windows are added to FoxTabs when it's started. LoadWindows goes through all open IDE windows and for each acceptable one (any editing window, such as a PRG, the Class or Form Designers, the Database Designer, and so forth), calls the NewFoxTab method. NewFoxTab adds a new FoxTab object to its FoxTabs collection to maintain information about the window, raises the AddFoxTabEvent event, and asks its WindowsEvents member to bind the WM_SETFOCUS, WM_SETTEXT, and WM_DESTROY Windows events for that window to the WMEventHandler method of that member. Here's most of the code for NewFoxTab (some error handling code is omitted):

```
Function NewFoxTab(hWnd As Integer)
```

```
Local oException As Exception
Local oFoxTab As FoxTab
Try
  * Check if a FoxTab already exists for this handle
  If Not Empty(This.FoxTabs.GetKey(Transform(hWnd, "@0x")))
    Exit
  EndIf

  * Create a new instance of our FoxTab class
  oFoxTab = NewObject("FoxTab")

  * Set windows handle and window name properties
  oFoxTab.hWnd = Transform(hWnd, "@0x")
  oFoxTab.WindowName = This.getWindowTitle(hWnd)

  * Add the class to our collection
  This.FoxTabs.Add(oFoxTab, Transform(hWnd, "@0x"))

  * Raise the add FoxTab event
  RaiseEvent(This, "AddFoxTabEvent", oFoxTab)

  * Setup event bindings for this window
  This.WindowsEvents.SetBindings(hWnd)
EndTry
```

Before we look at how FoxTabsManager deals with Windows events in windows it manages, let's look at the class responsible for handling these events.

FoxTabsEventHandler

The FoxTabsEventHandler class is responsible for receiving and handling the Windows events FoxTabs is bound to. Interestingly, there isn't a lot of code in this class; when it receives notification of a Windows event it's prepared to handle, it raises an event of its own, which FoxTabsManager binds to. Thus, although Windows event handling is delegated to FoxTabsEventHandler, FoxTabsManager receives notification of these events as well.

As I mentioned earlier, the SetBindings method of FoxTabsManager binds the WM_CREATE Windows event to the WMEventHandler method of its FoxTabsEventHandler instance. WMEventHandler is a typical Windows event handler; it uses a CASE statement to handle the different types of Windows events it's called for. When a window is created (the WM_CREATE event), WMEventHandler raises the WindowCreateEvent event and binds itself to WM_SHOWWINDOW and WM_SETTEXT events of the new window. When the window is displayed (the WM_SHOWWINDOW event), WMEventHandler raises the WindowShowEvent event and unbinds itself from WM_SHOWWINDOW. When the window receives focus (the WM_SETFOCUS event), the window's caption is set or changed (WM_SETTEXT), or the window is destroyed (WM_DESTROY), WMEventHandler raises the appropriate event: WindowSetFocusEvent, WindowSetTextEvent, or WindowDestroyEvent. If an error occurs, WMEventHandler raises the LogError event. Finally, regardless of what happened, the Windows event is passed back to VFP using CallWindowProc, as I discussed in my January 2005 FoxTalk article.

```
Function WMEventHandler(hWnd As Integer, Msg As Integer, wParam As Integer, ;
  lParam As Integer)
Local oException As Exception
Local lnReturn As Integer
lnReturn = 0
Try
  * Handle each windows message case
  Do Case
    Case Msg = WM_CREATE
      * Raise the window create event
      RaiseEvent(This, "WindowCreateEvent", hWnd)

      * Bind to these events so we can add it to our
      * collection
      BindEvent(hWnd, WM_SHOWWINDOW, This, "WMEventHandler", 4)
      BindEvent(hWnd, WM_SETTEXT, This, "WMEventHandler", 4)
```

```
    Case Msg = WM_SHOWWINDOW
      If wParam # 0
        * Raise the window show event
        RaiseEvent(This, "WindowShowEvent", hWnd)
      EndIf

      * Unbind to this event as we do not require it
      * any more
      UnBindEvents(hWnd, WM_SHOWWINDOW)

    Case Msg = WM_SETFOCUS
      * Raise the window set focus event
      RaiseEvent(This, "WindowSetFocusEvent", hWnd)

    Case Msg = WM_SETTEXT
      * Raise the window set text event
      RaiseEvent(This, "WindowSetTextEvent", hWnd)

    Case Msg = WM_DESTROY
      * Raise the window destroy event
      RaiseEvent(This, "WindowDestroyEvent", hWnd)
  EndCase

Catch To oException
  * Raise error event
  RaiseEvent(This.Parent.Parent, "LogError", ;
    oException, "Exception caught while handling " + ;
    "windows message event." + Chr(13) + ;
    " hWnd:"   + Transform(hWnd, "@x0") + ;
    " Msg:"    + Transform(Msg, "@x0") + ;
    " wParam:" + Transform(wParam, "@x0") + ;
    " lParam:" + Transform(lParam, "@x0"))

Finally
  * Must pass the message on
  lnReturn = CallWindowProc(This.PrevWndFunc, hWnd, Msg, wParam, lParam)
EndTry
Return lnReturn
```

Windows event handling

When a Windows message is sent to a window managed by FoxTabs, WMEventHandler raises one of its own events. FoxTabsManager binds to these events so it's notified that something happened. WindowShow, bound to the WindowShowEvent event of FoxTabsEventHandler, which is raised when a window is created, calls the NewFoxTab method we looked at earlier to add a new member to the FoxTabs collection. WindowSetText is bound to the WindowSetTextEvent event of FoxTabsEventHandler, which is raised when the caption of a window is set or changed, so it changes the WindowName property of the appropriate FoxTab object in the FoxTabs collection and raises the OnChangeEvent event. WindowSetFocus, bound to the WindowSetFocusEvent event of FoxTabsEventHandler, raises the GotFocusEvent event. WindowDestroy is bound to the WindowDestroyEvent event of FoxTabsEventHandler; it removes the appropriate FoxTab object in the FoxTabs collection, raises the RemoveFoxTabEvent event, and unbinds all event handling for the window being closed.

That's it for FoxTabsManager. At the top level, when something happens to one of the windows it manages, one of four events—AddFoxTabEvent, OnChangeEvent, GotFocusEvent, or RemoveFoxTabEvent—fire. It's up to the user interface classes of FoxTabs to decide what to do about these events. Let's see how that works.

FoxTabsToolbar

FoxTabsApplication instantiates an instance of the FoxTabsToolbar class into its FoxTabsToolbar member. FoxTabsToolbar, a subclass of Toolbar located in FoxTabs.VCX, provides the main visual interface for FoxTabs (see Figure 1). Shortly after instantiating FoxTabsToolbar, FoxTabsApplication binds the AddFoxTabEvent, OnChangeEvent, GotFocusEvent, and RemoveFoxTabEvent events of FoxTabsManager to the AddFoxTab, OnChange, GotFocus, and RemoveFoxTab methods of FoxTabsToolbar. This shows one of the benefits of the event handling strategy employed by FoxTabs: neither FoxTabsManager,

responsible for handling Windows events, nor FoxTabsToolbar, responsible for the user interface, know anything about each other. FoxTabsManager publishes events without knowing who subscribes to them and FoxTabsToolbar subscribes to events without knowing who publishes them. FoxTabs could easily be extended to provide additional handlers for events without changing a line of code in the existing classes. What a flexible design!

Tabs are represented in the toolbar using instances of the FoxTabControl class (also located in FoxTabs.VCX), which uses images and a label to create a tab-like appearance. The AddFoxTab method, which fires automatically when a window is added to FoxTabs, instantiates a FoxTabControl object and calls various methods to set up the tab, including the caption for the label and the icon for the tab. GotFocus, called when a window receives focus, calls the SetFocus method of the appropriate FoxTabControl object so its label appears in the "active" color. OnChange is fired when the title of a window changes; it calls the SetWindowName method of the appropriate FoxTabControl object so its label has the correct name. RemoveFoxTab removes the appropriate FoxTabControl object when its associated window is closed.

The other methods in FoxTabsToolbar are primarily support methods, called from FoxTabControl objects or shortcut menu functions.

Other interesting things

Here are a few other interesting things about FoxTabs:

- The most recently used files displayed in the options menu are obtained from your resource file using the FoxResource class I discussed in my October 2004 FoxTalk article, "Mining for Gold in XSource."
- The Save and Close functions are implemented using SYS(1500), which programmatically fires the Save and Close functions in the File menu.
- FoxTabs implements a very interesting error handling mechanism: rather than using an Error method, potentially error-prone code is wrapped in TRY structures, and the CATCH block raises a custom LogError event in FoxTabsApplication. The LogError method simply logs the error to a text file, but because it uses RAISEEVENT() rather than a method call, other objects could easily bind to this event for different types of error handling needs.
- Configuration settings (those items displayed in the Options dialog) are stored in an XML file named FoxTabs.Config, the structure of which is modeled after Web.Config in ASP.NET applications. These settings are managed by a FoxTabs-specific subclass of a generic ConfigurationBlock class, which exposes settings as objects in a collection, making them easy to access.
- Because of its cool, event-based design, extending FoxTabs is pretty simple. Just for grins, I created an add-on in FoxTabsAddOn.PRG that simply displays events to the screen using WAIT WINDOW. (Of course, a real add-on would do something more useful.) FoxTabsAddOn.PRG adds a new FoxTabsDebugAddOn object to the oFoxTabs object and binds events between the two. The only complication in this class is that the name of a window is initially "*," so this code doesn't display the window's name until it's assigned one. To use this add-on, DO FoxTabsAddOn.PRG after running FoxTabs.APP.

```
if type('oFoxTabs.Name') = 'C'
  oFoxTabs.AddObject('FoxTabsDebugAddon', 'FoxTabsDebugAddon')
  bindevent(oFoxTabs.FoxTabsManager, 'AddFoxTabEvent', oFoxTabs.FoxTabsDebugAddon, ;
    'AddFoxTab')
  bindevent(oFoxTabs.FoxTabsManager, 'RemoveFoxTabEvent', oFoxTabs.FoxTabsDebugAddon, ;
    'RemoveFoxTab')
  bindevent(oFoxTabs.FoxTabsManager, 'GotFocusEvent', oFoxTabs.FoxTabsDebugAddon, ;
    'GotFocus')
  bindevent(oFoxTabs.FoxTabsManager, 'OnChangeEvent', oFoxTabs.FoxTabsDebugAddon, ;
    'OnChange')
endif type('oFoxTabs.Name') = 'C'
```

```
define class FoxTabsDebugAddon as Custom
  cNewWindow = ''

  function AddFoxTab(toFoxTab)
    This.cNewWindow = '*'
  endfunc

  function RemoveFoxTab(toFoxTab)
    This.DisplayMessage(toFoxTab.WindowName, 'is no longer managed by FoxTabs')
  endfunc

  function GotFocus(toFoxTab)
    This.DisplayMessage(toFoxTab.WindowName, 'just received focus')
  endfunc

  function OnChange(toFoxTab)
    do case
      case empty(This.cNewWindow)
        This.DisplayMessage(toFoxTab.WindowName, 'was changed')
      case toFoxTab.WindowName <> '*'
        This.DisplayMessage(toFoxTab.WindowName, 'is now managed by FoxTabs')
        This.cNewWindow = ''
    endcase
  endfunc

  function DisplayMessage(tcWindowName, tcAction)
    wait window 'Window ' + tcWindowName + ' ' + tcAction timeout 0.5
  endfunc
enddefine
```

- Although I haven't tried this, I think FoxTabs could be used in your own applications as a visual form manager. You probably wouldn't need FoxTabsManager or even FoxTabsApplication; you'd simply call methods of FoxTabsToolbar. For example, when the user opens a form, you call the AddFoxTab method to create a tab for it. When the user closes the form, you call RemoveFoxTab. When the form receives focus, you call GotFocus. You could either call these methods directly from an application object or use event binding to bind to the appropriate events of the forms, similar to what FoxTabsManager does for VFP IDE windows.

Summary

FoxTabs is not only a wonderful addition to the VFP IDE, it also provides a great example of some very cool code techniques, including extensive use of custom events, Windows event handling, interesting error handling ideas, and configuration settings management. I suggest you add FoxTabs to your developer toolbox, and keep an eye on www.vfpsolutionexplorer.com for news about VFP Solution Explorer, a tool I'm sure I'll be writing about (and using) in the near future.

Doug Hennig is a partner with Stonefield Systems Group Inc. and Stonefield Software Inc. He is the author of the award-winning Stonefield Database Toolkit (SDT); the award-winning Stonefield Query; the MemberData Editor, Anchor Editor, and CursorAdapter and DataEnvironment builders that come with Microsoft Visual FoxPro; and the My namespace and updated Upsizing Wizard in Sedna. Doug is co-author of the "What's New in Visual FoxPro" series, "The Hacker's Guide to Visual FoxPro 7.0," and the soon-to-be-released "Making Sense of Sedna and VFP 9 SP2." He was the technical editor of "The Hacker's Guide to Visual FoxPro 6.0" and "The Fundamentals." Doug wrote over 100 articles in 10 years for FoxTalk and now writes for FoxRockX (http://www.foxrockx.com). He spoke at every Microsoft FoxPro Developers Conference (DevCon) since 1997 and at user groups and developer conferences all over the world. He is one of the organizers of the annual Southwest Fox conference (http://www.swfox.net). He is one of the administrators for the VFPX VFP community extensions Web site (http://www.codeplex.com/VFPX). He has been a Microsoft Most Valuable Professional (MVP) since 1996. Doug was awarded the 2006 FoxPro Community Lifetime Achievement Award (http://fox.wikis.com/wc.dll?Wiki~FoxProCommunityLifetimeAchievementAward).