Put Access methods to work

Access and Assign methods give you a chance to intervene when a property is referenced or changed. Access methods are particularly useful for just-in time calculations.

Tamar E. Granor, Ph.D.

In my last two articles, I demonstrated BindEvent(), the VFP function that lets you set up a method to respond when an event fires. In this article and the next, I'll take a look at Access and Assign methods, another VFP approach to provide automatic behavior.

Access and Assign methods were introduced in VFP 6, and essentially allow you to define your own events in an application. Any property can have an Access method, an Assign method or both. The Access method fires whenever the property is referenced, while the Assign method fires whenever the property changes. Each of them allows you to change the value of the property as well as take other actions.

These methods are named by combining the property name, an underscore, and the word "Access" or "Assign." For example, an Access method for the property Enabled is called Enabled_Access.

The Assign method receives the new value (the one you're assigning) as a parameter. In the code, you can assign any value you want to the property, not just the one it receives as a parameter. So, among other things, you can use the Assign method to make a property read-only or read-only under some circumstances. Most often, though, I use Assign methods to ensure that certain things happen when the value of the property changes. In other words, an Assign method becomes an event upon which I can take action.

In the Access method, you can control what value the triggering code sees as the value of the property. The value you return from the Access method is used by the triggering code, even if the actual value stored in the property is different. For example, you could translate the value into a different language based on an application or system setting, or convert it from a convenient internal storage mechanism to a friendlier user format (say, numeric to character). Most often, I use Access methods to ensure that a property is up-to-date based on current settings. That is, an Access method lets me avoid having to ensure that a property gets updated every time the things it depends on change. I do the update when I actually need the value.

Adding Access and Assign methods to a property is somewhat different from adding custom methods to an object. In the Property Sheet, right-click on the property in question, and choose Edit Property/Method. In the Edit Property/Method dialog, there are checkboxes for Access Method and Assign Method; check one or both. Access methods are created with a single line of code that returns the property's value. Assign methods are created with the new value passed as a parameter and a single line assigning that value to the property. Once they exist, you can edit the code as you desire.

Many of the examples in this article are drawn from the same Library application used for the BindEvent() articles; the code is included in this month's downloads.

Using Access methods

As I said earlier, my principal use for Access methods is to update a property when it's needed. However, they also offer a workaround for a VFP bug.

Just-in-time updating

When the value of a property is based on other items that can change as the application runs, an Access method lets us look up or compute the value when it's needed. Doing so serves several purposes. First, we don't have to insert calls (or use Assign methods) to update the property every time one of its components changes. Second, it's a kind of encapsulation. Only the property has to know how to find its own value. If the rules for generating the value change, the only thing we have to change is the Access method (or a custom method it calls).

For example, in Figure 1. the block that says "Minor Alarm" indicates the overall status of the displayed node (a node here is a utility substation); the determination of a node's status involves check-

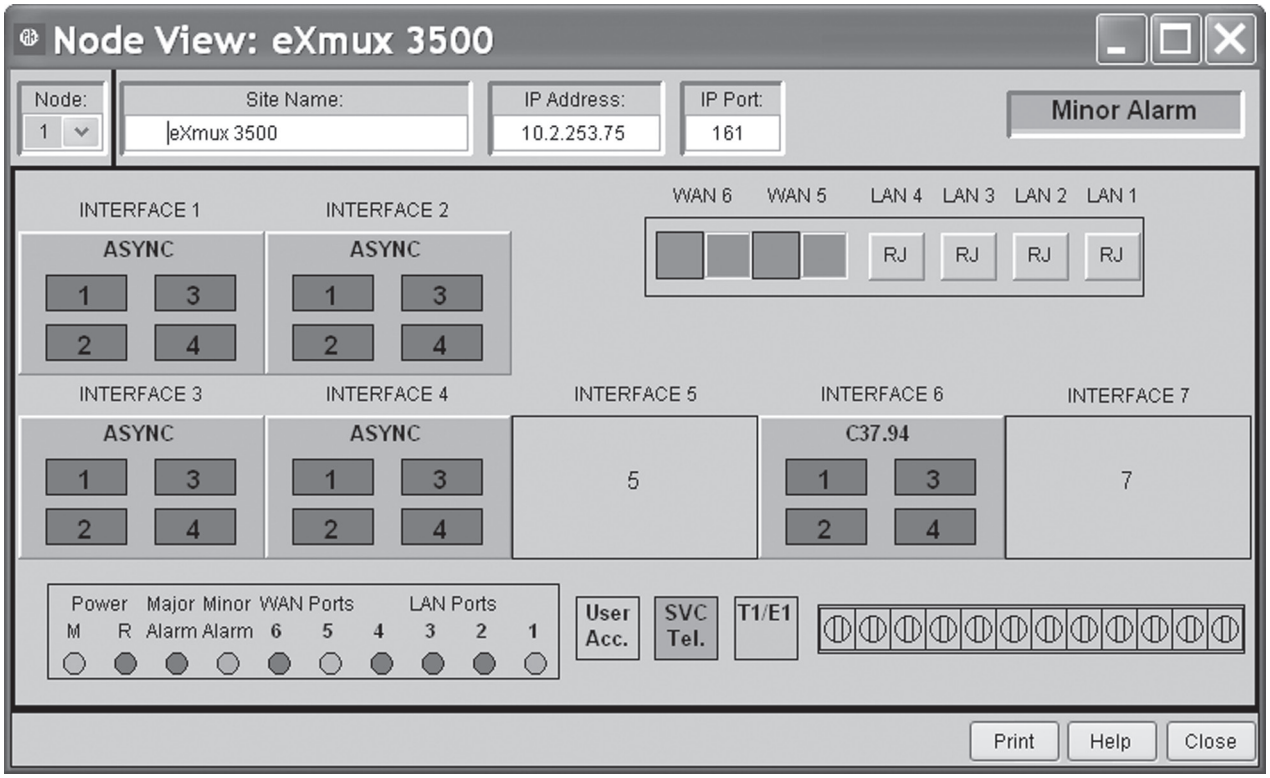

Figure 1. The node status shown in the upper right corner of this form is computed each time the form is refreshed. An access method for the node's cStatus property calls the appropriate method.

ing a number of conditions. The application monitors the corresponding hardware and when there are changes, updates the form. The node's status is stored in a property called cStatus; cStatus's Access method (cStatus_Access) calls another method that computes the current status and returns the computed value. The form then displays that value and colors the background around it appropriately (red for "Major Alarm", orange for "Minor Alarm", green for "Normal").

In the same application, a number of forms indicate the last time complete data was read from the hardware; Figure 2. shows an example. There's a lot of data to read, so some data is only read either by user request or when the form that displays the data item is open. Rather than updating the form-level timestamp each time an item is read, an Access method loops through all the relevant items and updates the form-level timestamp when the form is first displayed and each time it's refreshed.

The data-handling form class in the Library application, frmBizObjAware, uses this strategy to determine the first control on the form in tab order. The class has a custom property, cFirstControl; if the developer of a particular form sets the property at design-time, the specified control is treated as the first property. However, that's an easy thing for a developer to forget, so an access method for the property cycles through the controls on the form and sets the property based on the TabOrder of the various controls. Listing 1 shows the code in cFirstControl_Access.

Listing 1. This code ensures that a first control is specified on the form, even if the developer forgets to indicate which control comes first.

```
* Handle the possibility that no first control
* has been set
LOCAL oControl, oLowControl, nLowTab

IF EMPTY(THIS.cFirstControl)
   * Figure it out
   nLowTab = 1000000
   FOR EACH oControl IN THISFORM.OBJECTS
      IF PEMSTATUS(oControl, "TabIndex", 5) ;
        AND PEMSTATUS(oControl, "SetFocus", 5)
         IF oControl.TABINDEX < nLowTab
            oLowControl = oControl
            nLowTab = oControl.TABINDEX
         ENDIF
      ENDIF
   ENDFOR

   THIS.cFirstControl = oLowControl.NAME
ENDIF

RETURN THIS.cFirstControl
```

This code is used, for example, in the New button. After adding a new record and appropriately enabling and disabling controls, focus is set to the first control on the form. Listing 2 shows the code in the form class's New method.

Listing 2. This code in frmBizObjAware.New sets focus to the form's designated first control. The access method in Listing 1 is called to make sure that cFirstControl has a non-empty value.

```
* Add a record in this form
LOCAL lReturn

IF MethodExists(This.oBizObj, "New")
```

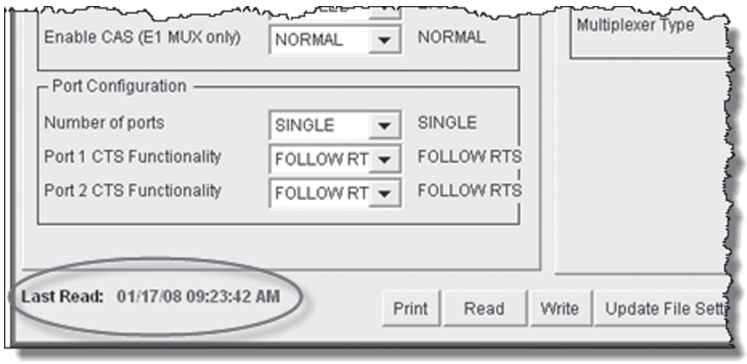

Figure 2. In this form, the last read value at the bottom indicates the oldest timestamp for any of the settings shown on the form. Rather than update each time a setting changes, an Access method finds the right value when it's needed.

```
   lReturn = This.BeforeNew()
   IF lReturn
      lReturn = This.oBizObj.New()
      IF lReturn
         This.lNewRecord = .T.
         This.lDataChanged = .F.
         lReturn = This.AfterNew()

         oFirstControl = EVALUATE("This." ;
           + This.cFirstControl)
         oFirstControl.SetFocus()
      ENDIF
   ENDIF

ENDIF

RETURN lReturn
```

Dynamic tooltips

In the form shown in Figure 1, we need to show tooltips for each of the numbered boxes, which represent ports; the content of the tooltip is determined by the current status of the port. An Access method provides an easy way to do what's needed; just build and return the appropriate string in the ToolTipText_Access method.

In the Catalog form of the Library application, it would be useful to be able to show all the details about a book's current status in a tooltip on the Copy page. (In fact, we might actually want to put the information on that page, but for the purpose of demonstrating this technique, we'll use a tooltip.) The information on that page is displayed in a container whose class is cntCopyInformation; Listing 3 shows the ToolTipText_Access method for that container.

Listing 3. The code in the ToolTipText_Access method calls a form method to build the tooltip.

```
*Check whether the book is out and build a
*tooltip with that info
RETURN ThisForm.GetBookDetail( ;
  ThisForm.cBarCode)
```

The form's GetBookDetail method looks up the book currently displayed and builds the appropriate string to display. Figure 3 shows an example.

Delegating tooltips for contained objects

The example in Figure 3 also demonstrates another use for Access methods. In a container like the one shown, we may want the same tooltip for all controls. One easy way to do that is to have the Access method for a control's ToolTipText return the parent's ToolTipText, along the lines of Listing 4.

Listing 4. To give a container and its contents the same tooltip, put code like this in the contained controls' ToolTipText_Access method.

```
RETURN This.Parent.ToolTipText
```

A little code in the base classes lets us set this up across the board, so we can turn it on for a given container by setting a single property. First, add a custom property to cntBase, the base container class; call it lBindToolTip. (Using the word "bind" here

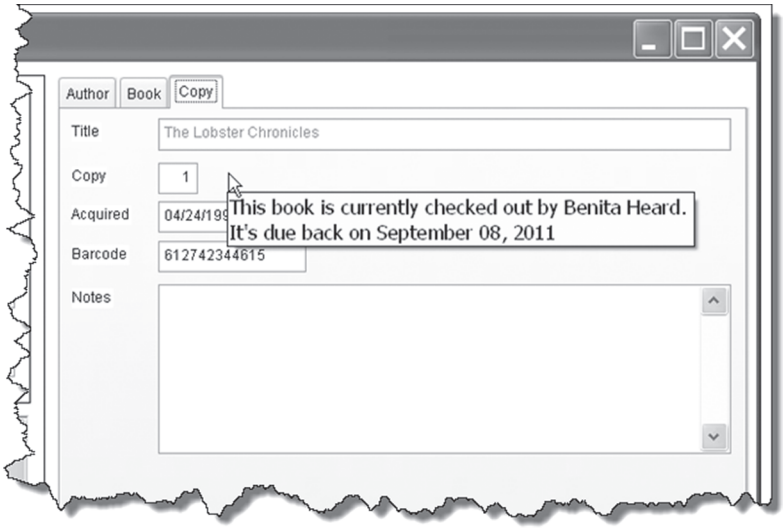

Figure 3. The tooltip for the container on the Copy page is built on the fly using the Access method of ToolTipText.

is a little misleading since it implies BindEvents, but in fact, we are really binding the container control's ToolTipText to the parent.) Then, in the ToolTipText_Access method for each control class that has a ToolTipText method, put the code in Listing 5. Then, all you have to do to ensure that everything in a container shows the same tooltip is set the container's lBindToolTip property to .T.

Listing 5. Put this code in the ToolTipText_Access method of each base control class.

```
* Check whether we're supposed to be passing
* tooltips up
LOCAL cTip
```

```
IF PEMSTATUS(This.Parent, "lBindToolTip", 5) ;
  AND This.Parent.lBindToolTip
   cTip = This.Parent.ToolTipText
ELSE
   cTip = This.ToolTipText
ENDIF

RETURN m.cTip
```

Grid component tooltips

A similar approach allows you to work around a bug in VFP 9 SP2; the contained objects of a grid don't show their own tooltips, but the tooltip of the grid. An Access method for the grid's ToolTipText property lets you drill down into the contained objects and use their ToolTipText instead.

In the library application, the top-level grid class, grdBase, has the code in Listing 6 in its ToolTipText_Access method. This code actually combines the technique in the previous section for propagating tooltips down from containers with the ability to give controls inside a grid their own tips. It checks first whether the parent of the grid has the lBindToolTip property set to .T. If so, it uses the parent's tip; if not, it drills into the grid and allows the controls inside to provide tips.

Listing 6. Use ToolTipText_Access to work around the VFP 9 SP2 bug regarding tooltips in grids.

```
* Check whether we're supposed to be passing
* tooltips up
LOCAL cTip

IF PEMSTATUS(This.Parent, "lBindToolTip", 5) ;
  AND This.Parent.lBindToolTip
   cTip = This.Parent.ToolTipText
ELSE
   * Let components have their own tooltips.
   * Look up the tooltip for the object
   * currently under the mouse.
   LOCAL aMousePos[1], oColumn, oControl

   cTip = ""

   IF AMOUSEOBJ(aMousePos) > 0
      oColumn = aMousePos[1]
      IF NOT ISNULL(m.oColumn) AND ;
         UPPER(oColumn.BaseClass) = "COLUMN"
         * First, grab column-level tip in
         * case we don't find something below
         cToolTip = oColumn.ToolTipText

         * Now, look for the right control.
          oControl = EVALUATE("oColumn." + ;
            oColumn.CurrentControl)
          IF NOT EMPTY(oControl.ToolTipText)
             cTip = oControl.ToolTipText
          ENDIF
       ENDIF
    ENDIF
ENDIF

RETURN m.cTip
```

In the application, the CheckOut form has a tooltip for the delete button in the third column; it's shown in Figure 4.

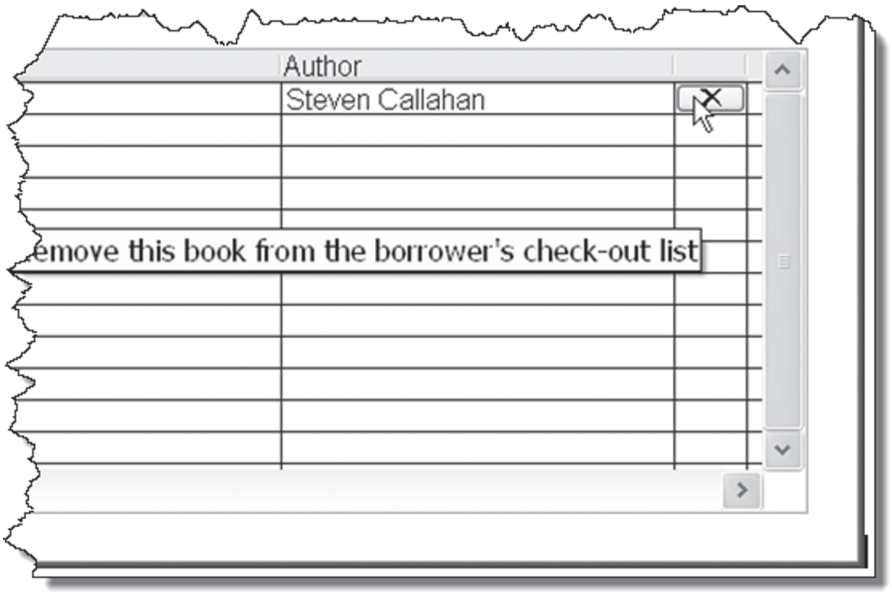

Figure 4. The tooltip shown comes from the button, not the grid. The ToolTipText_Access method finds the right tip to display.

Next up, Assign

I hope these examples give you some ideas as to how Access methods can enhance your applications. In my next article, I'll demonstrate Assign methods.

Author Profile

Tamar E. Granor, Ph.D. is the owner of Tomorrow's Solutions, LLC. She has developed and enhanced numerous Visual FoxPro applications for businesses and other organizations. Tamar is author or co-author of nearly a dozen books including the award winning Hacker's Guide to Visual FoxPro, Microsoft Office Automation with VisualFoxPro and Taming Visual FoxPro's SQL. Her latest collaboration is Making Sense of Sedna and SP2. Her books are available from Hentzenwerke Publishing (www.hentzenwerke.com). Tamar was a Microsoft Support Most Valuable Professional from the program's inception in 1993 until 2011. She is one of the organizers of the annual Southwest Fox conference. In 2007, Tamar received the Visual FoxPro Community Lifetime Achievement Award. You can reach her at tamar@thegranors.com or through www.tomorrowssolutionsllc.com.