Ahoy! Anchoring Made Easy

Doug Hennig

The new event binding feature in VFP 8 makes it easy to have objects resize or move themselves when their container is resized. In this month's article, Doug shows how you can do it by simply setting a couple of properties.

Until VFP 8, there were two ways to deal with what happens when a container, such as a form, is resized. One was to put code into the Resize method of the container that resized all the controls. That usually resulted in a ton of manually written code setting the Top, Left, Height, and Width properties of each control by control name, always a dangerous proposition (what if you rename a control?). The other way was to use a resizer object responsible for iterating through the controls in the form, resizing each appropriately. As I discussed in the December 1998 issue of FoxTalk ("Mining for Gold in the FFC"), the FoxPro Foundation Classes (FFC) that come with VFP provide a class, _Resizable, to handle this for you. I created a subclass of _Resizable called SFResizable that had better behavior, including allowing you to define how each control is resized. However, it again meant specifying controls by name and having to remember to update the SFResizable object when new controls were added to the form.

VFP 8 makes it much easier to handle this through a new feature called event binding. Instead of writing code in the container to handle all the objects or using a resizer object, each object will take care of its own resizing. Before we look at how to accomplish that, let's dig into event binding.

Event binding in VFP 8

Windows fires events when things happen. For example, when the user clicks on a button, the Click event of that button fires. That event is handled by any code you place into the Click method of the button. (Here, I distinguish between events, which are fired by the system when something happens, and methods, which are automatically called when the event fires.) You could say that the Click method is bound to the Click event. However, this binding is within a single object. What if one object wants to receive notification when an event occurs in another object? That's what event binding is about.

VFP 7 added the ability to bind to events in COM objects with the EVENTHANDLER() function. (With earlier versions, you could do event binding using the separate VFPCOM utility.) This allows us to, for example, do something in VFP when a message is received in Outlook or a document is opened in Word. VFP 8 extends this ability by providing event binding to native controls. This is done with the new BINDEVENT() function. Here's an example:

```
bindevent(Object1, 'Click', Object2, 'HandleClick')
```

This code sets up event binding between Object1 and Object2. In this case, when the Click event for Object1 fires, the HandleClick method of Object2 is automatically called.

Two other new functions are related to event binding. As its name implies, UNBINDEVENTS() turns off event binding. AEVENTS() fills an array with information about event binding.

BINDEVENT(), UNBINDEVENTS(), and AEVENTS() have various parameters that specify exactly what they should do. We won't go into any more detail about event binding here; see the VFP help topics for these functions for more information.

Handling container resizing

What does event binding have to do with container resizing? It gives us the ability to have each control be notified when its container is resized, and thus be responsible for what should happen itself rather than relying on other objects to handle it.

To implement this, I made changes to most of the visual classes in SFCtrls.VCX, my class library of VFP base class subclasses. The classes affected are SFCheckBox, SFComboBox, SFCommandButton, SFContainer, SFEditBox, SFGrid, SFImage, SFLabel, SFLine, SFListBox, SFOptionGroup, SFPageFrame, SFShape, SFSpinner, and SFTextBox. Since not every object needs to worry about its container being resized, I added an lHandleContainerResize property, .F. by default. I also added a HandleContainerResize method and cAnchor, nContainerHeight, nContainerWidth, nHeight, nLeft, nTop, and nWidth properties.

The Init method of these classes sets things up. If lHandleContainerResize is .T., meaning the object should do something when its container is resized, and the object is actually sitting in a container, it binds the Resize event of the container to the HandleContainerResize method. If the object is sitting on a page of a page frame, the object binds to the page frame rather than the page (that is, Parent.Parent), since pages don't resize. Init also saves the current size of the container and the current size and position of the object.

```
with This

* If we're supposed to bind to the Resize event of our
* container, and there is one, do so. If we're in a page
* of a pageframe, bind to the pageframe. Save the current
* size and position of the container and us.

  if .lHandleContainerResize and ;
    type('.Parent.Name') = 'C'
    if upper(.Parent.BaseClass) = 'PAGE'
      bindevent(.Parent.Parent, 'Resize', This, ;
        'HandleContainerResize')
    else
      bindevent(.Parent, 'Resize', This, ;
        'HandleContainerResize')
    endif upper(.Parent.BaseClass) = 'PAGE'
    .nContainerHeight = .Parent.Height
    .nContainerWidth  = .Parent.Width
    .nWidth           = .Width
    .nHeight          = .Height
    .nLeft            = .Left
    .nTop             = .Top
  endif .lHandleContainerResize ...
endwith
```

The HandleContainerResize method, automatically called when the container is resized because of event binding, adjusts the Height, Left, Top, and Width properties according to the setting of the cAnchor property. cAnchor should contain some combination of the values shown in **Table 1**.

Value	Anchor control to
L	Left edge of container
R	Right edge of container
V	Vertical center of container
T	Top edge of container
B	Bottom edge of container
H	Horizontal center of container

Table 1. Set the cAnchor property to a combination of these values to determine how the object should handle resizing.

"Anchoring" means that the various edges of the control are virtually (not actually) bound to part of the container. For example, if a control is anchored to the right edge of the form (cAnchor contains "R"), as the form is widened, the control will move to the right. If cAnchor is set to "LR", the control won't move since its left edge is bound to the left edge of the container but will instead become wider because its right edge is bound to the right edge of the container. You can use various combinations of anchor settings to achieve the desired result. For example, "LRTB" means the object will grow horizontally and vertically as the container is resized. Use "V" and "H" when you want an object to be centered (for example, "BV" will keep OK and Cancel buttons centered at the bottom of a form) or for objects that need to be resized that sit beside other objects that are also resized.

The code in HandleContainerResize does all the heavy lifting. AEVENTS() gives us a reference to the object whose event we're bound to, so we don't have to worry about whether it's our Parent or Parent.Parent (in the case of an object sitting on a page of a page frame) that we need to reference. We then determine how much the container's Height and Width changed by and use those differences to adjust the Top, Left, Width, and Height properties of the object, based on the setting of cAnchor. In the case of "L", “R”, “T”, and “B”, the adjustment is applied to the appropriate property. In the case of “V” and “H”, only half of the adjustment is used, since those settings mean the object is bound to the middle of the container.

```
local laEvents[1], ;
  lnHeight, ;
  lnWidth, ;
  lnHeightAdjust, ;
  lnWidthAdjust, ;
  lcAnchor
with This

* Get the new height and width of the container and how
* much it changed by.

  aevents(laEvents, 0)
  lnHeight       = laEvents[1].Height
  lnWidth        = laEvents[1].Width
  lnHeightAdjust = lnHeight - .nContainerHeight
  lnWidthAdjust  = lnWidth  - .nContainerWidth

* Adjust the width and left as appropriate.

  lcAnchor = upper(.cAnchor)
  do case

* Bound to the left and right edges: adjust the width.

    case 'L' $ lcAnchor and 'R' $ lcAnchor
      .Width = max(.nWidth + lnWidthAdjust, 0)

* Bound to the left and vertical center: adjust the width
* by half the change.

    case 'L' $ lcAnchor and 'V' $ lcAnchor
      .Width = max(.nWidth + lnWidthAdjust/2, 0)

* Bound to the right and vertical center: adjust the
* width by half the change and move the left by half the
* change.

    case 'R' $ lcAnchor and 'V' $ lcAnchor
      .Width = max(.nWidth + lnWidthAdjust/2, 0)
      .Left  = .nLeft + lnWidthAdjust/2

* Bound to the vertical center: move the left by half the
* change.

    case 'V' $ lcAnchor
      .Left  = .nLeft + lnWidthAdjust/2

* Bound to the right edge: move the left.

    case 'R' $ lcAnchor
      .Left  = .nLeft + lnWidthAdjust
  endcase

* Adjust the height and top as appropriate.

  do case

* Bound to the top and bottom edges: adjust the height.

    case 'T' $ lcAnchor and 'B' $ lcAnchor
      .Height = max(.nHeight + lnHeightAdjust, 0)

* Bound to the top and horizontal center: adjust the
* height by half the change.

    case 'T' $ lcAnchor and 'H' $ lcAnchor
      .Height = max(.nHeight + lnHeightAdjust/2, 0)
```

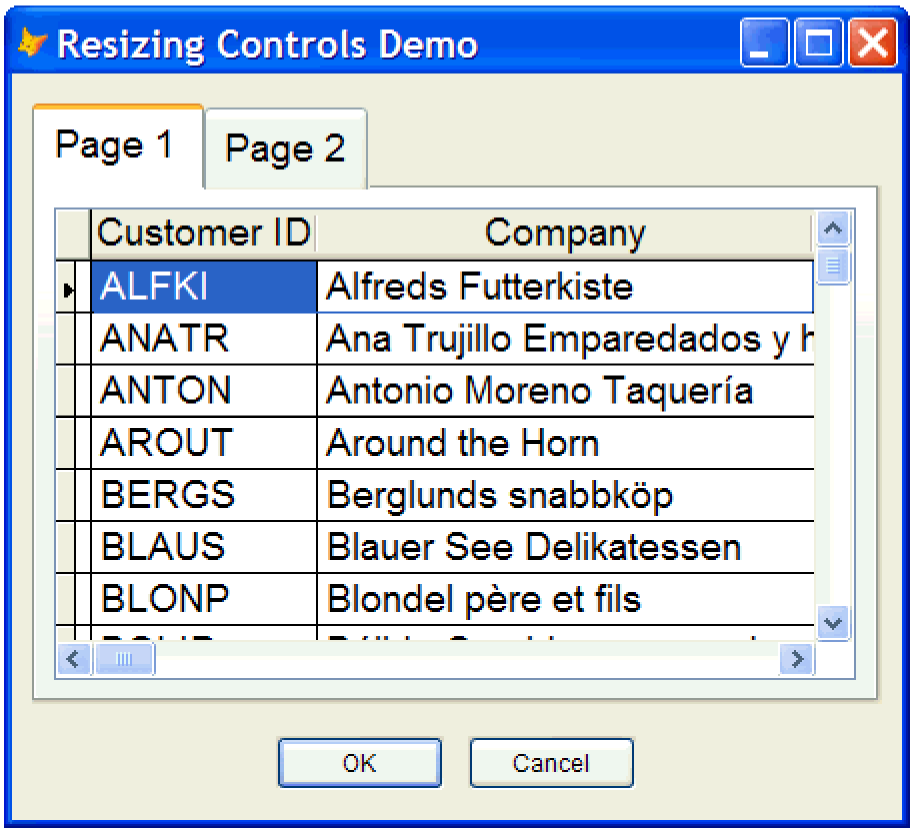
```
* Bound to the bottom and horizontal center: adjust the
* height by half the change and move the top by half the
* change.

    case 'B' $ lcAnchor and 'H' $ lcAnchor
      .Height = max(.nHeight + lnHeightAdjust/2, 0)
      .Top    = .nTop + lnHeightAdjust/2

* Bound to the horizontal center: move the top by half
* the change.

    case 'H' $ lcAnchor
      .Top    = .nTop + lnHeightAdjust/2

* Bound to the bottom edge: move the top.

    case 'B' $ lcAnchor
      .Top    = .nTop + lnHeightAdjust
  endcase
endwith
```

In addition to the changes in the visual classes, I added This.Resize() to the Init method of SFForm so all controls resize properly at startup.

Check it out

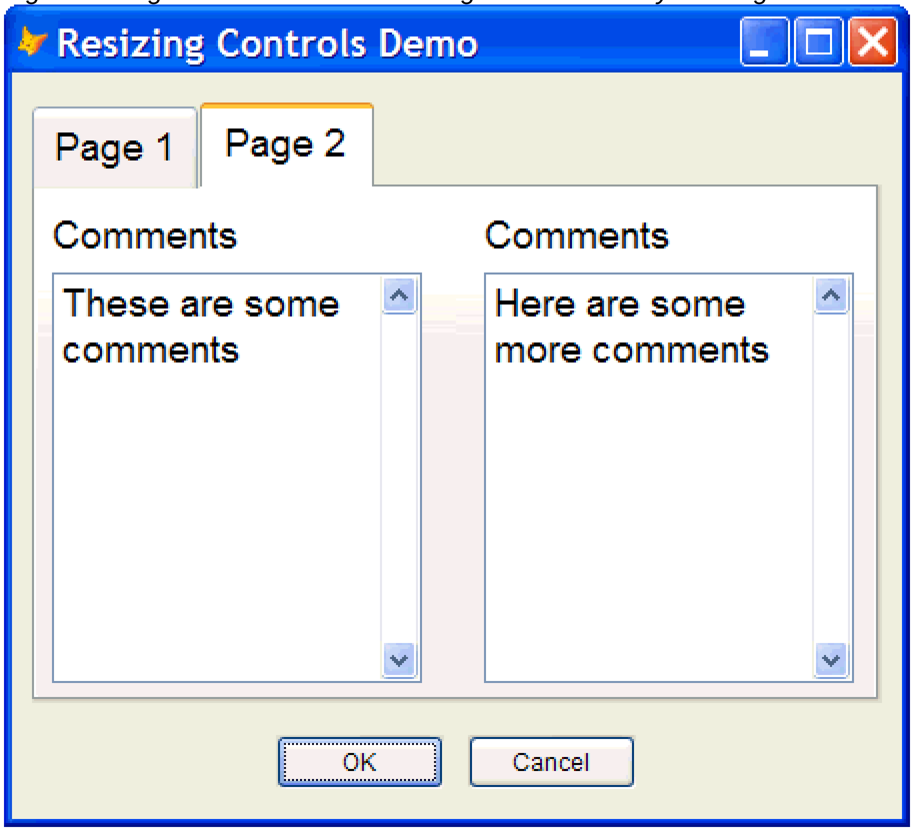
To see how resizing works, run DemoResizeBinding.SCX and resize the form. As you can see in the series of snapshots in **Figure 1** through **Figure 4**, the page frame, the grid in page 1, and the edit boxes in page 2 automatically resize themselves, the right-most label in Page 2 moves to the left or right, and the OK and Cancel buttons stay centered as the form is resized. No code was required to do this, just setting lHandleContainerResize to .T. and cAnchor as appropriate in each of these objects. In the case of the two edit boxes in Page 2, since they are beside each other and each resizes, they have to be anchored to the vertical center of the form so they don't bump into each other. The left edit box has cAnchor set to "LVTB" and the right one uses "RVTB".

Figure 1. Page 1 of DemoResizeBinding.SCX before any resizing.



Customer ID	Company
ALFKI	Alfreds Futterkiste
ANATR	Ana Trujillo Emparedados y h
ANTON	Antonio Moreno Taquería
AROUT	Around the Horn
BERGS	Berglunds snabbköp
BLAUS	Blauer See Delikatessen
BLONP	Blondel père et fils

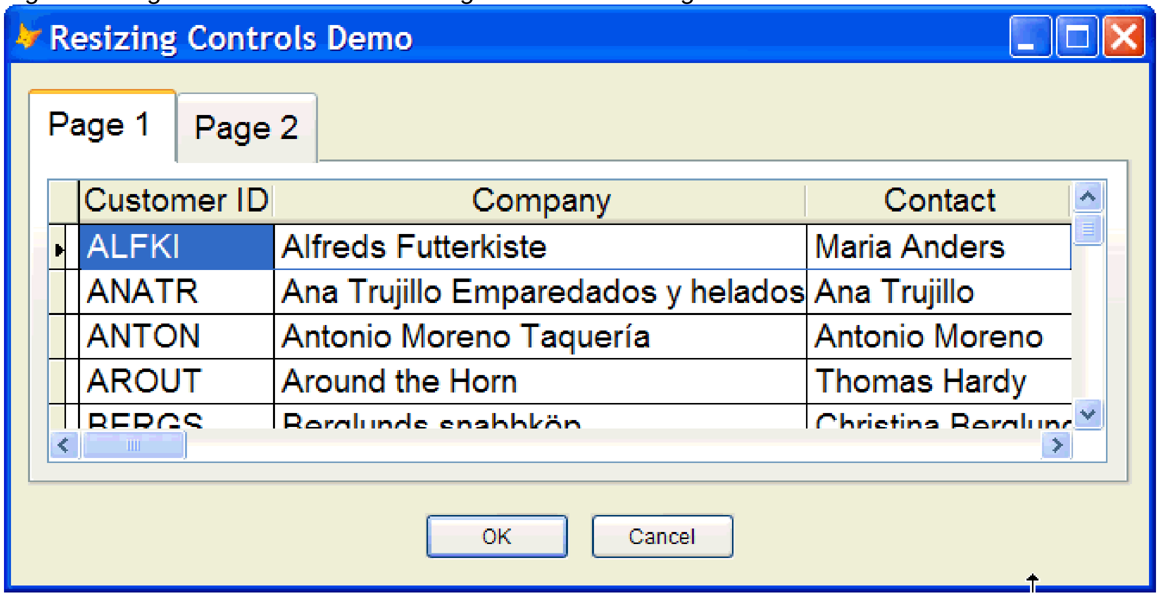
Figure 2. Page 2 of DemoResizeBinding.SCX before any resizing.



These are some comments

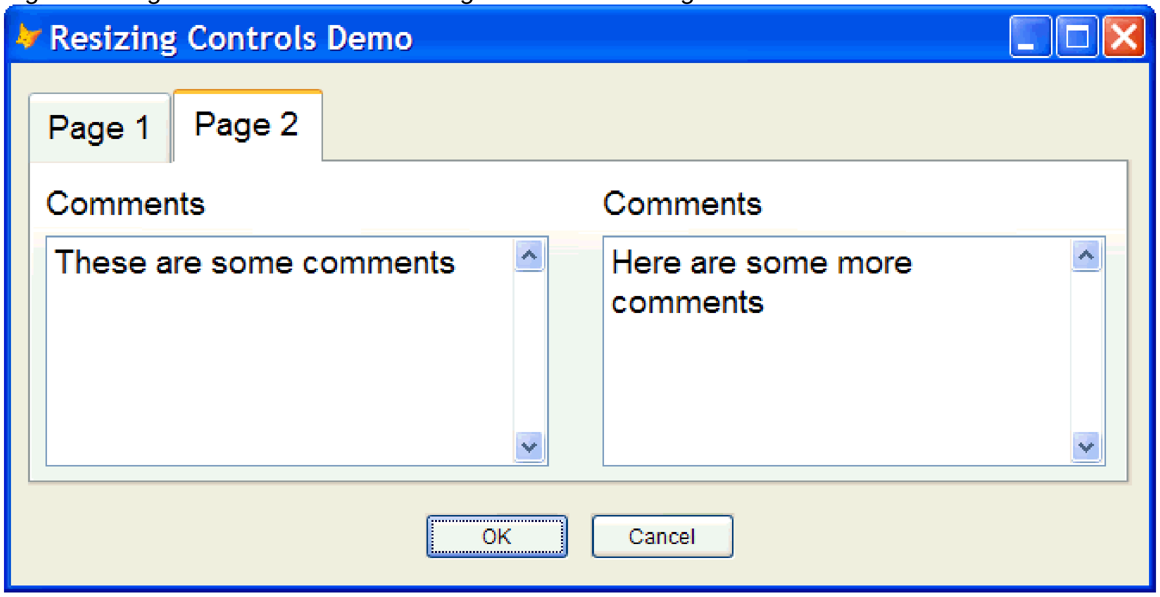
Here are some more comments

Figure 3. Page 1 of DemoResizeBinding.SCX after resizing.



Customer ID	Company	Contact
ALFKI	Alfreds Futterkiste	Maria Anders
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo
ANTON	Antonio Moreno Taquería	Antonio Moreno
AROUT	Around the Horn	Thomas Hardy
BERGS	Berglunds snabbköp	Christina Berglun

Figure 4. Page 2 of DemoResizeBinding.SCX after resizing.



Comments

These are some comments

Comments

Here are some more comments

Issues

There are a couple of issues with this mechanism I haven't worked out as of this writing. One is that when you programmatically change the Height and Width of a container, the HandleContainerResize of all member objects is called for each change, so this method gets called twice. It'd be nice to defer firing HandleContainerResize until both Height and Width are changed. The other issue is what happens when you click on the Minimize and Maximize buttons of the form. It appears that the Height of the page frame doesn't change until after the HandleContainerResize method of the objects are called, so while the width of these controls is adjusted properly, the height is not (although sometimes in my testing, it was Width that didn't change).

Summary

Event binding is a powerful new feature in VFP 8. In addition to making it possible to have objects resize or move themselves by simply setting a couple of properties, it can be used for all kinds of things. For example, a progress meter could update itself as a process proceeds, the buttons in a toolbar could enable or disable themselves according to what's happening in a data entry form, and so on. Spend some time checking out event binding, and I'm sure you'll come up with a lot of uses that formerly required a lot of ugly code to handle.

Doug Hennig is a partner with Stonefield Systems Group Inc. He is the author of the award-winning Stonefield Database Toolkit (SDT) and Stonefield Query, author of the CursorAdapter and DataEnvironment builders that come with VFP 8, and co-author of "What's New in Visual FoxPro 8.0", "What's New in Visual FoxPro 7.0", and "The Hacker's Guide to Visual FoxPro 7.0", from Hentzenwerke Publishing. Doug has spoken at every Microsoft FoxPro Developers Conference (DevCon) since 1997 and at user groups and developer conferences all over North America. He is a Microsoft Most Valuable Professional (MVP) and Certified Professional (MCP). Web: www.stonefield.com and www.stonefieldquery.com Email: dhennig@stonefield.com