January, 2001

Advisor Answers

Managing Grid actions

VFP 6.0

Q: I have code in my grid column header's Click event to sort data in the column - similar to the Microsoft Office programs. However, if I resize that column with my mouse, it causes the header's Click to occur, which runs the sorting code. Is there a way to resize a column with the mouse that does not also run the header Click event?

—Bob Thomsen (via Advisor.com)

A: You can't prevent the header's Click method from firing in this situation. Sizing a grid column always calls the Click method for the actual column being sized, which is the left-hand column of the two the mouse is between. However, there is a way to determine whether the Click came from a resizing action. The key is to use the grid's GridHitTest method, which lets you determine where you are in the grid.

GridHitTest was added in VFP 6 and provides some really useful information. You pass it coordinates and it tells you what component of the grid that point is over (or if it's not in the grid at all), as well as the relative row and column of the point. GridHitTest is an unusual method in that it expects several of its parameters to be passed by reference – that's how it can return multiple data values. Here's the syntax for GridHitTest:

```
GridHitTest( nXCoord, nYCoord, @nComponent, @nRelativeRow, @nRelativeColumn
                  [, @nPane ] )
```

The value stored in nComponent is a number that indicates which part of the grid that point is on. It's assigned 0 if the specified point isn't in the grid. For our task, the key values are 1, which indicates a header and 2, which indicates between headers. (The nPane parameter is for situations where the grid is split – it tells you which pane of the grid contains the specified point.)

To apply GridHitTest to prevent Click code from running, you need to get your hands on the point at which the Click occurred. The Click method doesn't have this information, but MouseDown does. It also turns out that you have to test the point before the move occurs, so

the header's MouseDown method is a good place to call GridHitTest. Once you've done so, you need to store the nComponent value to make it available in the header's Click method.

Here are the steps you need to take to prevent your custom Click code from firing when you resize a column:

1) Add a custom property to the form. Call it nLastComponent.

2) Add this code to the MouseDown method of each header:

```
LPARAMETERS nButton, nShift, nXCoord, nYCoord
LOCAL nComp, nRelRow, nRelCol
This.Parent.Parent.GridHitTest(nXCoord, nYCoord, @nComp, @nRelRow, @nRelCol)
ThisForm.nLastComponent = nComp
```

3) In the header's Click method, use code like this to check whether it's a real click or a resize:

```
IF ThisForm.nLastComponent = 1
   * This is actually a click. Put your custom code here.
ELSE
   * This is a resize. Behave appropriately.
ENDIF
```

The biggest problem with this solution is that you need to put code in two methods of every header. Clearly, a better solution is to subclass the Header class, and put the code you need there. This raises two (related) problems.

First, headers can't be subclassed using the Class Designer. The solution to this problem is to create your subclass in code. Here's code for a header subclass that does the job. Note that I've moved the nLastComponent property from the form to the header for better encapsulation.

```
DEFINE CLASS hdrResize AS Header
nLastComponent = 0
PROCEDURE MouseDown
LPARAMETERS nButton, nShift, nXCoord, nYCoord
LOCAL nComp, nRelRow, nRelCol
This.Parent.Parent.GridHitTest(nXCoord, nYCoord, @nComp, @nRelRow, @nRelCol)
This.nLastComponent = nComp
RETURN
PROCEDURE Click
IF This.nLastComponent = 1
   * This is actually a click. Put your custom code here.
ELSE
   * This is a resize. Behave appropriate.
ENDIF
```

```
RETURN
ENDDEFINE
```

The second problem is more difficult. You can't tell the grid to simply use a particular header class. Instead, you need to change the header of each column. You can do this by removing the base class header and substituting your custom header class. While it's possible to do so at design-time with a builder, that approach raises other problems that make it more trouble than it's worth. The robust way to do this is to replace the headers at runtime by putting code in the grid's Init method.

Here's the code – it assumes that the header class above is stored in a file called HdrClass.PRG. It gives the new header for each column the same name and caption as the one it's replacing.

```
FOR EACH oColumn IN This.Columns
   FOR EACH oControl IN oColumn.Controls
      IF UPPER(oControl.BaseClass) = "HEADER"
         cHdrName = oControl.Name
         cCaption = oControl.Caption
         oColumn.RemoveObject(cHdrName)
         oColumn.NewObject(cHdrName, "hdrResize", ;
                                     "hdrClass.PRG")
         oColumn.&cHdrName..Caption = cCaption
      ENDIF
   ENDFOR
ENDFOR
```

Of course, once you're replacing base class headers with your own custom class, you can put whatever code you think appropriate in the class. Your code to re-order the grid is one obvious candidate for this treatment.

In addition, if you're using a grid subclass, you can do the job once by adding custom properties to hold the header class name and class library. Then, in the grid class's Init method, check those properties and if they're not empty, run the loop above, substituting the properties for the hard-coded names I used.

You'll find the simple Header class and the code for the grid's Init on this month's Professional Resource CD.

–Tamar