March, 2001

Advisor Answers

Preventing movement in a list

VFP 7.0/6.0/5.0/3.0

Q: I have a listbox with the MoverBars property set to .T. There are some items in the list that I don't want users to be able to move. (These items are blank.) Is there an event that is triggered by moving a MoverBar? I tried to use the InteractiveChange to catch it, but it only occurs after the move.

–Owen McPeak (via CompuServe)

A: The first thing I did to try to figure this out was to turn on Event Tracking in the debugger to see exactly the sequence of events that occur when the user drags a bar with the mover button. Here's the resulting list:

```
form1.list1.MouseDown(1, 0, 33, 64)
form1.list1.MouseUp(1, 0, 32, 127)
form1.list1.InteractiveChange()
form1.list1.Click()
form1.list1.When()
form1.list1.Message()
```

Looking at the mouse position in MouseDown and MouseUp, it appears that the key event for catching (and preventing) a move is MouseDown. My next step was to see whether putting NODEFAULT in the MouseDown method prevents the move; it does. That reduces the problem to figuring out, in MouseDown, when the user is trying to move one of the bars we don't want moved.

There are two components to that information. The first part is determining whether the mouse is over a mover button or actual data. It turns out that no matter what you do to the font or font size of the list, the mover buttons stay the same size. (I also checked to be sure that nothing in the Appearance page of the Windows Display Properties dialog changes the size of the mover button.) Since that's the case, I was able to figure out how wide the mover buttons are by using the AMouseObj() function. I positioned my mouse right at the edge of the button (you can do this in design mode or at runtime) and then from the Command Window (which I reached using the CTRL+F2 shortcut), I executed the function:

```
AMouseObj(aMResults)
```

After a few tests, I ascertained that the mover button is 17 pixels wide, beginning at the left edge of the list.

The harder part is determining whether the mouse is on one of the items that's not to be moved. In your case, that's an item containing the empty string. Neither the Value nor the ListIndex properties of the list is changed until later in the event sequence. In MouseDown, they still contain the old values. In fact, in MouseDown, there's no property that tells you which list item you're on. That leaves brute force.

The solution I found is to compute the item number, using the mouse position and the size of an item. The FontMetric() function tells us how tall one line is in a specified font and the leading (the space between lines) for that font. Using this information, we can figure out how many lines down in the list the mouse is, at this moment.

That leaves only one issue. What if the list has been scrolled down? We need to take that into account and compute the actual position in the list, not just the number of rows down from the top of the list.

Putting it all together, the code needed in MouseDown is surprisingly short:

```
LPARAMETERS nButton, nShift, nXCoord, nYCoord
LOCAL nPosition, nRowHeight, nRow
IF nXCoord <= 17 + This.Left
   * Mouse is on mover bar, so compute row
   nPosition = nYCoord - This.Top
   nRowHeight = ;
      FontMetric(1, This.FontName, This.FontSize)+;
      FontMetric(4, This.FontName, This.FontSize)
   * Round up in computation
   nRow = CEILING(nPosition/nRowHeight) + ;
               This.TopIndex – 1
   IF EMPTY(This.List[ nRow ])
      * The condition for this IF can be changed to whatever
      * is appropriate for the rows that shouldn't be moved
      NODEFAULT
   ENDIF
ENDIF
```

It's also possible to move the items in a list using the Ctrl+UpArrow and Ctrl+DnArrow keys. Preventing that is much simpler since we know where the keyboard focus is. This code in KeyPress does the trick:

```
LPARAMETERS nKeyCode, nShiftAltCtrl
```

```
* CTRL key and UpArrow or DnArrow
IF BITTEST(nSHiftAltCtrl, 1) AND ;
   INLIST(nKeyCode, 141, 145)
   IF EMPTY(This.List[ This.ListIndex ])
      * The condition for this IF can be changed
      * to whatever is appropriate for the rows
      * that shouldn't be moved
      NODEFAULT
   ENDIF
ENDIF
RETURN
```

This month's Professional Resource CD contains a form (KillMove.SCX) that demonstrates the technique.

I considered one alternative approach. You can put straight lines into a list by specifying "\-" as the bar contents. Those lines don't have mover bars. Unfortunately, I found that VFP doesn't handle the combination of straight lines and mover bars properly. Depending on the font size and other considerations, either the other bars can't be moved at all, or moving them causes visual inconsistencies. Microsoft has acknowledged this behavior as a bug.

–Tamar