Cool Tools by Craig Boyd, Part II

Doug Hennig

Doug Hennig continues his examination of cool tools provided to the VFP community by Craig Boyd.

Last month, I discussed several tools generously provided to the VFP community by VFP MVP and guru Craig Boyd, including his encryption library, progress bar control, and splitter control. This month, we'll look at some more of his tools: a calendar control and a scrolling container control.

Calendar control

I use the Microsoft Date and Time Picker ActiveX control extensively to provide an easy way for users to enter dates. It provides a control similar to a text box for date entry, but has automatic validation on the various parts of the date. There's also a down arrow button that, when clicked, displays a calendar where the user can visually choose the date. However, there are a number of issues with this control:

- Since it's an ActiveX control, you have to install and register an OCX (MSCOMCT2.OCX) along with your application. Also, if you instantiate it programmatically rather than dropping it on a form, you'll run into licensing issues unless you manually update the Windows Registry to install a license for the control.
- This control is pretty old, so it doesn't visually appear like modern controls. It doesn't support Windows XP themes, for example.
- It doesn't allow empty dates, so you have to write code to deal with that situation.

Craig has created a pure VFP code calendar class, available for download from http://www.sweetpotatosoftware.com/files/vfpcalendar.zip. Because it's VFP code, there are no ActiveX issues, and since source code is provided, you can change the control's appearance and behavior if you wish.

Figure 1 shows one of the samples included in Craig's download, which has the calendar control in a VFP form. In addition to a nice looking calendar, notice that the cells for 6, 12, 21, and 27 have different highlighting. I'll discuss how to do that later.

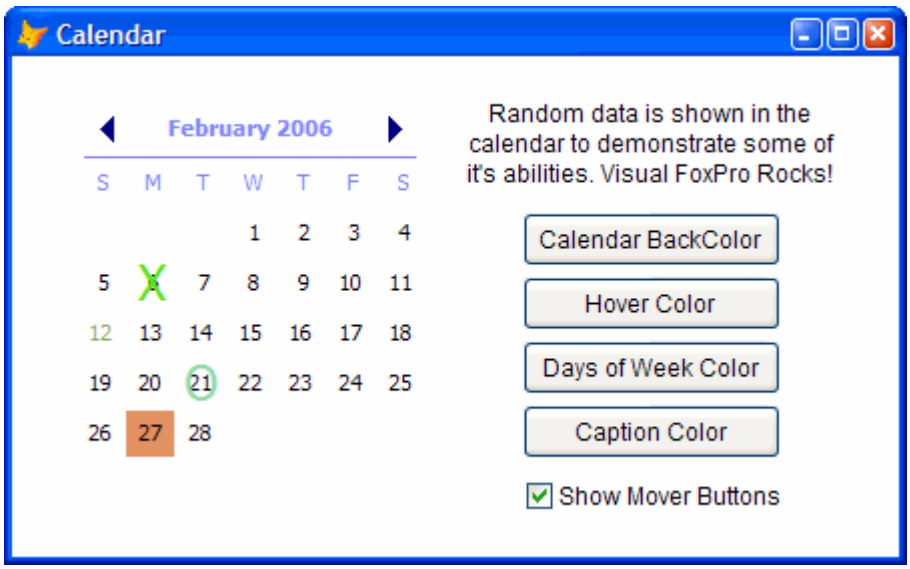

Figure 1. Craig's calendar control is a nice looking control written completely in VFP.

In addition to the calendar control, Craig has provided Date and DateTime picker controls that, like the Microsoft control, provide a text box with a drop-down calendar, but are much more visually appealing. **Figure 2** shows an example of these controls.

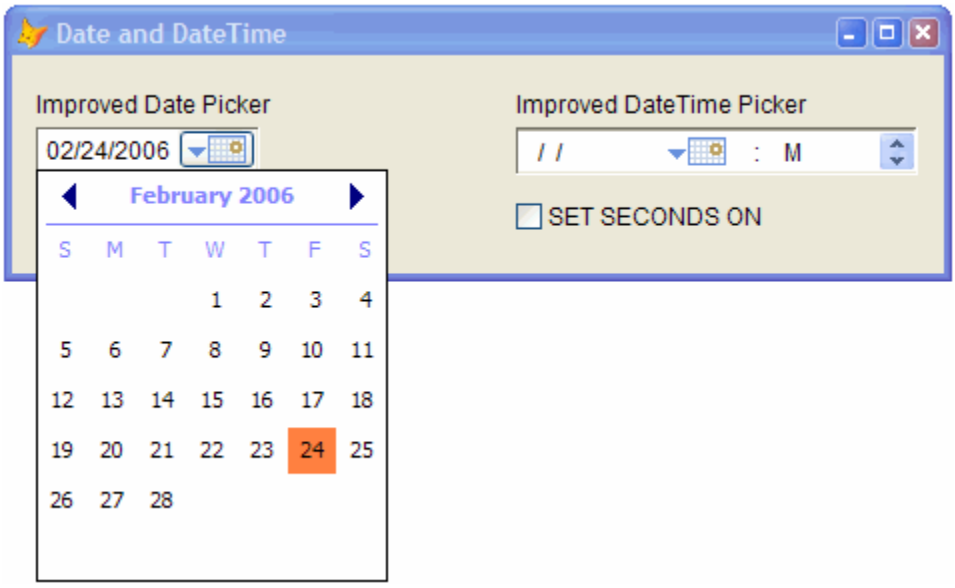

Figure 2. The Date and DateTime picker controls are nicer looking than the Microsoft control.

To use the calendar control, drop an instance of CalMonth from VFPCalendar.VCX onto a form. CalMonth has several properties that control its appearance, including CalendarBackColor, CaptionForeColor, DaysForeColor, DaysOfTheWeekForeColor, HighlightForeColor, and OverlayForeColor. If you change one of these properties programmatically, you have to use a somewhat arcane set of steps to ensure the calendar is properly updated: call the SetColors method, assign the CurrentDate property of the CalEngine1 member to itself (so its Assign method fires, which has some necessary behavior), and then call Refresh. (It seems like a method of CalMonth, such as RefreshCalendar, would make this easier.) The other property of interest, DateSelected, contains the date selected in the calendar; you might want to initialize that to DATE() to highlight today's date, or to the value of a Date field so that date is automatically selected. After the user has selected a date, use the value of that property as necessary.

Using the Date and DateTime picker classes is even easier. Drop a CalDatePicker or CalDateTimePicker object onto a form and set ControlSource to the desired value, such as a Date or DateTime field. The Value property of the control contains the Date or DateTime entered by the user.

In addition to the properties I mentioned earlier, there are other ways to control the appearance of the calendar, such as with overlays, but before we discuss that, let's see how Craig built the calendar control. CalMonth is a container class consisting of a header container, a grid, and a non-visual engine object. That's right; the calendar part is actually a regular VFP grid. The grid's properties are set so it has no gridlines, no scrollbars, no record or delete mark, and no header; at design time, it doesn't look like a grid. The grid has seven columns, each of which contains a CalDay object.

CalDay is also a container class. It consists of two text boxes and a command button. One of the text boxes, txtDay, displays the date. The other, txtOverlay, displays overlay information. The When methods of both text boxes return .F. (this is actually done in a parent class) so they cannot receive focus. The command button doesn't appear (Style is set to 1-Invisible) but its MouseDown method selects the date the cell represents in the grid.

The secret to the grid appearing like a calendar is the BackStyle_Access method of CalDay. Craig explains how this works in detail in the August 19, 2005 entry in his blog (http://www.sweetpotatosoftware.com/SPSBlog/default.aspx), but the short explanation is that the BackStyle property of every object in a column of a grid is accessed for every cell of the grid, so if you create an Access method for that property, it'll be called for every instance of the object, giving you full

control over how that object appears in each individual cell. That's why, in Figure 1, some cells appear like day column headings, some appear in different colors, and some have "overlay" objects. I won't go over the code in CalDay.BackStyle_Access as it's fairly long, but I recommend examining it yourself to see just how cool this powerful technique is. Craig has other examples on his Web site that show a different image in each cell of a grid or use a grid to display bar graphs.

Since the calendar is really is grid, there must be a cursor the grid is bound to. The CalEngine class, an instance of which is contained in CalMonth, is responsible for creating that cursor. Its CreateCursor method creates a cursor with the following columns: DayNum1 through 7, which represent the day number for columns 1 through 7 in the grid, Overlay1 through 7, which contain overlay characters to display in the columns, OverColor1 through 7, which contain the foreground colors to use for the overlay characters, DayColor1 through 7, which contain the foreground colors for the day numbers, and BackColor1 through 7, which contain the background colors for the day numbers. Thus, there are five fields for each column in the calendar (for example, DayNum1, Overlay1, OverColor1, DayColor1, and BackColor1 for the first column) and one record for each row. With all of these fields, you have a lot of control over how each individual cell appears.

For example, ExampleOptions.SCX, one of the sample forms Craig provides, overrides the CreateCursor method of the CalEngine1 object in its CalMonth instance to insert random values into the OverColor, DayColor, and BackColor fields of random columns. It also sets certain Overlay fields to "X" or "O"; in Figure 1, the cell for the 6th has an "X" overlay while the 21st has an "O." Every time you run this form, it looks different (not something I'd recommend in a production application, but it makes for a nice demo).

Craig's calendar control is a great addition to every VFP developer's toolbox.

SFCalMonth

As you probably know, I always like to tweak tools a bit to make the appearance and/or behavior fit my needs, and Craig's calendar control is no exception. One of the things I like about the Microsoft control is the ability to quickly select a particular month: clicking on the month in the calendar header displays a menu of months you can select from rather than having to click the arrow buttons one at a time. Changing years is a little clunkier: clicking on the year displays up and down buttons that add or subtract one year at a time, so selecting the birth year for someone of my, ahem, age, means a lot of button clicking. Fortunately, it was fairly easy to add a menu for the months to CalMonth, and I added one for years too.

I subclassed CalMonth, creating SFCalMonth in SFCalendar.VCX. Adding a menu of months or years simply means adding code to some method of the CalMonthMover1 control (which represents the header for the calendar). Unfortunately, the month and year display in CalMonthMover1 are in a single text box. I didn't feel like writing the ugly code required to figure out whether the mouse is over the month or the year, so I decided that clicking would display the month menu and right-clicking would display the year menu. The Click method has this code:

```
local lnI, ;
  ldDate
define popup MonthPicker from mrow(), mcol()
for lnI = 1 to 12
  define bar lnI of MonthPicker ;
    prompt cmonth(date(2006, lnI, 1))
next lnI
on selection popup MonthPicker ;
  deactivate popup MonthPicker
activate popup MonthPicker
if not empty(prompt())
  ldDate = This.Parent.calengine1.currentdate
  This.Parent.calengine1.currentdate = ;
    date(year(ldDate), bar(), day(ldDate))
  This.setcaption()
endif not empty(prompt())
```

There's similar code in RightClick to display the year, although I decided to use a range of years from five years past to five years ahead of the current year in the calendar. If you work with a wider range of years, such as birth dates, you might want a different mechanism to quickly select the year.

Another change I made has to do with the nature of a grid. While Craig set the AllowCellSelection property of the grid to .F. to prevent most grid behaviors, the grid still acts like a grid in some ways, which is undesirable when you want it to be a calendar. For example, if you click on a day to select it, and then press the Page Dn key (thinking that would perhaps display the next month), the grid scrolls so none of the days are visible. I figured handling this would be pretty easy: simply override the KeyPress method and use NODEFAULT for Page Dn. Unfortunately, that didn't work, and a quick peek at the help topic for KeyPress showed why: that method is ignored if AllowCellSelection is .F. So, I had to make a few changes to support this: turn AllowCellSelection back to .T. and then handle date selection manually. Here's the code from the Click method of the grid to re-enable date selection with the mouse. Notice it uses the GridHitTest method, which I'll confess I haven't used before, to figure out which row and column the user clicked in, and simply fires the MouseDown method of the button object in the appropriate column, which has the logic of handling date selection.

```
local lnWhere, ;
  lnRow, ;
  lnCol
lnWhere = 0
lnRow   = 0
lnCol   = 1
This.GridHitTest(mcol(Thisform.Name, 3), ;
  mrow(Thisform.Name, 3), @lnWhere, @lnRow, @lnCol)
if lnWhere = 3 and lnRow > 1
  go lnRow
  This.Columns(lnCol).CalDay1.CalButton1.MouseDown()
endif lnWhere = 3 ...
```

While I was adding code to KeyPress to handle Page Dn, I figured I may as well add support for other key presses to make the calendar easier to use. Page Dn moves the calendar ahead a month, while Page Up moves it back a month. The left, right, up, and down arrows move the selected date in the expected direction.

```
lparameters tnKeyCode, ;
  tnShiftAltCtrl
local ldDate, ;
  llDateChanged
with This.Parent
  do case

* Handle PgDn: move ahead one month.

    case tnKeyCode = 3
      ldDate = gomonth(.DateSelected, 1)
      llDateChanged = .T.

* Handle PgUp: move back one month.

    case tnKeyCode = 18
      ldDate = gomonth(.DateSelected, -1)
      llDateChanged = .T.

* Handle up arrow: move back a week.

    case tnKeyCode = 5
      ldDate = .DateSelected - 7
      llDateChanged = .T.

* Handle down arrow: move ahead a week.

    case tnKeyCode = 24
      ldDate = .DateSelected + 7
      llDateChanged = .T.

* Handle left arrow: move back a day.

    case tnKeyCode = 19
```

```
      ldDate = .DateSelected - 1
      llDateChanged = .T.

* Handle right arrow: move ahead a day.

    case tnKeyCode = 4
      ldDate = .DateSelected + 1
      llDateChanged = .T.
  endcase

* If the date was changed, handle it.

  if llDateChanged
    store ldDate to .calengine1.Currentdate, ;
      .DateSelected
    .Calmonthmover1.SetCaption()
    .Refresh()
    nodefault
  endif llDateChanged
endwith
```

Scrolling container control

In my December 2004 column, "Mining for Gold in XSource, Part 3," I discussed a class included in the source for the Task List that provides a scrolling region in a form. It uses the Microsoft Flat Scrollbar ActiveX control, so again you have to be sure to install and register the proper OCX, and like the Microsoft Date and Time Picker control, it doesn't respect Windows XP themes. Craig has created his own scrolling container control, available from http://www.sweetpotatosoftware.com/files/vfpscrollbar.zip.

Figure 3 shows this control in action. You can see that I scrolled the window to display parts of an edit box and a page frame, as well as a command button. Notice the scroll bars look themed.

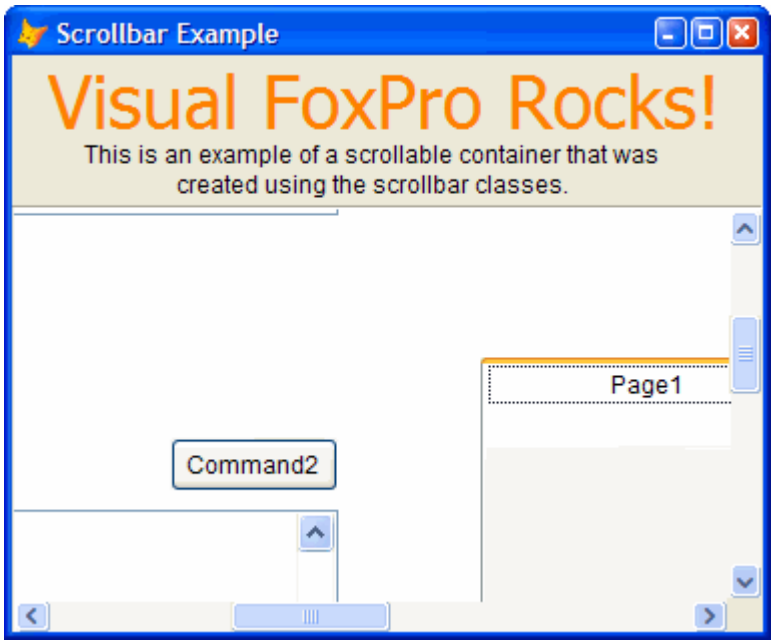

Figure 3. The scroll bar control allows you to create a scrolling region in a form.

The thing that's kind of mind-blowing about this control is that there are no images at all. Everything is drawn with shapes: the boxes at each end of the scroll bar, the thumb on the scrollbar, even the four little lines in the middle of the thumb. In his blog on this class (the August 27, 2005 entry), he states that it took over five hours to create this control, and I believe it.

All of the classes making up the scrolling container are in VFPScrollBar.VCX. The main class is SBScrollContainer. It's a container class with horizontal and vertical scrollbars at the bottom and right edges, respectively. However, you can control which scrollbars are visible by setting the Scrollbars property

to 0 for none, 1 for horizontal, 2 for vertical, and 3 (the default) for both. The ScrollableHeight and ScrollableWidth properties specify the size of the scrollable region.

To use SBScrollContainer, drop it onto a form and add the controls you want inside the scrolling region to the container. You can size SBScrollContainer as necessary, but realize that its components don't reposition themselves at design time, so they may not look quite. However, when you run the form, the components are sized and positioned correctly for the container's size, and are even anchored so resizing the form works properly.

Let's look at the scrolling container control more closely to see how Craig did it. SBScrollContainer is a container with ScrollbarHorizontal and ScrollbarVertical objects. Scrollbar_Assign ensures only the appropriate components are visible, and the Assign methods for ScrollableHeight and ScrollableWidth set the Max properties of the ScrollbarVertical and ScrollbarHorizontal objects, respectively. The Change method of each scrollbar object, which fires when the user clicks in the scrollbar, clicks on the buttons at either end, or drags the thumb, adjusts the Top or Left properties of each contained object so they appear to scroll.

ScrollbarHorizontal and ScrollbarVertical are subclasses of Scrollbar. This class has properties indicating how many pixels to scroll when the user clicks the buttons (SmallChange, which defaults to 25) or in the scrollbar (LargeChange, set to 100) and the range of motion in the scrollbar (Min and Max, set to 0 and 500, respectively). The Value property indicates the current scroll position, and its Assign method fires the Change method, so it acts like an event.

The genius (or madness) of the scrollbar classes is how they're built. As I mentioned earlier, there are no images, so each of the boxes and lines that appears in the scrollbar is actually composed of multiple shapes of varying colors and lengths to provide a 3-D, themed appearance. I won't bother going into details on this construction; feel free to poke around the classes yourself and see how Craig painstakingly put each line and shape into place. It's sort of like building a model bridge out of toothpicks, but the results look great.

SFScrollContainer

Once again, being Mr. Picky, I have one quibble with SBScrollContainer. If you run the sample form that comes with Craig's download, notice as you scroll, you can see part of the page frame in the small square region to the right of the horizontal scrollbar and below the vertical scrollbar; you can see this in the area I've highlighted in **Figure 3**.

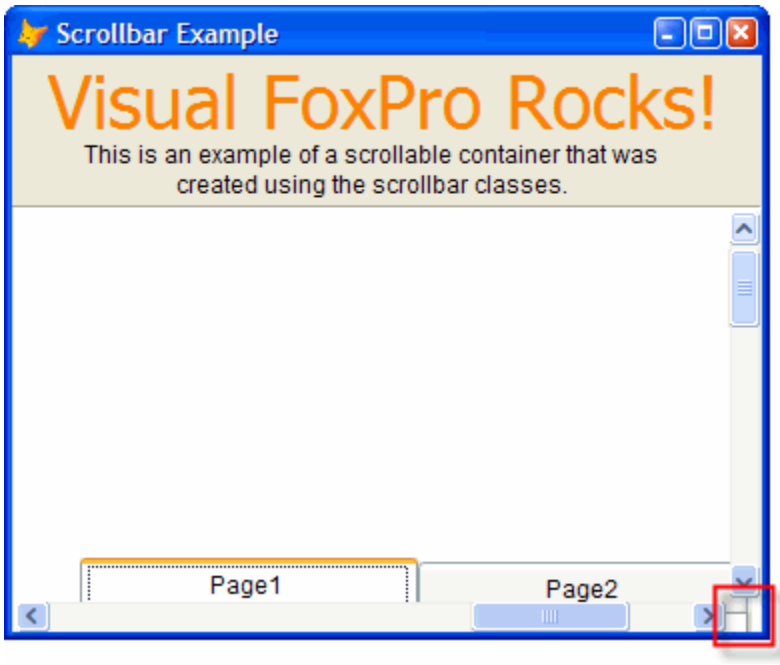

Figure 4. As you scroll, controls may bleed through in the corner of the scrolling region.

Fortunately, this is easy to fix. I created a subclass of SBScrollContainer called SFScrollContainer in SFScrollbar.VCX. I added a small shape in the square region exhibiting bleed through; this shape covers the "hole" in the scrolling container so none of its contained controls will appear in that region. I added the following code to the PositionScrollbars method, called from Setup, which is itself called from Init; this method ensures the scrollbar components are sized and positioned correctly, so this code performs that task for the shape.

```
dodefault()
with This
  .Shape1.Left   = .scrollbarhorizontal1.Width - 1
  .Shape1.Top    = .scrollbarvertical1.Height  - 1
  .Shape1.Width  = .Width  - .Shape1.Left
  .Shape1.Height = .Height - .Shape1.Top
  .Shape1.Anchor = 12
endwith
```

To see a sample of SFScrollContainer at work, run SFExample.SCX. It's a clone of Craig's Example.SCX, but uses SFScrollContainer rather than SBScrollContainer.

There's only remaining problem with SBScrollContainer that I haven't fixed yet: if you scroll the container so a control is partly obscured by a scrollbar, you can get bleed through in the scrollbar itself. For example, scroll the example form so the command button is half-visible, and then hover the mouse pointer over the button; you'll see the button completely drawn, covering part of the scrollbar. Changing ZOrder of the controls doesn't help.

Summary

In this article, I looked at two tools provided by Craig Boyd: a calendar control and a scrolling container. I actually planned to cover a third control that provides XP-like panels to VFP developers, but ran out of space, so I'll discuss that next month.

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, and the MemberData Editor, Anchor Editor, New Property/Method Dialog, and CursorAdapter and DataEnvironment builders that come with VFP. He is co-author of the "What's New in Visual FoxPro" series and "The Hacker's Guide to Visual FoxPro 7.0," all from Hentzenwerke Publishing. Doug has spoken at every Microsoft FoxPro Developers Conference (DevCon) since 1997 and at user groups and developer conferences all over the world. He is a long-time Microsoft Most Valuable Professional (MVP), having first been honored with this award in 1996. Web: www.stonefield.com and www.stonefieldquery.com Email: dhennig@stonefield.com