I Got Rendered Where? Part II

Doug Hennig

Thanks to the new ReportListener in VFP 9, we have a lot more control over where reports are rendered. Last month, Doug Hennig showed a listener that collaborates with a custom preview window to provide a "live" preview surface. This month, he takes it a step further and adds support for finding and highlighting text.

Last month, I showed a ReportListener subclass called DBFListener that renders a report to a cursor. The cursor contains information about each object in the report: the type of object (field, label, rectangle, etc.), its content (in the case of fields and labels), page number, horizontal and vertical position, and size. I also showed a custom preview window that uses the cursor to provide a "live" preview surface, one in which the user could click on a specific report object and fire an event in the form. This month, I'm going to extend the behavior of the preview form to support finding and highlighting text.

Finding text

Clicking on the Find button in the preview form's toolbar calls the FindText method of the preview form (the SFPreviewForm class in SFPreview.VCX). That method prompts the user for some text to find, calls ClearFind to clear any existing find settings, tries to find the specified text in the output cursor created by DBFListener, and calls the HandleFind method to deal with the results.

```
local lcText, ;
  lcObject
with This
  lcText = inputbox('Find:', 'Find Text')
  if not empty(lcText)
    .ClearFind()
    select (.cOutputAlias)
    .cTextToFind = lcText
    locate for ;
      upper(This.cTextToFind) $ upper(CONTENTS)
    .HandleFind(found())
  endif not empty(lcText)
endwith
```

FindNext, called from the Find Next button in the toolbar, simply uses CONTINUE to find the next instance and also calls HandleFind to handle the results.

HandleFind deals with the results of the search. If a record was found containing the desired text, HandleFind calls the AddHighlightedItem method to add the record number and page number of the record to the oHighlightedItems collection of items to highlight (we won't look at this method). One complication of highlighting the found text is that if the preview form is sized smaller than page size, the highlighted text may be outside the visible area of the page. So, HandleFind calls the ScrollToObject method to scroll the form so the highlighted text is within the visible area. It then either calls DrawPage or DisplayPage, both of which I discussed last month, depending on whether the first instance of the found text appears on the currently displayed page or on a different page. (In case you're wondering, ScrollToObject must be called before DrawPage or DisplayPage because the highlighting process, which we'll look at later, needs to have the highlighted area within the current viewable area.) If no matching record could be found, HandleFind beeps (using the Windows API MessageBeep function, declared in Init), clears the existing find information, and redraws the current page, thus clearing any previously highlighted text.

```
lparameters tlFound
local lcObject, ;
  loObject
with This

* If we found the object, add it to the collection
* of highlighted objects and flag that the Find Next
* function can be used.
```

```
  if tlFound
    lcObject = transform(recno())
    .AddHighlightedItem(lcObject, PAGE)
    .lCanFindNext = .T.

* Scroll the form if the found object isn't currently
* visible. If the object is on the current page,
* redraw it. Otherwise, display the proper page.

    scatter name loObject
    .ScrollToObject(loObject.Top/10, ;
      loObject.Height/10, loObject.Left/10, ;
      loObject.Width/10)
    if PAGE = .nCurrentPage
      .DrawPage()
      .RefreshToolbar()
    else
      .DisplayPage(PAGE)
    endif PAGE = .nCurrentPage

* We didn't find the text, so clear the find
* settings.

  else
    MessageBeep(16)
    .ClearFind()
    .DrawPage()
    .RefreshToolbar()
  endif tlFound
endwith
```

The ScrollToObject method does what its name suggests (I find that's usually the best way to name a method <g>)—it scrolls the form as necessary so the specified location is visible. This involves checking the various ViewPort* properties to see what part of the total image is currently displayed in the viewable area and calling SetViewPort to change the viewable area if necessary.

```
lparameters tnTop, ;
  tnHeight, ;
  tnLeft, ;
  tnWidth
local lnNewTop, ;
  lnNewLeft, ;
  lnVPos, ;
  lnHPos
with This
  lnNewTop  = .ViewPortTop
  lnNewLeft = .ViewPortLeft
  lnVPos    = tnTop  + tnHeight
  lnHPos    = tnLeft + tnWidth
  if not between(lnVPos, .ViewPortTop, ;
    .ViewPortTop + .ViewPortHeight)
    lnNewTop = lnVPos - .ViewPortHeight/2
  endif not between(lnVPos ...
  if not between(lnHPos, .ViewPortLeft, ;
    .ViewPortLeft + .ViewPortWidth)
    lnNewLeft = lnHPos - .ViewPortWidth/2
  endif not between(lnHPos ...
  if lnNewTop <> .ViewPortTop or ;
    lnNewLeft <> .ViewPortLeft
    .SetViewPort(lnNewLeft, lnNewTop)
  endif lnNewTop <> .ViewPortTop ...
endwith
```

Highlighting text

How is the found text highlighted? The DrawPage method, which I discussed last month, calls the HighlightObjects method to ensure any objects on the current page that exist in the collection of items to highlight are drawn in a manner that makes them stand out.

```
local lcObject, ;
  lcRepObject, ;
  loRepObject
with This
  for each loObject in .oHighlightedItems
    if loObject.Page = .nCurrentPage
      lcRepObject = loObject.Name
      loRepObject = evaluate('.oContainer.Object' + ;
        lcRepObject)
      .HighlightObject(loRepObject)
    endif loObject.Page = .nCurrentPage
  next loObject
endwith
```

The actual work of highlighting a specific object is done in HighlightObject. Remember that the page displayed in the preview window is really a GDI+ image, so we can't do much with it. As we saw last month, SFPreviewForm creates invisible shape objects (not really invisible, but without a border so they don't appear on the preview surface) for each rendered item in the current page of the report. So, to make the found text stand out, we simply have to make the object representing the text visible. You might think the code could simply do something like this:

```
Object.BorderWidth = 2
Object.BorderStyle = 1
Object.BorderColor = rgb(255, 0, 0)
```

However, that doesn't work for two reasons. First, even though it's transparent, the shape overlays the report image, blocking off the text the user wants to see. Second, the form continually flashes. It turns out that if something is changed when Paint fires (in SFPreviewForm, Paint calls DrawPage, which calls HighlightObjects, which calls HighlightObject, which would change the border of the shape if we used this code), it starts a vicious cycle: Paint fires, which changes the drawing surface, which causes Paint to fire, which changes the drawing surface, and so on. So, instead, we'll use GDI+ to draw a box around the shape.

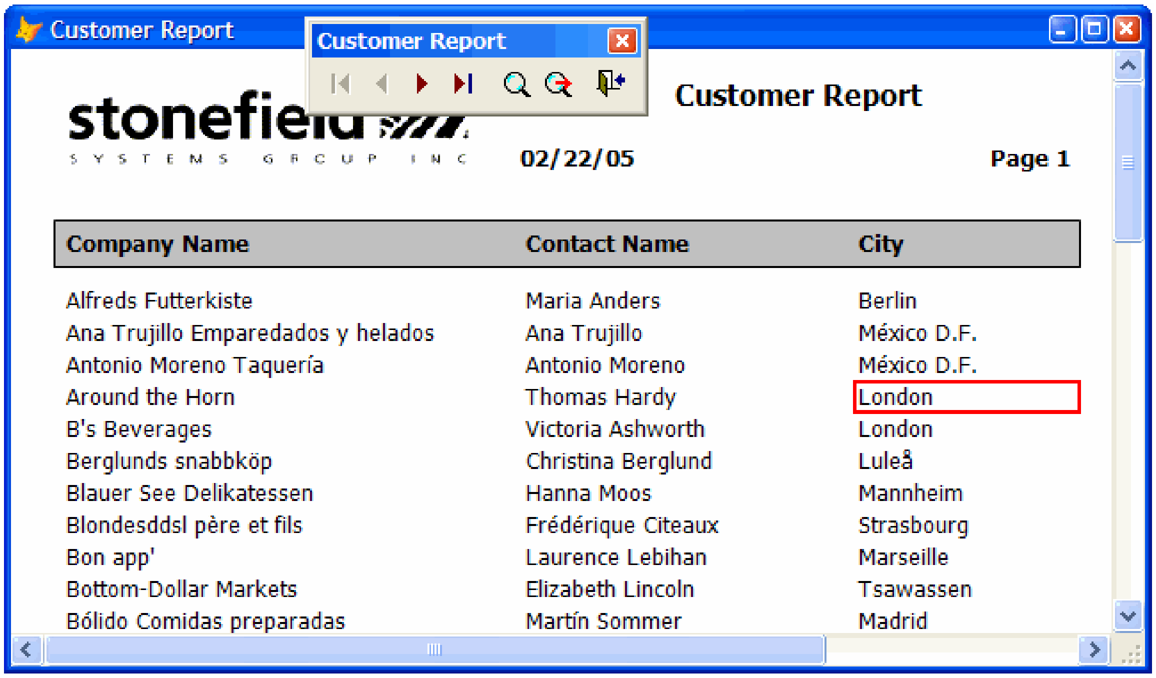
The Init method of the form instantiates a GpGraphics object from _GDIPlus.VCX in the FFC subdirectory of the VFP home directory. This class library, which provides an easy-to-use wrapper for GDI+ functions, was created by Walter Nicholls, and he discussed some of its classes in the August and September 2004 issues of FoxTalk. GDI+ needs a handle to the surface it draws on, so you normally call the CreateFromHWnd method of GpGraphics, passing it the HWnd property of the form. However, I ran into a snag using it with SFPreviewForm: when the ScrollBars property is set to anything but 0-None, none of the GDI+ drawing I did showed up on the form. Fortunately, Walter generously pointed out to me that drawing should actually occur on the child window of the form (the part inside the form that scrolls is actually another window), and that I could get a handle to the child window using the GetHandle Windows API function.

HighlightObject uses the GpGraphics object to draw a rectangle of the appropriate size and position for the shape object being highlighted. Set the custom nHighlightColor and nHighlightWidth properties of the form to the RGB value for the desired color and width of the box; they're set by default to 255 (Red) and 2. Note that the RGB values used by VFP are somewhat different than those used by GDI+, so the code in HighlightObject rearranges the color value so it works with GDI+.

```
lparameters toObject
local lnX1, ;
  lnY1, ;
  lnX2, ;
  lnRed, ;
  lnGreen, ;
  lnBlue, ;
  lnARGB, ;
  lnHWnd, ;
  loColor, ;
  loPen
with This

* Figure out the upper left corner of the box.

  lnX1 = toObject.Left + .oContainer.Left
  lnY1 = toObject.Top  + .oContainer.Top

* Figure out the ARGB color to use from the RGB
* value.

  lnRed   = bitand(.nHighlightColor, 255)
  lnGreen = bitrshift(bitand(.nHighlightColor, ;
    0x00FF00), 8)
  lnBlue  = bitrshift(bitand(.nHighlightColor, ;
    0xFF0000), 16)
  lnARGB  = 0xFF000000 + 0x10000 * lnRed + ;
    0x100 * lnGreen + lnBlue

* Get the handle of the form to draw on (since we're
* a scrollable form, we need to get the child of the
* form, not the form itself). Since we may have
* scrolled the form, we need to do this each time.

  #define GW_CHILD 5
  lnHWnd = GetWindow(.HWnd, GW_CHILD)
  .oGraphics.CreateFromHWnd(lnHWnd)

* Draw the box.

  loColor = createobject('GpColor', lnARGB)
  loPen   = createobject('GpPen')
  loPen.Create(loColor, .nHighlightWidth)
  .oGraphics.DrawRectangle(loPen, lnX1, lnY1, ;
    toObject.Width, toObject.Height)
endwith
```

Check it out

Let's see how it works. Run TestSFPreview.PRG to run a report and preview it using SFPreviewForm. Click the Find button in the toolbar and when prompted, enter something that appears multiple times, such as "London." You'll see a red box appear around the first instance of that text; see Figure 1 for an example. Click the Find Next button and notice that the box around the first instance disappears and the second instance is now highlighted. As you continue to click Find Next, you may see the preview form automatically scroll or move to another page so the found instance is always visible. Once no more instances are found, you'll hear a beep and no text will be highlighted.



Customer Report

02/22/05 Page 1

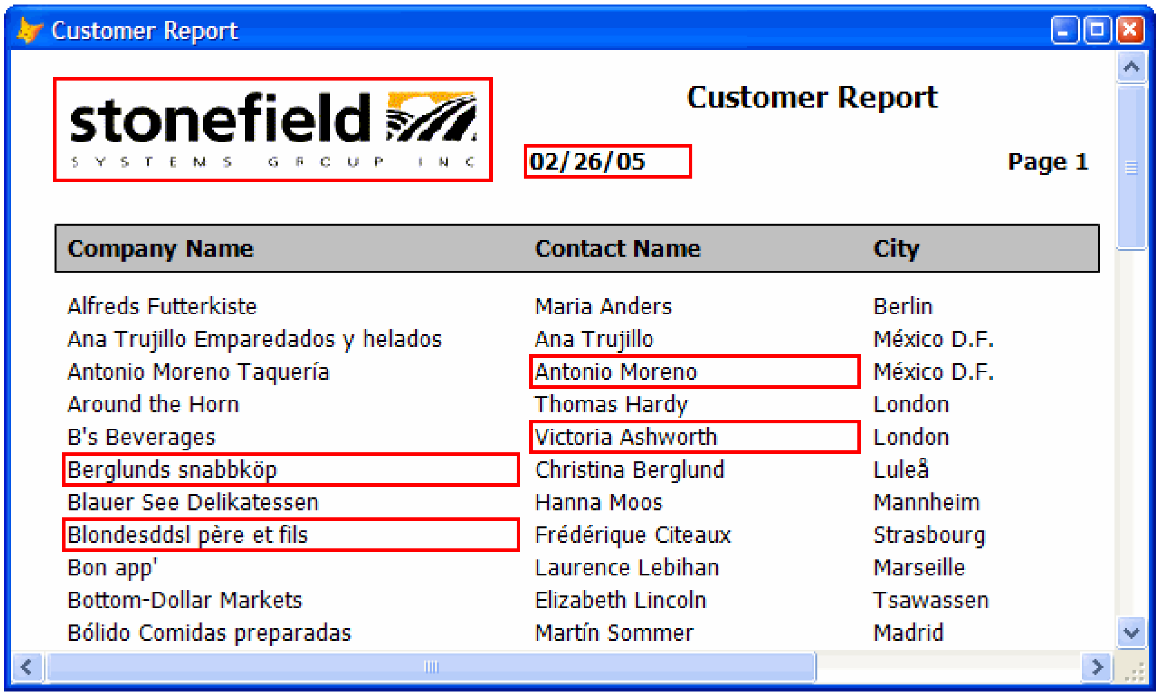
Company Name	Contact Name	City
Alfreds Futterkiste	Maria Anders	Berlin
Ana Trujillo Emparedados y helados	Ana Trujillo	México D.F.
Antonio Moreno Taquería	Antonio Moreno	México D.F.
Around the Horn	Thomas Hardy	London
B's Beverages	Victoria Ashworth	London
Berglunds snabbköp	Christina Berglund	Luleå
Blauer See Delikatessen	Hanna Moos	Mannheim
Blondesddsl père et fils	Frédérique Citeaux	Strasbourg
Bon app'	Laurence Lebihan	Marseille
Bottom-Dollar Markets	Elizabeth Lincoln	Tsawassen
Bólido Comidas preparadas	Martín Sommer	Madrid

Figure 1. SFPreviewForm highlights found text, adding an important feature to a report preview window.

One thing you might be curious about is why I used a collection to hold the highlighted items when only one item is highlighted at a time during a find. That's because I wanted to support the possibility of highlighting multiple items at a time. Perhaps you want the find functionality to find and highlight all instances of the found text rather than one at a time. Perhaps you want to support bookmarks, with each bookmarked item being highlighted.

Here's a simple example: TestSFPreview.PRG binds the ObjectClicked and ObjectRightClicked events to the OnClick and OnRightClick methods of a click handler class (I discussed how events are handled in last month's article). When you click an object, that object becomes highlighted. To unhighlight the object, right-click it. Figure 2 shows the results after I clicked on various objects.

```
loHandler = createobject('ClickHandler')
bindevent(loForm, 'ObjectClicked',      loHandler, 'OnClick')
bindevent(loForm, 'ObjectRightClicked', loHandler, 'OnRightClick')

* A class to handle object clicks.

define class ClickHandler as Custom
  procedure OnClick(toObject, toForm)
    local lcObject
    toObject.Top    = toObject.Top/10
    toObject.Left   = toObject.Left/10
    toObject.Width  = toObject.Width/10
    toObject.Height = toObject.Height/10
    lcObject = transform(toObject.RecordNumber)
    toForm.AddHighlightedItem(lcObject, ;
      toObject.Page)
    toForm.DrawPage()
  endproc

  procedure OnRightClick(toObject, toForm)
    local lcObject
    lcObject = transform(toObject.RecordNumber)
    toForm.oHighlightItems.Remove(lcObject)
    toForm.DrawPage()
  endproc
enddefine
```



Customer Report

02/26/05

Page 1

Company Name	Contact Name	City
Alfreds Futterkiste	Maria Anders	Berlin
Ana Trujillo Emparedados y helados	Ana Trujillo	México D.F.
Antonio Moreno Taquería	Antonio Moreno	México D.F.
Around the Horn	Thomas Hardy	London
B's Beverages	Victoria Ashworth	London
Berglunds snabbköp	Christina Berglund	Luleå
Blauer See Delikatessen	Hanna Moos	Mannheim
Blondesddsl père et fils	Frédérique Citeaux	Strasbourg
Bon app'	Laurence Lebihan	Marseille
Bottom-Dollar Markets	Elizabeth Lincoln	Tsawassen
Bólido Comidas preparadas	Martín Sommer	Madrid

Figure 2. Because it uses a collection, SFPreviewForm supports multiple highlighted objects.

Summary

Over the past several months, I've discussed the new ReportListener in VFP 9 and some of the ways it can be used to provide features we could only dream of in earlier versions. Among the things I've looked at are dynamically formatting text, hyperlinking objects within a report, having a "live" preview surface, and highlighting objects in a report. I'm going to move on to another topic next month, but rest assured that we'll find other cool things to explore in this area in the future.

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 North America. 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