Hyperlink Your Reports

Doug Hennig

Last month, Doug Hennig discussed the new ReportListener class in VFP 9 and how it can be used to control report output in ways that previously wasn't possible. This month, he looks at how output generated from reports can have live hyperlinks so some action is performed when they're clicked.

Wouldn't it be cool if you could tell VFP to add a hyperlink to a field in a report? Then the user could click on the hyperlink to navigate to some related information. For example, a report showing customers and their Web sites or email addresses would have live links; clicking on a Web site link would navigate their browser to that URL. Even more interesting would be the ability to navigate to something else within your application. For example, clicking on a company name in a report could bring up the customers data entry form with that company as the selected record.

Because the report preview window that ships with VFP doesn't support live, clickable objects in a report, the easiest way to implement this is using HTML, which natively supports hyperlinks.

Hyperlinking reports

VFP comes with a report listener that outputs HTML (the HTMLListener class built into ReportOutput.APP and in _ReportListener.VCX in the FFC folder), but I was sure it would require a lot of work to get it to support hyperlinks. However, I was pleasantly surprised to discover how little effort it required.

First, a little background. HTMLListener is a subclass of XMLDisplayListener, which is a subclass of XMLListener, which is a subclass of _ReportListener, the class I discussed last month and recommended you normally use as the parent class for your own listeners. When you use HTMLListener, either directly by instantiating it and using it as the listener for a report or by specifying OBJECT TYPE 5 in the REPORT command, it actually generates XML for the report (this is performed by its parent classes), then applies an XSL transform to the XML to generate the HTML. The XSLT to use is defined in the GetDefaultUserXSLTAsString method.

The default XSLT used by HTMLListener is very complex, and not being much of an XSL expert, I thought it might be an overwhelming task to figure out what to change to add support for hyperlinks. However, as I started poking through GetDefaultUserXSLTAsString, I discovered the following:

```
<xsl:when test="string-length(@href) &gt; 0">
  <A href="{@href}">
  <xsl:call-template name="replaceText"/>
  </A>
</xsl:when>
```

This XSL adds an anchor tag to the HTML if there's an HREF attribute on the current element in the XML. This is cool—it means HTMLListener already supports hyperlinks! However, searching for "HREF" turned up no hits in XMLDisplayListener or XMLListener, so how to add that attribute to an element, especially dynamically?

After doing some more poking around, I found that the attributes of a particular element were set in the GetRawFormattingInfo method of XMLListener. So, I subclassed HTMLListener and added the behavior I want to this method.

The following code, taken from HyperlinkListener.PRG, provides a listener that generates a hyperlink on an object in a report if that object's User memo contains the directive "*:URL =" followed by the expression to use as the URL.

```
define class HyperlinkListener as HTMLListener ;
  of home() + 'ffc\_ReportListener.vcx'
  QuietMode = .T.
    && default QuietMode to suppress feedback
  dimension aRecords[1]
    && an array of information for each record in FRX

* Before we run the report, go through the FRX and
* store information about any field with our expected
```

```
* directive in its USER memo into the aRecords array.

  function BeforeReport
    dodefault()
    with This
      .SetFRXDataSession()
      dimension .aRecords[reccount()]
      scan for atc('*:URL', USER) > 0
        .aRecords[recno()] = ;
          alltrim(strextract(USER, '*:URL =', ;
          chr(13), 1, 3))
      endscan for atc('*:URL', USER) > 0
      .ResetDataSession()
    endwith
  endfunc

* If the current field has a directive, add the URL
* to the attributes for the node.

  function GetRawFormattingInfo(tnLeft, tnTop, ;
    tnWidth, tnHeight, tnObjectContinuationType)
    local lcInfo, ;
      lnURL
    with This
      lcInfo = dodefault(tnLeft, tnTop, tnWidth, ;
        tnHeight, tnObjectContinuationType)
      lcURL  = .aRecords[recno('FRX')]
      if not empty(lcURL)
        .SetCurrentDataSession()
        lcInfo = lcInfo + ' href="' + ;
          textmerge(lcURL) + '"'
        .ResetDataSession()
      endif not empty(lcURL)
    endwith
    return lcInfo
  endfunc
enddefine
```

The BeforeReport event fires just before the report runs. It uses the SetFRXDataSession method to select the data session the FRX cursor is in, and then scans through the FRX and puts the URL expression for any object that has the directive into an array. It calls ResetDataSession at the end to restore the data session the listener is in.

The GetRawFormattingInfo method uses DODEFAULT() to perform the usual behavior, which generates the attributes for an XML element as a string. It then checks the appropriate array element (the data session for the FRX cursor was selected by code in XMLListener before this code executes) to see whether the current object in the report has the directive, and if so, adds an HREF attribute to the XML element. It calls SetCurrentDataSession to select the data session used by the report's data and uses TEXTMERGE() on the URL expression because the expression will likely contain something specific for each record, such as <<CustomerID>>.

That's it! Let's look at some examples of how we can use this listener.

Example 1: Live links to URLs

Links.FRX is a simple example that shows how this listener works. It reports on the Links table, which has a list of company names and their Web sites. The website field in the report has "*:URL = http://<<trim(website)>>" in its User memo. Links.PRG runs this report, using HyperlinkListener as the report listener, and uses the _ShellExecute class in the FFC to display the HTML file in your default browser. **Figure 1** shows the results.

```
loListener = newobject('HyperlinkListener', ;
  'HyperlinkListener.prg')
loListener.TargetFileName = fullpath('Links.html')
report form Links object loListener
loShell = newobject('_ShellExecute', ;
  home() + 'ffc\_Environ.vcx')
```

```
loShell.ShellExecute(loListener.TargetFileName)
```

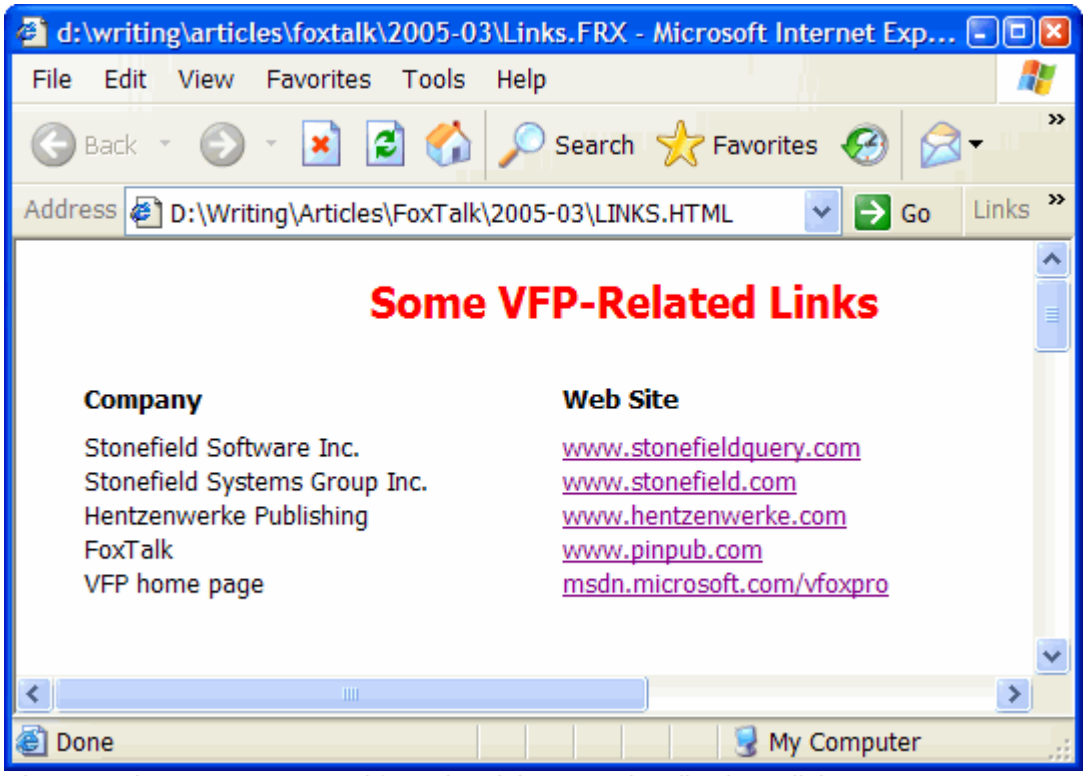

Figure 1. The HTML generated from the Links report has live hyperlinks.

Example 2: Drilldown reports

HyperlinkReports.SCX is a more complex example. As you can see in **Figure 2**, it presents a list of customer information. However, this HTML is displayed in a Web Browser ActiveX control imbedded in a VFP form rather than a browser window. Clicking on a company name runs a report of orders for that customer and displays it in the form, as shown in **Figure 3**. The orders report also has a hyperlink that returns the display to the customer list. So, this form provides drilldown reports.

Figure 2. HyperlinkReports.SCX shows a customer report with each customer hyperlinked to their orders.

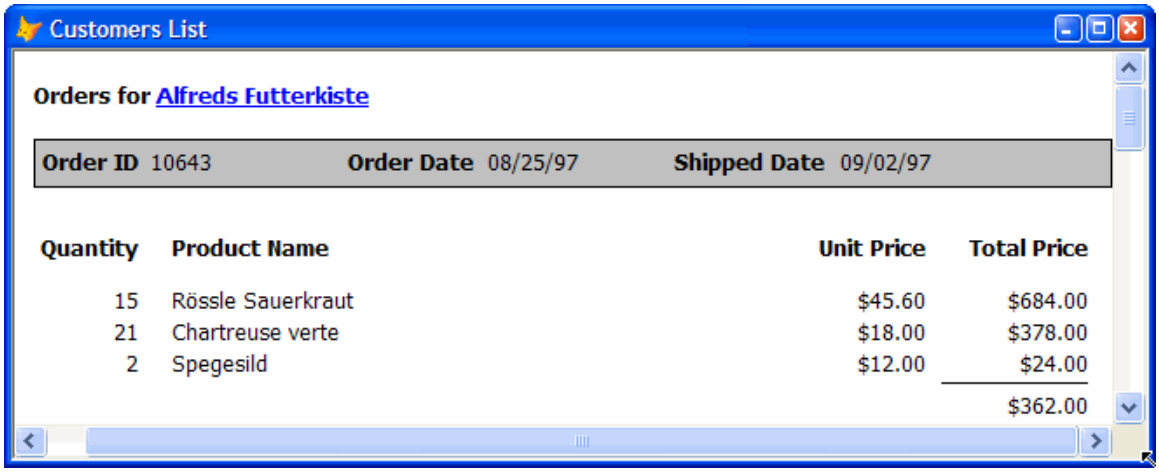

Figure 3. Clicking on a customer's link displays a report of the orders for that customer.

The Init method of the form uses the HyperlinkListener class to generate a hyperlinked HTML file from the HyperlinkCustomers report, and then calls the ShowReport method to display it in the Web Browser control. It also maintains a collection of HTML files generated by the form so they can be deleted when the form is closed.

```
with This

* Create a collection of the HTML files we'll create
* so we can nuke them all when we close.

  .oFiles = createobject('Collection')

* Create the customers report.

  .oListener = newobject('HyperlinkListener', ;
    'HyperlinkListener.prg')
  .oListener.TargetFileName = ;
    fullpath('HyperlinkCustomers.html')
  report form HyperlinkCustomers object .oListener
  .oFiles.Add(.oListener.TargetFileName)

* Display it.

  .ShowReport()
endwith
```

The ShowReport method simply tells the Web Browser control to load the current HTML file:

```
local lcFile
lcFile = This.oListener.TargetFileName
This.oBrowser.Navigate2(lcFile)
```

Rather than generating orders reports for every customer and hyperlinking to them, I decided to generate the reports on demand when a customer name is clicked. To do that, I needed to intercept the hyperlink click. Fortunately, that's easy to do: simply put code into the BeforeNavigate2 event of the Web Browser control.

To tell BeforeNavigate2 that this isn't a normal hyperlink, I used a convention of "vfps://," which stands for "VFP script," rather than "http://." The code in BeforeNavigate2 looks for this string in the URL to be navigated to, and if found, executes the code in the rest of the URL rather than navigating to it. For example, the User memo of the CompanyName field in HyperlinkCustomers.FRX has the following:

```
*:URL = vfps://Thisform.ShowOrdersForCustomer('
<<CustomerID>>')
```

The HyperlinkListener report listener will convert this to an anchor tag such as <a href="vfps://Thisform.ShowOrdersforCustomer('ALFKI')"> for the customer with a CustomerID of ALFKI. When you click on this hyperlink in the Web Browser control, BeforeNavigate2 fires and the code in that event strips off the "vfps://" part and executes the rest. It also sets the Cancel parameter, passed by reference, to .T. to indicate that the normal navigation should not take place (similar to using NODEFAULT in a VFP method). Here's the code for BeforeNavigate2:

```
LPARAMETERS pdisp, url, flags, targetframename, ;
  postdata, headers, cancel
local lcMethod
if url = 'vfps://'
  lcMethod = substr(url, 8)
  lcMethod = left(lcMethod, len(lcMethod) - 1)
    && strip trailing /
  &lcMethod
  cancel = .T.
endif url = 'vfps://'
```

The ShowOrdersForCustomer method, executed when you click on a company name, runs the HyperlinkOrders report for the specified customer, displays it in the Web Browser control, and adds the file name to the collection of files to be deleted when the form is closed.

```
lparameters tcCustomerID
with This
  .oListener.TargetFileName = ;
    fullpath(tcCustomerID + '.html')
  report form HyperlinkOrders object .oListener ;
    for Orders.CustomerID = tcCustomerID
  .ShowReport()
  .oFiles.Add(.oListener.TargetFileName)
endwith
```

The CustomerName field in the HyperlinkOrders report has "*:URL = vfps://Thisform.ShowCustomers()" in its User memo so clicking on this hyperlink in the report redisplays the customer list.

Example 3: Launching a VFP form

CustomerReport.SCX is similar to HyperlinkReports.SCX, albeit a little simpler. It too hosts a Web Browser control that displays the HTML from a report, EditCustomers.FRX, which looks the same as the previous example. However, clicking on a customer name in this form displays a maintenance form for the selected customer.

EditCustomers.FRX is a clone of the HyperlinkCustomers report used in the previous example, but has "*:URL = vfps://Thisform.EditCustomer('<<CustomerID>>')" in the User memo of the CompanyName field instead. The form's EditCustomer method, called from BeforeNavigate2 when a customer name is clicked, launches the Customers form, passing it the CustomerID for the selected customer. The Customers form is a simple maintenance form for the Customers table, with controls bound to each field and Save and Cancel buttons.

NavPaneListener

MVP Fabio Vazquez has created another kind of listener that has hyperlinks, albeit for a completely different purpose. His NavPaneListener, available for download from http://ReportListener.com, provides an HTML report previewer with a table of contents for the report. As you can see in **Figure 4**, a thumbnail image of each page is shown at the left and the current page is shown at the right. Clicking on a thumbnail navigates to the appropriate page.

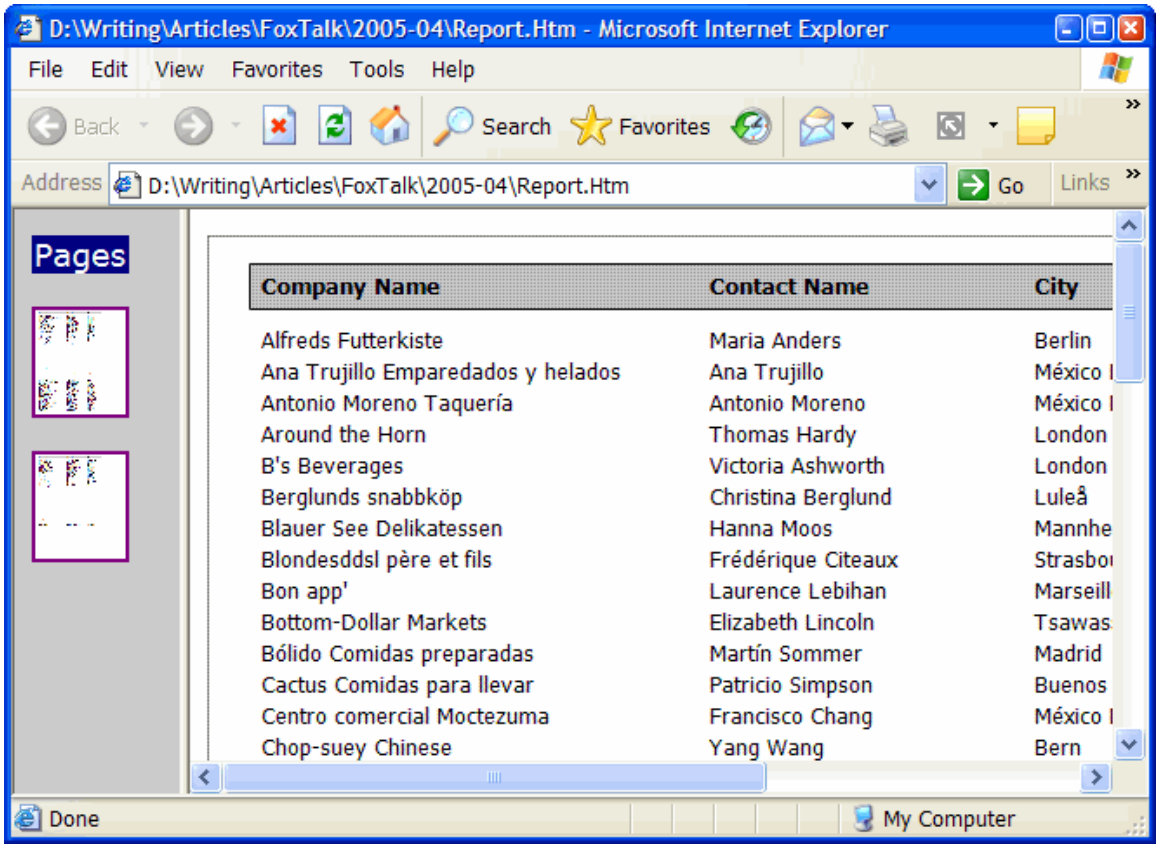

Figure 4. Fabio Vazquez's NavPaneListener creates an HTML report previewer with a table of contents.

Like HyperlinkListener, NavPaneListener is quite simple. Its OutputPage event, called as each page is to be output, simply generates a GIF file for the page by calling itself again with the appropriate parameters. tnDeviceType is initially -1, meaning no output, because the listener's ListenerType property is set to 2. In that case, OutputPage calls itself, passing the name and path for a GIF to generate (the cPath property defaults to the current directory) and a device type value which indicates a GIF file. On the second call, in addition to the normal behavior (generating the specified GIF file), OutputPage passes the name and path to the AddPage method of a collaborating object stored in the oNavigator property. (Note: I translated some of Fabio's code into English to make it more readable.)

```
procedure OutputPage(tnPageNo, teDevice, ;
  tnDeviceType)
  local lnDeviceType
  with This
    do case
      case tnDeviceType = -1  && None
        lnDeviceType = 103     && GIF
        .OutputPage(tnPageNo, .cPath + 'Page' + ;
          transform(tnPageNo) + '.gif', lnDeviceType)
      case tnDeviceType = 103
        .oNavigator.AddPage(teDevice)
    endcase
  endwith
endproc
```

After the report is done, the navigator object creates a couple of HTML documents, one that defines a frameset with the table of contents in the left frame and the contents to display in the right frame, and one that contains the table of contents as thumbnails of the GIF files hyperlinked to display the full-size GIF file

in the content frame. The navigator object then automates Internet Explorer to display the frameset document.

Summary

Notice that none of the code in any of these examples is complicated, nor is there much of it. As a result, it takes only a few moments to implement reports with live hyperlinks, drilldown reports, reports that launch some VFP form or other action, or reports with navigation panes. This truly shows the power of report listeners!

Next month we'll look at a similar topic—report previews that perform some action when clicked—but using an entirely different technique that'll give us abilities such as text search and bookmarks.

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