Web Page Components

Doug Hennig

With its fast and powerful string functions, VFP is a great tool for generating HTML. This month, Doug Hennig shows how you can treat Web pages as a collection of reusable HTML "components", to easily generate both static and dynamic Web sites.

When I was recently revamping my Web sites (www.stonefield.com and www.stonefieldquery.com), I found myself copying and pasting between pages a lot. When I'm coding, copying and pasting sets off alarm bells in my mind because of all the potential problems and extra work involved. With an object-oriented language like VFP, we strive to create reusable code through inheritance and other mechanisms. What I wanted was something similar for HTML: the ability to reuse different parts of a document in other documents. I'll call these pieces "components". Ideally, I'd like to be able to create a Web page by simply combining different components in the proper order.

After reading a chapter in the excellent *MegaFox: 1002 Things You Wanted to Know About Extending Visual FoxPro* by Marcia Akins, Andy Kramek, and Rick Schummer, I was inspired to create the tools that I need to support this in VFP. This month's article is the result.

Data driving the process

The chapter in MegaFox that describes generating HTML from VFP states that the best way to do this is to data-drive the process. That means that the code that generates the HTML should do so by reading not only the content from tables, but also the order the content appears in and even the classes to use to perform the generation. I decided to create three tables for this: PAGES, which specifies the Web pages to generate, CONTENT, which contains information about the components used in a Web site, and PAGECONTENT, which serves as a join table between the other two, specifying which components appear on each page as well as how they should be formatted (for true reuse, the format of a component should be separate from its content). The structures of these tables are shown in Tables 1, 2, and 3.

Table 1. PAGES.DBF specifies the Web pages to generate.

FIELD	TYPE	PURPOSE
ID	I (auto-inc)	Primary key
PAGE	C(30)	The name of the page
TITLE	C(60)	The title for the page
TEMPLATE	C(60)	The name of a template page to use for static content (such as the <HEAD> section)
INDENT	N(2)	How much to indent generated content if formatting is desired

Table 2. CONTENT.DBF contains information about the components used in a Web site.

FIELD	TYPE	PURPOSE
ID	I (auto-inc)	Primary key
HEADER	C(60)	The header for the component
CONTENT	M	The body of the component
FOOTER	C(60)	The footer for the component
HTML	L	.T. if CONTENT contains HTML that should not be formatted

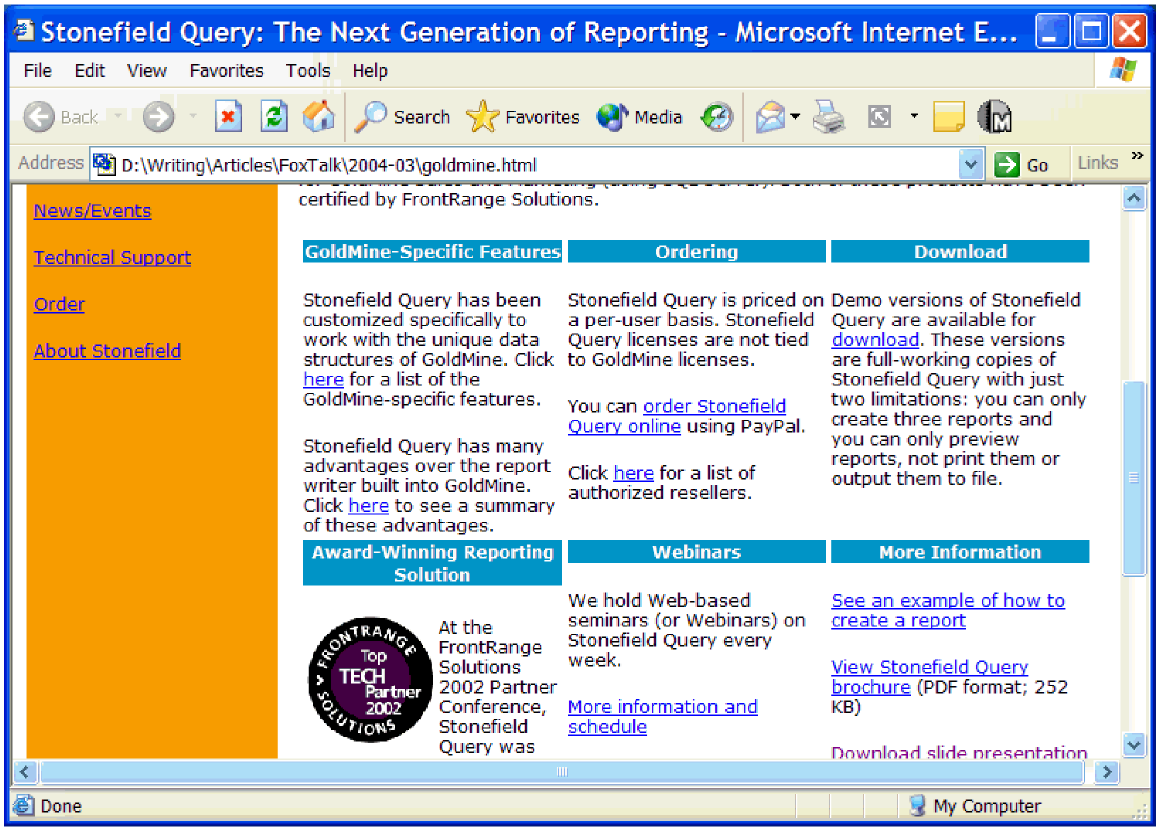
Table 3. PAGECONTENT.DBF specifies which components appear on each page.

FIELD	TYPE	PURPOSE
ID	I (auto-inc)	Primary key
PAGEID	I	The ID for a PAGE record
CONTENTID	I	The ID for a CONTENT record
ORDER	I	The order the component should appear in on the page
PARENT	I	The ID of a record in PAGECONTENT that this component is a child of
CLASS	C(30)	The name of a class used to render this component
LIBRARY	C(30)	The name of the library containing the class specified in CLASS
PROPERTIES	M	Code that sets properties of the class specified in CLASS

A few explanations are in order. First, rather than generating all of the HTML for a given Web page, I decided that only the part that varies from page to page should be generated. For example, many Web sites have a consistent look because they use many of the same components on each page, such as a company logo and a navigation bar. It doesn't make sense to generate this static HTML for each page, so PAGES.DBF has a TEMPLATE field that specifies an HTML file used as the template for the page. A placeholder in this file is replaced with the generated HTML. To change the overall appearance of a Web site, simply alter the template file or specify a different one.

A second comment is that for consistent appearance, most components will use styles defined in a CSS (cascading style sheet) file. Like a style in Microsoft Word, a CSS style applies a consistent format, such as font and color, to HTML elements. Some of the classes we'll look at later have properties in which you can store the names of CSS styles to use for the various elements of a component.

A final note is that many of my Web pages have small sections of text that are laid out in several columns of a table. For want of a better name, I called these components "snippets". In Figure 1, the "GoldMine-Specific Features", "Ordering", and "Download" sections are snippets.


Stonefield Query: The Next Generation of Reporting - Microsoft Internet E...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address D:\Writing\Articles\FoxTalk\2004-03\goldmine.html
Go Links
News/Events
Technical Support
Order
About Stonefield
Done
My Computer

certified by FrontRange Solutions.

GoldMine-Specific Features

Stonefield Query has been customized specifically to work with the unique data structures of GoldMine. Click here for a list of the GoldMine-specific features.

Stonefield Query has many advantages over the report writer built into GoldMine. Click here to see a summary of these advantages.

Ordering

Stonefield Query is priced on a per-user basis. Stonefield Query licenses are not tied to GoldMine licenses.

You can order Stonefield Query online using PayPal.

Click here for a list of authorized resellers.

Download

Demo versions of Stonefield Query are available for download. These versions are full-working copies of Stonefield Query with just two limitations: you can only create three reports and you can only preview reports, not print them or output them to file.

Award-Winning Reporting Solution

FRONTRANGE SOLUTIONS Top TECH Partner 2002

At the FrontRange Solutions 2002 Partner Conference, Stonefield Query was

Webinars

We hold Web-based seminars (or Webinars) on Stonefield Query every week.

More information and schedule

More Information

See an example of how to create a report

View Stonefield Query brochure (PDF format; 252 KB)

Download slide presentation

Figure 1. This Web page has several snippets.

The design of these tables not only allows for component reuse (a given component can appear on more than one page by simply having more than one record in PAGECONTENT with the same value for CONTENTID), it allows a component to appear differently on different pages because the class used to render a component is specified in PAGECONTENT rather than CONTENT.

The HTML rendering classes

The classes that render the components specified in the CONTENTS table are defined in SFHTML.VCX. SFHTML is the parent class for all of the other classes in this class library. It's a subclass of Custom with only one custom method, Render, which is abstract in this class.

SFHTMLComponent is used to render HTML for a component. It has properties to hold the header, body, and footer (cHeader, cBody, and cFooter) of the component, as well as the CSS styles to use for these items (cHeaderClass, cBodyClass, and cFooterClass). Although spaces are irrelevant in HTML, you may want to format the HTML for the component with indents to make it more human-readable. The nIndent property indicates how much to indent the HTML by, and nChildIndent indicates how much extra to indent certain structures (such as <TR> and <TD> tags within a <TABLE>). The lCRAfterContent property determines whether a carriage return and line feed are added after each line of the HTML. The Render method creates and returns the HTML for the component. It mostly just delegates tasks to other methods so this class can more easily be subclassed. We won't look at the other methods due to space limitations, but they're fairly simple. Notice that Render declares lcHTML as PRIVATE rather than LOCAL because the class uses text merge to output the component's HTML to this variable, and we want to avoid scoping issues.

```
private lcHTML
local loException as Exception
with This
  lcHTML = ''

* Set up text merge and output each part of the
* component.

  try
    .SetupTextMerge()
    .RenderHeader()
    .RenderBody()
    .RenderFooter()

* Just throw any errors to the next level handler.

  catch to loException
    loException.Procedure = This.Name + '.' + ;
      loException.Procedure
    throw

* Turn off text merge and clean up.

  finally
    .CleanupTextMerge()
  endtry

* Add a line break to the end of the HTML.

  lcHTML = lcHTML + .cLineBreak
endwith
return lcHTML
```

SFHTMLContainer provides a container for one or more components. Each component is stored in a collection contained in the oContentCollection property (the collection is instantiated in the Init method). The Render method simply calls the Render method of each component and concatenates the results together.

```
local lcHTML, ;
  loContent
with This
  lcHTML = ''
  for each loContent in .oContentCollection
    loContent.nIndent = .nIndent + .nChildIndent
    lcHTML = lcHTML + loContent.Render()
  next loContent
endwith
return lcHTML
```

SFHTMLSnippetContainer is a subclass of SFHTMLContainer that's used to hold snippet components. It has an nColumns property that specifies how many columns across the page to place the snippets into. Its Render method is more complex than SFHTMLContainer's because it has to create an HTML table to hold the snippets and place the HTML output for each snippet into a cell in the table. The cSnippetContainerClass and cSnippetCellClass properties specify the CSS styles to use for the table and for the cells.

```
local lcIndent1, ;
  lcIndent2, ;
  lcIndent3, ;
  lnIndent4, ;
  lcHTML, ;
  lnRows, ;
  lnRow, ;
  lnColumn, ;
  lnTopic, ;
  loContent, ;
  loException as Exception
with This

* Determine the indentation we'll need at each level and
* the character to add to the end of each line.

  try
    lcIndent1   = space(.nIndent)
    lcIndent2   = space(.nIndent + .nChildIndent)
    lcIndent3   = space(.nIndent + .nChildIndent * 2)
    lnIndent4   = .nIndent + .nChildIndent * 3
    .cLineBreak = .GetLineBreak()

* Start the HTML with the table for the snippets. Figure
* out how many rows we'll need.

    lcHTML = lcIndent1 + ;
      '<table class="' + .cSnippetContainerClass + ;
      '">' + .cLineBreak
    lnRows = ceiling(.oContentCollection.Count/.nColumns)

* Process each row. Start by creating a row, then process
* each column by outputting a cell and having the
* appropriate topic object render its output into that
* cell.

    for lnRow = 1 to lnRows
      lcHTML = lcHTML + lcIndent2 + '<tr>' + .cLineBreak
      for lnColumn = 1 to .nColumns
        lnTopic = (lnRow - 1) * .nColumns + lnColumn
        if lnTopic <= .oContentCollection.Count
          lcHTML    = lcHTML + lcIndent3 + ;
            '<td class="' + .cSnippetCellClass + '" ' + ;
            'width="' + transform(int(100/.nColumns)) + ;
            '%">' + .cLineBreak
          loContent = .oContentCollection.Item(lnTopic)
          loContent.nIndent = lnIndent4
          lcHTML = lcHTML + loContent.Render()
          lcHTML = lcHTML + lcIndent3 + '</td>' + ;
            .cLineBreak
        endif lnTopic <= .oContentCollection.Count
      next lnColumn
      lcHTML = lcHTML + lcIndent2 + '</tr>' + .cLineBreak
    next lnRow
    lcHTML = lcHTML + lcIndent1 + '</table>' + .cLineBreak

* Just throw any errors to the next level handler.

  catch to loException
    loException.Procedure = This.Name + '.' + ;
      loException.Procedure
    throw
  endtry
endwith
return lcHTML
```

Individual snippet topics are rendered with the SFHTMLSnippetTopic class, which is simply a subclass of SFHTMLComponent with the cHeaderClass, cBodyClass, and cFooterClass properties filled with the appropriate CSS styles to use.

SFHTMLCalendar is a more complicated subclass of SFHTMLComponent that renders calendars. The nMonth and nYear properties contain the month and year for the calendar. Various properties, such as cCalendarTableClass and cCalendarDayClass, contain the names of the CSS styles to use for the appropriate parts of the calendar. The content to display for each day comes from a table whose name is specified in the cTable property. This table must have a field containing the date on which something will occur, which is stored in the cDateField property, and a field containing the information for that date, specified in the cContentField property. You can also specify a filter to apply to the table (for example, if you have a table containing all types of training classes but want to only show certain types in a given calendar) in the cFilter property. We won't look at the code in this class for space purposes, so feel free to examine it yourself.

The last class we'll examine is SFHTMLPage. This class, based on SFHTML, is responsible for rendering an entire HTML page. Its Render method is data-driven because it reads the content to output from the CONTENT, PAGES, and PAGECONTENT tables described earlier. It first figures out what components it needs and adds each to a collection. It then spins through the collection and asks each object to render itself. Finally, it inserts the combined HTML of all objects into the appropriate place (marked with a placeholder specified in the cInsertText property) of the template file specified in the TEMPLATE column of the PAGES table.

```
lparameters tcPageName
local lcTemplate, ;
  lcID, ;
  loContent, ;
  loException as Exception, ;
  lcParent, ;
  loParent, ;
  lcHTML, ;
  lcPage
with This

* Ensure we have the name of the page to create.

  if vartype(tcPageName) <> 'C' or empty(tcPageName)
    throw 'The name of the page to generate was not ' + ;
      'specified.'
  endif vartype(tcPageName) <> 'C' ...

* Get the content for our page. Give an error if there
* isn't any.

  try
    select PAGECONTENT.ID, ;
        CONTENT.HEADER, ;
        CONTENT.CONTENT, ;
        CONTENT.FOOTER, ;
        CONTENT.HTML, ;
        PAGECONTENT.PARENT, ;
        PAGECONTENT.CLASS, ;
        PAGECONTENT.LIBRARY, ;
        PAGECONTENT.PROPERTIES, ;
        PAGES.TEMPLATE, ;
        PAGES.INDENT ;
      from PAGECONTENT ;
      join PAGES ;
        on PAGECONTENT.PAGEID = PAGES.ID ;
      left outer join CONTENT ;
        on PAGECONTENT.CONTENTID = CONTENT.ID ;
      where upper(PAGES.PAGE) = upper(tcPageName) ;
      order by PAGECONTENT.ORDER ;
      into cursor CONTENTFORPAGE
    if _tally = 0
      throw 'There is no definition for the ' + ;
```

```
        tcPageName + ' page.'
    endif _tally = 0

* Ensure the template file we're supposed to use exists.

    lcTemplate = alltrim(TEMPLATE)
    do case
      case empty(lcTemplate)
        throw 'No template file was specified.'
      case not file(lcTemplate)
        throw 'The template file ' + lcTemplate + ;
          ' was not found.'
    endcase

* Create a collection for the content objects.

    .oContentCollection = createobject('Collection')

* Go through the content cursor and add objects to our
* collection.

    scan
      lcID = transform(ID)
      try
        loContent = newobject(alltrim(CLASS), ;
          alltrim(LIBRARY))
      catch to loException ;
        when loException.ErrorNumber = 1733
        throw 'The class specified for page content ' + ;
          ID ' + lcID + ' was not found.'
      catch
        loException.Procedure = This.Name + '.' + ;
          loException.Procedure
        throw
      endtry
      with loContent
        .cHeader = alltrim(HEADER)
        .cBody   = CONTENT
        .cFooter = alltrim(FOOTER)
        .lHTML   = HTML
        .nIndent = INDENT
        if not empty(PROPERTIES)
          execscript(PROPERTIES)
        endif not empty(PROPERTIES)
      endwith

* If this item doesn't have a parent, add it to our
* collection. Otherwise, add it to the parent object.

      if empty(PARENT)
        .oContentCollection.Add(loContent, lcID)
      else
        try
          lcParent = transform(PARENT)
          loParent = .oContentCollection.Item(lcParent)
          loParent.oContentCollection.Add(loContent, ;
            lcID)
        catch to loException
          throw 'Invalid parent for page content ID ' + ;
            lcID + '.'
        endtry
      endif empty(PARENT)
    endscan

* Now tell each object to render itself.

    lcHTML = ''
    for each loContent in .oContentCollection
      lcHTML = lcHTML + loContent.Render()
    next loContent
```

```
* If we don't have any content, we have a problem.

    if empty(lcHTML)
      throw 'There is no content for this page.'
    endif empty(lcHTML)

* Insert the HTML into the template.

    lcPage = strtran(filetostr(lcTemplate), ;
      .cInsertText, lcHTML)

* Throw any errors up to the next level handler.

  catch to loException
    loException.Procedure = This.Name + '.' + ;
      loException.Procedure
    throw
  endtry
endwith
return lcPage
```

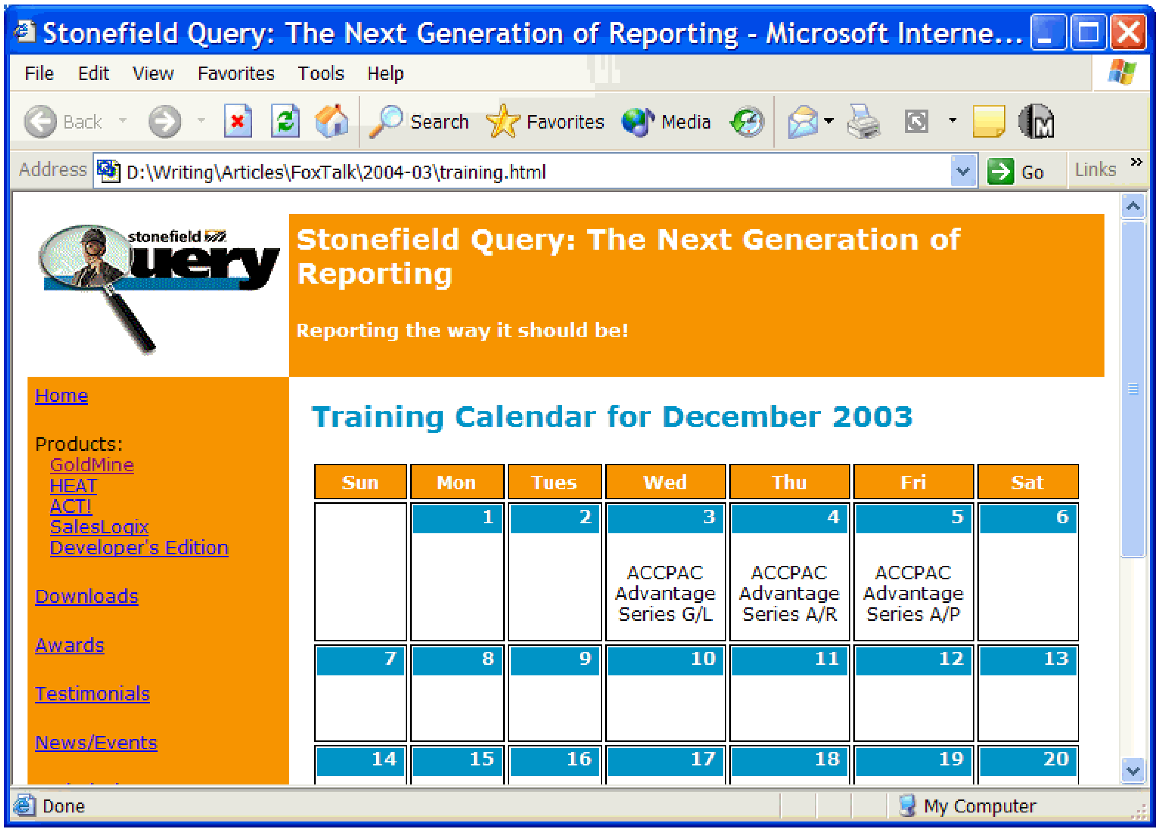
There are a lot of other classes you can create, depending on your needs. For example, you may wish to show some data (such as from a VFP or SQL Server table) in a grid or even a chart. By subclassing SFHTMLComponent, you can create classes that render content into any form you wish.

Check it out

Now that we've looked at a lot of code, let's see how it works. To generate one of the pages for my Web site, I added a record to the PAGES table that specified the name of the page to create, and I created several topics in the CONTENT table, each one of which represents a component to display in the page. I then added records to the PAGECONTENT table that joins the various topics to the page and specifies the type of component to use for each topic. Finally, I created a small test program called TESTPAGE.PRG to generate the page and display it in my default browser. Figure 1 shows the page generated from this program.

```
set classlib to SFHTML
loPage = createobject('SFHTMLPage')
lcHTML = loPage.Render('goldmine.html')
lcFile = sys(5) + curdir() + 'goldmine.html'
strtofile(lcHTML, lcFile)
declare integer ShellExecute in SHELL32.DLL ;
  integer nWinHandle, string cOperation, ;
  string cFileName, string cParameters, ;
  string cDirectory, integer nShowWindow
ShellExecute(0, '', lcFile, '', '', 0)
```

Using nearly identical code (only the name of the page to generate is different), TESTCALENDAR.PRG generates the calendar shown in Figure 2.


Stonefield Query: The Next Generation of Reporting - Microsoft Interne...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address D:\Writing\Articles\FoxTalk\2004-03\training.html
Go Links
Done
My Computer

Stonefield Query: The Next Generation of Reporting

Reporting the way it should be!

Home

Products:
GoldMine
HEAT
ACT!
SalesLogix
Developer's Edition

Downloads

Awards

Testimonials

News/Events

Training Calendar for December 2003

Sun	Mon	Tues	Wed	Thu	Fri	Sat
	1	2	3 ACCPAC Advantage Series G/L	4 ACCPAC Advantage Series A/R	5 ACCPAC Advantage Series A/P	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20

Figure 2. The SFHTMLCalendar class generates calendars such as this one.

Generating a Web site

There are at least a couple of ways you can use the classes in SFHTML.VCX to generate the pages of a Web site. First, you can generate a static Web site by spinning through the PAGES table and generating each page specified. GENERATEWEBSITE.PRG does just that:

```
local loPage, ;
  lcFile, ;
  lcHTML
set classlib to SFHTML
loPage = createobject('SFHTMLPage')
use PAGES
scan
  lcFile = trim(PAGE)
  lcHTML = loPage.Render(lcFile)
  strtofile(lcHTML, lcFile)
endscan
use
```

A more interesting use is to generate pages dynamically. The user's browser will access an ASP or ASP.NET page that instantiates a VFP COM DLL to render the desired HTML. Here's a program (GENERATEASP.PRG) that generates ASP pages for each of the pages specified in PAGES.DBF. Each ASP page has nearly identical content (although that could certainly be changed): it instantiates the WebPages class in the WebPages.DLL and calls the GeneratePage method with the name of the desired page in PAGES.DBF.

```
local lcHTML, ;
  loPage, ;
  lcFile

* Generate the ASP content.
```

```
set textmerge off
text to lcHTML noshow
<%Language="VBScript"%>
<html>
<head>
<title><<trim(TITLE)>></title>
<link rel="stylesheet" type="text/css" href="stonefield.css">
</head>
<body>
<%
dim WebApp
set WebApp = Server.CreateObject("WebPages.WebPages")
Response.Write(WebApp.GeneratePage("<<trim(PAGE)>>"))
%>
</body>
</html>
endtext

* Now create the ASP pages.

set classlib to SFHTML
loPage = createobject('SFHTMLPage')
use PAGES
scan
  lcFile  = forceext(trim(PAGE), 'asp')
  strtofile(textmerge(lcHTML), lcFile)
endscan
use
```

WebPages.DLL is built from WebPages.PJX. This project contains SFHTML.VCX and WebPages.PRG, which has the definition for the WebPages class. The WebPages class simply instantiates SFHTMLPage and calls its Render method with the specified page.

```
define class WebPages as Session olepublic
  function GeneratePage(PageName as String) as String
    local loPage, ;
      lcHTML, ;
      loException as Exception
    try
      set path to justpath(_vfp.ServerName)
      set classlib to SFHTML
      loPage = createobject('SFHTMLPage')
      lcHTML = loPage.Render(PageName)
    catch to loException
      comreturnerror('WebPages', 'Error #' + ;
        transform(loException.ErrorNo) + ;
        ' occurred in line ' + ;
        transform(loException.LineNo) + ' of ' + ;
        loException.Procedure + ': ' + ;
        loException.Message)
    endtry
  return lcHTML
  endfunc
enddefine
```

To try this out, run GENERATEASP.PRG, then copy the generated ASP pages to the root of your IIS Web site (such as C:\INetPub\WWWRoot) or some other virtual directory. Open WebPages.PJX and build a multi-threaded DLL from it. Then open your browser and enter “http://localhost/goldmine.asp” for the URL. You should see the Web page shown in Figure 1.

Summary

This month, we looked at a set of classes that allow you to generate a Web page as a collection of reusable HTML components. However, as you’ll soon discover when you start working with it, populating the tables that drive the classes isn’t much fun in BROWSE windows. Next month, we’ll look at a front-end UI for these tables to make that task much easier.

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, author of the CursorAdapter and DataEnvironment builders that come with VFP 8, and co-author of "What's New in Visual FoxPro 8.0", "What's New in Visual FoxPro 7.0", and "The Hacker's Guide to Visual FoxPro 7.0", 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 Microsoft Most Valuable Professional (MVP) and Certified Professional (MCP). Web: www.stonefield.com and www.stonefieldquery.com Email: dhennig@stonefield.com