Visual FoxPro and the Internet

Whil Hentzen

In this session, you will learn how to use Visual FoxPro to automate the maintenance of web pages and how to allow users to query databases across the web.

For a web site to be successful, its content must regularly be updated. However, this maintenance can be a time and labor intensive (and relatively boring) task. A worthwhile technique is to use have the users enter the data that will eventually make up the web page's content into a Visual FoxPro application Then, the application can be responsible for generating new versions of the HTML pages, keeping the content of the site fresh and lively. You will learn the general makeup of a web page and see the structure of a Visual FoxPro program that updates a live web page from the contents of a database.

The other common request of Visual FoxPro developers is to publish database contents across the web so that it is accessible to a wide range of users. Unlike the first topic, the purpose here is to allow the user to enter criteria into a web page to select a subset of data from a database and then display that result set in a new web page. The advantage is that the data and the application are maintained in one centralized location but the data and the engine providing access to that data are available to anyone - at any location - with the appropriate permissions. You will learn how to create a web page with a variety of controls that can be used for entering criteria, how to send the criteria to a Visual FoxPro application that will perform the data retrieval, and how to format the result set in a new web page for viewing by the user.

Why the Internet?

- Vast storehouse of information available from anywhere.
- Critical mass has been reached, so new sites and content is being added rapidly.
- Tools are coming into being that allow more than read-only access to static data.
- The mechanism is a new way to centralize applications and data while making their functionality available on a wider basis.

The Internet will have a huge impact on the way you will do development in the future. The tools aren't all there yet, and traditional development isn’t finished, but the days are numbered.

Internet Basics

The Internet is a collection of TCP/IP networks. TCP/IP is a simple protocol that allows computers to talk to each other and is in wide use around the world. It comes with Windows95 and Windows NT, and can be downloaded for use with Windows 3.1.

Each of these networks has one or more computers on it, and each computer has an IP address. An IP address has the form of four numbers separated by periods - such as 123.101.099.215. Since these addresses are difficult to remember, an IP address can be mapped to a domain name such as microsoft.com or hentzenwerke.com. The mechanism that maps an IP address to a domain name is called a Domain Name Server (DNS).

One common use of the Internet is to store files that can be viewed through a piece of software called a browser. These files are called web pages, and Netscape and Internet Explorer are the two most popular browsers. These tools interpret data in the files and display the data in a specially formatted way, much like the Browse command in FoxPro displays data contained in a .DBF file as a spreadsheet-like view of rows and columns. The browsers available today have rather limited capabilities in that they can display text and images in a variety of formats and provide some basic capabilities for the user to interact - but nothing like the interactivity that we are used to with recent and current versions of FoxPro.

Web pages can be located on any computer on the Internet, but that computer must have a software application called a web server installed in order to "serve" the web pages to other computers requesting access to them. Common web server packages include Microsoft's Internet Information Server (IIS) and Commerce Builder from the Internet Factory.

When a user wants to access a web page on the Internet, they simply enter the address of the computer into their browser, and the contents of the page is displayed in the browser. When you install a browser, you can specify the DNS to be used when you enter domain names instead of the more cryptic IP addresses. Then, when you enter a domain name, the browser looks up the IP address via the DNS and connects to the computer with that address.

Since one of the appealing attributes of a web site (the computer that hosts a web page) is that web pages can be linked through the use of hypertext links, a computer may have dozens, hundreds or more web pages on it. The web server software usually has a setting that specifies which web page on the computer is the default, or initial web page to load when a user accesses the computer, similar to a startup program in a FoxPro application or an AUTOEXEC batch file on a DOS-based PC.

HTML Formatting

The preceding discussion has assumed the existence of these files called "web pages" that presumably contain text and images. Let's discuss what these web pages look like for a minute.

These pages are often referred to as HTML pages, because the language used to describe the special codes and commands is called HyperText Markup Language (HTML). An example of a simple HTML page would look like this when viewed in a text editor:

```
<HTML>
<HEAD>
<TITLE>This is a title</TITLE>
</HEAD>
<H1><FONT COLOR="BLUE">This is a heading </FONT></H1>
<BODY>This is body text</BODY>
</HTML>
```

You'll see that this looks a lot like, say, a WordPerfect document with Reveal Codes on.

There are a large number of commands that allow you to specify how to specify the placement and appearance of objects, including text, images and interactive controls, on a page. However, keep in mind that the fundamental premise of a HTML page is that it is static - any interaction with the page requires a refresh of the page, unlike with an application like Visual FoxPro, where the interaction and refresh can be done on a very granular basis - down to an individual control.

Image:

```
<IMG SRC="sample.gif" WIDTH="100" HEIGHT="125" ALIGN="MIDDLE" ALT="Programming VFP"
BORDER="0">
```

Anchor (hypertext link)"

```
<A HREF="sample2.html">This is a hypertext link</A>
```

Simple Controls (text box, radio button, command button):

```
<FORM METHOD="POST" ACTION="http://hentzenwerke.com/cgi-win/wwcgi.dll?getart">
<B>Author:</B>
<INPUT TYPE="TEXT" NAME="Author" SIZE="20">
<BR>

<B>Keywords:</B>
<INPUT TYPE="RADIO" NAME="allorany" VALUE="all" CHECKED>
All
<INPUT TYPE="RADIO" NAME="allorany" VALUE="any">
Any

<INPUT TYPE="SUBMIT" NAME="SubmitButton" VALUE="Search for these articles">
<BR>
<INPUT TYPE="RESET" NAME="SubmitButton" VALUE="Search for these articles">
```

Other types of objects that can be placed in a web page include tables and ActiveX controls.

Automated Web Page Formatting

A short time ago, it was pretty common to hear the claim "This web site generated by Visual FoxPro" and if you were in awe of the developer, you were not alone. As the saying goes, "Any technology sufficiently advanced is indistinguishable from magic." Well, it's not nearly as hard as it seems.

For a web site to be successful, its content must regularly be updated. As you have seen, it's not difficult to modify the HTML file that makes up a web page. However, while it's not difficult, it's not interesting either. And that's one of the fundamental reasons for database applications - automating the boring, repetitious work so that the difficult, intellectual tasks remain for us humans. Melding a database application with a web site is a natural.

The first step is to determine what part of the site will remain constant, and which part will be regularly updated. If the site is designed in advance with the intent of automated updating, the task of automatically generating the pages on a regular basis becomes easier.

The second step is to create the table(s) that will serve as the source for the new information. Given a "What's New" page that has a date and a short text description for each entry, the supporting table could have the following structure:

```
WHATSNEW.DBF
nOrder  mDate  mDe
-2      Memo   Memo
-1      Memo   Memo
 1      Memo   Memo
 2      Memo   Memo
```
etc.

The mDate field contains the text string for the date (such as November 15, 1996) and the mDe field contains the description of what was new on that date. The nOrder field is used to put the records in the proper order (by date), to avoid having to parse out the contents of the mDate field.

There's a trick embedded in the table here. Since the data displayed on the page will most likely be formatted with special codes, it makes sense to keep those codes in a central location so that they can be easily maintained. The record with nOrder -1 contains the beginning formatting codes for each field and the record with nOrder of -2 contains the ending formatting codes for each field.

The third step is to create a skeleton HTML file that contains the static information for the page. Generally, this is a mockup of the page with special text strings ("semaphores") that will be replaced with specially formatted data from the database. The string "whatsnewlist" is a semaphore for the data and the string "//date// is a semaphore for the current date.

The Visual FoxPro program is relatively straight forward. The program, stripped down to its essentials, looks like this:

```
* CREAWHAT.PRG
*
* create temporary table for intermediate processing
*
create table JUNK (mMAINDEF m, mHISTDEF m)
insert into  JUNK (mMainDef) values ("")
*
* bring skeleton HTML files into memo fields for manipulation
*
append memo mMainDef from MAINDEF.HTML
append memo mHistDef from HISTDEF.HTML
*
* create memory variables that contain these blocks of text
*
scatter memo memvar
*
* get formatting codes for each block of text
*
select mDate, mDe from WHATSNEW ;
  where nOrder < 0 ;
  order by nOrder descending ;
  into array aCode
*
* we only put the five most recent events on the page
* the rest goes on a linked page called History
*
select mDate, mDe from WHATSNEW ;
  where recno() > recc() - 5 ;
  order by nOrder descending ;
  into array aLastFive
*
* create a text string that consists of the formatting codes
* and the five most recent entries
*
m.lcMain = ""
for I = 1 to 5
  m.lcMain = m.lcMain ;
  + aCode[1,1] + aLastFive[I,1] + aCode[1,2] ;
  + aCode[2,1] + aLastFive[I,2] + aCode[2,2]
endfor
*
* now stuff the newly created text string into the
* HTML page skeleton memory variable
*
m.lcWorkerMain = m.mMainDef
m.lcWorkerMain = strtran(m.lcWorkerMain, "whatsnewlist", m.lcMain)
m.lcWorkerMain = strtran(m.lcWorkerMain, "//date//", dtoc(date()))
*
* (same process works for the history page)
* create a dummy table that we can stuff the memvar data into
* (LLFFs are a nuisance)
*
create table JUNK (mMainOut m, mHistOut m)
insert into JUNK (mMainOut, mHistOut) values (m.lcWorkerMain, m.lcWorkerHist)
*
*create memvars for the HTML file names
*
m.cNaFiMain = "main.html"
m.cNaFiHist = "history.html"
*
* create the HTML files - finally!
*
copy memo mMainOut to &cNaFiMain
copy memo mHistOut to &cNaFiHist
```

(This program shows the essential steps - no error trapping or fault handling has been provided in order to make the code easy to follow and understand.)

The last step is to move the new pages, once generated, onto the server that hosts the pages. Depending on your server set up, this may not be necessary.

Database Publishing

Given the ease with which the contents of a database can be put into HTML format, it only makes sense to go the next step and allow the user to specify the contents that they want to see. The basic scenario is to paint a web page that contains HTML controls that allow the user to enter information that can be interpreted as criteria to be used in a query against a database, send the information that the user has entered to a Visual FoxPro application, execute the VFP application (which generally takes the form of a query that returns a result set), and then format the result set in HTML and present the final page to the user. Let's look at this process in more detail.

A web page can contain certain controls with which the user can interact. These include many of the controls with which we are familiar. These include data entry controls such as text boxes, option buttons, check boxes, combo boxes and list boxes, and action controls such as command buttons.

The page also has a "method" attached to it that is executed when the user initiates an action. In this case, it's the pressing of the command button that sends the request on its way. The ultimate destination of the request is a Visual FoxPro application that will accept and process the data that the user entered, and, hopefully, return some data.

Let's look at what happens to the request "on its way."

All requests from a user start out on an HTML page as either a hypertext link or an action command from an input form. In our case, it's the latter. The input form contains the name of a program that is to be executed. This program could be a Visual FoxPro application, but in practice, this would be an awful choice. The VFP application is very large, takes a long time to load, and each request from a page would cause that application to be reloaded. Imagine if you had to start up VFP each time the user saved data or moved to another record!

A better way would be to load VFP and have it waiting - in memory - for a request. We'll then use a connector application that will send data to VFP from the HTML page. The name of the program in the input form - some people call it a script - is an EXE or a DLL that references the connector application.

In this example, the connector application is wwcgi.dll.

```
<FORM METHOD="POST" ACTION="http://hentzenwerke.com/cgi-win/wwcgi.dll?getart">
```

This "script" executes and then returns an HTML page upon completion. The items after the "?" are parameters that can be passed to the VFP program. This connector application, wwcgi, is usually written in C, and uses a high speed interface and low resource requirements on the part of the server. The connector application can be EXE files that use CGI (the standard Internet protocol that describes how input and output is to be handled) or in-process DLLs that are loaded by the web server.

There are a number of these connector applications: the one I am showing, WWCGI, comes with the shareware package West Wind Web Connection.

Once control is turned over to the Visual FoxPro application by the connector application, VFP goes out and gets information from the web server.

Looking at this situation from the other side, VFP is running and polling the environment for CGI requests that are sent in from the connector application. With Web Connection, the CGIMAIN.PRG program is running and waiting.

```
* CGIMAIN.PRG
* sample code only!
set proc to CGIServ
*
* starts up server and creates an object that contains methods to handle CGI requests
*
oCGIServer = CREATE("wwCGIServer", "Process")
*
* sets up polling mode and displays a status window
* this is simply a timer that does a look-around every once in a while
oCGIServer.show()
```

When WWCGI.DLL is called (by pressing the Submit button on the form), it formats the request so that the wwCGIServer object can see it and process it. The "Process" UDF within the VFP application is called by a method of the oCGIServer object when a request is received.

```
*************************************************************************
FUNC PROCESS.PRG
* object that contains CGI information (the request)
lparameters loCGI
*
* do my own specific query program (GET ARTicles)
* and pass the object (reference to the as a parm
*
do GETART with loCGI

return
```

The GETART program is the only custom piece required - and it's the workhorse of the system. It does the actual fetching of the data from the tables and then uses methods of the wwCGI object to prepare and format the results for presentation in a browser via HTML. Again, like earlier samples in this document, only the basic code necessary for understanding the process has been shown.

```
* GETART.PRG
* routine to actually get data from a set of tables

*
* get memvars from HTML page that user entered data into
*
m.lcAuthor = loCGI.GetFormVar("Author")
m.lcKeyw   = loCGI.GetFormVar("Keyword")
* etc.
decl laPubs[1]
m.lnCount=loCGI.GetFormMultiple(@laPubs)

*
* use these vars to find out how many records will be retrieved
*
select count(*) from TUFR_BI ;
  where m.lcAuthor $ cAuthor ;
  into array aHowMany

* if number of hits is too big to display, inform user and return
if aHowMany[1] > 50
  loHTML.SendLn("Too many articles. Narrow search and try again.")
else
  select cTitle as "Title", cAuthor as "Author", cPub as "Publication", ;
      dDate as "Pub_Date", mDesc as "Description" ;
    from TUFR_BI ;
  where m.lcAuthor $ cAuthor ;
  into curs csrArts
  *
  * display the results
  *
  loHTML.ShowCursor()
endif

*
* tell user what parms they had initially entered
*
loHTML.SendPar()
loTHML.SendLn("Author requested: " + m.lcAuthor+"<BR>")
loTHML.SendLn("Keywords: " + m.lcKeyw+"<BR>")
* etc.

return .t.
```

Now that VFP has the parameters against which to run the query, it does so, and creates a result set to be returned. The server gets the page and sends it back to the browser for the user.

This example uses Rick Strahl's Web Connection as an example of a tool that provides (1) a mechanism to call a VFP application from a web page (wwcgi.dll), (2) an interface for getting data from a web page from within the VFP app, and (3) a mechanism to format and return a result set to the web server. Why write your own when you can take advantage of the work that your predecessors have already done?

The technology shown in this presentation is relatively robust and reliable for use today. New technologies are being introduced that extend the ability of the Internet into the realm of read/write applications and the pioneers today will be mainstream tomorrow. Today is an appropriate time to begin to get involved in using the Internet as another platform for delivering the information systems that you have already been building.