## Figures:
## File for Subscriber Downloads:

Your First Database Web Application – Part III

Whil Hentzen

Last month I discussed setting up a Web server with the intent to use Web Connection (or another application framework) for creating a database application with VFP, and then how to set up your development machine. This month, I'll discuss setting up your project, introduce a strategy for constructing your Web programs quickly and reliably, and show you a couple of tips and tricks that will speed your development along.

The Web Development Process

Just like regular application development in the LAN world, there are thousand different types of user interfaces and a thousand different approaches. However, just like in the LAN world, applications fundamentally boil down to three parts: Input, Calculations, and Output.

In many respects, web development is more straightforward because the input portion of the application is much more restricted – thus, your options aren't as numerous. Basically, you display static text, perhaps ask the user for some input, and then branch off into one or more directions according to the user's wishes. The new destination is again a simple display of static text and perhaps some more user input. Each page that you display is either a static HTML page that resides on your server, or is generated by a VFP program. There's a one-to-one correspondence between each page you generate and a supporting VFP program. (Actually, in Web Connection, each page is generated by a function inside a PRG, but that's picking nits at this point.)

In other words, user interfaces, and the logic that supports them, is much more linear than event driven. Here's how you might go about producing an application for the web.

Lay out pages (simple ones - like dBase III)

Just like when you prototype using VFP for the interface, you'll need to design the interface that the user sees. I often use Front Page because I get the interface set up much like the user is going to see it – and then cut out the HTML that Front Page generates and use that for the static portion of my page. Remember, you're going to be writing a program or function for each page that the user can navigate to or encounter via a request.

Develop logic inside forms

Once you've laid out each page as the user is going to see them, you'll need to figure out what's going on behind the scenes. In other words – what each program is going to do. (I'm ignoring static pages here cuz, by definition, there's nothing to do programmatically.) There are basically four parts to this.

The first is to do any necessary housekeeping before the page loads – this may include gathering information from the previous page's input controls, or saving that data to a table.

The second is to do any calculations necessary in order to present this page. Calculations include

(1) gathering information from other pages. An example would be if you were going to make a behind-the-scenes request to another site, like authorizing a credit card charge

(2) gathering data from a database. An example would be if the user has asked for information based on a parameter they've entered – here is where you would do the database query

(3) formatting information or doing other business logic. For example, based on a parameter they've input (or one you've gathered behind the scenes), you will display one set of information or another.

Third, you'll actually draw the page. To make maintenance easier, you'll do all the logic to produce the page in one place, and then product the page later on – instead of doing a bit of logic, a bit of presentation, a bit more logic, a bit more presentation, and so on…

Finally, you'll need to provide for the actions that the user can perform on the form in terms of executing actions and calling other programs. This is the equivalent of drawing dummy forms in a VFP prototype and simply placing command buttons on each form that call subsequent elements of the user interface.

Build a program for each form

Once you know what you've got to do, now comes the fun part – writing the code that makes it happen. Remember that each function is basically procedural code. You'll want to build a debugging scaffold (see the May, 2000 issue of FoxTalk) into your code so that you can find the problems easier. Since you'll be running from an EXE, you won't be trapping errors within the interactive environment like you might be used to. As a result, I've found it handy to build in trapping statements that verify the data types of parameters, input fields, and variables as well as display intermediate values.

Suppose you're building a string of values, and your program is hanging up in the middle of a series of concatenations. Web Connection, for example, will display the message "Data Type Mismatch" in the browser window when you call the function with the error, but typically you want more than that. I put switches in the program to show the intermediate results as part of the output to the browser, and when, er, I mean IF a function blows up, I can set the switch to display the results up to that point.

For example, you may want to check that an email address is valid before further processing.

```
m.llDebug1 = .f.
m.lcEmail = lower(m.lcEmail)
* check to see if email is OK
if ("com" $ m.lcEmail ;
     or "org" $ m.lcEmail ;
     or "net" $ m.lcEmail) ;
   and ;
   ("." $ m.lcEmail and "@" $ m.lcEmail)
  * valid email - continue
  m.lcPage = "Welcome, " + m.lcEmail
else
  * invalid email
  m.lcPage = "An invalid email address was entered"
endif

if m.llDebug1
  m.lcPage ;
    = "***** DEBUG OUTPUT *****" + <br> ;
    + "m.lcEmail =" ;
    + tran(m.lcEmail) + <br> ;
    = "***** DEBUG OUTPUT END *****" + <br> ;
    + m.lcPage
endif
```

Build the EXE

Depending on how your application will be deployed, you may need to compile your application into executable form. Visual FoxPro can build an EXE and register it as a COM server in one step.

Run the EXE

Most people – including myself – forget this step – they're so anxious to get to the browser window and see their creation in action that they just call up the browser and enter in the URL with the call to the function. Unfortunately, if you don't run your application, the call from the browser will just hang and you'll be wondering why you're not getting any results.

Call main page of project

You can start testing your application in one of two ways. Either you can call your program directly by typing a URL that includes a call to WC.DLL in your browser, or you can embed that URL in a static HTML page (much like your website could do), and launch that page via your browser.

Create shortcuts

You'll see that you will run your EXE and call the main page of your application over and over. You'll find it useful to have a pair of shortcuts in your task bar – one for launching the EXE, and another to launch your main page.

I've gone one step further, and created a static HTML page that contains links to each of the parts of the application. For example, during development I keep a copy of a dummy function like "HelloThereWorld" that just displays the current date and time. Calling this function tells me if my server is up and running properly. So the static HTML page has a link to HelloThereWorld. Another link is to the main page of the application, and a third is to the Admin page of Web Connection.

This HTML page actually has two columns – one for calling these functions on the development machine's server, and a second column that calls the identical functions on the live server.

Sure, you can type the URLs into the address of the browser, or you could use the history feature, but I find it a lot less frustrating to have the calls coded there right for me to click on, instead of trying to navigate through a history list that might be dozens or even hundreds of addresses long.