WebRAD

V35
Whil Hentzen

So how to people learn to use apps? They hack away at it, get it working, and then go on to the next one, most likely not having a chance to clean up the mess they made on their first two or three attempts. If you're looking at building a web app, or perhaps you've built one or two and are dissatisfied with the process so far, this is the session for you.

Purpose

The purpose of this session is to give you a framework and a process for building web-based database applications quickly. The example we'll use is an online store – complete with customer lookup, edit and entry, item selection from a predefined SKU table, shipping calculation, credit calculation, and finally confirmation of a successful order. While we build this application, you'll see how to set up your environment, how to structure pieces, what the process looks like, and how to rearchitect your applications once you've developed some experience.

Definition of RAD

First, let's talk about RAD. You hear the term 'quickly and easily' used in just about every demo that every player in the IT arena gives these days. Whether it's building COM+ applications, installing routers into a server farm, or upgrading your virus signature files throughout a company, everyone claims "quickly and easily." Let's face it – this is not quick, nor is it easy. If it were, we'd all be out of work. The average computer user still isn't quite sure just exactly why Word won't run on Linux, or, for that matter, what Linux and Windows even are.

So let's talk about RAD. In a formal sense, "RAD" was coined by James Martin in the 1970's as a result of his research on software engineering. It refers to a specific approach regarding application development. In the three decades since, though, the term has been bastardized in every conceivable way to the point that it's unrecognizable and virtually useless, except as a generic moniker for "cool and fast" by marketing folks.

What we're after, though, what you're after – what drew you to this session – is the title "WebRAD" – the implicit promise that you're going to learn how to build web database applications faster.

So how do you do it faster?

1. You use tools that you already know how to use – as long as they are appropriate for the job
2. You use tools that are mature and reliable
3. You use tools that are well documented and are easy to use
4. You use tools that do the work for you
5. You use tools that help you avoid mistakes
6. You use tools that help you find the mistakes you make
7. You use tools that have a large support group

Visual FoxPro and Web Connection running on NT 4 or Windows 2000 fills each of these requirements.

1. You know how to use

Presumably you're here because you're a VFP developer. WC is built completely in VFP – with the exception of a couple of DLLs that you install and never look at again.

2. Mature and reliable

VFP has been around for years – and the majority of the work you'll be doing is using parts that have been around forever. ASP, by contrast, "you run into problems when you start moving to proper n-tier design that utilize components." (MSDN Magazine, September, 2000) Part of the reason is the immaturity of the platform and lack of proper architecture. DLL Hell is the most well-known manifestation of this problem.

3. Well documented and easy to use

Fox is well documented – well, not to be self-serving, but there is a lot of documentation on VFP these days – thousands of pages on detailed topics written by experienced developers who use VFP.
While easy to use may be a subjective term, VFP is relatively easy to use as it's been around a long time and there is a large knowledge base of building "xbase" applications in the 'xbase' language and object.method and object.property syntaxes. Furthermore, the level of componentization required and available for VFP/WC applications is relatively mature.

4. Do the work for you

VFP has 2500 commands, functions, properties events and methods. Just about anything you want to do is there in front of you, or, with FoxTools or the WinAPI, just a step away.

Web Connection is an incredibly rich framework for building web applications. Want to create a stateful application – one where data persists from one browser screen to the next? It's a pair of two simple function calls in WC. Want to update the live EXE running on your server from another physical location? It's built into WC. Want to call FedEx or UPS to get shipping rates based on data you've fetched from customer input? The tools you need are part of WC.

Together, the two allow you to worry about the business rules and the look and feel of your application – VFP and WC handle the plumbing and low level functions. And, of course, it's fast and highly scalable – Unless you're building the knowledge base for the IRS during tax season or a new online store for Wal-Mart, Fox and WC will be able to handle the load.

5. Avoid mistakes

6. Find mistakes you make

Given that you're using VFP, the maturity of the language and tools means you have a rich development environment that has many features to help you avoid making mistakes and find mistakes you make. Furthermore, given that you can easily separate the UI, the business logic, and the data handling – and yet identify problems quickly. Contrast this with the way that the failure of creating an object in an ASP page responds: "Unable to create object", leaving you to wonder which of many steps was responsible. Furthermore, given that the ASP architecture propagated spaghetti code (MSDN, Sept 2000), current web applications using multiple components tend to be hard to develop and even harder to maintain and modify.

7. Use tools that have a large support group

While the Fox group may not be the largest on the planet, it certainly is as vocal and ardent as any group – and there are a multitude of resources…

RAD vs. State of the Art

Before I go any further, I want to make clear that "Faster and more reliable" is in direct conflict with "State of the art". By definition, state of the art technology is on the leading edge – it hasn't had time to mature (that means it's buggy), it is often not documented well, and if pieces aren't missing or incompletely implemented, the interfaces with other components often are. The advantage to state of the art is new capabilities that aren't present in current technology. You have to weigh the capabilities of new state of the art vs. reliable and mature RAD tools. Given this, let's proceed with RAD.

How a Web Application Runs

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. This protocol comes with Windows9x, Windows NT, and Windows 2000, 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). When you set up a computer to operate on a network, you'll set up TCP/IP properties to identify the IP address and the DNS servers that your computers will talk to.

Specifying the DNS to be used when you enter domain names instead of the more cryptic IP addresses is done in the DNS tab of the TCP/IP Properties tab. 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.

How a Web Server works

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. This process is called rendering.

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), Apache Server, and Commerce Builder from the Internet Factory.

When you run an application like Word or Visual FoxPro and open a file like a document or a program, the application makes a request of another piece of software to "get" that file. That "other piece of software" is the operating system. When the application receives the file from the operating system, it then displays that file – which could be natively stored in some inscrutable binary format - in a form that is pleasing to humans like you and me. A Word document shows up with bold and italics and certain fonts and so on; a program file shows up with line breaks, color-coded keywords, and so on. The application interprets "secret codes" in the file, and then performs other functions to display – or render - it.

A browser and a Web Server do the exact same thing. Typing www.hentzenwerke.com/herman.html in your browser is akin to selecting the File, Open command in another application. The browser sends that request – the request for the "herman.html" file at the www.hentzenwerke.com web site – to another piece of software to "get" that file – but this time, that "other piece of software" is the web server – and it's not running on the same computer – but on another one on the TCP/IP network.

When you have an application properly set up on an OS like Windows, and then open a file through File, Open, the application and the operating system know how to talk to each other. When you request a file of a web server through a browser, and they're both set up through a protocol like TCP/IP, the browser and web server know how to talk to each other as well. It's just a bit more involved than a simple File, Open command.

If you're requesting a file that's located on a web server "somewhere" in the world, you enter the name of that file server – like www.accuweather.com - in your browser. Remember the "default.html" specification

in the Documents tab of the Default Web Site Properties dialog? If you don't specify a filename after www.accuweather.com, the web server at accuweather will dish up that default page. If there was a file named "herman.html" in the Local Path of www.hentzenwerke.com, you could open that file by entering www.hentzenwerke.com/herman.html, even if herman.html wasn't the default page.

As I mentioned, every web server in the world has a unique IP address - the four number address: 121.45.100.199. When you set up your Internet connection, you possibly specified the location of a "Domain Name Server" (another address like 121.45.100.3.) This is a machine that contains a mapping between 121.45.100.199 and a text string like www.homeontherange.com so you can enter an easy to remember URL instead of trying to remember all those numbers.

If, on the other hand, you're requesting a file that's on your own test server, you can simply specify the address of the server, like so:

http://192.168.1.10/default.html

Notice that you need to prefix the dot address with http:// in order to tell the browser what communication protocol to use, and that you will name the file you want to grab as well. Your test server doesn't have an entry in the Domain Name Server scheme of things, so you don't have access to your server that way. You'll notice, of course, that this dot address is the IP Address of your web server that you specified when connecting it to your network as discussed earlier in this article.

Of course, you'll actually have to have a file called DEFAULT.HTML in your web server's home directory, but that's the topic of another session. There are lots places on the Web that you can find out more about writing HTML – or you could just fool around with Front Page for a while and save the results as DEFAULT.HTML and stick them into your home directory.

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.

Web Server Hardware and Software

You can use most any lowly Pentium-class machine as your web server – particularly when you're just starting out. A P90 with 64 MB and a couple gigabytes of hard disk space will do just fine for the time being, but you'll probably want to add a bit of memory –128 MB will be plenty. Format the hard disk, and create two partitions. One will be for your operating system and application software; the other is for your data, so that you don't have to reinstall your data each time you have to reinstall NT.

If you're using NT, apply the latest Service Pack and the NT Option Pack. Doing so will give you the choice to install Internet Information Server, a piece of software that runs on your web server machine as an NT service.

You'll need to configure the TCP/IP protocol properties when you include the machine on your network. Give your web server an IP address like 192.168.1.10. I'll come back to that later.

You change options in IIS through the Microsoft Management Console, a new UI to NT that will eventually be used to get to all of your NT services, like Transaction Server and Index Server. The only challenge is finding it. Select Start, Programs, NT 4.0 Option Pack, Microsoft Internet Information Server, Internet Service Manager, and you'll see the Microsoft Management Console, Version 1.0 appear. Yes, version 1.0 – always a comfort.

MMC uses an Explorer style interface to display, in the left window, a list of services, like IIS and MTS, and in the right, details on the highlighted service. Under IIS, you'll see the name of your web server

machine, and under that, a list of services, such as Default FTP site, Default Web Site, Admin Web Site, and Default SMTP Site. In the right window, you'll see those names and their status – either Running or Stopped. You can use a menu option or a toolbar button to start a service that is stopped – make sure your Default Web Site service is started.

Next, select the Default Web Site node, right-click, and select Properties from the context menu. The Default Web Site Properties dialog that displays has two tabs of particular interest at this time. The Home Directory allows you to specify where IIS - your web server software – is going to look for the files that make up your web site (the files that are going to be viewed by the user when they access your site), and the Documents directory allows you to specify which file in the home directory is automatically loaded.

Create a directory named something like "my1stweb" in the root of your D directory, and specify that directory under the "When connecting to this computer, content should come from Local Path" text box in the Home Directory tab. When you select the Documents directory, you'll see a check box and a list box that allow you to specify that "default.html" is the default document. Just leave this as is for the time being – I'll come back to it in a minute.

Directions for Windows 2000 aren't included for the time being – not until the first service pack is released.

Your web server is now ready.

Connector

Web Connection is made of two separate components that comprise the full Web Connection framework: One is an ISAPI connector that communicates with the Web server. (The other is a Visual FoxPro based framework that knows how to communicate with this ISAPI extension.)

The general idea of the ISAPI connector is to take the request information that a Web request provides - HTML Form variables (HTTP POST data), Server Information (server port, Software etc.), Client information (Client IP address, Browser etc.) and Request state information (such as cookies and authentication info) - and pass it forward to a Visual FoxPro server application that waits for incoming requests.

The server then uses the incoming request data as input to the request processing, which uses standard Visual FoxPro code to generate HTTP output. In most cases this output will be HTML, but it can also be XML, or binary content such as a data file, a Word or PDF document for example. This result data is then returned as a string from your VFP server code.

Your Application

Your application is Visual FoxPro code compiled into an application, such as an EXE running in conjunction with the VFP runtime. This application runs on the web server, and is called via the ISAPI connector by a URL, like so:

http://www.somedomainname.com/wc.dll!YourApp~SomeMethodInTheApp

Generally (and very roughly) speaking, your application will consist of a main application program that will be used to launch other functions found in the main application program. Each function will map to a specific page that the user can encounter. You can, of course, architect this many different ways, but for the time being, consider this as a good starting point for your application.

Data

Your website will also be on the web server, and includes both static HTML pages, images and files, and data that your VFP application accesses. Each of these types of data should be in their own directory both for easy of maintenance as well as security.

During a Hit

Your VFP application will be running on your web server continuously, waiting to hear from a user. When a user makes a request of the web server, like clicking on a link like this:

http://www.somedomainname.com/wc.dll!YourApp~SomeMethodInTheApp

your application goes into action. The ISAPI connector detects the hit and routes the request to your application, much like a VFP READ EVENTS command lies in wait for an action from a user. The SomeMethodInTheApp method fires, and your VFP program runs. Most often, your VFP program will do some sort of database manipulation, perhaps including data that the user had typed into their browser, and then return results that are formatted such that they look proper in a browser.

Setting up a Development Server

Just as you generally wouldn't want to do your development work on your production application and server, you don't want to work on your web application on your live web server. You can use your own development box for web development and then deploy to the live web server once you're ready for production.

Web Server

IIS requires NT Server or Windows 2000 Server, and you're likely not going to be running Server on your development machine. Fortunately, Microsoft provides "personal" versions of IIS for both Windows 9x, NT and 2000. (For NT, get PWS from the NT Option Pack.) The names vary slightly according to which version you're using, from Personal Web Server to Peer Web Services, but you can refer to all of them as "PWS" to make it clear you're talking about web server software that runs on your own development computer, and not some live server in the back room.

Just like IIS on a live server, you configure PWS to point to a home directory. It will attempt to create and point to a directory named InetPubs\wwwroot on your C drive. This is stupid – this is where your data is going to be, for gosh sake. Create InetPubs\wwwroot on drive D (or wherever the development copy of your website will be), but you should never point to drive C - in preparation for the inevitable reformatting of drive C one day.

If you plan on having more than one application running from your website, it would be a good idea to create separate application directories underneath wwwroot. For example,

wwwroot\MyAppOne
wwwroot\MyAppTwo

This way you can keep each application's files separate, and reduce the risk of one application stomping on another application's files. You can keep each application running separately – if you have to reconfigure or bring down one app, you're less likely to bring down another app if it's located in a different directory.

Once PWS is running your development machine, you'll see a "Personal Web Server is running" icon in your system tray. You can access and configure PWS through this icon.

Web Connection

Setting up Web Connection is generally very easy – run the installation program and follow the prompts. There are a couple places where you might find additional information useful.

First of all, this should be obvious, but in case it isn't, you'll need the Visual FoxPro development environment on the machine that you're installing Web Connection on. You are building VFP apps with WC, and, more directly, WC will run some VFP programs upon conclusion of the its installation.

Second, it'll ask you where you want to install WebConnect, but the wording isn't clear. Web Connection is an application framework, much like FoxExpress, MaxFrame or Mere Mortals – and thus comprises both a set of framework classes from which you'll build your apps, and some wizards and programs that help you build your apps.

In this step, you're selecting where you want to install the Web Connection framework and wizards. It will defulat to drive C, but I would argue that it doesn't belong on drive C anymore than any other data or development programs belong on C. I suggest that you install Web Connection on your development drive, such as drive F. The default directory is \wconnect.

Note that when you create a new project in Web Connection, the new project wizard will put the project in this Web Connection installation directory. I'll show you how to change that to a better place shortly.

Third, you'll be asked to select the location of InetPub\wwwroot – select the directory you made on drive D or wherever – not the default on drive C.

Setting up your development structure and environment

You've got your website located in D:\InetPub\wwwroot, and your development drive, including the Web Connection framework classes and wizards in F:\wconnect. Now you'll need to set up a directory for web application development, and some directories for databases and the like.

We create top level directories on our development drive for each customer. Under that top level directory we create subdirectories for each application. You could follow this structure, or you could create a separate top level directory for all of your web applications. For sake of convenience during this discussion, I'll go with the second choice. Under the top level web applications directory, create separate subdirectories for each separate web application, like so:

F:\WebApps\MyWebAppOne
F:\WebApps\MyWebAppTwo
F:\WebApps\MyWebAppThree

Under each directory, create the normal directories that you use in your day to day development life. For example, we put the project and the final executable in the root of the application directory, and the source code underneath the application directory in a subdirectory called SOURCE. So the contents and structure for AppOne would look like this:

F:\WebApps\MyWebAppOne\AppOne.PJX
F:\WebApps\MyWebAppOne\AppOne.PJT
F:\WebApps\MyWebAppOne\AppOne.EXE
F:\WebApps\MyWebAppOne\Source\<lots of source code files>

Mapping to the Live Server

Both the development and live servers have InetPub\wwwroot directory structures, and they both will have data directories.

One tricky part of setting up your environment is getting the paths to your files, particularly your data, set up correctly. One way you might want to try while starting out is to create top level directories on both your development drive and your live web server for your data. This way, you can control access to the files through discrete security settings to the directories themselves. Once you get more experienced, you can become more and more sophisticated about controlling security.

Assuming you're going to have more than one web application, your data directory might look like this:

F:\WebData\MyWebAppOne
F:\WebData\MyWebAppTwo
F:\WebData\MyWebAppThree

And your data files for each app lie in their own directory. Naturally, if these applications are going to share data files, you'll need to change your approach, but that's best addressed after you become more experienced.

The application directories, on the other hand, don't map to anything on the live server, because they're used for development. The end result – an executable file like an EXE – is placed in the website directory on the live server.

Where might you put those EXEs on your development machine while you're testing them out? You've got two choices. The first is to copy them to the appropriate location for your website – InetPub\wwwroot\MyAppOne – because that's where any support files, such as images or other static pages will be. If, on the other hand, you're testing pure data, you don't need to go that far – you can just run the EXE from where it was built – as long as it can still find its data. If you're referencing your data from the root, like so: \WebData\MyAppOne, you shouldn't have any problems.

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.

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 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. 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're 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 may display the page one way 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 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.

Build 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 EXE

Most people – including myself – forget this step – they're so anxious to get to the browser window and see their creation in action. 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.

A Sample Application

A robust but easy to understand web application is that of an online store.

Online store description

In this example, we'll suppose you have a table with a number of available items, each identified with a unique SKU. In this table, we'll have the normal attributes you'd expect of an item table, such as a description, a price, a shipping weight, and a status code indicating whether the item is in stock, or on back order, and if on back order, how long the back order is expected to take.

We'll also assume that you have an existing customer database, and that existing customers have a login name and password that they can use to retrieve their name and address information.

The user will navigate to the store's main page where they will be greeted with instructions on how to place an order. Having these instructions all spelled out at the beginning will help them understand what they are to do and what will happen each step of the way.

Main page

Besides presenting the instructions on how to place an order, this main page will also have a pair of text boxes where they can enter their login name and password so that the system can pull up an existing record instead of making the customer enter the information for each new order.

The user will enter their login name and password if they have one, and then click on the Continue button to go to the Customer Information page.

Step 1: Customer information

If the user has already bought something from the store, we've got their name and address. If they've entered their login name and password, this page will display the information on file and allow them to change it. If the login name and password they entered wasn't found, this page will inform them so. If they didn't enter a login name and password, this page will display an empty form so they can enter their information.

When finished, the user will click the Continue button to go to the Items page.

Items

In this simplified model of an online store, all the items that a customer can purchase are listed in a single scrolling page. The availability of each time – whether it's In Stock or Back-Ordered – will also be displayed. Obviously, this isn't Sears.

The customer will enter quantities for each; when done, they'll click the Continue button to go to the Shipping page.

Shipping

One of the biggest complaints from customers while shopping online is not knowing the shipping charges until the order has been processed. This is a shame, because it's relatively easy to calculate shipping once the items and shipping location have been identified.

This page will display the various options available for shipping, such as Surface or Next Day Delivery, and the costs associated with each option.

The customer will select a shipping method and click the Continue button to go to the Order Verification page.

Order verification

This page will display all of the relevant information about the order, including

Customer name
Shipping address
List of items and quantity purchased
Gross price
Reminder about availability
Shipping method chosen
Shipping costs
Discounts and other price-altering factors
Final price

The customer isn't required to perform any particular action here; they'll simply click the Continue button to go to the Payment Processing page.

Payment processing

This page will prompt the user to enter their credit card number and other relevant information. When they are done entering their information, they will click the Continue button to have their order processed and paid for, and to then go to the Order Confirmation page.

Order confirmation

The results of the credit card processing as well as order processing will be displayed on this page. Hopefully they'll get a confirmation number, an order number, and other information (for example, if they're buying something with a software or electronic component, access to download that material.)

Architecture of a Page

Just like a well-constructed form in Visual FoxPro should have a specific architecture – with, for example, a data environment, Load, Init, Show and Activate methods, data-bound controls, and so on – a Web page also should have a consistent architecture. Let's look at one possible structure that you could use while developing Web pages.

Request vars from previous page

The first thing you're going to do is capture data from the previous page. For example, if you've got text boxes named cNaF (first name), cNaL (last name), and cNaO (company/organization name) in the Customer page, the code in the top of the Items page, using Web Connection functions, would look like this:

```
lcNaF=Request.Form("cNaF")
lcNaL=Request.Form("cNaL")
lcNaO=Request.Form("cNaO")
```

Update session vars

Presumably, you'll want to save this information throughout the user's session – in other words, across multiple page hits. The code to save the name data, using Web Connection functions, including the Session object, would look like this:

```
THIS.oSession.SetSessionVar('cNaF',m.lcNaF)
THIS.oSession.SetSessionVar('cNaL',m.lcNaL)
THIS.oSession.SetSessionVar('cNaO',m.lcNaO)
```

Database hit

Now that you're done with the previous page, it's time to attend to the functions of this page. The first thing to do is go out and grab data from databases. For example, in the Customer page, this is where you would use the Login Name and Password as parameters to find an existing customer's record:

```
select cNaF, cNaL, cNaO from CUSTOMER ;
  where CUSTOMER.cNaLogin == m.lcNaLogin ;
  and CUSTOMER.cPassword == m.lcPassword ;
  into array aCust
```

From here, of course, you might save this information to properties of the Session object:

```
THIS.oSession.SetSessionVar('cNaF',aCust[1])
THIS.oSession.SetSessionVar('cNaL', aCust[2])
THIS.oSession.SetSessionVar('cNaO', aCust[3])
```

You may not only be hitting your own database. Here's where you want to get data from other places – such as other web sites. The code, using functions in Web Connection, to get rates from the UPS web site, based on information you've already gotten, such as the customer's Postal Code and the weight of the package,

```
*  Make initial connection to UPS
oHTTP.HTTPConnect('www.ups.com')
*  Build the name-value pairs
oHTTP.AddPostKey('ActionCode','4')
oHTTP.AddPostKey('RateChart', 'Regular Daily Pickup')
oHTTP.AddPostKey('ShipperPostalCode', m.lcZipShipper)
oHTTP.AddPostKey('PackageActualWeight', m.lcPackageActualWeight)
oHTTP.AddPostKey('ServiceLevelCode', m.lcServiceLevelCode)
oHTTP.AddPostKey('ConsigneePostalCode', m.lcZipCust)
oHTTP.AddPostKey('ConsigneeCountry', m.lcCountryCust)
oHTTP.AddPostKey('ResidentialInd', m.lcOneIfRes)
* Make the call and get the result
m.liResult=oHTTP.HTTPGetEx('/using/services/rave/qcost_dss.cgi', ;
                    @m.lcHTML, @m.liText)
```

Calculations

At this point, you've got all the data you're going to need to build your page – but it may not be in the form that you want to present to the user. Here's where you perform the calculations that perform the business logic and assemble the strings for display to the user.

Configure input fields

In many cases, you're going to want to get input from the user. This may be as simple as presenting a free-form text box, like so:

```
m.lcPage = m.lcPage + '<font size="2">Name, First * '
m.lcPage = m.lcPage + '<font size="2"><input name="cNaF" size="10" tabindex="1">
```

You could display controls conditionally. For example the following code will display an option button for selecting shipping via UPS Ground, together with the cost for doing so – if this method is indeed appropriate for the order (it wouldn't be if you were shipping overseas, for example.)

```
if m.llShipCostGroundUPS
 * no button
else
  m.lcPage = m.lcPage ;
    + '<input type="radio" value="GND" name="cShipMethodCode">UPS Ground' ;
    + " Cost: $" + allt(str( m.lnShipCostGroundCust ,10,2)) ;
    + '<br>'
endif
```

Define the Form action

There are two parts ot a form action – the definition of the action, like so:

```
m.lcPage = ' <form action="/ols/wc.dll?olsprocess~POP4" method="post"> '
```

And the control that initiates the action, such as a button defintion, like so:

```
m.lcPage = m.lcPage + ' <p><input type="submit" value="Continue to Step 4: Verify Order and Enter
Payment Information" name="B1" tabindex="99"></p> '
```

```
m.lcPage = m.lcPage + ' </form> '
```

Typically, I put the definition of the action at the beginning of the program for the page, and follow it immediately with the title fo rthe apge:

```
m.lcPage = ' <form action="/ols/wc.dll?olsprocess~POP4" method="post"> '
m.lcPage = m.lcPage + '<h3>3. Choose Shipping Method and Calculate Shipping Costs.</h3>  '
```

Present the page

Finally, it's time to present the page to the user. Web Connection has a standard function, StandardPage(), that you can call direct to display a text string, like so:

```
THIS.StandardPage("Stop Pre Publications On-Line Ordering",;
        m.lcPage)
```

Naturally, you could subclass StandardPage() to customize it and make it fit with the rest of your site.

Rearchitecting

Once you've been through a couple of applications, you start to get ideas about what you're doing over and over again – the long way – and what is working out OK. Two fundamental precepts behind RAD are modularization and reuse. These two actually work arm in arm. As you modularize your code, you break long monolithic chunks of code into smaller modules that can be replaced by reusable components. Let's look at the three primary areas you should address.

Wrapper calls for database hits

The story line goes, "What if you decided to replace your Fox database with SQL Server? Do you want to replace, manually, every call to your data with new calls to a new back end?" The idea is to wrap the calls to your data with generic functions that work regardless of what type of data you're working with. For example, let's look at that simple call to the CUSTOMER table shown earlier:

```
select cNaF, cNaL, cNaO from CUSTOMER ;
  where CUSTOMER.cNaLogin == m.lcNaLogin ;
  and CUSTOMER.cPassword == m.lcPassword ;
  into array aCust
```

There are a couple of advantages to embedding this SQL SELECT directly into your program code: it's fast, and it's easy to do. But if you have to go through 10,000 lines of code and change this code to refer to SQL Server, well, that'll be a huge pain. Instead, you could create a function that goes out and gets the data and returns the appropriate values. The function, located in a separate data handling class, could be replaced according to the back end database desired.

Framework functions for UI

The same philosophy can be used for the user interface – fortunately, Web Connection has already provided a number of wrappers for common user interface needs. StandardPage() is one, for example.

Encapsulate business logic in functions, classes or components

Finally, consider creating a function library or class for your business logic, and making calls to that library or object as needed throughout your application. As you determine the types of actions you perform over and over, you'll be able to call upon an ever-growing toolbox of robust, well-tested functionality, and thus be able to plug additional capabilities into your Web applications with a minimum of effort.

Conclusion

RAD means being able to crank out applications quickly and efficiently, in a reliable and repeatable fashion. You need tools that you have experience using, that are mature and reliable, and that are well architected and documented.

The combination of Visual FoxPro and a web application framework, such as Web Connection, satisfies these requirements today.