Course XXX-09

Creating Applications with IIS and VFP

By Whil Hentzen

Access to Data

Access to data on the Web works differently than in the traditional LAN-based architecture that you're used to. Let's examine why, and then discuss the ramifications.

HTTP: An analogy

Unlike the LAN, where you are either extricably linked to thedata (in a pessimstic locking scenario) or a mere keystroke away (in an optimistic locking scenario), the browser has no direct connection to the database.

This relationship is similar to a word processor that retrieves a file from a network. You Open the .DOC file, and now it's in memory on your own PC, regardless of where the original.DOC file was. There isn't a "live" or "continuous" connection between the words you're editing and the file on disk. You need to execute a File Save in order make the connection again.

And during the File|Save, what happens is that you are actually creating a brand new connection. You can prove this because if you took out your handy wire cutters, you could be working on the .DOC in your PC's memory, and have a pal snip the wire between your PC and the rest of the network. You might not notice until you went to save – you'd then find out that your connection to the network was gone.

Same thing with the Web – your connection is temporary – lasting only long enough to grab the goodies and get out. Let's be more specific:

1. The browser (the "client") sends a request to the web server. This is analogous to executing File Open in our word processor.
2. The web server (a piece of software such as IIS running on a separate box) recieves the request, and goes to fetch the data as described in the request.
3. This data may be on the same box, in a simple format like a set of DBFs or an MDB, or it may be on yet another separate box – the database server. In either case, the data is returned to the web server.
4. The web server returns the data to the client (browser), and the connection is broken.
5. The browser gets a file/chunk of data from the server, and displays it for the user, just as Word can display the contents of a .DOC file in a pleasing format for the user.

Obviously, you're going to through a paradigm shift when it comes to providing access to data on the Web. What are the ramifications?

1. You'll be working with set-based data sets instead of the whole enchilada.

2. The interface capabilities are significantly more limited than what you're using to in VFP.

3. HTTP protocol is disconnected so you don't have the same performance capabilities.

Set based data sets

The Web is very much a client-server mechanism, and the connection can be very tenuous at times. Even more so than a traditional client-server relationship, you want to minimize traffic over the wire. Get used to working with one or a few records, as opposed to being able to SKIP or BROWSE through a couple hundred thousand records (and their myriad relatives.)

Limited interface capabilities

During the early 1990's, we bemoaned the limitations of the FoxPro 1.0 and 2.x user interfaces – a minimal number of events and limited control over what the user was doing and what you were able to control. A far cry from the incessant READs of dBASE and FoxBASE interfaces, to be sure, but we spent an incredible amount of time devising workarounds. With the form designer tools of VFP (and similar tools like VB), a whole new world opened up to us: extremely granular control over every aspect of the interface.

As they said in the movie Mad Max Beyond Thunderdome, "No matter where you go, there you are." And here we are, back where we were. While accelerating, the capabiltiies of today's browsers simply don't match what we have on the desktop with our other tools. Many have compared them to dBASE III or FoxBASE. I won't argue where we stand on the ladder of evolution – but we've taken a few steps back for the time being.

HTPP protocol

You'll have to rework your data access mindset and strategies, being more careful about what you're asking for and when. With VFP's blinding speed, you could get away with being lazy – don't have the right data? Just grab another batch through a SQL SELECT. Don't know what you want? No matter – grab as much as you want, and then winnow down the result set later.

You'll approach your data differently now.

Plumbing

Production site requirements

What you need and where you can get it.

First, start off with a clean machine – all the way down to formatting the drive and installing everything from scratch. It's simply too easy for remnants of something that was supposed to have been completely uninstalled to be left around and cause trouble when you least expect it.

Second, install NT Server 4.0. Internet Information Server 2.0 comes with NT Server.

Third, install Service Pack 3 for NT 4.0. This includes IIS 3.0. SP3 is available for download at www.microsoft.com, among other places. (Of course, you need NT 4.0 to use it.) You'll now see Programs/Microsoft Internet Server (Common) on the Start menu.

Fourth, Visual Studio Enterprise 6. You can get SP3 and IE4 from this package as well. Then you are asked which type you want to install: Custom, Products or Server Applications. Select Server. This will install a number of items, including NT Option Pack 4.0, Front Page Server Extensions, Visual FoxPro Server, Visual InterDev Server, and MS Data Access Components.

NT Option Pack includes updates to some components, including from IIS3 to IIS4.

You'll now see Programs/Windows NT 4.0 Option Pack/Microsoft Internet Information Server/Internet Service Manager, which brings up Microsoft Management Console (MMC).

Development software - Server

Same as Production Server

Development software - Workstation

Now let's look at what you'll want to do for your workstation.

First, again, start off with a clean machine. This is probably more important since you're more likely to have installed something goofy on a workstation. It's also more of a nuisance, again, because you're more likely to have a ton of things already installed.

Install Windows NT Workstation 4.0, and Service Pack 3 for NT 4.0.

Install Visual Studio Enterprise, Custom. You'll get Internet Explorer 4.0 as well as Visual FoxPro and Visual InterDev.

MSDN – comes with Visual Studio – have several gigabytes free, because you'll want to install the whole thing.

Hints

Use a server-workstation network setup. You can play around with everything on a single machine, but for actual development, you'll want to use a local and a master.

Start with a clean install. FDISK is your best friend when preventing problems from the get-go.

Mentally get ready to install more than once. Get all of your files ready, keep them in a single place. Some people like a messy desk; others have to have everything just so. I'm one of the latter, and so I prefer to keep everything as clean and tidied up as possible, and that includes my hard disk and my Random Access Memory.

Document what you do, what happens, and what you've accomplished. It's very easy to install something the third or fourth time, and not realize that you forgot Step 7. Be particularly sure you document which version of a tool you're installing.

Active Server Pages

What is ASP and where does it come from?

"ASP" stands for Active Server Pages. It is a technology that allows processing to be done on the web server, with simply the results being sent back to the browser – in HTML format, which can be understood by any browser! ASP essentially is VBScript that runs on the server, and that generates platform-independent code.

ASP comes along for the ride with IIS 3.0 and later, and takes the form of a DLL in the WINNT\SYSTEM32\INETSRV directory on your web server machine.

A web server is a piece of software running on a machine that handles requests for HTML files. After installing the web server software, you configure it to identify where the "root" of the web application is, and then you place your main web application files there. Then, when the web server receives a request for a page, it looks for that file and returns it to the browser, much like Word will open up a file and present it to the user during the execution of a File, Open command.

If the web server supports ASP, and encounters a request for a page ending with an .ASP extension, it will execute the ASP.DLL. The ASP.DLL does additional processing of the information contained on the ASP page, and then returns control back to the web server, which then dishes information back to the browser.

By the way, an "ASP page" may sound redundant, but is the officially supported nomenclature from Microsoft. When you think about it, it actually makes sense – the "Active Server Pages" is the name of a technology, and "pages" are files on a server. So just as you can refer to "HTML pages" or DHTML pages" or "My Aunt Martha's pages", it is correct to refer to Active Server Pages pages. It just sounds a bit funny, and you do get the red squiggly lines under the second "pages" in Word.

All I need to know about VBScript I learned at DevCon

Explaining VBScript in a few pages here is like explaining "What kind of software can you build with that Visual FoxPro?" to your father-in-law as he's on his way to bar for a refill. But we can go over the essentials quickly enough.

The <SCRIPT> tag

VBScript code is placed between <SCRIPT> and </SCRIPT> tags.

```
<SCRIPT LANGUAGE = "VBScript">

<INPUT NAME="Button1" TYPE="BUTTON" VALUE="Click Here" OnClick='MsgBox
"Holy High Tech, Batman!"'>

</SCRIPT>
```

There are a number of keywords, including RUNAT which specifies where the VBScript is to be executed.

Running VBScript on browsers that don't handle it

Accommodating technology-impaired browsers – place your VBScript code between comment tags: <!-- and -->

```
<SCRIPT LANGUAGE = "VBScript">

<! - -

<INPUT NAME="Button1" TYPE="BUTTON" VALUE="Click Here" OnClick='MsgBox
"Holy High Tech, Batman!"'>

 - - >

</SCRIPT>
```

VBScript language elements

Code Continuation

The underscore is used to continue long lines of code, like so:

```
If (expression1 or expression2) and _
   (expression3 or expression4)
```

Variable declarations

Unlike Visual Basic, VBScript only supports one type of language element – the variant. A Variant is a variable that can represent any kind of data – character, numeric, date, and so on. This is not a concern for Visual FoxPro developers, as we have lived with these all of our lives. However, a variant's subtype can distinguish between types, and this can cause errors in your code, just as when you try to add a character variable and a date variable in VFP.

Static and multi-dimensional variables

Static variables are the same as variables we use in FoxPro; multi-dimensional are arrays. They are both created with the DIM keyword. Note that the array index starts with zero.

Variable scope

Variable scope is similar to Visual FoxPro, with the Dim, Private and Public statements giving you control over scope similar to Local and Global in VFP.

Operators

Operators are used to perform calculations. They include the standard suite of addition (+), subtraction (-), multiplication (*), floating point division (/), integer division (\), exponentiation (^), string concatenation (&), modulus (MOD), and Boolean operators such as AND, OR, NOT, XOR, value comparators (>, <, =), and an object comparator: IS.

Control Structures

VBScript uses a standard array of logic control structures, including IF, FOR, CASE and DO. Probably the most common is the IF/THEN/END IF structure.

Functions

VBScript comes with a wide range of functions. These include (but are not limited to):

Data-conversion: Int, Fix, Hex, Oct, Cbool, Cbyte, Cble, Chr, Cint, CLng, CSng, CStr, and Val

Data validation: IsArray, IsDate, IsEmpty, IsError, IsNull, IsNumeric, IsObject, and VarType

String Manipulation: LSet, Mid, RSet, CStr, InStr, Lcase, Left, Len, LTrim/Rtrim, Right, Str, StrComp, String, UCase, Val

Date/Time: Cdate, Date, DateSerial, DateValue, Day, Hour, IsDate, Minute, Month, Now, Second, Time, TimeSerial, TimeValue, Weekday

You can already guess what most of these are used for because of their similarity to Visual FoxPro keywords.

Running VBScript on the server

Enclose VBScript between <% and %> tags when it is to be run on the server.

VBScript Subroutines

Once your application grows past “trivial”, you’ll want to break it into pieces. Subroutines (we know them as “functions” or “procedures”) are one way to do so.

```
<TITLE>Hi Ho</TITLE>
<BODY BGCOLOR = "YELLOW">
<%=MyMessage()%>

<SCRIPT LANGUAGE=VBSCRIPT RUNAT=SERVER>
Function MyMessage
Mymessage = "It's off to work we go…"
end function
</SCRIPT>
```

This code sample places a button on the form, and then calls the subroutine from the onclick method of the button:

```
<HTML>
<TITLE>Hi Ho</TITLE>
<BODY BGCOLOR = "YELLOW">
<form>
<INPUT TYPE=BUTTON VALUE="Press me!" NAME="B1">
</form>

<SCRIPT LANGUAGE=VBSCRIPT>
Sub B1_onclick
msgbox "You have pressed me"
end sub
</SCRIPT>
</BODY>
</HTML>
```

Server-side Include Files

Use the code <!-- #INCLUDE FILE=”myfile” --> to insert “myfile” into an ASP page. Note that the Include file is inserted just before the web server processes the script, and so the include file is added to the ASP file just before it executes. A typical use for an Include file is for a common header or footer. Here is what an .ASP page that contains calls to header and footer INCLUDE files looks like:

```
<TITLE> My Application’s Title </TITLE>
<!-- #INCLUDE FILE = “fsheader.html” - - >
<! - - your application goes here - ->
<!-- #INCLUDE FILE = “fsfooter.html” - - >
```

This is what FSFOOTER.HTML looks like:

```
<html>
<body>
<p><a href="fshumour.htm">Back to main Humour page</a> </p>
<p><a href="index.htm">Back to main DevStudio Apartment page</a></p>
</body>
</html>
```

GLOBAL.ASA

Just as your Visual FoxPro applications typically have a main PRG file that controls the rest of the application. In an ASP application, the GLOBAL.ASA file serves the same purpose. This file is placed in the root directory of the application, and defines the beginning and end of an application as well as the beginning and end of an individual user’s session.

GLOBAL.ASA detects four important events: Application_OnStart and Application_OnEnd, which are both application specific, and Session_OnStart and Session_OnEnd, which are user specific.

A GLOBAL.ASA shell would look like this:

```
<SCRIPT LANGUAGE=VBScript RUNAT=Server>
Sub Session_OnStart
End Sub

Sub Session_OnEnd
End Sub

Sub Application_OnStart
End Sub

Sub Application_OnEnd
End Sub
</SCRIPT>
```

ASP objects and components

Despite the richness of the language as describer earlier, VBScript is still fairly limited in some areas, and one of those areas is accessing data files. VBScript has no functions that allow access to a database, or to manipulate files on disk.

ASP objects and components fill this need. These are simply ActiveX components – DLLs that you already may know about and use in your VFP applications.

ASP objects are intrinsic to VBScript – and consist of a series of five: Application, Session, Request, Response, and Server. ASP components, on the other hand, are DLLs that you make yourself (or buy elsewhere.)

Application object

The application object is similar to a global application object in VFP, in that it can hold and maintain variables throughout the life of the application. However, unlike an application object in VFP which is private to each user, the ASP application object is available to all users at the same time.

```
<%Application.Lock%>
<%Application("Today") = Date%>
<%Application.Unlock%>
```

Note that we used the methods Lock and Unlock in order to restrict access to the Today variable while we were assigning it a value. The variable is global to the application, and so you have to deal with concurrency – more than one user trying to make a change at the same time. (The Lock method locks the entire Application object – not just the variable you are working with.)

Session object

The session object is more similar to the VFP application object, in that it's scope is a single user. The syntax is the same:

```
Session("UserName") = "My Name"
Session("Password") = "mypassword"
```

This type of code could be typically contained in a GLOBAL.ASA subroutine:

```
<SCRIPT LANGUAGE=VBScript RUNAT=Server>
Sub Session_OnStart
  Session("UserName") = "My Name"
  Session("Password") = "mypassword"
End Sub
</SCRIPT>
```

Request object

When a browser sends data to a web server, it uses "form submission" processes. The form has <FORM> tags together with data input controls such as text boxes and action controls such as buttons.

```
<FORM METHOD="POST" ACTION="sdartdb.asp">
<P><INPUT TYPE="TEXT" NAME = "T1"</P>
<P><INPUT TYPE="SUBMIT"></P>
</FORM>
```

When you push the Submit button, a text string gets sent to the web server. The Request object on the server grabs this text string and parses it into a format the server can understand, like so:

```
<%=Request.Form("T1")%>
```

There are additional uses of the Request object, such as hyperlinks and grabbing properties of the client.

Response object

So now that we've got data, what do we do with it? We gotta send it back to the browser, right? We use the Response object:

```
<%Response.Write("Rushmore Technology rocks!")%>
```

However, writing the phrase "Response.Write" over and over again could get pretty tedious. More tedious, than, for example, writing "@SAY" and "@GET" a hundred times, I suppose. So you can use the "=" sign as a shorthand:

```
<%="Rushmore Technology rocks!"%>
```

The Response object has a number of properties, and one very handy property is the Expires property. Setting the Expires property to 0 will prevent the browser from caching the results of the previous request of the page, which is what you want if the data being sent back to the browser is continually changing:

```
<%Response.Expires = 0%>
The current time is <%= Now %>
```

Server object

The Server object is sort of like your utility PROC file (or utility class, in VFP.) It will provide system services and access to a variety of unrelated but important methods and properties. The one that you're going to use often is the CreateObject method.

```
Set oVFP = Server.CreateObject("X.Y")
```

Where X.Y is an ActiveX control – in this case, a .DLL built in VFP. Other examples would look like these:

```
Set oWord = Server.CreateObject("Word.Basic")
Set oXL = Server.CreateObject("Excel.Sheet")
Set oLetter = Server.CreateObject("Mail.Connector")
```

The object names, oWord, oXL, and so on, are just my own object names – they're not tied to specific products or components. You could name them oHerman and oBetty if you wanted to.

VFP COM Objects

"The Middle Tier"

By now you can probably recite the pieces of an n-tier application in your sleep: you've got the user interface component, the business rules component, and the data storage component. And just like normalization, while you can go deeper, those three are really the meat and potatoes of the matter.

Since we're talking about Web-based applications, it's obvious that the user's browser is the user interface component. Kind of a shame, because we're kind of getting used to all the wonderfully slick interface tricks we can play with Visual FoxPro. Spoiled, even, I'll bet.

But all is not lost, because there's just as much power under the hood of VFP. Using Visual FoxPro to build the business rules engine is a joy because of it's power, speed, and flexibility. If you've scanned the current literature, you've seen what it takes other tools to do even simple database processing that we merely sniff at. Visual FoxPro is very much at home here.

But before we going into the details, I should mention that you can use Visual FoxPro or another back end as the data source tier. To make these example simple, we'll use VFP tables, but it's not that much of a leap, once you've got these mechanisms mastered, to talk to other data back ends.

Creating a DLL

Do you remember that commercial where the mother is sitting in the kitchen, reading a magazine, and nuking a cake or some brownies in the microwave? Meanwhile, the family is patiently waiting in the dining room for Mom to finish working on this masterpiece. Just before she delivers the goodies, she rumples her blouse, musses her hair and sprinkles some flour on her face to give the appearance of having worked for hours. "They'll think it took all day" is the tag line.

Building and installing a COM component can be pretty much the reverse. If all goes smooth, it's a ten minute job. But there are a number of tricks and traps that can make this ten minute process an exercise in endurance and stamina.

First, remember that a .DLL is just like any another Windows component – you instantiate it, maybe pass it some parameters, and get a result back. In this simple case, we're going to pass some parameters that represent data that the user has entered via their browser, and we're going to return a result set that is a formatted HTML page. In the middle, the DLL will do some processing with the parameter(s)s passed to it.

(In order to keep things simple for this first pass, we're going to avoid all that error trapping nonsense.) All this code does is accept a parameter, concatenate an exclamation mark and then a second copy of the parameter, and return that value. Clearly, once this part works, using the parameter to retrieve data from (or stuff data into) a database is simple.

```
* vfpmain1.prg
define class vfpserv1 as custom olepublic
```

```
name = "vfpserv1"
func main
lpara m.tcX
m.lcRetVal = m.tcX + "!" + m.tcX
return m.lcRetVal
endfunc
enddefine
```

Now that we've created this program, create a project (let's say, named VFPSERV1), and add this program as the main program.

Next, select the Build option, select the Build COM DLL, check the Regenerate Components ID's checkbox, and press OK in order to build it. The status bar will flash a few messages, including one about registering the Type Libraries, and you're done. Well, not completely done. Now that you've built the DLL once, you're going to have to go back in and do it a second time, but this time to make it a multiple use DLL – a new capability in VFP 6.

With the VFP Project Manager window open, select the Project, Project Info dialog from the main menu. Select the Servers tab, and change the Instancing drop down from Single-Use to Multiple-Use. Select OK, and rebuild, being sure to check the Regenerate Component ID's checkbox again. (You have to do this second build because the Instancing drop down is not enabled on brand new projects.)

Testing in VFP first

You must ensure that your DLL is completely bug free. If an error occurs, VFP will throw a dialog – and guess where that's going to happen? On the web server! Since there is probably no one around monitoring the server, it's just going to sit there – and what has happened from the point of view of the user is that the page has hung. And they'll close their browser and not come back.

Not only errors will cause the application to freeze from the point of view of the user. Suppose a file isn't where it's supposed to be? VFP will display the Open dialog – on the web server, which results in the same effect – the application will appear to have hung for the user.

You'll want to test your DLL from within VFP first. Calling a DLL from within VFP is simple – using the DLL described above, the following two lines of code (resulting in the third line) will do just fine:

```
oX = createobject("VFPSERV1.VFPSERV1")
? oX.main("herman")
herman!herman

* you may wish to release as well
oX=.NULL.
```

Installing the DLL on the web server

Once you've successfully built and tested your DLL, you'll want to put it on the server. The easiest way is to use the VFP Setup Wizard to build an application you can install on your web server – and the application simply consists of the DLL (and the VFP runtime.)

Calling from an ASP

There are actually two files involved in calling the DLL we just built. (Both files reside on the web server, of course.) The first is an HTML page that contains controls that will gather data from the user and allow the user to submit that request to the server. Here are the three most interesting lines in SDARTDB.HTM:

```
<!-- Sdartdb.htm -- >
<form ACTION="sdartdb.asp" method="POST">
<input type="text" name="T1" size="50" tabindex="1">
<p><input type="submit" value="Submit" name="B1" tabindex="98">
```

The first line defines what will happen when you push the Submit button: a new page, SDARTDB.ASP, will be called.

The second line displays a 50 character text box (named "T1") for the user to enter data.

The third line displays a command button that sends the request to the server.

Now let's look at the second file – the actual ASP file. Again, there are just a few interesting lines in SDARTDB.ASP:

```
<%
  set oVFPObject = Server.CreateObject("VFPServ1.VFPServ1")
  lcSearchText     = Request.Form("T1")
  cGetArticleResults = oVFPObject.main(lcSearchText)

  Response.Write("<p>The criteria you entered are:</p>")
  Response.Write("<br>")
  Response.Write("Search Text: " & lcSearchText)
  Response.Write("<p>The Articles that match are...</p>")
  Response.Write(cGetArticleResults)
  Response.Write("<p><a HREF='sdartdb.htm'>Back</a> <br></p>")
  'Release object variable
  Set oVFPObject = Nothing
%>
```

I've divided this code into two parts. The first part instantiates the DLL and gets the data. The second part returns it to the user.

In the first part, the first line issues a createobject to the DLL and creates an object reference. Then we create a local variable from the data the user entered into the "T1" text box. Finally, we call a method of the DLL (the "main" method), using the local variable as a parameter, and get a return value back.

Note that this is all working "behind the scenes" – there is no user interface in the DLL. This is important to remember because this DLL is running on the web server. If the DLL fails, the user's browser is just going to stand still while a VFP error message displays on the web server – millions of miles away.

In the second part, we simply generate a number of HTML strings, including the value that the user had originally entered, and the return value. We'll also be kind to the user by giving them a link back to the main page.

Tips

Errors happen. Some are our fault, and some, well, they just happen. A few typical errors you will likely run into are:

“File Access Denied.” This one occurs at the end of the Build process in VFP, and it’s because you’ve already instantiated the DLL from an ASP page. Once you’ve done so, the DLL stays in memory. You can run the ASP page in a separate memory space (mark the checkbox in the Home Directory tab of your Web Site Properties dialog under MMC).

“Can’t create object.” This one occurs when your HTML page is calling the ASP page. There are a whole host of possibilities here.

Have you set the DLL to multiple use? If you call the ASP page a second time with a single use DLL, you can often get this error.

Have you set your permissions properly? The default user is IUSR_machine, where “machine” is the name of the web server (e.g. IUSR_TAZ, IUSR_WEBSTER, etc.) By default, this user only has guest permissions. You’ll need to change them so that the user has access to a variety of directories; which ones depend on how your application is configured and where it’s components are. At a minimum, the user will need access to the location of the DLL, databases, and probably a temp directory. You can add this user to the Administrative Group as a temporary measure.

“Invalid class string.” This means you typed the wrong DLL or class name in your createobject function in the ASP page.

Just like voting in certain large Midwestern cities, reboot early, and reboot often. Even the most innocuous changes can fail to register – usually at the most inopportune times.

ADO

What is ADO and where does it come from?

Accessing data isn't as easy as it used to be. Back in the olden days, we could count on data being pretty homogeneous, all residing in a single type of structure. Our biggest challenges were importing data into our preferred type of storage from a foreign source, and normalizing (and then denormalizing) what it was that we had collected.

But data is increasingly coming in a variety of formats – and have to stay in those formats. For example, we may want to have at data on a mainframe – in pre-relational formats as well as normalized structures. And at data in a server database – like Oracle, SQL Server, or Ingres. And at data on the desktop – in Visual FoxPro, Jet, and Paradox. But it gets worse – we also want to get at data in non-normalized data, such as spreadsheets, multi-media files, email, and a whole host of specialized structures. As a result, we have to access that data in their native format – and that's, er, a challenge.

Enter Universal Data Access, a new Microsoft architecture that provides access to nearly every type of data you could want to get your hands on. UDA is a single interface that allows you access to data in it's native format – a much better solution than having to convert data regularly.

UDA is comprised of a number of components, including ActiveX Data Objects, Remote Data Services, ODBC and OLE DB. OLE DB is a system-level interface to data throughout the enterprise – it's the tool that talks directly to the data sources. And, as you can imagine, often that's a complex interface to deal with. ADO is an application-level programming interface that is supported by a variety of development tools. In other words, it's a wrapper for OLE DB. Our application talks to ADO, and ADO handles the rest of the path through OLE DB on to the data itself.

But what is "ADO" or "OLE DB" or any of these other things? They are a set of DLLs and related files that you can find in your Program Files\Common Files\System directory. For example, under Program Files\Common Files\System, you'll find an ADO directory that contains MSADO15.DLL, MSADER.15.DLL, and MSADOR15.DLL. Just as an .APP file in Visual FoxPro can provide functionality via classes or functions contained in it, these DLLs provide the functionality provided by ADO services.

The ADO object model

So you're saying to yourself: "How do I know what objects I can create from ADO? How do I know that the connection object has a Provider property? How do I know that the RecordSet object has an Open method?" You can find about 200 books at the local bookstore on ADO. But we can do a quick pass right now.

The ADO model includes seven major objects that are organized in a hierarchical structure:

```
Connection
Command
Recordset
Field
Parameter
```

```
Property
Error
```

The Connection object allows you to connect to your database.

The Command object allows you to specify a command that you want to execute on a database.

The Recordset object is used to manipulate a set of rows within the database. This subset could be the entire database, but, more likely, will be a small subset for performance and management reasons.

The Field object, like the recordset object, applies to the database – in this case, to a specific field in the database.

The Parameter object can be used to specify parameters for executing a command against the database, such as parameters that are passed to a stored procedure.

The Property object contains specific properties that are defined by the service provider (the provider of the interface.)

The Error object collects error information generated when attempting to perform a database operation that fails.

Other sessions here at DevCon spend more time on the plumbing of ADO. For now, all you have to know is that it can be accessed in an ASP page just like our VFP COM objects. Let's look at a couple of examples just for fun.

Connecting to data and manipulating recordsets

Here is code that will create a series of objects inside VFP:

```
oRS = Createobject("adodb.recordset")
oCN = createobject("adodb.connection")
```

If you did this in VFP and then displayed memory, you'll see something like this:

```
RST    Pub    O    ADOR.RECORDSET
CNN    Pub    O    ADODB.CONNECTION
```

Here is the same code that will create a series of objects inside an ASP page with a SQL Server data source:

```
<%
Dim oCN
oCN = createobject("adodb.connection")
oCN.Open "Publications", "sa", ""
Dim oRS
Set oRS = Createobject("adodb.recordset")
oRS.open "SELECT * from Publishers", oCN
%>
```