

# Your First Web Application With VFP

Hector J. Correa  
[hector@hectorcorrea.com](mailto:hector@hectorcorrea.com)  
[www.hectorcorrea.com](http://www.hectorcorrea.com)  
October/2001

## Content

- [What is a Web Application](#)
- [Types of Web Applications](#)
- [Structure of a Web Application](#)
- [VFP on a Web Application](#)
- [Setting up sample programs](#)
- [Code Samples using VFP for Thin Client Web Applications](#)
  - [HelloWorld.ASP](#)
  - [VFPSample.ASP – VFP on the data tier](#)
  - [Customers\\_\\*.ASP](#)
  - [VFPCOMSample.ASP – VFP on the middle tier](#)
  - [Employee\\_\\*.ASP](#)
  - [Proxy.ASP](#)
- [Code Samples using VFP for Thick Client Web Applications](#)
- [A Final note](#)
- [References](#)

## What is a Web Application

A clear and simple definition of a web application can be found in the Microsoft Computer Dictionary, which defines it as:

"A software program that uses HTTP for its core communication protocol and delivers Web-based information to the users in the HTML language."

As you can see from this basic definition, the key aspect of a web application is that it uses HTTP (HyperText Transfer Protocol)<sup>1</sup>. HyperText Transfer Protocol is the protocol behind the World Wide Web. Every time you go to a web page and click on a link or submit a form, an HTTP command is sent to a web server to process your request.

An interesting observation that we can be made on this definition is, although most web applications are used through the Internet, the technologies for web applications can also be applied to internal networks (intranets) as well.

---

<sup>1</sup> As we will see later in this document, the use of HTML is very important for thin client web application, but becomes less important on thick client web applications.

## Types of Web Applications

There are basically two types of web applications: thin client web applications and fat client web applications (also called thick client web applications).

A thin client web application is where the client portion of the application (the piece of the application that the users execute) is a program that does not have built-in intelligence about the application. In this type of web application, the client portion merely knows how to display information to the users and allow them to input information. Furthermore, the client portion does not know (or knows very little) how to perform validations on information entered by the user. Nor does it know how to process this information in order to return results to its users. In most thin client applications, the web portion of the application is a web browser like Internet Explorer or Netscape Navigator.

A typical example of a thin client web application is Amazon.com. When you go to the Amazon home page the browser presents a screen with an option to input the name of the book. You then input the title of a book and click the submit button. At this point, the browser does not have a clue whether what you just entered matches an existing book, or how much a book with that title will cost. The only thing that the browser knows to do is to pass your request (using HTTP) to a web server. This web server will return to your browser the results of the search, either a message telling you that this book does not exist or a list of books that match the name that you indicated (with prices, pictures, discounts and so forth).

In a fat client application, the client portion of the application is a full-blown Windows application with a rich user interface. There are two factors that make *fat* a fat application. One is the fact that the client portion of the application is something that the users do not have already installed on their computers and that they need to install (e.g. an application developed in Visual FoxPro or Visual Basic). The other factor is the fact that the client portion of the application may have built-in rules and processes as part of the application. In other words, it may know how to process information.

There are several advantages and disadvantages on either type of web application. When using a thin client web application, users normally do not need to have anything installed on their computers except a browser. Additionally, since the application does not actually reside on the user's computer, you can update the server portion the application and the users will get the updates immediately. On the other side, thin client applications are limited to the user interface that you can build using a browser, which is far more limited than what you can achieve in a regular Windows application developed in Visual FoxPro or Visual Basic.

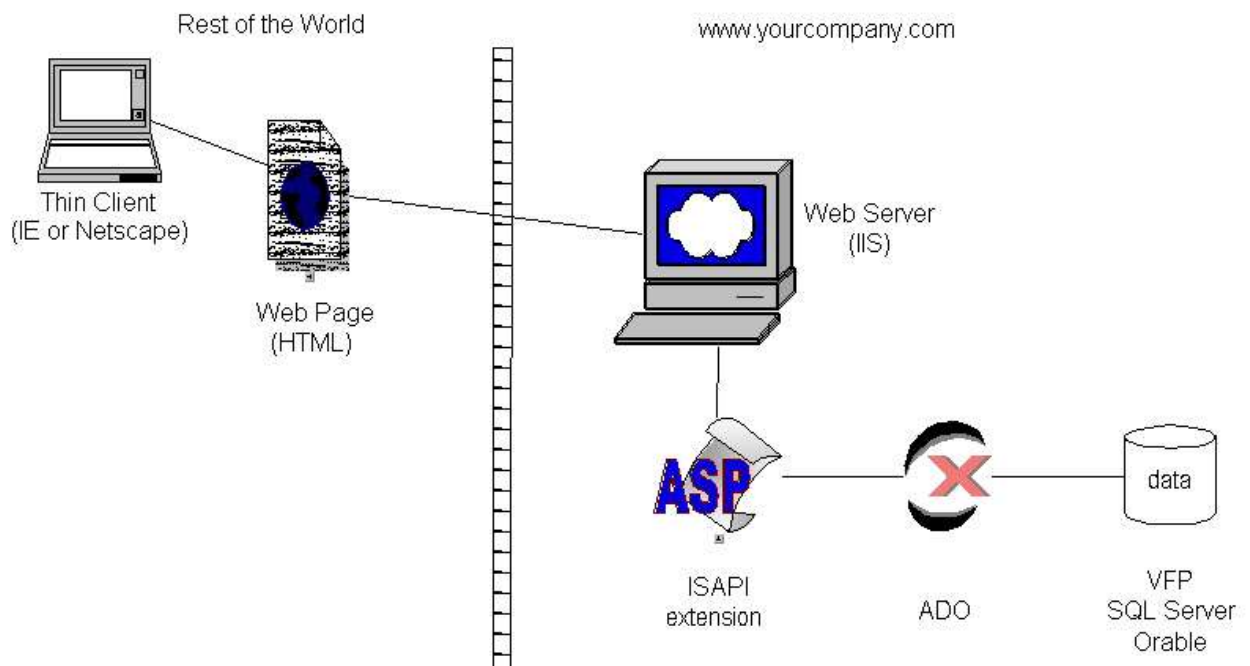
## Structure of a Web Application

Web applications are client/server applications developed using n-tier architectures. A client server application is split into, at least, two components: a client

component that the users run on their individual workstations when they use the application and a server component that resides on a server computer and provides some central functionality under the hood. When a client/server application is split into more than two components (tiers) it is called an n-tier application.<sup>2</sup>

This n-t-tier concept is probably the most challenging concept to master when developing your first web application, especially for those of us who were used to creating monolithic applications where everything was self-contained in a single executable. Try keeping this n-tier architecture concept in mind as you work on your web applications and your life will be much easier.

In a web application there are typically three tiers involved: a presentation tier, a middle tier, and a data tier. In a thin client web application, the presentation tier is usually a web browser such as Internet Explorer or Netscape Navigator. The middle tier, on the other side, typically consists of a web server like Internet Information Server (IIS) and an ISAPI extension such as Active Server Pages (ASP). Finally, the data tier can be carried out by any database management system (VFP, SQL Server, or Oracle.) The following picture depicts this structure:



The following table explains how these technologies work together in a typical thin client web application.

Presentation Tier	Middle Tier	Data Tier
-------------------	-------------	-----------

<sup>2</sup> There are several other ways to describe and define *client/server structure* and *n-tier architecture*, the explanations presented in this document represent typical scenarios.

The user launches a **web browser**, inputs a URL like [www.yourcompany.com/catalog.asp](http://www.yourcompany.com/catalog.asp) and hits enter.

At this point, the web browser sends an HTTP request for a document called `catalog.asp` to a web server called [www.yourcompany.com](http://www.yourcompany.com)

A **web server** receives the user request and determines (based on the document extension) that the document needs to be processed by an **ISAPI extension** called Active Server Pages (ASP)

Active Server Pages loads the `catalog.asp` document and parses it out.

While parsing the document, ASP connects to a VFP database using ADO and reads the list of items available in the catalog table.

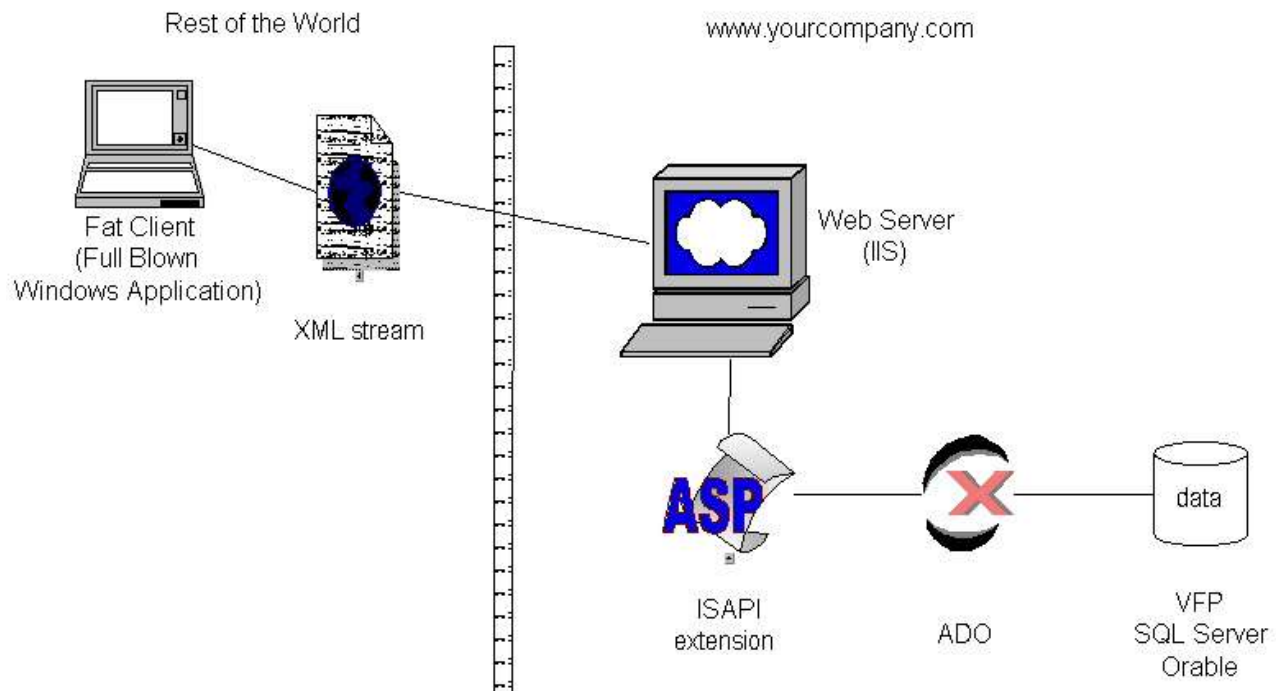
The **VFP database** is read through ADO.

The code in `catalog.asp` creates an HTML document listing all records found in the catalog table and passes it back to the web server.

The **web server** returns the HTML document to the web browser.

The **web browser** renders the received document and displays it to the user.

For fat client web applications the only piece that changes is that the presentation tier is realized by a full-blown Windows application developed in Visual FoxPro or Visual Basic. The following picture depicts this structure:



It is a very well known fact that the technologies involved on the middle tier are very dynamic and new technologies are appearing every day. There are several web servers available for the Windows platform from several different vendors and each of them provides its own set of advantages and disadvantages. Likewise, the number of ISAPI extensions available for Visual FoxPro developers is growing each day. Keep in mind that, although the technology behind web applications is changing rapidly, the basic structure behind these applications remains the same: client/server with an n-tier architecture. This is true whether you are working on Windows, Unix, or planning on using .NET in the future.

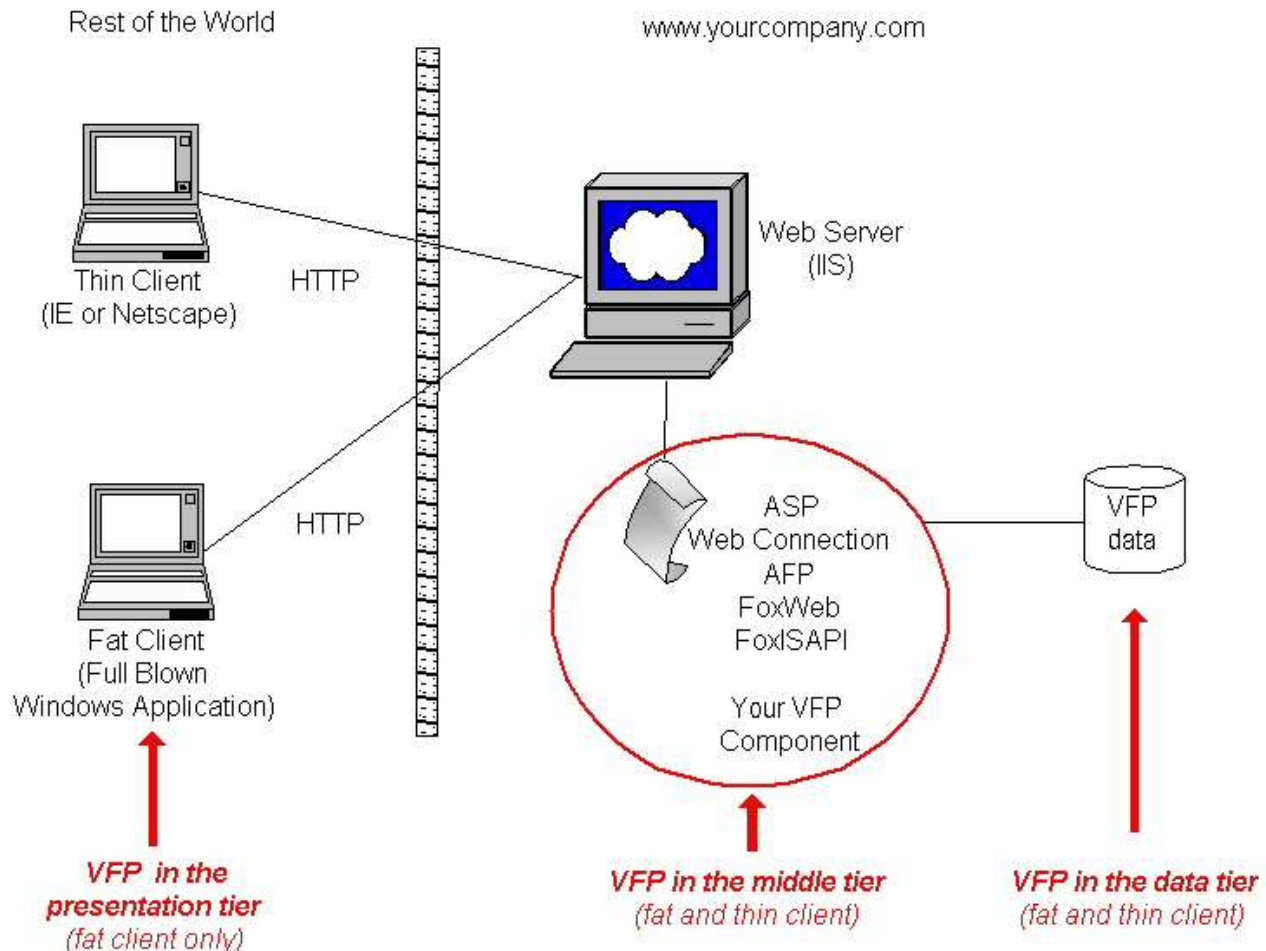
## VFP on a Web Application

Visual FoxPro can be used to develop pieces of software for any of the three tiers involved in a web application.

On one hand, VFP provides a powerful database engine that can be used in the data tier of a web application.

There are several combinations for using VFP in the middle tier, starting with using it through Active Server Pages to using any of the VFP-specific third party products like West-Wind Web-Connect, Active FoxPro Pages, or FoxWeb to name a few.

Finally, since VFP is also a very powerful tool to create Windows applications, VFP can be used in the presentation tier when developing fat client web applications. The following diagram shows where VFP can be used in a web application.



As a programming language, VFP possesses a unique feature among most languages: VFP can be used on any of the three tiers. Most languages, including Visual Basic and Visual C++, can be used only on the presentation and the middle tiers. Nevertheless, since VFP comes with a database engine built-in, it can be used in the data tier in addition to the presentation and middle tiers.

## Setting up the sample programs

In order to run and test the sample programs included with this document, you will need to have Internet Information Server (IIS) running on your machine. Below is a brief explanation on how to install IIS and create a virtual directory under Windows 2000 Professional. These instructions are just a jump-start for your first web application and are

by no means exhaustive. I highly encourage you to look at the references at the end of this document for more complete explanation on how to properly configure IIS for production environments<sup>3</sup>.

## Installing IIS

To install Internet Information Server under Windows 2000 Professional:

1. Go to Control Panel
2. Click on Add/Remove Programs
3. Click on Add/Remove Windows Components
4. Make sure the component "Internet Information Server (IIS)" is checked.

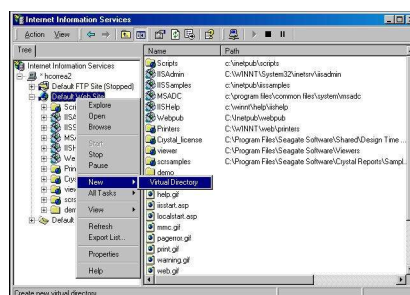
## Testing that IIS is running properly

To test that IIS is running on your computer, launch your browser, input the URL <http://localhost> and hit enter. If after this you see a web page with a welcome message to IIS 5.0 then IIS is running properly. If, on the other hand, you receive a web page that says: "The page cannot be displayed" then something is not properly setup.

## Creating a Virtual Directory in IIS

To run the examples explained in this document you will need to create a Virtual Directory in Internet Information Server. Below are the steps that you will need to follow under Windows 2000 Professional to create a Virtual Directory.

1. Create a directory on your local hard disk (e.g. `c:\g1gdw_web`)
2. Copy all sample files to this directory
3. Create a virtual directory in Internet Information Server
  - a. Go to the *Control Panel*
  - b. Go to *Administrative Tools*
  - c. Click on *Internet Services Manager*
  - d. Open the IIS tree and look for *Default Web Site* branch (if *Default Web Site* is stopped, right click on it and select *Start*)
  - e. Right click on *Default Web Site* and select *New + Virtual Directory* (see diagram)



<sup>3</sup> Internet Information Server comes with Windows NT 4 and Windows 2000. Under Windows 95 and 98 you will need to use Personal Web Server (PWS). Please review the references at the end of this document for instructions on how to set up PWS or IIS under Windows NT 4.

- f. Enter an alias for this new virtual directory (e.g. `FirstWebApp`)
- g. Select the directory that you created in step 1 (e.g. `c:\glgdw_web`)
- h. Allow **read** and **run scripts** privileges to this directory.

### ***Testing your virtual directory***

Once you are sure that IIS is running, you can test your virtual directory. To do this, input the URL <http://localhost/firstwebapp> (or whatever alias you used in step *f* above.) Again, if you get a welcome page with the list of sample programs then you are on the right track. If, on the other hand, you receive a web page saying: "The page cannot be displayed" then the virtual directory `FirstWebApp` is not properly set up.

## **Code Samples using VFP for Thin Client Web Applications**

The files with source code for the samples described below can be viewed and edited with Visual FoxPro or Visual Interdev. I would recommend editing HTML and ASP files with Visual Interdev and PRG files with Visual FoxPro. Nevertheless, since all of them are basically text files, you can edit them with any text editor (including Notepad.)

To run any of these examples, you just need to launch your web browser and enter the URL <http://localhost/firstwebapp>. This will bring you a web page from which you can launch any of the code samples.

Keep in mind that, before running some of these examples (`VfpCOMSample.ASP`, `Employee_*.ASP` and `Proxy.ASP`) you will need to build a few DLLs in your computer. Instructions to build these DLLs are provided on the next pages.

The following sections will describe the important pieces of each of the samples provided with this white paper. At the beginning of each sample there is a list of the files where the source code is included. I would suggest you to open the source code with Visual FoxPro or Visual Interdev and review it as you read through this document.

### ***HelloWorld.ASP***

**Source code file:** `HelloWorld.ASP`

**Description:**

This first example does not involve VFP at all. It shows how to create a simple Active Server Page to display current date and time on a web page. It also shows how to use statements like `if/endif` and `for/next` inside your ASP pages.

Scripting code in an ASP is identified by the `<%` and `%>` markers. The ASP engine will parse out everything inside these two markers as it reads the web page. In this example, the first scripting code is the declaration of the scripting language that you will use, in this case, VB Script:



```
<%@ LANGUAGE = VBScript %>
```

The next scripting piece used in this document tells ASP that you want to get the current date and time. Note that this code will be evaluated at runtime and the appropriate date and time will be displayed on the browser rather than the scripting code:

```
<%=now%>
```

The last scripting code in this example shows how to use VB Script to create loops and conditions. This code creates a loop (with **for/next**) that will iterate five times and display the text “line number” followed by the appropriate number. We also use an **if/endif** statement to display the word “(three)” on the third iteration:

```
<%  
dim i  
for i=1 to 5  
%>  
    Line number: <%= i %>  
    <%if i=3 then %>  
        (three)  
    <%end if%>  
    <br>  
<%next%>
```

As mentioned before, this example does not use VFP on any of the tiers, yet it allows you to see how easy it is to script code in ASP. On our next examples we will build additional functionality based on this basic sample.

## ***VfpDataSample.ASP – FoxPro in the data tier***

**Source code file:** VFPDataSample.ASP

**Description:**

This next example will use an Active Server Page to display VFP data in a web page. In other words, this example uses VFP in the data tier.

Active Server Pages do not know how to talk to VFP data directly. In fact, ASP do not know how to talk to any database management system. Nevertheless, when using ASP you can use ADO to talk to a database. In this example, we will read some data from the TasTrade database that comes with VFP.

*If you are not familiar with ADO, there is a separate example, VFPADOSAMPLE.PRG, included in the samples that show how to use ADO from a VFP program. You might want to check this program first to become familiar with the use of ADO in VFP and then come back to this program to see how it can be used inside ASP.*

There are three main pieces in the VFPDataSample.ASP example and most of them are fully related to the way ADO works more than to the way ASP works. The first one establishes a connection to a VFP database using ADO inside an ASP.

```
<%  
set oConnection = Server.CreateObject( "adodb.connection" )  
oConnection.Provider = "vfpoledb.1"  
oConnection.ConnectionString = "data source=c:\glgdw_web\data\tastrade.dbc"  
oConnection.Open
```

The next piece of code creates an ADO record set. An ADO record set is analogous to a VFP cursor in the sense that it is merely a collection of records in memory. Our record set is just the list of all records in the customer table in the TasTrade database.

```
set oRecordSet = Server.CreateObject( "adodb.recordset" )  
oRecordSet.Open "select * from customer", oConnection  
%>
```

Finally, the rest of the code is the piece that actually works with the customer records (i.e. the data stored in our record set.) First, it displays the *name of the first three fields* in our record set. Then, it displays the *actual values of these fields* for the current record. Lastly, it skips to the next record and displays values of these three fields for this record.

```
<p><b>Name of first three fields are:</b><br>  
<%= oRecordSet.Fields(0).Name %><br>  
<%= oRecordSet.Fields(1).Name %><br>  
<%= oRecordSet.Fields(2).Name %><br>  
</p>  
  
<p><b>Value of these three fields for current  
record are:</b><br>  
<%= oRecordSet.Fields(0).Value %><br>  
<%= oRecordSet.Fields(1).Value %><br>  
<%= oRecordSet.Fields(2).Value %><br>  
</p>  
  
<p><b>We've just moved to the next record.</b>  
<%= oRecordSet.MoveNext %>  
</p>  
  
<p><b>Value of these three fields for this record now are:</b><br>  
<%= oRecordSet.Fields(0).Value %><br>  
<%= oRecordSet.Fields(1).Value %><br>  
<%= oRecordSet.Fields(2).Value %><br>  
</p>
```

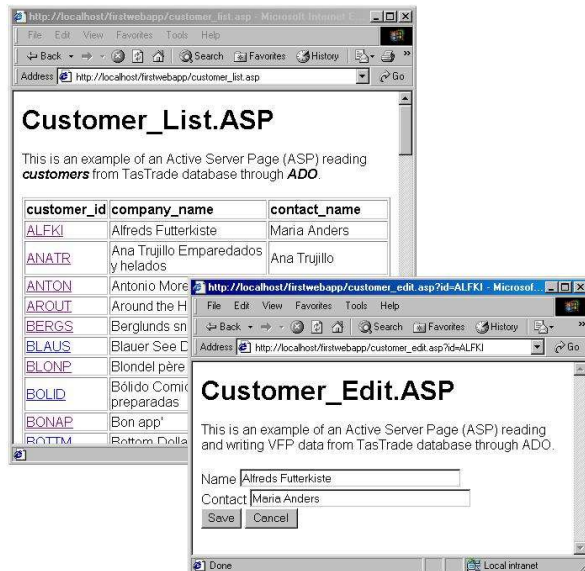
As you can see from this example, working with VFP data inside an Active Server Page is fairly simple. Once you have learned ADO's objects and methods, the code is fairly straightforward.

## Customers\_\*.ASP

**Source code files:** Customer\_List.ASP, Customer\_Edit.ASP and Customer\_Save.ASP

### Description:

This example shows how to use ASP to read customer data from the TasTrade database and make changes to this information.



There are three ASP files included in this example:

- Customer\_List.ASP displays a list of all customers in the TasTrade database.
- When a user clicks on a customer, Customer\_Edit.ASP displays the customer information and lets the user make changes to it.
- When users click “save,” a third page (Customer\_Save.ASP) is executed. This page is never shown to the user. Instead, it redirects the user back to the customer’s list page.

The Customer\_List.ASP file follows basically the same structure as the previous example (VFPDataSample.ASP.) It connects to the TasTrade database, creates a record set of the customers and then creates an HTML page to list all these records in an HTML table. Below is the piece of code that creates this HTML table.

```
<table border=1>
<tr>
<%
dim i
for i=0 to 2%>
    <td><b><%=oRecordSet.Fields(i).Name %></b></td>
<%next%>
</tr>

<!-- Loop through records -->
<%do while Not oRecordSet.EOF%>
<tr>
    <td><a href=customer_edit.asp?id=
        <%=oRecordSet.Fields(0).Value %>
        <%=oRecordSet.Fields(0).Value %></a></td>
    <td><%=oRecordSet.Fields(1).Value %></td>
    <td><%=oRecordSet.Fields(2).Value %></td>
</tr>
<%oRecordSet.MoveNext
loop%>
</table>
```

There are two loops in this code. The first one, a **for/next** command, creates three columns in the HTML table to list the field names as headers. The next loop, a **do while/loop** command, scans through all the records in the record set and adds them one

by one, to the HTML table. Notice how the first column for each record is displayed as a hyperlink. This hyperlink will call our Customer\_Edit.ASP page and will pass the customer ID (e.g. **customer\_edit.asp?ID=ALFKI**).

Code in Customer\_Edit.ASP has two main pieces. The first one creates a record set with only one customer (the customer passed as a parameter by Customer\_List.ASP) In ASP, we can use the QueryString collection from the Response object to retrieve the name values passed to a URL.

```
' Build WHERE clause for SQL Statement
' based on parameters indicated in URL
' (e.g. customer_edit?id=ALFKI)
dim cFilter, cSQL
if len( Request.QueryString("id") ) = 0 then
    cSQL = "select * top 1 from customer order by 1"
else
    cFilter = "where customer_id = '" & Request.QueryString("id") & "'"
    cSQL = "select * from customer " & cFilter
end if

' Requery our recordset
set oRecordSet = Server.CreateObject( "adodb.recordset" )
oRecordSet.Open cSQL, oConnection, adOpenForwardOnly
```

The second piece of code in this file creates an HTML form to display this customer information and let the user edit it. It is important to notice that HTML forms are not bound to data as VFP form are. In an HTML form, we need to read the data that we want to display (which we did in the previous code), manually insert it into the form (see code below) and manually save it (which we will do in Customer\_Save.ASP).

```
<form name="frmCustomer"
      action="customer_save.asp"
      method="post">

<input type=hidden
      name="txtCustID"
      size=40
      value="<%=oRecordSet.Fields(0).Value %>">

Name <input type=text
      name="txtCustName"
      size=40
      value="<%=oRecordSet.Fields(1).Value %>"><br>

Contact <input type=text
      name="txtCustContact"
      size=40
      value="<%=oRecordSet.Fields(2).Value %>"><br>

<input type=submit value="Save" name=cmdSave>
<input type=submit value="Cancel" name=cmdCancel>

</form>
```

In this HTML form, whenever the user pushes the “Save” or “Cancel” button, our Customer\_Save.ASP file will be called. The process to save changes to this file is pretty

straightforward. We basically just create a SQL-Update statement to tell VFP (via ADO) the new values for the customer record.

```
dim cSQL
cSQL = "update customer " & _
      "set " & _
      "company_name = '" & Request.Form( "txtCustName" ) & "', " & _
      "contact_name = '" & Request.Form( "txtCustContact" ) & "' " & _
      "where " & _
      "customer_id = '" & Request.Form( "txtCustID" ) & "'"

oConnection.Errors.Clear
oConnection.Execute cSQL
```

Remember, the call to our Customer\_Save.ASP was done from the HTML form defined in Customer\_Edit.ASP. When Customer\_Save.ASP receives the control, we are using Request.Form() to gather the values that the user entered in Customer\_Edit.ASP.

Finally, once changes have been processed, Customer\_Save.ASP redirects the user back to the Customer\_List page.

```
Response.Redirect "customer_list.asp"
```

All of these previous examples use pure ASP scripting code to read and write VFP data. Although this technique is very popular, scripting code has its drawbacks. For one, the language to write scripts in ASP is limited to whatever capabilities VB Script (or Jscript) provide you. You do not have an object-oriented language like VFP at your disposal when writing scripting code. The following examples will show you how you can write VFP code and use it inside your ASP files.

## ***VfpCOMSample.ASP – FoxPro in the middle tier***

**Source code files:** VFPCOMSample.ASP, VFPCOMSample.PRG and VFPCOMSample.PJX

### **Description:**

In previous examples we have seen how we can use ADO to talk to our database. ADO is a COM Server. But, guess what? We can create our own COM Servers with VFP! With this in mind, you can create a VFP application, compile it as a COM Server, and use it to read VFP data (or any other data) in your Active Server Pages. In other words, we are going to use VFP on the middle tier.

There are two files involved in this example. One is a VFP program file (yes!) and the other is an ASP file. Let's start by looking at VFPCOMSample.PRG file. This is a regular VFP program file where we define an OLE public class (it is very important that your class is marked as OLE public, otherwise, you will not be able to use it from within ASP later on.) As you can see below, code in this class is rather simple. We are just creating a class with two methods: About() and SaySomething().

```
DEFINE CLASS SimpleClass AS SESSION OLEPUBLIC
```

```

FUNCTION About()
RETURN "SimpleClass.About() 1.0"

FUNCTION SaySomething()
    local cRetVal
    cRetVal = "VFP datetime() = " + ttoc( datetime() ) + ENTER +;
              "VFP sys(2015) = " + sys( 2015 ) + ENTER +;
              "VFP SYS(0) = " + SYS( 0 )

RETURN cRetVal

ENDDEFINE

```

Once we have defined our OLE public class, we proceed to compile it into a DLL. To do this, compile the project VFPCOMSample.PJX using the following line in VFP command window:

```
BUILD MTDLL vfpcomsample.dll FROM vfpcomsample
```

This will create a multithreaded DLL and will register it in the Windows registry. Now, let's go and use our new VFP COM Server from within an ASP page. The code in VFPCOMSample.ASP is pretty simple. It basically creates an instance of our SimpleClass defined in our VFP COM Server and then calls the SaySomething() method.

```

<% set oVFPCOMSample = Server.CreateObject( "VFPCOMSample.SimpleClass" ) %
>
<%= oVFPCOMSample.SaySomething() %>
<%set oVFPCOMSample = nothing %>

```

Although this example is very simple and does not use any VFP data, it certainly shows how easy it is to integrate VFP with ASP.

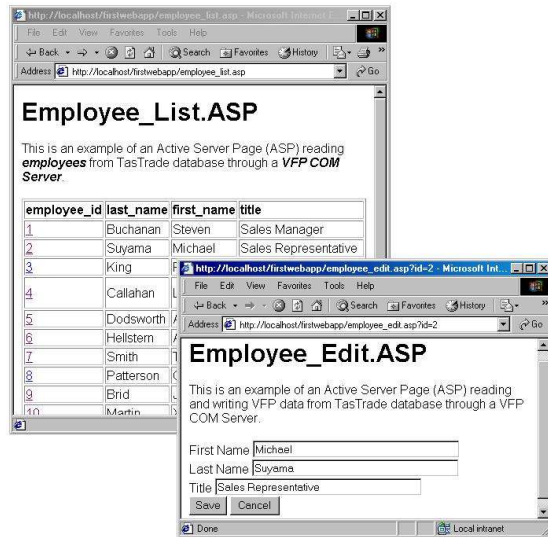
## ***Employee\_\*.ASP***

**Source code file:** Employee\_List.ASP, Employee\_Edit.ASP, Employee\_Save.ASP, Employee.PRG and Employee.PJX

### **Description:**

This example shows how to use a VFP COM Server through ASP to read employee data from the TasTrade database and make changes to this information.

There are four files included in this example:



- Employee\_List.ASP displays a list of all employees in the TasTrade database.
- When a user clicks on an employee, Employee\_Edit.ASP displays the employee information and lets the user make changes to it.
- When users click “save,” a third page (Employee\_Save.ASP) is executed. This page is never shown to the user. Instead it redirects the user back to the employees’ list page.
- Supporting all three ASP pages is a VFP COM Server (Employee.DLL) built and based on Employee.PRG

Let’s first take a look at Employee.PRG. This program defines an Employee class that will be available through COM (i.e. it is marked as OLE PUBLIC) Inside this class there are four methods: Init(), List(), GetField(), and Update().

Code in Init() method basically ensures that the environment is properly set up. This is a crucial step in a VFP COM Server because when a VFP COM Server is instantiated, it will start up with the default settings for VFP – not with your typical development settings. Also, the default directory needs to be explicitly set, otherwise, your class will not be able to find data files.

The List() method creates an HTML table with all employees in TasTrade. This code mimics the functionality that we described in Customer\_List.ASP. However, this time, the code is implemented with plain VFP code.

The Edit() method creates an HTML form with the employee information for a given employee ID.

The Update()method receives a list of fields to update and issues a SQL-Update command to update the employees table. Again, these two methods are implemented using plain VFP code.

Below is a reduced version of this class. As you can see, it uses pure VFP code.

```
DEFINE CLASS employee AS Session OLEPUBLIC

    FUNCTION Init()
        SET RESOURCE OFF
        SET EXCLUSIVE OFF
        SET CPDIALOG OFF
        SET DELETED ON
        SET SAFETY OFF
```

```

        SET DEFAULT TO c:\glgdw_web\data
        SET PATH TO c:\glgdw_web\data
RETURN DODEFAULT()

FUNCTION List()
    * see code in source file
RETURN cHTMLTable

FUNCTION Edit( cEmployeeID)
    * see code in source file
RETURN cHTMLForm

FUNCTION Update( cFields, cEmployeeID )
    * see code in source file
RETURN cSQL

ENDDEFINE

```

Once you have the Employee class in VFP, you can build our COM Server using the following command in the VFP command window:

```
BUILD MTDLL employee.dll FROM employee
```

This will create a multithreaded DLL and will register it in the Windows registry. Now, let's see how you can use this DLL inside your ASP pages and have it do the work for you using VFP methods instead of coding them in ASP.

The code below shows how Employee\_List.ASP looks. As you can see, this code is much simpler than the code that we used in Customer\_List.ASP. Basically, what this code does is: it instantiates our Employee class and calls List() method to do the job. As you can see, there is no ADO involved here. All data access is done through the Employee class that was coded using plain VFP code.

```

<%set oEmployee = Server.CreateObject( "employee.employee" ) %>
<%=oEmployee.List() %>
<% set oEmployee = nothing %>

```

Code for Employee\_Edit.ASP is also rather simple. It retrieves the employee ID and calls our VFP class to create an HTML form.

```

<%
set oEmployee = Server.CreateObject( "employee.employee" )
dim cEmployeeID
cEmployeeID = Request.QueryString( "id" )
%>
<%=oEmployee.Edit( cEmployeeID ) %>
<%set oEmployee = nothing%>

```

Finally, Employee\_Save.ASP is very similar to the code that we used in Customer\_Save.ASP. Yet, in this case, we are using our VFP COM Server to process the update rather than using ADO.

```

dim cFieldList
cFieldList = "first_name = '" & Request.Form( "txtEmpFirstName" ) &"', "&_

```



```

        "last_name = '" & Request.Form( "txtEmpLastName" ) & "'", "&_
        "title = '" & Request.Form( "txtEmpTitle" ) & "'" "

dim cEmployeeID
cEmployeeID = Request.Form( "txtEmpID" )

set oEmployee = Server.CreateObject( "employee.employee" )
oEmployee.Update cFieldList, cEmployeeID

```

As you can see, the amount of code required in our Active Server Pages has been reduced considerably. Keep in mind that this code has not vanished. You just moved it to a VFP class. Still, VFP is a language more powerful than VB Script (or Jscript). On top of that, you are a VFP developer and know how to program in VFP. By using a VFP COM Server the amount of VB Script and Jscript that you need to learn (and mess with) is reduced dramatically. You are, indeed, using VFP to power the Internet ☺

There is one caveat when using VFP COM Servers in Active Server Pages. Once a COM Server (like our Employee.DLL) has been instantiated in an ASP file, Internet Information Server (IIS) will lock this DLL in memory to improve performance in future calls. However, once a DLL is locked in memory, you cannot replace it with a newer version. In order to replace it, you will need to shutdown IIS momentarily and restart it again. In a production environment this is usually unacceptable since it basically means shutting down the shop for a few seconds. This is not a VFP specific problem, it happens to all DLLs instantiated via ASP, regardless of the language in which they were developed.<sup>4</sup>

Fortunately, for VFP developers, there is a work-around for this problem. The following example explains it.

## ***Proxy.ASP***

**Source code files:** Proxy.ASP, Proxy.PRG, MyProgram.PRG, MyOtherProgram.PRG and Proxy.PJX

### **Description:**

A common approach when working with ASP and VFP is to create a class in VFP that will serve as a *proxy* between ASP and your program files in VFP. This proxy class will not have much functionality in it. It will merely receive the name of the function that you want to execute and the name of the file where this function resides. Below is an example:

```

DEFINE CLASS Proxy AS SESSION OLEPUBLIC

    FUNCTION Do( cCommand, cProgramFile )

        IF '.PRG' $ UPPER( cProgramFile )
            SET PROCEDURE TO &cProgramFile
        ENDIF

        xRetVal = &cCommand
    
```

---

<sup>4</sup> Rick Strahl's book has an excellent description of this problem on pages 118-121.

```

        * Clear the PRG from memory so
        * we can replace it later on.
        SET PROCEDURE TO
        CLEAR PROGRAM &ProgramFile

RETURN xRetVal

FUNCTION Init()
    SET RESOURCE OFF
    SET EXCLUSIVE OFF
    SET CPDIALOG OFF
    SET DELETED ON
    SET SAFETY OFF
RETURN DODEFAULT()

enddefine

```

To use this class, you just need to build a COM Server by issuing the following command from the VFP command window:

```
BUILD MTDLL proxy.dll FROM proxy
```

File Proxy.ASP makes use of this VFP COM Server. This file creates an instance of our VFP proxy class and makes several calls to invoke VFP functions [like DateTime()] and some custom functions [like About() and FindOldestEmployee()] included in separate program files.

```

<%set oVfpProxy = Server.CreateObject( "proxy.proxy" ) %>
<%=oVfpProxy.Do( "DATETIME()" ) %>
<%=oVfpProxy.Do( "About()", "c:\glgdw_web\myprogram.prg" ) %>
<%=oVfpProxy.Do( "FindOldestEmployee()", "c:\glgdw_web\myotherprg.prg" ) %>
<% set oVfpProxy = nothing %>

```

The cool thing about this class is that once you have created this class, you will never need to replace it again and, thus, you will not run into the DLL locking problem with IIS. In addition, since your functionality is not embedded in the DLL, but rather is distributed in separate program files (PRGs), you can replace them anytime you want and IIS will not complain!

This concept is extremely powerful and it is unique to VFP since it is the only language that allows you to compile programs from inside another program. Most of the third-party web development tools for VFP developers make use of this concept.

## Code Samples using VFP for Thick Client Web Applications

### ***FatClient.SCX***

**Source code files:** FatClient.SCX, EmployeeFat.PRG, Employee\_List\_Fat.ASP and Employee\_Save\_Fat.ASP

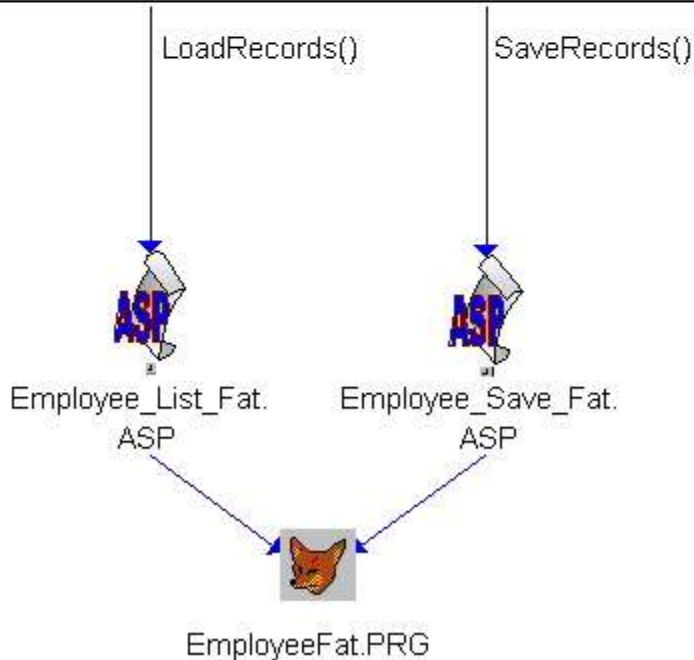
**Description:**

This example uses a VFP form to display and edit employee information. Although the data displayed comes from VFP tables, all access to this data is done through HTTP. The idea of this is that you can run the VFP form from anywhere in the world and read and write VFP data by using HTTP.

**Fat Client Sample**

URL:

Employee_id	Last_name	First_name	
1	Buchanan	Steven	Sales Man
2	Suyama	Michael	Sales Rep
3	King	Robert	Sales Rep
4	Oz	Wizard	Inside Sal
5	Dodsworth	Anne	Sales Rep
6	Hellstern	Albert	Business
7	Smith	Tim	Mail Clerk



There are four files included in this example:

- FatClient.SCX is a VFP form that displays employees' information and let you edit this information. It represents the presentation tier in this case.
- Employee\_List\_Fat.ASP is an ASP file that calls a VFP function in EmployeeFat.PRG to return a list of employees as an XML string.
- Employee\_Save\_Fat.ASP is an ASP file that calls a VFP function in EmployeeFat.PRG to save employee information for a given employee.
- EmployeeFat.PRG implements methods to actually read and save employee information.

Let's start by looking at the VFP form. There are two main methods in this form: LoadRecords() and SaveRecords()

LoadRecords() retrieves employee information, but contrary to a typical VFP application, it does not open tables directly. Instead, it goes through HTTP and asks a web page (Employee\_List\_Fat.ASP) to return the list of employees. Our ASP page returns an XML string with that list. We convert this XML string to a VFP cursor and assign it to a grid. Code for LoadRecords() is shown below:

```

* Connect to URL using XMLHTTP.
oXMLHTTP = CREATEOBJECT( "Microsoft.XMLHTTP" )
oXMLHTTP.Open( "GET", ;

```

```

        "http://localhost/firstwebapp/employee_list_fat.asp",;
        .F. )
oXMLHTTP.Send()

* Convert XML string to a VFP cursor.
* (Our URL must have returned an XML
* string for this to work.)
XMLTOCURSOR( oXMLHTTP.ResponseText, "EmployeeCursor" )

* Assign VFP cursor to grid.
SELECT EmployeeCursor
thisform.grdData.RecordSource = "EmployeeCursor"
thisform.grdData.Refresh()

```

Code for SaveRecords is rather simple as well. This method connects to a web page (Employee\_Save\_Fat.ASP) and sends the fields that need to be updated via XMLHTTP method Send().

```

* Instanciate XMLHTTP.
oXMLHTTP = CREATEOBJECT( "Microsoft.XMLHTTP" )

cString = "id=" + TRANSFORM( employee_id ) + "," +;
          "last_name=[" + TRANSFORM( last_name ) + "], " +;
          "first_name=[" + TRANSFORM( first_name ) + "]"

oXMLHTTP.open( "POST",;
              "http://localhost/firstwebapp/employee_save_fat.asp",;
              .F. )
oXMLHTTP.setRequestHeader( "Content-Type",;
                          "text/xml; charset=utf-8" )
oXMLHTTP.send( cString )

```

Code in both ASP files is pretty similar to what you have seen in the previous samples for thin clients. This time, however, these ASP files do not return HTML text to be rendered by the browser, instead they return XML strings to be parsed by your VFP form.

Below is code in Employee\_List\_Fat.ASP file. Notice how you are just passing a request to the Proxy class to call EmployeeList function inside EmployeeFat.PRG. This function returns an XML string with a list of employees.

```

<%set oVfpProxy = Server.CreateObject( "proxy.proxy" ) %>
<%= oVfpProxy.Do( "EmployeeList()", "c:\glgdw_web\employeeefat.prg" ) %>
<%set oVfpProxy = nothing%>

```

Code for Employee\_Save\_Fat.ASP file is displayed below. The first thing this page does is retrieve data that was sent by your VFP form. To do this, you are using Request.BinaryRead method. Then, by using the Proxy class, you call VFP function EmployeeSave() and pass the parameters received from the VFP form.

```

<%
' sData will contain values passed to this
' ASP file through XMLHTTP.Send() method.
dim sData
sData = Request.BinaryRead( Request.TotalBytes )
sData = BinaryToString( sData )

```

```

' Concatenate sData to our EmployeeSave()
' function.
dim sCall
sCall = "EmployeeSave( '" & sData & "' )"

' Call EmployeeSave() function through our
' Proxy class.
if len( sData ) > 0 then
    set oVfpProxy = Server.CreateObject( "proxy.proxy" )
    Response.Write oVfpProxy.Do( sCall, "c:\glgdw_web\employeeefat.prg" )
    set oVfpProxy = nothing
end if
%>

```

## A Final Note

As you can see VFP is a powerful tool that can be user to create all types of web applications including thin and thick client web applications.

In addition, VFP developers are very fortunate when it comes to web development tools. There are several well-known third-party tools that are targeted at VFP developers and provide a framework for web applications. The following table lists the most common VFP products for web development (listed alphabetically):

Product	Company	Web Page
Active FoxPro Pages	ProLib	<a href="http://www.afpages.com/">http://www.afpages.com/</a>
EEVA Web Server	Eetasoft	<a href="http://www.eetasoft.ee/ewbserv.htm">http://www.eetasoft.ee/ewbserv.htm</a>
FoxWeb	Aegis Group	<a href="http://www.foxweb.com">http://www.foxweb.com</a>
Visual Web Builder	EPS Software	<a href="http://www.eps-software.com">http://www.eps-software.com</a>
Voodoo Web Controls	EPS Software	<a href="http://www.eps-software.com">http://www.eps-software.com</a>
Web-Connect	West-Wind	<a href="http://www.west-wind.com">http://www.west-wind.com</a>

## References

### Books

Wong, Clinton. *Web Client Programming with Perl*. O'Reilly & Associates, Inc., 1999.

Strahl, Rick. *Internet Applications with Visual FoxPro 6.0*. Hentzenwerke Publishing, 1999.

### White Papers

Strahl, Rick. *Using VFP COM Objects with Active Server Pages*. West-Wind Technologies ([www.west-wind.com](http://www.west-wind.com)), 2000.

Strahl, Rick. *Building distributed Web Applications with Visual FoxPro*. West-Wind Technologies ([www.west-wind.com](http://www.west-wind.com)), 2000.

Strahl, Rick. *Internet enabling Visual FoxPro applications*. West-Wind Technologies ([www.west-wind.com](http://www.west-wind.com)), 1999.

## Articles

Bromberg, Peter. *Publish your home IP address to your favorite web server*. Egghead Café ([www.eggheadcafe.com](http://www.eggheadcafe.com)), 2001.

Esposito, Dino. *Exchanging Data Over the Internet using XML*. MSDN Magazine, April 2000.

Hentzen, Whil. *Your First Database Web Application, Part 3*. FoxTalk, March 2001.

Hentzen, Whil. *Your First Database Web Application, Part 2*. FoxTalk, December 2000.

Hentzen, Whil. *Your First Database Web Application*. FoxTalk, March 1999.

Onisick, Stephan. *Saving Session Variables for future use*. Universal Thread Magazine ([www.universalthread.com](http://www.universalthread.com)), July 2001.

Strahl, Rick. *Active Server Pages and Object Messaging*. FoxPro Advisor. September 1998.