Chapter 4 Your First Web Connection Application

Once you've got the Web Connection demo running, you're probably all anxious to do your own thing. In this chapter, I'll show you how to create your first Web Connection application, how to run and test it, how to start modifying code, and, gasp, even how to access data!

I'm going to start you out on the right foot from the very beginning. When you create a project using the Web Connection wizard, Web Connection will place the project in the WCONNECT directory—along with all of the Web Connection framework code—and your project very definitely doesn't belong there.

As noted in the introduction to this book, there are many possible development environment architectures, and I'm not purporting to claim that the approach I describe in this chapter is the only one, nor the best one. But it's straightforward and, for lack of a common standard, one that you can use as you get started. As you progress through this book, you may find techniques and styles described later on by Harold and Randy to be more to your liking. Feel free to mix and match as you desire.

This chapter's sample application

In this chapter, I'm going to show you how to create your first Web Connection application. I'll start with a variation of "Hello World," just so you can follow the process involved in creating, running, and testing all of the pieces. Then I'll show you how to reference static files in your application, such as simple HTML pages and image files.

Next, I'll create some methods that allow the application to talk to data. In order to keep things simple, I'll use a single table of companies and products that are categorized by industry category and type of product. Using a search screen that allows the user to query this table by industry category and product type, I'll show you how to get input from the user, query the database, and return the results of that query. Once that's working, I'll discuss how to provide simple maintenance functions—add, edit and delete—to the query functionality just built.

Finally, I'll move the application to a live server, just to show you the pieces involved. Setting up the live server (both hardware and software) will be discussed in complete detail in Chapter 12, "A Web Connection Application from Start to Finish," but this will give you a running start.

A review of your directory structure

I'm going to assume that you've already gone through the drill in Chapter 3, "Installing, Configuring, and Testing Web Connection," so VFP and Web Connection are installed and running. However, just to make sure, I'll review what your directory structure should look like. For the sake of this discussion, I'll assume that all of your development is being done on drive E, and that the Web Connection framework files are in E:\WCONNECT. (That's because the machine I'm using while writing this chapter has a CD-ROM drive D.) If you have a one-drive machine, just replace "E" with "C" in the following discussion.

First, you'll need a directory in which to place your new Web Connection project. If this is being done for a customer, you could create a customer directory on drive E, and then a subdirectory underneath that customer's directory for this project. If you're just goofing around, why don't you create a directory called, say, "WCAPPS," and underneath it, create a directory for your first project, say, WR04. That stands for WebRAD, Chapter 4.

Next, you'll need a directory where you'll put the Web site files for this project. To make your life easier, how about E:\INETPUB\WWWROOT\WR04?

You'll also need a directory for those all-important temp files. I suggest E:\WC_MSG\WR04.

Finally, you'll need a directory where you'll put your data files for this project. Again, for sake of argument, I'll use E:\WSDB\WR04 (the "WSDB" stands for "Web Site Databases"). Whatever you do, *don't* put your data underneath the INETPUB\WWWROOT directory, because on your live server, that directory will be accessible by anyone who can get to your Web site.

Creating your project

The first step in creating your new Web Connection project is to start VFP and load Web Connection. I know, you thought I was going to say, "Take out a piece of paper and do your analysis and design." Well, for the time being, I'll assume that you've already done that. I mean, this is a simple one-table system.

So, once you've loaded VFP, changed to the \WCONNECT directory, and run WCSTART, you should have the Web Connection menu pad added to the end of your VFP system menu, as shown in **Figure 1**.

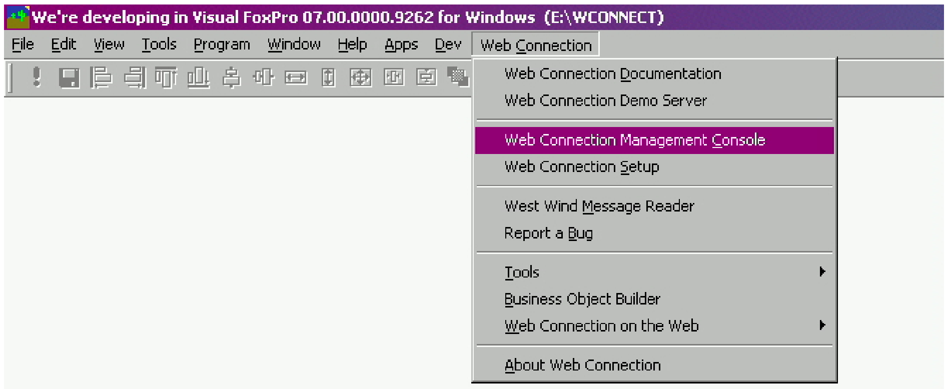

We're developing in Visual FoxPro 07.00.0000.9262 for Windows (E:\WCONNECT)
File Edit View Tools Program Window Help Apps Dev Web Connection
Web Connection Documentation
Web Connection Demo Server
Web Connection Management Console
Web Connection Setup
West Wind Message Reader
Report a Bug
Tools
Business Object Builder
Web Connection on the Web
About Web Connection

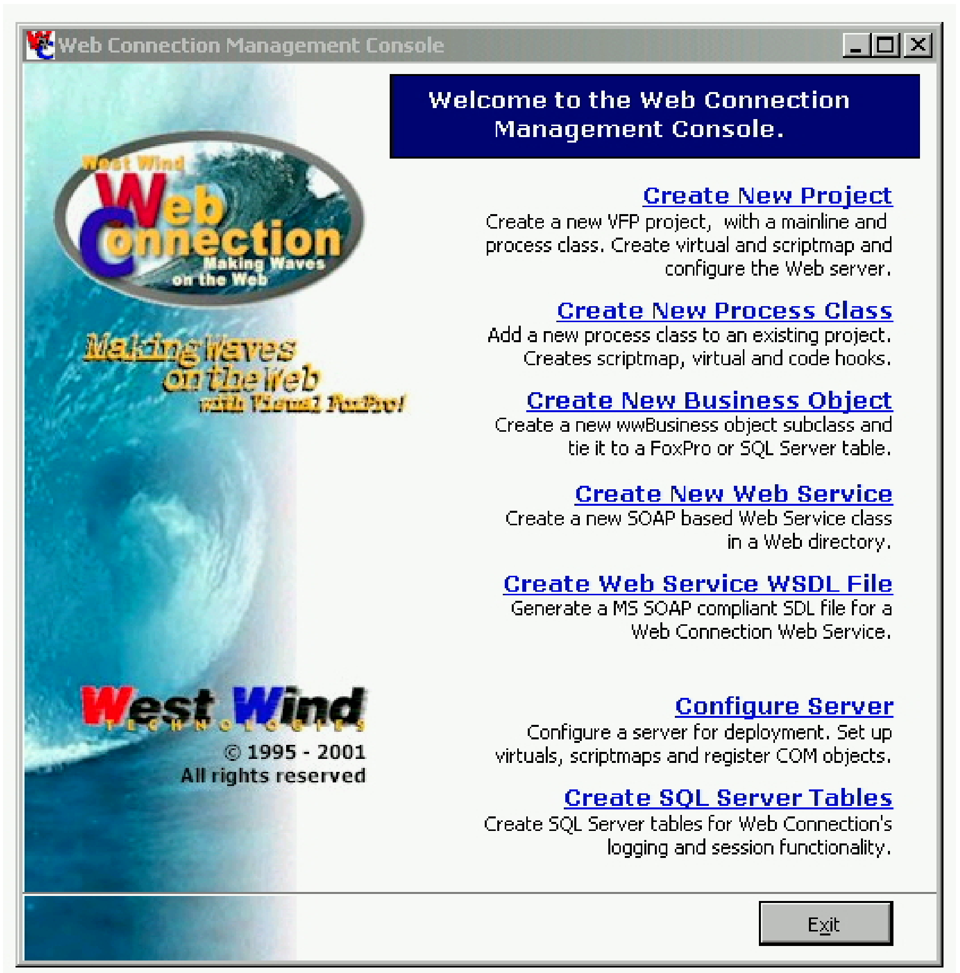
***Figure 1**. Running WCSTART adds a Web Connection pad to the Visual FoxPro system menu.*

You may want to double check that you're set up properly. Make sure that the current VFP directory is \WCONNECT and that the path (using SET('path')) is:

```
\wconnect
\wconnect\classes
\wconnect\tools
\wconnect\wwipstuff_samples
.\classes
..\
.\console
```

Some of our sharp-eyed readers have noted that they put this in a WAIT WINDOW NOWAIT box as part of the setup code—making it easier to notice a mistake due to a setup routine that went astray.

Select the Web Connection Management Console menu option, and you'll be greeted with the main Web Connection Management Console dialog, as shown in **Figure 2**.


Web Connection Management Console
West Wind
Web Connection
Making Waves on the Web
Making Waves on the Web with Visual FoxPro!
West Wind Technologies
Exit

Welcome to the Web Connection Management Console.

Create New Project
Create a new VFP project, with a mainline and process class. Create virtual and scriptmap and configure the Web server.

Create New Process Class
Add a new process class to an existing project. Creates scriptmap, virtual and code hooks.

Create New Business Object
Create a new wwBusiness object subclass and tie it to a FoxPro or SQL Server table.

Create New Web Service
Create a new SOAP based Web Service class in a Web directory.

Create Web Service WSDL File
Generate a MS SOAP compliant SDL file for a Web Connection Web Service.

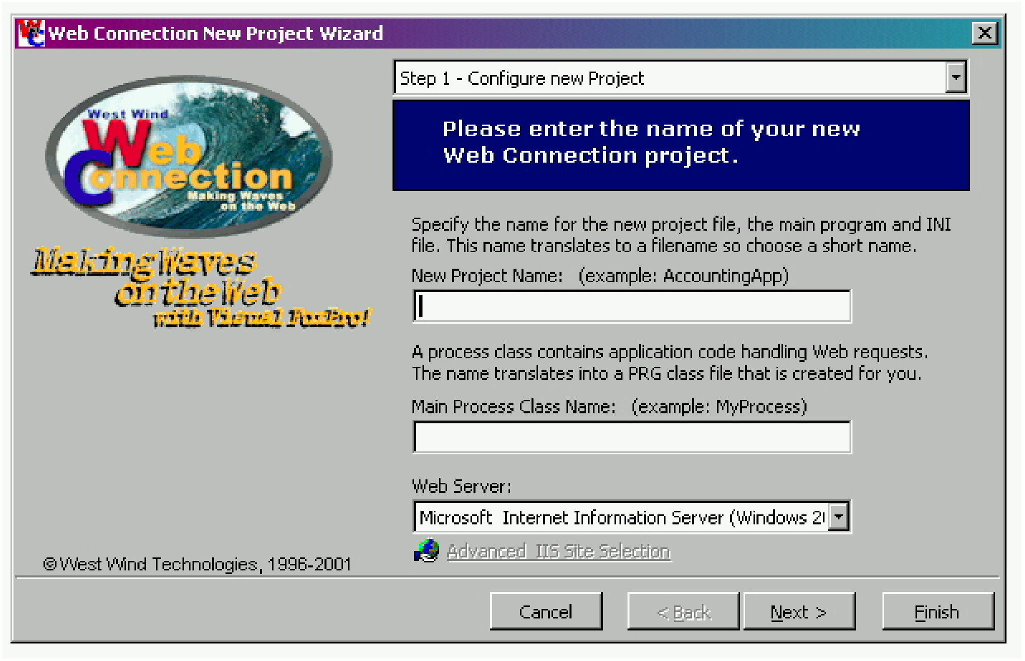
Configure Server
Configure a server for deployment. Set up virtuals, scriptmaps and register COM objects.

Create SQL Server Tables
Create SQL Server tables for Web Connection's logging and session functionality.

© 1995 - 2001
All rights reserved

***Figure 2**. Launch the Web Connection Management Console to create new projects.*

Select the Create New Project hyperlink in the Web Connection Management Console dialog, and you'll get the first step in the New Project Wizard, as shown in **Figure 3**. Note that the Web Connection Help file also loads at this point—depending on the position of the moons as well as the karma of your development machine, the Help file may overlap the wizard dialog. Just Alt-Tab to get back to the wizard if needed.


Web Connection New Project Wizard
Step 1 - Configure new Project
West Wind Web Connection Making Waves on the Web
Making Waves on the Web with Visual FoxPro!
©West Wind Technologies, 1996-2001
Advanced IIS Site Selection
Cancel
< Back
Next >
Finish

Please enter the name of your new Web Connection project.

Specify the name for the new project file, the main program and INI file. This name translates to a filename so choose a short name.

New Project Name: (example: AccountingApp)

A process class contains application code handling Web requests. The name translates into a PRG class file that is created for you.

Main Process Class Name: (example: MyProcess)

Web Server: Microsoft Internet Information Server (Windows 2(

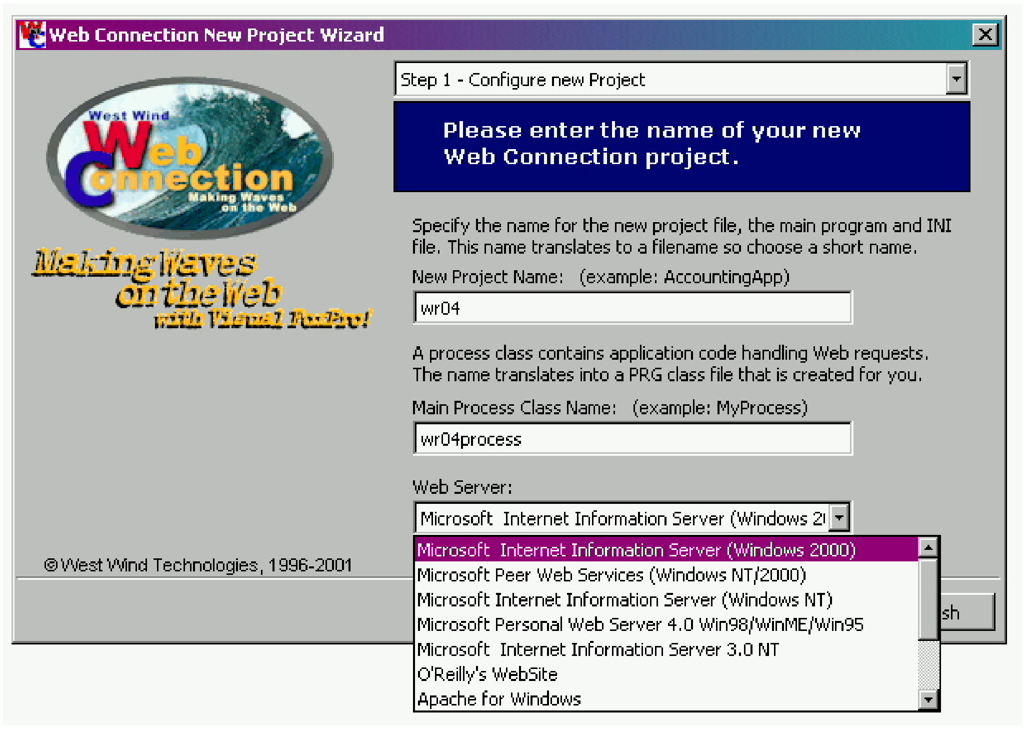
Figure 3*. The first step in creating a new Web Connection project is to name it.*

Enter the name for your project (this will become the name of your VFP project, so don't use a really long name), and then the name for your main process class (for the time being, just add the name of your project in front of the word "Process"). In future projects, you can change your mind about your main process class naming conventions.

Pick the Web server you're running on your development box (hopefully you took my advice from Chapter 3 and are running NT or 2000, so you can just grab IIS and be done with it)—see **Figure 4**.

Click Next to go to Step 2 in the wizard. There are two pieces to this step. The first is to select the name of a virtual directory.

A virtual directory provides the Web server with a mapping from a realm name like www.yoursite.com to a specific physical location where the server can find files. This physical location can be either a subdirectory under the Web root with the same name, or it can be any other physical location that you desire.


Web Connection New Project Wizard

Step 1 - Configure new Project

Please enter the name of your new Web Connection project.

Specify the name for the new project file, the main program and INI file. This name translates to a filename so choose a short name.

New Project Name: (example: AccountingApp)

wr04

A process class contains application code handling Web requests. The name translates into a PRG class file that is created for you.

Main Process Class Name: (example: MyProcess)

wr04process

Web Server:

Microsoft Internet Information Server (Windows 2

Microsoft Internet Information Server (Windows 2000)
Microsoft Peer Web Services (Windows NT/2000)
Microsoft Internet Information Server (Windows NT)
Microsoft Personal Web Server 4.0 Win98/WinME/Win95
Microsoft Internet Information Server 3.0 NT
O'Reilly's WebSite
Apache for Windows

©West Wind Technologies, 1996-2001

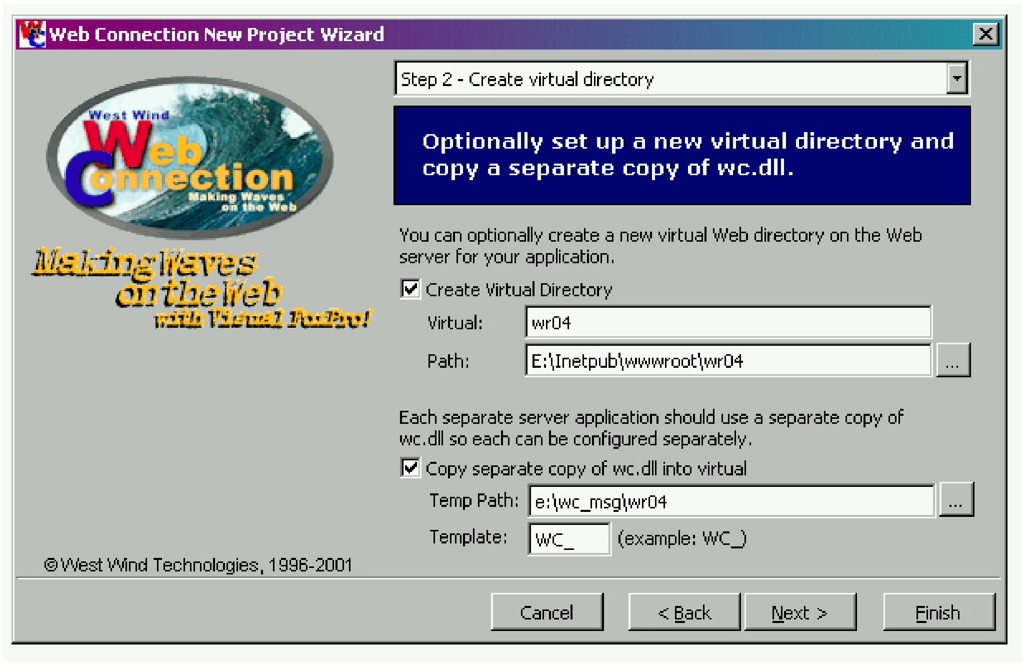
Figure 4. *The Configure New Project dialog after a new project has been named.*

As a result, then, the user will be entering a domain name and a virtual name into their browser, and the Web server will be translating that information to a physical location on a hard disk.

It looks like creating a virtual directory is optional, but I suggest that you choose to do so. You'll want Web Connection to act as if there is a virtual directory whether you use a different name for the physical location or not. You'll be setting properties for that virtual that are different from the properties of the default Web site.

Check the Create Virtual Directory check box, and then enter the name for your virtual directory, and point to the actual path that it will represent. Note that Web Connection will try to guess what the path will be, by appending the name of your project to the location of the Web server home directory, as shown in **Figure 5**. It's perfectly fine for the virtual name and the subdirectory name to be the same, and, in fact, doing so can be less confusing.

Note that by checking the Create Virtual Directory option, the Web Connection wizard will actually have IIS create a virtual directory. You could do this yourself, of course, by going into IIS and creating a virtual directory, but by letting Web Connection do it, Web Connection also sets other things properly—things like script maps and setting up access permissions properly. We'll cover this stuff later in Chapter 12, but it's good to know that for now, it's being taken care of for you.


Web Connection New Project Wizard
Step 2 - Create virtual directory
West Wind Web Connection Making Waves on the Web
Making Waves on the Web with Visual FoxPro!
©West Wind Technologies, 1996-2001
Cancel
< Back
Next >
Finish

Optionally set up a new virtual directory and copy a separate copy of wc.dll.

You can optionally create a new virtual Web directory on the Web server for your application.

[✓] Create Virtual Directory

Virtual: wr04

Path: E:\Inetpub\wwwroot\wr04

Each separate server application should use a separate copy of wc.dll so each can be configured separately.

[✓] Copy separate copy of wc.dll into virtual

Temp Path: e:\wc_msg\wr04

Template: WC_ (example: WC_)

***Figure 5**. Creating a virtual directory and copying the WC.DLL.*

The dialog makes the second part of Step 2 in the Web Connection Setup Wizard confusing. The dialog appears to be asking whether you want to copy a separate copy of the WC.DLL (and create a Web Connection INI file to go with it) into the virtual or not. However, what's not clear is that this task must be performed—if you don't have the wizard do it for you, you'll have to do it yourself manually. The dialog is asking whether you want Web Connection to do it for you, or if you want to do it yourself later. (Blech!) There is not an alternative of not making a separate copy one way or another. Let Web Connection do it!

Thus, you'll want to check the "Copy separate copy" check box.

Next, change the Temp Path value to match the folder you create for temporary files for this project. In this example, that temp path would be E:\WC_MSG\WR04.

The template is the extension used for files placed in the temp file directory. If you place temp files for each Web Connection application into their own temp directory, then you can leave this as "WC_"—if, for some reason, you need to have multiple applications put their temp files all in the same directory, then you'll want unique template names for each application, so that each application creates files with different extensions.

Click Next to go to Step 3 in the wizard, where you can define a script map. Like the temp files, scripts maps are powerful and important, but require a bit of explanation. Please take the time to read through this!

Remember that a Web server can process three types of files:

- Those that are simply sent back to the user's browser, such as straight HTML
- Those that are executed (such as EXEs and DLLs)

- Those that are pre-processed, such as ASP files

To understand what a script map is, let's look at how a URL is processed at the server. When you enter a URL with a file name that has an extension of, say, HTML or GIF, the server knows that it's to find the file and deliver a copy back over HTTP to the user.

Some files, however, well, you don't want to send them to the user. Instead, you want them to be processed on the server. EXEs and DLLs, for example, right? Servers can be configured to automatically run certain types of files instead of delivering the file back to the user. That's what the "Execute" setting for a virtual directory means—that EXEs and DLLs, for example, are to be executed on the server instead of delivering the EXE or DLL file back to the user like an HTML file would be.

For reference, go back to Chapter 3, and look at Figures 5 (for NT) and 13 (for Windows 2000). You'll see where you set the Scripts/Execute permissions for a directory on the Web server.

Now, let's go one step further. Some files aren't just delivered back to the user, nor are they "executed" on the server. ASP files, for example, fall into this third category. When a Web server gets a URL that includes a file with an ASP extension, like this:

```
http://www.somewebsite.com/SomeFile.ASP
```

the Web server knows that this file should be handled in a special way. How does it know this? There's a lookup table in IIS, associated with the Web server that has the "ASP" extension in it. (See the "Setting up script mapping" section of Chapter 12 for details about how to set up script maps directly in IIS.)

The Web server sees the "ASP" extension and automatically looks for a DLL called ASP.DLL. (The server knows this because the entry in that lookup table I just mentioned matches the ASP.DLL next to the extension "ASP.") The ASP file is processed by the ASP.DLL, and the results are returned by the Web server to the user.

So far, so good, eh?

Now, what if you could define your own extension, similar to "ASP," and then attach it to some DLL? Groovy, eh? In fact, it'd be even groovier if you could use that lookup table I just mentioned to do so, right?

A Web Connection script map is just this—a definition of an extension for a file that is processed by the WC.DLL. (So, when you think about it, a script map should actually be called a "script extension," doncha think?) This definition is placed in the lookup table automatically from Web Connection by this New Project Wizard (see, I told you we were just taking a short detour). And, of course, the Web Connection framework knows how to interpret a URL that has a predefined script map.

So, instead of entering a string into your browser like this:

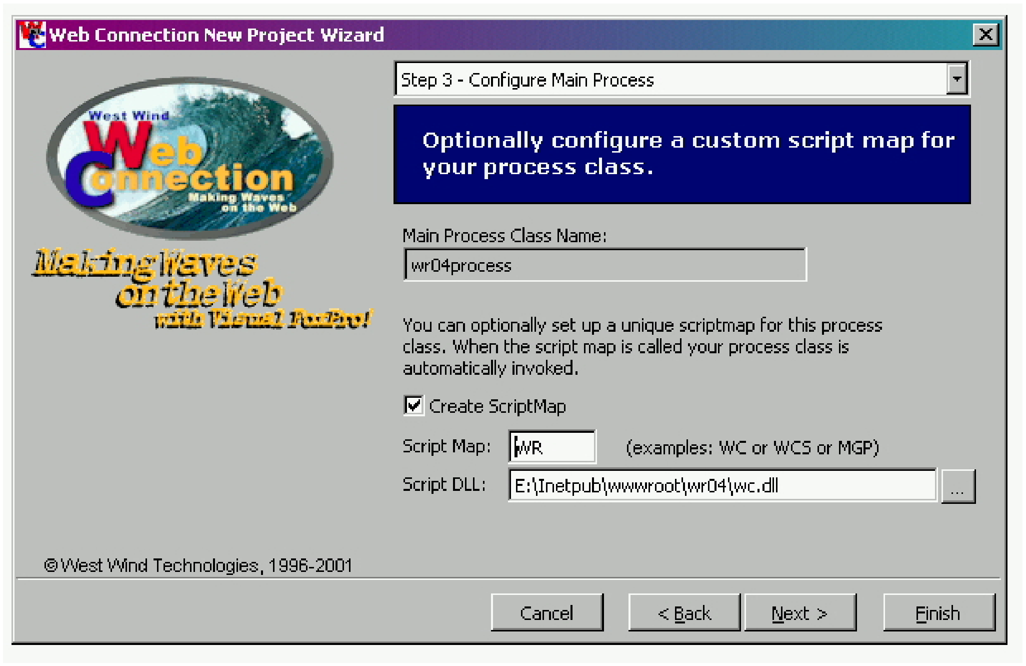
```
http://www.somesite.com/wc.dll?SomeClass~SomeMethod
```

you could enter a script map, XXX (remember, you could think of "XXX" as a "script extension" if that makes it easier), that would be mapped to the WC.DLL in the IIS lookup table. Then, you could enter this string instead:

```
http://www.somesite.com/SomeMethod.XXX
```

This provides you with a couple of benefits that I'll talk about in depth later. The bottom line, for now, is that Step 3 in the New Project Wizard is where you determine whether or not you want to define a script map, and, if so, what it is. See **Figure 6**, where I define a script map of "WR," and link it to the WC.DLL in the WR04 directory.

There's actually a lot to using script maps the first time around. I discuss the topic in more depth in the section "Using script maps instead of calls to WC.DLL" later in this chapter, and Harold covers it in more detail in "Setting up script mapping" in Chapter 12.


Web Connection New Project Wizard
Step 3 - Configure Main Process
West Wind Web Connection Making Waves on the Web
Making Waves on the Web with Visual FoxPro!
Cancel
< Back
Next >
Finish

Optionally configure a custom script map for your process class.

Main Process Class Name: wr04process

You can optionally set up a unique scriptmap for this process class. When the script map is called your process class is automatically invoked.

[✓] Create ScriptMap

Script Map: WR (examples: WC or WCS or MGP)

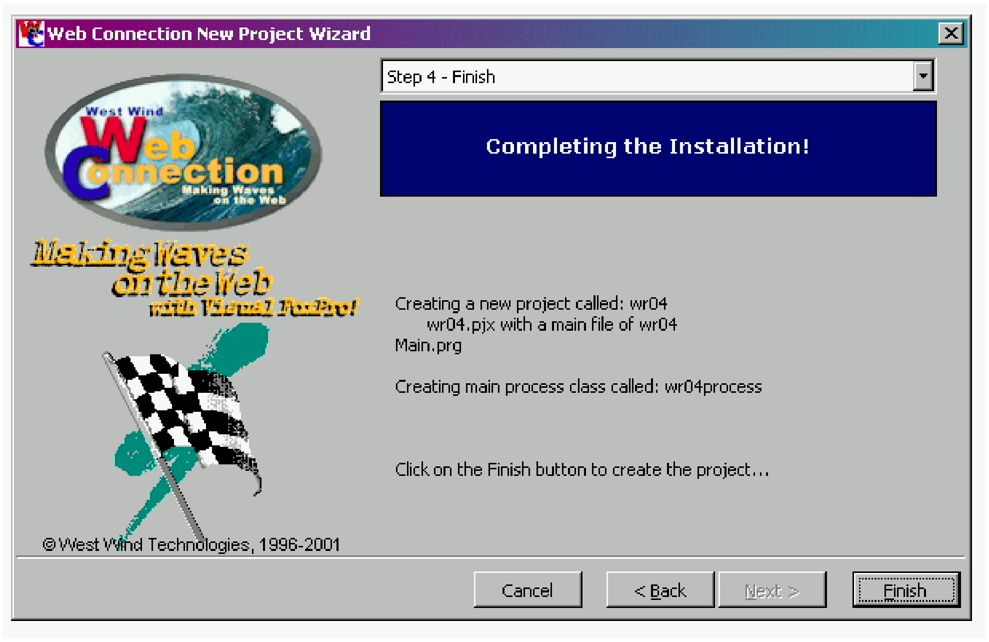
Script DLL: E:\Inetpub\wwwroot\wr04\wc.dll

©West Wind Technologies, 1996-2001

***Figure 6**. Naming the script map and identifying the script map DLL.*

Okay, back to the wizard. Click the Next button, and then, finally, create the project by clicking the Finish button in the Step 4 dialog, as shown in **Figure 7**. Ever wonder what the difference is between clicking the Finish button in the Step 3 dialog in Figure 6, and clicking the Finish button in the Step 4 dialog in Figure 7?

You could shortcut Step 4 by clicking on Finish in Step 3—it's just that the final page in Step 4 shows you what all of the selections you've made are in a summarized screen. If you realized that you made a mistake, you could go back via the Back button and change something. You could certainly click Finish earlier if it is offered.


Web Connection New Project Wizard
Step 4 - Finish
Completing the Installation!
West Wind Web Connection Making Waves on the Web
Making Waves on the Web with Visual FoxPro!
©West Wind Technologies, 1996-2001
Cancel
< Back
Next >
Finish

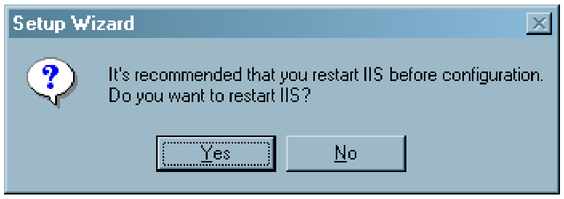
Creating a new project called: wr04
wr04.pjx with a main file of wr04
Main.prg

Creating main process class called: wr04process

Click on the Finish button to create the project...

***Figure 7**. The Finish dialog displays the settings you've chosen.*

As of Version 4.0, Web Connection will then offer to stop and restart IIS for you, as shown in **Figure 8**. If you select Yes, a DOS box will display stopping and starting command lines. It may take a minute or two for the operation to complete on your machine.



It's recommended that you restart IIS before configuration.
Do you want to restart IIS?

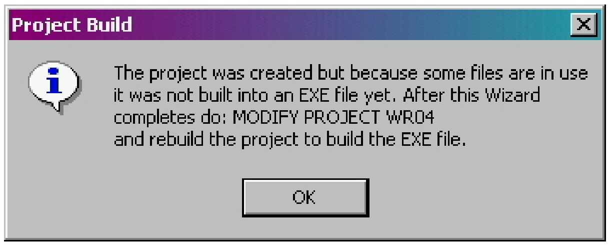
***Figure 8**. Web Connection offers you the opportunity to automatically stop and restart IIS after clicking Finish in the wizard.*

The Web Connection wizard will now do its thing, creating a project file, adding files to it, and copying files as you directed with the wizard. Eventually, the wizard will attempt to build an EXE from the project, as the WAIT WINDOW in **Figure 9** shows, but will fail because some files are in use.

Compiling project: into EXE file...

***Figure 9**. The WAIT WINDOW indicates that the New Project Wizard is attempting to create a new EXE file.*

You'll get the error message in **Figure 10**. Click OK and move on. However, do *not* follow the instructions in the Project Build error dialog! At this point, the project has been created, but it's still residing in the \WCONNECT directory. The next step is to move the project where it belongs *before* creating your first EXE.

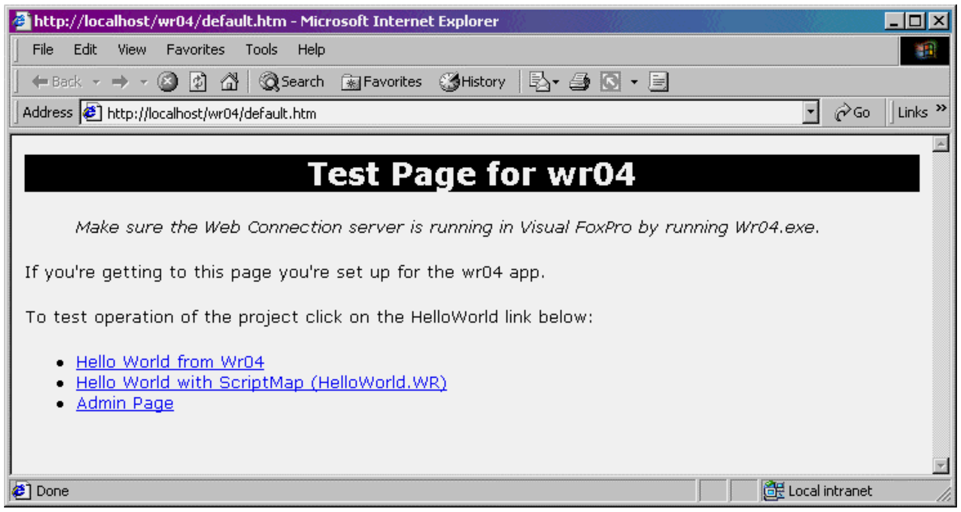

Project Build

The project was created but because some files are in use it was not built into an EXE file yet. After this Wizard completes do: MODIFY PROJECT WR04
and rebuild the project to build the EXE file.

***Figure 10**. Ignore this error message during the New Project Wizard startup. You'll move your project before creating your first EXE anyway.*

Unfortunately, even if the project build fails, Web Connection will launch a test page for the new project, as shown in **Figure 11**. I say "unfortunately" because if you click on any of the hyperlinks in the test page, the hit will fail. The reason they will fail is because the Web Connection server isn't running—after all, the EXE didn't even get built in the previous step, so it certainly couldn't be running!

You might want to keep the browser open to this page, and after you move your project and build the EXE, you'll be ready to test, instead of having to open up your browser again and enter the test URL.


http://localhost/wr04/default.htm - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost/wr04/default.htm
Go Links

Test Page for wr04

Make sure the Web Connection server is running in Visual FoxPro by running Wr04.exe.

If you're getting to this page you're set up for the wr04 app.

To test operation of the project click on the HelloWorld link below:

- Hello World from Wr04
- Hello World with ScriptMap (HelloWorld.WR)
- Admin Page

***Figure 11**. The test page for your new project allows you to test the server and the script map.*

Before I get to moving the project, though, take a look at the contents of the address bar in the browser. You'll see that the link to this test page doesn't include the great big long URL to

E:\INETPUB\WWWROOT\WR04—just simply the call to the Web server (localhost or 127.0.0.1), the virtual directory (wr04), and the name of the test page (default.htm). If you wanted to, you could have set up your virtual directory to extend way deep into the bowels of the file system directory, pointing to a subdirectory buried in layers of subdirectories, but still reference a test page simply via the virtual directory name.

Now, after you've clicked OK in the error dialog in Figure 10, you'll be returned to the Visual FoxPro IDE, and your Project Manager window and the Web Connection Management Console window will both still be open. I've found it easiest to close the WCMC window, and then get out of Fox completely, in order to make sure there aren't any loose files open.

Moving your project

If you open up Windows Explorer, you'll see a bunch of new files in the E:\WCONNECT directory (or whatever you named the directory where you installed the Web Connection framework), most of which start with the name of your project, as shown in **Listing 1**.

Listing 1. *The contents of the Web Connect directory after creating project WR04.*

```
 Volume in drive E is DEV
 Volume Serial Number is 9D58-C911

 Directory of E:\wconnect

01/11/25  04:44p        <DIR>          .
01/11/25  04:44p        <DIR>          ..
01/11/25  04:40p                 2,909 bld_Wr04.prg
01/11/25  03:29p        <DIR>          classes
01/10/17  11:41p                   210 config.fpw
01/11/22  04:21p        <DIR>          console
01/11/03  11:41a               670,203 console.EXE
01/11/25  04:44p                     0 dirl.txt
01/11/22  04:21p        <DIR>          FoxCentral
01/11/22  04:36p        <DIR>          html
01/11/22  04:21p        <DIR>          scripts
01/10/18  09:14p                19,892 setup.EXE
01/10/18  09:14p                20,404 setup70.EXE
01/11/22  04:21p        <DIR>          SoapSamples
01/11/22  04:21p        <DIR>          templates
01/11/22  04:21p        <DIR>          tools
01/11/22  05:23p                 1,348 wcdemo.ini
01/11/22  04:43p                 6,585 wcdemomain.FXP
01/11/22  04:36p                14,914 wcDemoMain.prg
01/11/03  11:50a             3,672,992 wconnect.chm
01/10/17  10:18p                 8,804 wconnect.h
98/07/27  02:19a                 1,078 wconnect.ico
00/04/14  12:20p                44,404 wconnect.wav
01/10/20  03:46p                   559 wconnect_override.h
01/11/22  04:43p                 3,070 wcstart.FXP
01/10/10  03:40a                 3,967 wcstart.PRG
01/11/25  04:40p                   217 Wr04.ini
01/11/25  04:42p               143,088 Wr04.PJT
01/11/25  04:42p                 3,143 Wr04.PJX
01/11/25  04:40p                11,842 Wr04Main.prg
01/11/25  04:40p                 2,111 wr04process.prg
01/04/16  01:07a                 3,329 wwbanners.DBF
```

```
01/11/22  04:44p        <DIR>          wwdemo
01/11/22  04:22p        <DIR>          wwDevRegistry
01/10/17  10:07p                75,776 wwipstuff.dll
01/11/22  04:22p        <DIR>          wwIPStuff_samples
01/11/22  04:22p        <DIR>          wwreader
01/11/22  05:23p                   661 wwRequestLog.DBF
01/11/22  05:23p                 1,280 wwRequestLog.FPT
01/11/22  04:27p        <DIR>          wwthreads
01/10/18  09:24p                 9,379 _readme.htm
              26 File(s)      4,722,165 bytes
              15 Dir(s)   1,201,762,304 bytes free
```

The one file that has your project name in it, but doesn't begin with the name of your project is at the beginning of this listing: BLD_WR04.PRG.

If you still haven't created the project directory that you're going to keep your project files in, now is the time to create it, as well as the SOURCE directory underneath.

Next, move (don't copy) all of your project's files from the \WCONNECT directory to your project directory. (Don't forget to close those files in VFP first! That's why I just exit VFP completely.) Then move the XXProcess.PRG (where XX is the name of your project) files to the source directory. Do not move XXMain.PRG to your source directory! If you do, and try to run the PRG, Web Connection will look for your application's INI file in the source directory. When it doesn't find it, you'll end up with unexpected results that are very hard to track down. (The error message "WWC_SERVER is not found" is one symptom.) You should end up with something like this (using "WR04" as the name of the project):

```
E:
  \WCAPPS
         \WR04
         bld_wr04.prg
         wr04.ini
         wr04.pjx
         wr04.pjt
         wr04main.prg
            \SOURCE
            wr04process.prg
```

Get back into Fox, change to that project's directory by selecting the project's menu option (set up via the GOFOXGO tool described in Chapter 3), open the project, click on the Rebuild project option button, check the Recompile All Files check box, and all of the files should be brought into the project if they weren't already. Remember that the GOFOXGO routine I described in the previous chapter includes "SOURCE" as part of the path when changing to the project's directory, so you may have to fiddle around a bit if you don't do it that way.

If you don't use the GOFOXGO mechanism, you'll find that the project will complain about not being able to find the Web Connection framework files, so you'll need a different way to make sure your paths are set up properly. One way is to manually set the default directory and path. After loading Visual FoxPro, you could type the following into your Command Window:

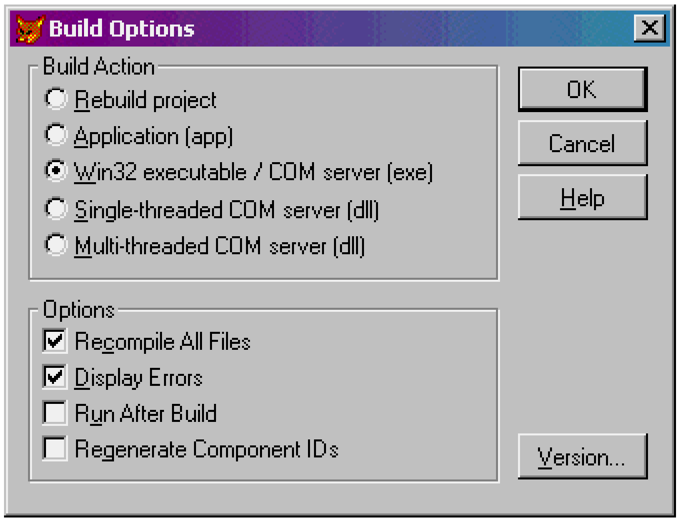
```
  set defa to <name of your application directory>
  set path to source; \wconnect; \wconnect\classes\; \wconnect\tools\;
\wconnect\wwipstuff_samples\
```

Another way, espoused by our technical editor, is to create a CONFIG.FPW file in the project's directory that specifies the path, and then create a desktop shortcut that opens VFP in that directory.

Testing your project

It's white-knuckle time now. Time to build your EXE and test it.

First, rebuild your project again, and be sure to select the Win32 executable/COM server option group in the Build Options dialog, as shown in **Figure 12**.



Build Action
- ( ) Rebuild project
- ( ) Application (app)
- (●) Win32 executable / COM server (exe)
- ( ) Single-threaded COM server (dll)
- ( ) Multi-threaded COM server (dll)

Options
- [✓] Recompile All Files
- [✓] Display Errors
- [ ] Run After Build
- [ ] Regenerate Component IDs

***Figure 12**. Be sure to check the Recompile All Files check box when rebuilding your EXE.*

Next, run the EXE by double-clicking on it in Explorer. (Technically, you could also DO WR04.EXE in the Command Window in VFP.) You'll get the server window as shown in **Figure 13**. Running the executable—what we know as "running a program" in LAN and client/server apps—is referred to as "running the server." At some point, you may be asked, "Is the Web Connection server running?"—this question means, "Have you run the EXE so that the server window is displayed?" like in Figure 13. (There are cases where you won't have a visible instance of the server, but we'll cover that in Chapters 12 and 14.)

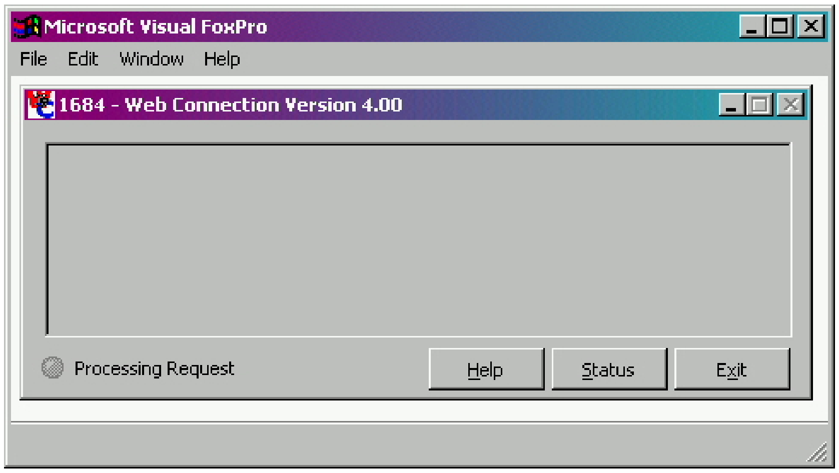
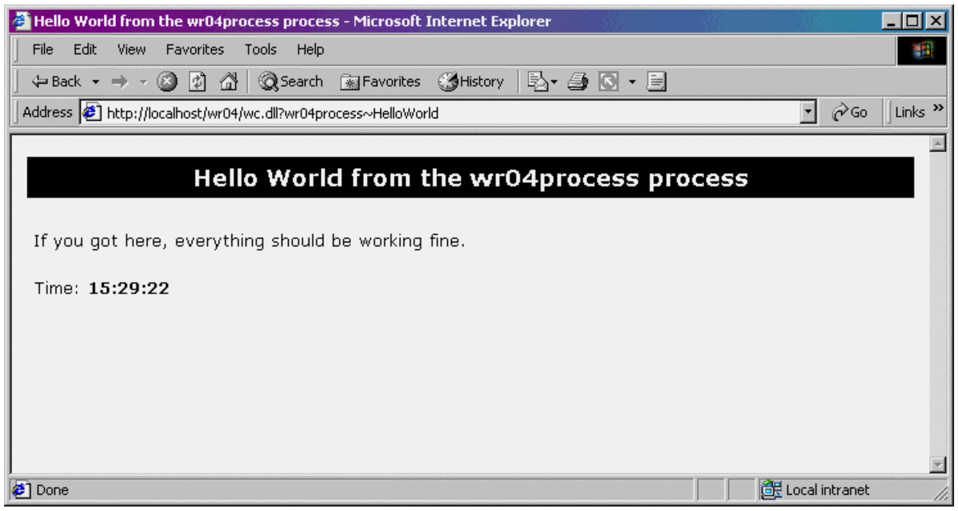


***Figure 13**. The Visual FoxPro Web Connection server window.*

So far, so good. Now it's time to test whether it's really working. Open up your browser, and enter:

```
http://localhost/wr04/default.htm
```

or whatever the name of your virtual directory is. (What you are doing here is displaying the default.htm file that's located in E:\INETPUB\WWWROOT\WR04.) You'll get the same browser window as shown in Figure 11. Click on the first hyperlink, and you should get a new page in your browser, as shown in **Figure 14**.


Hello World from the wr04process process - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost/wr04/wc.dll?wr04process~HelloWorld
Go Links
Done
Local intranet

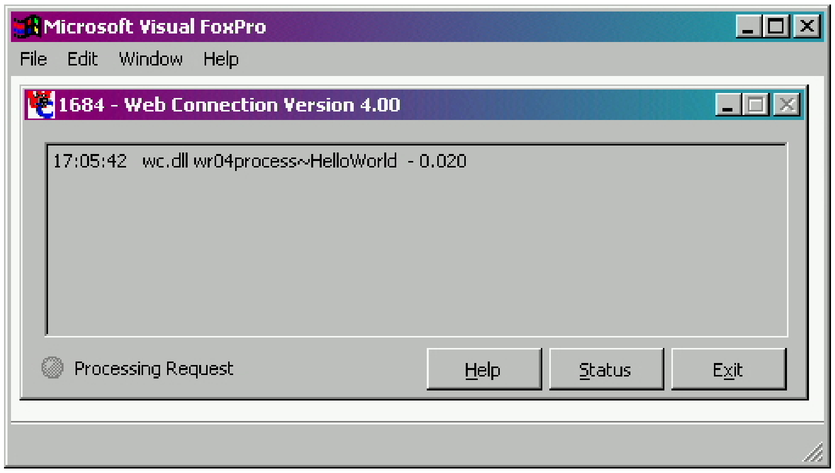
Hello World from the wr04process process

If you got here, everything should be working fine.

Time: **15:29:22**

***Figure 14**. Your first successful Web Connection request from your own project.*

Look over to your server window, as shown in **Figure 15**, and you should see a line of text describing what the server just did—process a request and make a call to the Hello World method of the WR04Process class. Where did this Hello World method come from? It's always generated as part of the Process class that is created when you create a new project from the Web Connection Management Console.



```
17:05:42   wc.dll wr04process~HelloWorld  - 0.020
```

***Figure 15**. The Visual FoxPro Web Connection server window displays the results of hits to the server.*

After you've made your first successful hit, you can click the Status button in the Web Connection server window and see that the WC.INI button is finally enabled.

Now it's time to start making some modifications. Go ahead and click the Exit button in the server window.

Optimizing your development environment—just a bit

Before we start cranking out oodles of Fox code, though, it's worth spending a few moments to make your life easier. You're going to find that you will be doing a few things over and over again—running the server, and making page requests in your browser.

You'll notice that for this first test, I had you build an EXE and run it from outside of Visual FoxPro. Do you have to do that each time you test? Yeah, it's not a big deal to build an EXE, switch over to Explorer, find the EXE, and double-click on it, or to open the drop-down in the browser, find the link you want, and execute it. But it's still more work than you really want… What if we could make both of those functions single clicks? Or what if we didn't even have to build an EXE at all?

As the saying goes, the idea is to keep the iteration time shorter than your attention span. Personally, I need all the help I can get. I'll show you how to speed up time if you want to build an EXE each time, and then how to avoid building the EXE at all, until you're ready to deploy to your live Web server.

EXE shortcut

The first thing to do is create a shortcut to the EXE on your taskbar.

Open Explorer, click on your EXE file, and drag it to the Quick Launch toolbar in the taskbar (the area immediately to the right of the Start button), and you'll be all set. Well, almost all set. Once you have created a half-dozen apps, you'll end up with a bunch of EXE shortcuts files that all look alike—they'll all have the same orange Fox head for the icon.

As a result, before you drag the EXE to the taskbar, assign a different icon to the project. If you're short of artistic skills, you can find a bazillion ICO files on drive C using Find in Explorer. Place a copy of an icon you like in your project directory.

Note that this doesn't work with just any old ICO file—the icon has to be a 32x32 icon to be able to attach, and the icon for the EXE won't display in Explorer unless there's also a 16x16, 16-color icon in the ICO file. Unfortunately, there's no way from Explorer to positively tell what type of icon an ICO file is. In the Explorer Search tool, you'll see the size of the various ICO files—1K files don't contain both the 32x32 and 16x16 icons, while it seems that the 2K files usually do. But I've only been able to positively confirm this by opening the icon in a tool like IconEdit32. See the online Help under the topic "Project Tab, Project Information Dialog Box." You can get IconEdit32 from ***www.zdnet.com/pcmag/pctech/content/16/12/ut1612.001.html****, and you can find a shareware version called IconEdit Pro at* ***www.iconedit.com****.*

Then, open the Project Manager (don't have it docked), and select the Project | Project Info menu option from the VFP system menu. Select the Project tab, click on the Attach Icon check box in the lower right, as shown in **Figure 16**, and navigate to the ICO file you want. I grab a copy of the ICO file I want and put it in the project's directory so that I have it for the life of the project.

Next time you build your EXE, this icon file will be attached to the EXE. However, it will still not be attached to the shortcut in the Quick Launch toolbar. Right-click on the shortcut in the Quick Launch toolbar, select Properties, click on Change Icon, and, using the Browse command button, navigate to the ICO file (*not* the EXE file that's provided as the default file for the Current Icon). Click on OK in the Change Icon dialog, then on Apply and OK in the Shortcut Properties dialog, and your new icon will display instead of the orange Fox head on the Quick Launch toolbar.

From now on, you can just click on the EXE shortcut in the Quick Launch toolbar after building the EXE inside VFP. But that's still a lot of work. How about running your Web Connection server from within your Visual FoxPro development environment?

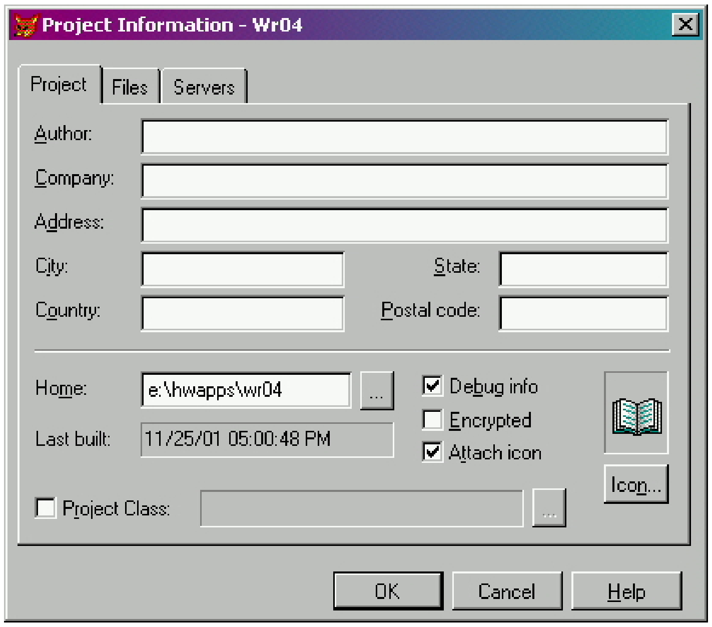


Figure 16*. Use the Project Information dialog to attach an icon to the EXE for easier identification in the taskbar.*

Running your Web Connection server inside VFP

When you're building regular Visual FoxPro applications, you're probably used to testing your programs by issuing a

```
do someprogram
```

command in the VFP Command Window. Wouldn't it be nice if you could do this with your Web Connection applications as well? Well, you can, usually.

Either issue the

```
do wr04main
```

command in the VFP Command Window, or, if you're the type to keep your Project Manager open, click on the wr04main item in the Code tab, and then click the Run button. (I keep the Project Manager open all the time with the wr04main item highlighted, so I just have to keep hitting Run—about as few steps as is possible.)

Note that occasionally Web Connection and Visual FoxPro get confused. You may need to rebuild your project completely, or even get of Visual FoxPro and start it up again in those cases.

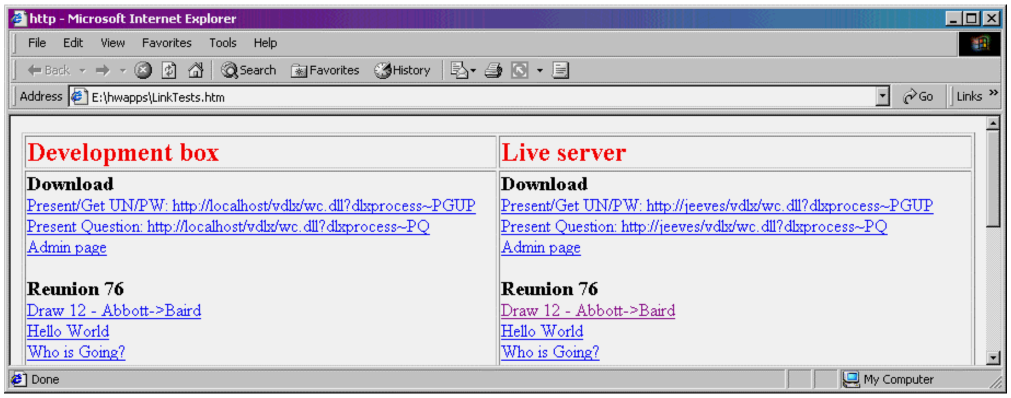
HTML page shortcut

In just about every project you work on, you're going to have multiple calls to Web Connection methods. After a while, the list of URLs in your browser address drop-down can be pretty lengthy—particularly if you use your browser for other things, like surfing the Web. Thus, it can be difficult to find the exact hyperlink you're looking for.

What I've done is create a dummy HTML page and embed all of the URLs I regularly use in it. (These URLs have the full link, including the http://localhost prefix, to make sure that the request is routed through the Web server, not just the operating system's file system.) Then I put a shortcut to that HTML page on my taskbar. Thus, it's just two clicks to get to any URL I need—one to call up the dummy HTML page, and a second one for the real link.

Note that the shortcut to the dummy HTML page is just opening the HTML page in the browser via the file system—not through the Web server—but that's OK, because we don't need that page processed through the Web server. The key is that the links in the dummy HTML page make the correct requests through the Web server.

In fact, I take this idea one step further. I divide the dummy HTML page into two columns—the left column contains links for the development machine, and the right column has matching links that go to the live Web server. That way, I can test the live server from my development box with just a couple clicks. See **Figure 17** for an example.


http - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address E:\hwapps\LinkTests.htm
Go Links
Done
My Computer

Development box	Live server
Download	**Download**
Present/Get UN/PW: http://localhost/vdlx/wc.dll?dlxprocess~PGUP	Present/Get UN/PW: http://jeeves/vdlx/wc.dll?dlxprocess~PGUP
Present Question: http://localhost/vdlx/wc.dll?dlxprocess~PQ	Present Question: http://jeeves/vdlx/wc.dll?dlxprocess~PQ
Admin page	Admin page
Reunion 76	**Reunion 76**
Draw 12 - Abbott->Baird	Draw 12 - Abbott->Baird
Hello World	Hello World
Who is Going?	Who is Going?

Figure 17. *Create a dummy HTML page with links to your development box and your live server for easy access to your Web Connection application's functions.*

Want life to be even better? Make this dummy HTML page your home page in your browser, so you can get to this page with the Home button any time your browser is open.

Another option is to create a new folder under your "Favorites." Call it something like "Development projects." Then you can add the main page for each of your virtual directories to this folder. You can also create another folder for the "Live server" links. The disadvantage to this technique is that you don't see the development and live links side-by-side, as you do on the dummy HTML page.

Help file shortcuts

Hopefully you'll find this book so spellbinding that you'll be referring to it day and night. But it can be inconvenient to keep flipping through it—why not refer to the CHM file that comes along with the source code downloads?

I've found it very useful to put shortcuts to several CHM files on the taskbar. The shortcuts I have on my taskbar open the Hacker's Guide, the Web Connection Help file, and this book.

As our technical editor mentioned, you may find that you can sure get a lot of shortcuts building up that way. Barbara puts her shortcuts into groups, and has the groups on the taskbar.

Opening up and adding code to the project

The previous discussion is all well and good, but you're itching to get your hands dirty, aren't you? Okay, let's open up the project and start writing our own code.

Get back into Fox, switch to your project's directory, and click on the Code tab in the Project Manager, as shown in **Figure 18**. I personally keep the Project Manager docked, and just open the Code tab itself.

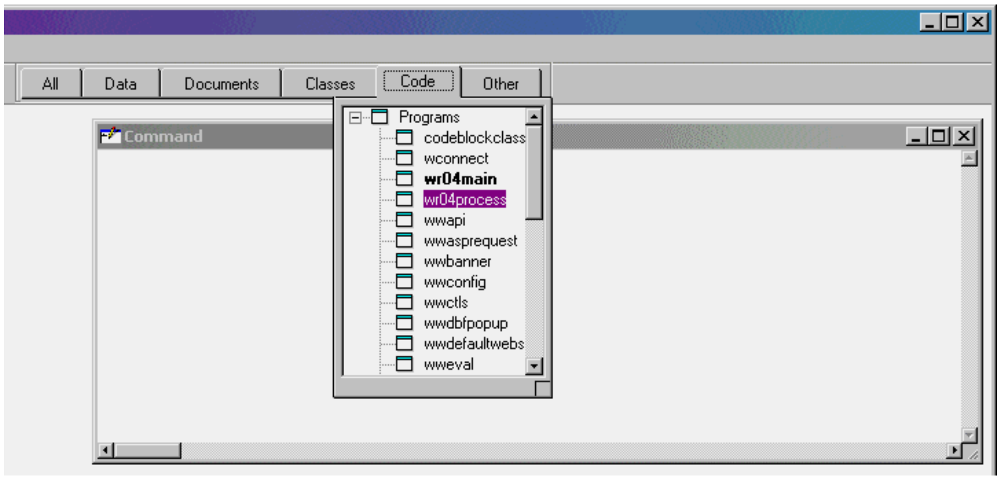


***Figure 18**. Open up the process class program via the Code tab in the Project Manager.*

The action happens in the WR04Process program, so open it up and look for a method called Hello World, as shown in the following listing.

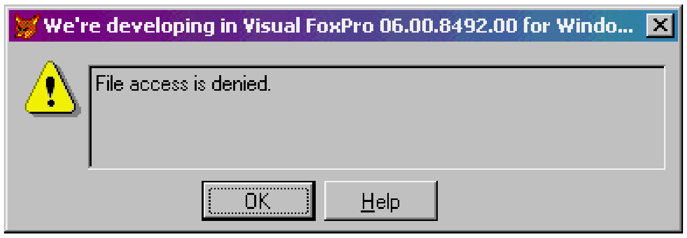
```
*********************************************************************
FUNCTION HelloWorld()
************************

THIS.StandardPage("Hello World from the WR04Process process",;
                  "If you got here, everything should be working fine.<p>" + ;
                  "Time: <b>" + TIME()+ "</b>")
```

```
ENDFUNC
* EOF WR04Process::HelloWorld
```

Now, there are two types of programmers: the cautious ones, and the ones who are now selling insurance. Thus, I won't blame you if you want to take it slowly and make a slight tweak to the existing Hello World method, just to prove to yourself that you can do it. How about the following?

```
THIS.StandardPage("Well, go-oo-oolly, Andy, looky here! ",;
                  "Wait till I go tell Aunt Bea!<p>" + ;
                  "Time: <b>" + TIME()+ "</b>")
```

To make it easy on yourself, just copy the existing THIS.StandardPage call and modify the copy. Then comment out the original call. If you rebuild the project, creating a new EXE, you very well might end up with an error message like in **Figure 19**. (In fact, if you don't run into this at one point or another in your Web Connection programming career, you're just not trying very hard.)


We're developing in Visual FoxPro 06.00.8492.00 for Windo...

File access is denied.

Figure 19*. You can't rebuild an EXE if it's already running.*

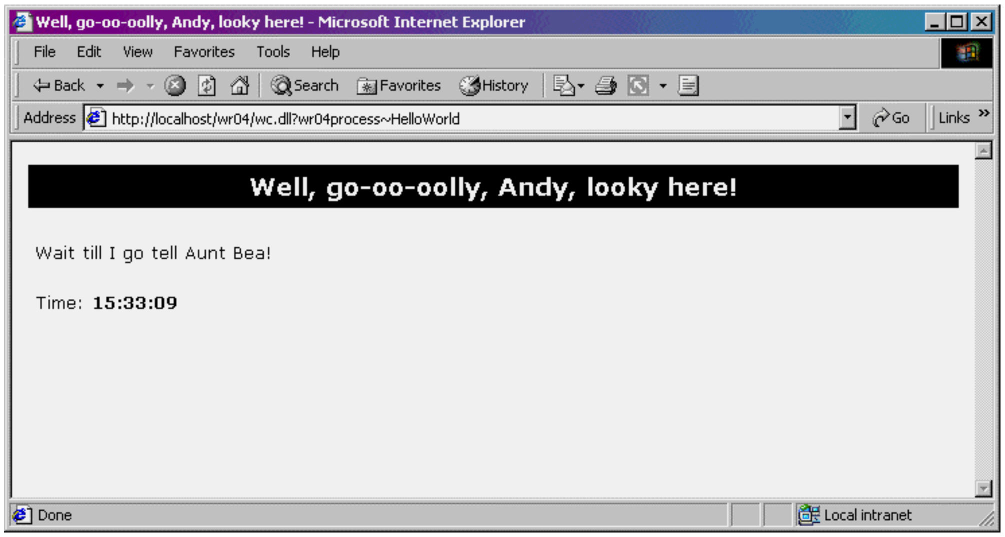
What's happened is that you haven't shut down the Web Connection server (the window in Figure 14).

Click on the icon in the taskbar, click on the Exit button in the server window, and then rebuild the EXE again. Run the EXE (using that nifty shortcut on your taskbar that's part of the GoFoxGo program!), and then enter the following request in your browser:

```
http://localhost/wr04/default.htm
```

If you click the "Hello World" link this time, you should end up with a new page in your browser, like in **Figure 20**.

Okay, so we're genuine programmers now. But we're just running an EXE and returning the results of a simple function. Let's move on.


Well, go-oo-oolly, Andy, looky here! - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost/wr04/wc.dll?wr04process~HelloWorld
Go Links
Done
Local intranet

Well, go-oo-oolly, Andy, looky here!

Wait till I go tell Aunt Bea!

Time: **15:33:09**

Figure 20. *The results of your revised Hello World program.*

Accessing static (non-Web Connection) files

It's time to expand the scope of this application to include static files. One example might be an image file that you want to embed in your generated Web page.

You'll want to do one thing first. That "thing" is to put code in your Web Connection app that will reference (and display) other files.

Referencing static files

The next step is to make some modifications to the process program that contains the Hello World function we just modified. I'm going to show you how to create a new function that references a static image file, and call that function instead of Hello World in the URL.

First, shut down the server (click on the Exit button in the server window).

Next, fire up VFP (if it's not already running), open your project, and open the WR04Process program file. Add the following code, say, in front of the Hello World function. (You could put it after, instead, if you wanted to—just make sure that your new function is located after the DEFINE CLASS command near the beginning of the code, and before the ENDDEFINE statement at the end of the Process program file.)

```
********************************************************************
FUNC ShowStatic()
********************************************************************
m.lcTitle = "This is a WC request with a static file"
m.lcPage ;
  = [This is the West Wind Logo:] ;
  + [<br>] ;
  + [<p><img border="0" src="images\wconnect.gif" ] ;
  + [ width="180" height="105"></p>] ;
  + [That was the West Wind Logo.]
```

```
this.StandardPage(m.lcTitle, m.lcPage)
ENDFUNC
```

You'll notice that the HTML code in this new function refers to a GIF file that's located in the "images" directory, wherever that is. We'll set that up in a second.

The other thing to notice is that in the Hello World function, the actual data was sent to StandardPage(). In this new function, I'm using the same standard Web Connection function—StandardPage()—but feeding the StandardPage() function two parameters via memory variables. You'll want to use this technique yourself, as the text strings you assemble as part of your Web page output may be rather large, and assembling them inside StandardPage() would be, er, awkward.

Setting up static files

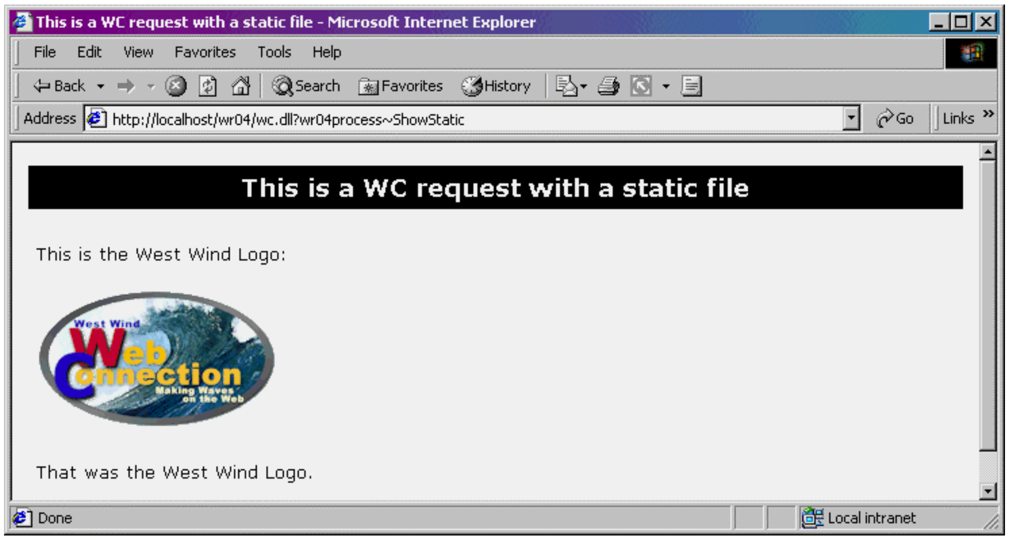
Now, about that image file. Create a directory called IMAGES underneath \INETPUB\WWWROOT\WR04, and put the image file you want to display in your HTML page in that directory. The image file WCONNECT.GIF can be found under \wconnect\html\formimages. If you don't, or you forget, or you mess up the spelling, the image won't display in the page—and you'll get a big red X where the image should be.

Running your app

Time to test. Run WR04MAIN or rebuild your EXE, and run the EXE (using that nifty taskbar shortcut described earlier). Then open up your browser. Enter the following URL:

```
http://localhost/wr04/wc.dll?wr04Process~ShowStatic
```

You should get a page returned in your browser that looks like **Figure 21**.


This is a WC request with a static file - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost/wr04/wc.dll?wr04process~ShowStatic
Go Links
Done
Local intranet

This is a WC request with a static file

This is the West Wind Logo:

That was the West Wind Logo.

***Figure 21**. The results of a Web Connection hit that references a static file.*

Accessing data

It’s a pretty silly database application that doesn’t actually have data in it, eh? Well, the same type of techniques I used to reference static files can be used, after a fashion, to reference data. Neat-o!

The sample application I’m going to demonstrate in this section is a simplified version of the Resource Database from Hentzenwerke’s Web site. If you’re not acquainted with it, the Resource Database contains listings on a variety of developer resources—consultants, training videos, events, third-party tools, vertical market applications, and so on. Users can select the type of resource they’re looking for—utilities, for example—as well as keywords, and then display all listings that match that query.

In this example, I’m simply going to allow the user to select records that match a particular type of resource. In other words, I’m going to pull out all records where the value matches the contents of the Type field.

Setting up your database

The first thing to do is set up the resource table. The table, RES.DBF, is going to go into the WSDB\WR04 directory. The structure looks like this.

```
iidres I
ccategory c(10)
ctype c(20)
cnares c(100)
cnaf c(15)
cnal c(15)
cnao c(50)
ca1 c(30)
ca2 c(30)
ca3 c(30)
ccity c(20)
cstate c(10)
czip c(15)
ccountry c(25)
cgeoloc c(10)
cvoice c(15)
cfax (c15)
ctollfree1 c(15)
ctollfree2 c(15)
curl c(100)
cemail1 c(50)
cemail2 c(50)
mdesc m
cadded c(10)
tadded t
cchanged c(10)
tchanged t
```

The domain for the cType field contains the following values: Books, Consultants, Events, Magazines, Training, Courseware, Tools, Vertical Market Apps, and Videos. Thus, the user will be able to query the table for all records in any one of these types.

Opening your database

The next thing to do is enable your application to see the data. Remember, unlike the static image file we used in the previous example, your data will not be in the Web server directory structure, and, thus, will not be directly visible to the app.

What we need is a "file opening" mechanism, much like you probably use in your day-to-day desktop applications. Open up the WR04Main program, and look for the SetServerProperties method. In this method, you'll see the following line:

```
*** Add any data paths - SET DEFAULT has already occurred so this is safe!
```

This is where you'll want to add your file opening code. The following code *would* work:

```
*** Add any data paths - SET DEFAULT has already occurred so this is safe!
select 0
use \WSDB\WR04\RES shared
```

but it would be a bad idea. Hard-coding the entire path name in the USE command, that is. If you just have one or two tables, maybe you could get away with it—at least for introductory testing. But it's a bad habit to get into.

Why? First of all, if your app grows (and apps never get smaller, do they?), and you add more tables, you potentially enter into a maintenance nightmare. Imagine if you had 10 or 15 or 20 tables—and then you decided to change the location of where your data is stored.

Second, while I've been recommending to you that you keep your development and live machine structures similar, that's not always practical, or possible. And if that's the case—either now, or becomes the case in the future—then you'll need to create a file opening mechanism that can do double duty: open tables in one location on your development box and in another location on the live server.

So, what's a better way? There are actually two issues here: 1) setting the path to the location of the tables in one place, so you don't have to make multiple changes if the path changes; and, 2) providing a mechanism so that you can point your app at tables in one location during development and a second location during production.

You can handle the first issue by setting your path under the "Add any data paths" line, and then creating a separate function that opens tables. For example:

```
*** Add any data paths - SET DEFAULT has already occurred so this is safe!
set path to (set('path')+'; YourPath')
this.OpenTable("RES")
```

And then, right after the ENDFUNC statement of the process method in the server, you'd have a function called OpenTable that would look like the code in **Listing 2** that our technical editor uses.

***Listing 2**. The OpenTable function to generally open tables.*

```
function OpenTable
parameter tcTableName, tcAlias
* Opens table named in tcTableName parameter.
* If tcAlias is specified, table will be
* opened with that alias.
* Leaves table/alias selected
* Returns .T. if successful, .F. if not

local llSuccess, lcOldOnError

lcOldOnError = on('Error')
on error llSuccess = .F.

llSuccess = .T.
if empty(tcAlias)
 if not used(tcTableName)
  select 0
  use (tcTableName) again shared
 else
  select (tcTableName)
 endif
else
 if not used(tcAlias)
  select 0
  use (tcTableName) again alias (tcAlias) shared
 else
  select (tcAlias)
 endif
endif

on error &lcOldOnError

return llSuccess
```

A better way is to read the data path from the application's INI file (WR04.INI) instead of hard-coding it with a SET PATH command. First, you need to include your data path in your application's INI file. In the bottom section of WR04.INI, I've modified the Datapath line to reference the \wsdb\wr04\ path that heretofore had been referenced in the SET PATH command.

```
[Wr04process]
Datapath=\wsdb\wr04\
Htmlpagepath=e:\inetpub\wwwroot\wr04\
```

Next, here's the code you'd use in the SetServerProperties method of WR04MAIN instead of the SET PATH (or DO PATH) business. And while you're at it, you might add a belt and suspenders to the SET PATH statement, and throw a couple of debugging statements in as well:

```
*** Add any data paths - SET DEFAULT has already occurred so this is safe!
lcDataPath = THIS.oConfig.oWR04process.cDataPath
debugout "lcDataPath is " + m.lcDataPath
this.cPathOriginal = set("path")
set path to this.cPathOriginal + "; " + (m.lcDataPath)
m.lcX = set("path")
debugout "Path is now " + m.lcX
this.OpenTable("RES")
```

The first line, with the reference to THIS.oConfig.oWR04Process.cDataPath, is explained more fully in Chapter 11, "Managing Your Configuration"; for the time being, you can replace the "WR04Process" string with the name of your own project.

Note that if you forget this step—adding code to open your tables—the code you write in your process method will eventually try to do something with a table, and since that code knows nothing about, nor can it find that table, it'll pop open a File Open dialog on your server box. Since the user of this application—running a browser in Texas or Germany or New Zealand—isn't sitting in front of your server, they can't deal with the dialog.

You'll also want to know that you don't have to explicitly close the table at any point. Since your application (and, presumably, your data) will be available all the time, you won't need to close the table. Whenever you shut down your Visual FoxPro Web Connection server, the table will automatically be closed, just like any Visual FoxPro executable does. You may be wondering if one instance of an open table is shared by all the hits serviced by the WC server—and, yes, it is.

Now that the table is available for access, it's time to get stuff out of it.

Presenting parameter selections to the user

We're going to present an HTML page to the users that allows them to choose which records they want to retrieve from the database. Technically, this could be done with a static page that you put together in FrontPage or another HTML editing tool. However, we're going to build it from a function that we're going to put in our WR04Process class, and then call via Web Connection.

Doing so provides two benefits. First, we'll call a second WR04Process function from this page, so you'll start to get an idea of how to daisy-chain one page to the next. And, second, it's just a short jump to make this static HTML page data-driven. For example, the categories are currently hard-coded in the HTML code, but it would be pretty easy to create the categories on the fly by querying the resource database and building the list of query options from those values.

Just so you know where we're heading, **Figure 22** shows you a picture of the parameter page that we're going to build in a moment.

Note that the URL in the address bar has a typical Web Connection call, with the final function named "GetParms." In your own live application, you'd typically have a hyperlink that says, "Click here," and would have the following link in it:

```
Click <a href="/wr04/wc.dll?wr04process~GetParms">here</a>
```

You can probably guess that we're going to need a new function in WR04Process.PRG. It's going to look like **Listing 3**.

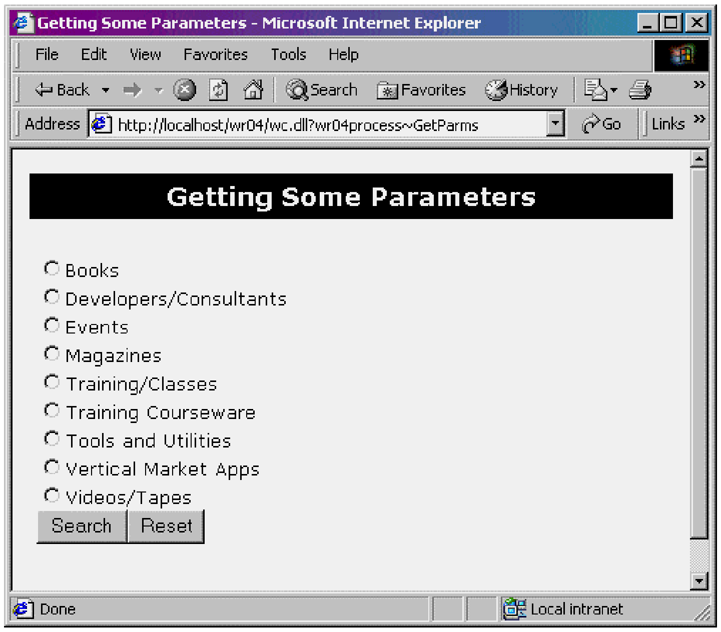


Figure 22. *The query screen requests parameters from the user.*

Listing 3. *The GetParms function that is called by a Web Connection call.*

```
*********************************************************************
FUNC GetParms()
*********************************************************************
m.lcTitle = "Getting Some Parameters"
m.lcPage ;
  = [<form action="/wr04/wc.dll?wr04process~GetAndShowResults" method="post">]

m.lcPage ;
  = m.lcPage ;
  + [<input type="radio" value="Books" name="RadioType">Books<br>] ;
  + [<input type="radio" value="Consultants" name="RadioType">] ;
    + [Developers/Consultants<br>] ;
  + [<input type="radio" value="Events" name="RadioType">Events<br>] ;
  + [<input type="radio" value="Magazines" name="RadioType">Magazines<br>] ;
  + [<input type="radio" value="Training" name="RadioType">] ;
    + [Training/Classes<br>] ;
  + [<input type="radio" value="Courseware" name="RadioType">] ;
    + [Training Courseware<br>] ;
  + [<input type="radio" value="Tools" name="RadioType">] ;
    + [Tools and Utilities<br>] ;
  + [<input type="radio" value="Vertical Market Apps" name="RadioType">] ;
    + [Vertical Market Apps<br>] ;
  + [<input type="radio" value="Videos" name="RadioType">Videos/Tapes<br>]

m.lcPage ;
  = m.lcPage ;
  + [<input type="submit" value="Search" name="B1" tabindex="98">] ;
  + [<input type="reset" value="Reset" name="B2" tabindex="99"><br>] ;
  + [</form>]

this.StandardPage(m.lcTitle, m.lcPage)
ENDFUNC
```

There are four important pieces to this little bit of Visual FoxPro and HTML code.

The first thing to realize is that this is building HTML strings in Visual FoxPro. I used the square bracket delimiters because more often than not, single and double quotes will be needed in the strings presented to the user.

The second is the option button (those old-fashioned HTML folk still call it a radio button) syntax code. You can learn more about this syntax in Chapter 9, "How a Web Page Works," but it's important now to point out that the "type" describes what kind of input control is going to be displayed—an option group, a text box, a check box, or whatever.

The "value" is the value of the control that you'll look for when determining which option button the user selected, while the "name" is the name of the control that you'll use to determine which control on the HTML page the user was working with. This is the same as how it works in Visual FoxPro.

If you had two option groups, one for the Type of Resource and the other for the Geographic location, you'd could name one of them "RadioType" and the other "GeographicLocation." The values for the RadioType option group would be "Books," "Videos," "Events," and so on, while the values for the GeographicLocation option group would be "Alabama," "Alaska," "Arkansas," and so on. The text string after the close of the angle bracket is the text that will display on the HTML page after the option button image.

And, finally, the <br> is an HTML code for a line break—much like a carriage return/line feed combination.

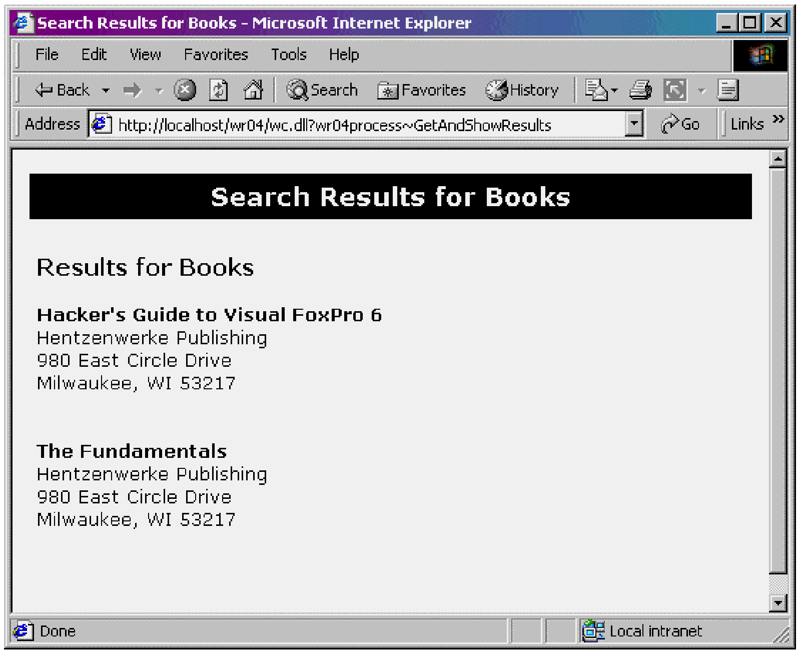
The third piece is the definition of the command buttons on the form. The types of command buttons are predefined—"submit" will execute whatever is defined as the form action, while "reset" will set the values in all controls on the page back to the values they held when the form was first displayed. So, if you seed a control with a value and then display the page like that, but the user then edits that value and then clicks Reset, the value of the control goes back to the original value.

The fourth and final piece is the definition of the form action—what will happen when the Submit button is clicked. In our case, we're going to make a call to a second WR04Process function, called GetShowResults, and it's this function that will determine which option group button was selected, do the query of the table, and then display the results.

Digging data out of your database

Next, we need to run a query using the parameters that the user selected, and then present the result set to the user in another page as shown in **Figure 23**. As Garth said in "Wayne's World" when they were setting up the satellite relay to intercept Mr. Big's cell phone calls, "This is almost too easy!"

Listing 4 shows the code for the GetAndShowResults function.


Search Results for Books - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost/wr04/wc.dll?wr04process~GetAndShowResults
Go Links
Done
Local intranet

Search Results for Books

Results for Books

Hacker's Guide to Visual FoxPro 6
Hentzenwerke Publishing
980 East Circle Drive
Milwaukee, WI 53217

The Fundamentals
Hentzenwerke Publishing
980 East Circle Drive
Milwaukee, WI 53217

Figure 23*. The results of the query screen.*

Listing 4*. The GetAndShowResults function that is called by a Web Connection call.*

```
**********************************************************************
FUNC GetAndShowResults()
**********************************************************************
local m.lcSearchForWhat, m.lcUSearchForWhat, m.li, m.lcPage

* get the value of the control from the GetParms page
m.lcSearchForWhat = alltrim(request.form('RadioType'))
m.lcUSearchForWhat = upper(m.lcSearchForWhat)

* dig the data out of the table
sele cNaRes, cNaF, cNaL, cNaO, cA1, ;
     cA2, cA3, cCity, cState, cZip, ;
     cVoice, cFax, cTollFree1, cTollFree2, cEmail1, ;
     cEmail2, cURL, left(mDesc,200) ;
     from RES ;
  where upper(cType) = m.lcUSearchForWhat ;
  order by cNaRes ;
  into arra aWhat

if _tally = 0
  * no resources for this Type
  m.lcTitle = "Search Results for " + m.lcSearchForWhat
  m.lcPage = [Type = ] + upper(cType) + [!<br>]
  m.lcPage = m.lcPage + [UpperSrchForWhat = ] + m.lcUSearchForWhat  + [!<br>]
  m.lcPage = m.lcPage + [No Records for ] + m.lcSearchForWhat
```

```
else
  * label the page with the name of the resource
  m.lcTitle = "Search Results for " + m.lcSearchForWhat
  m.lcPage = [<font face="Verdana" size="3">]
  m.lcPage = m.lcPage + [<b>Results for ] + m.lcSearchForWhat + [</b></font>]
  m.lcPage = m.lcPage + [<br><br>]

  * loop through all resources found
  for m.li = 1 to alen(aWhat,1)
    m.lcPage = m.lcPage + [<font face="Verdana" size="2">]
    m.lcPage = m.lcPage + [<b>]

    * resource name
    m.lcPage = m.lcPage + alltrim(aWhat[m.li,1])
    m.lcPage = m.lcPage + [</b><br>]

    * company name
    m.lcPage = m.lcPage + iif(!empty(aWhat[m.li,4]), ;
                          alltrim (aWhat[m.li,4]) + [<br>] , "")
    * Address 1
    m.lcPage = m.lcPage + iif(!empty(aWhat[m.li,5]), ;
      alltrim (aWhat[m.li,5]) , "")
    * Address 2
    m.lcPage = m.lcPage ;
      + iif(!empty(aWhat[m.li,6]), [, ]+ alltrim (aWhat[m.li,6]), "" )
    * Address 3
    m.lcPage = m.lcPage ;
      + iif(!empty(aWhat[m.li,7]), [, ]+ alltrim (aWhat[m.li,7]), "" ) + [<br>]
    * City/State/Zip
    m.lcPage = m.lcPage + alltrim (aWhat[m.li,8]) + [, ]
    m.lcPage = m.lcPage + alltrim (aWhat[m.li,9]) + [  ]
    m.lcPage = m.lcPage + alltrim (aWhat[m.li,10]) + [<br>]

    * space between this resource and the next
    m.lcPage = m.lcPage + [<br><br></font>]
  next
endif

* output the results
this.StandardPage(m.lcTitle, m.lcPage)
ENDFUNC
```

There are, again, several specific pieces of code that are important. The first is the Request.Form function. It grabs the value of the control passed as the parameter—in this case, the RadioType option group. (Remember that "RadioType" is the name we specifically gave to the control in the GetParms function—we could have called that option group "Herman" or "Hilda," and then would have issued a Request.Form("Herman") call.)

The second thing is to convert the value to all uppercase. The reason I don't do this when assigning the value to the variable m.lcSearchForWhat is that I'll want the proper case version around when I display the results to the user.

You'll probably find that, in the beginning, interpreting the value of the controls on HTML pages is one of the trickier things to get right. It seems that the value is always the wrong case, or its type is not what you thought it would be—numeric when you thought it would be character, or vice versa. As a result, having the original value from the control around is useful when you have to debug.

Once I've found what type of Resource the user is interested in, I grab the appropriate records from the RES table, and then put together the results string that are going to be displayed to the user by moving through the resulting array row by row.

Once the whole string has been assembled, I use Web Connection's StandardPage() function to output the results.

You'll notice that in this piece of code, I don't use one huge statement to put together the value for m.lcPage. This is because it can be tough to hunt down a syntax error in a 25-line string concatenation. If you build a series of smaller strings, Fox will tell you exactly which line to look at if you mismatch parentheses or forget a comma—and it happens to all of us. And Fox is plenty fast at building strings—there generally isn't a performance penalty.

Even so, some of the strings can be hard to decipher at first—a well-structured convention for formatting your code will help readability and debugging considerably.

As you can see, using StandardPage can be flexible, to a point. But what if you wanted a different look for your title? Building the entire page into a single string can also become a pain. Fortunately, the Response object has two functions—Write and WriteLn—that allow you to write out a single line at a time. The difference between them is that with Write, you must specify where the carriage returns go. WriteLn includes a carriage return at the end of each line. There are also HTMLHeader and HTMLFooter functions to make creating a more customized header and footer easier, and ShowCursor for a quick way to display a table. An example of how to use these functions is shown in **Listing 5**, using the same output we generated earlier.

***Listing 5**. The GetAndShowResults function using an alternate set of functions for HTML presentation.*

```
*********************************************************************
FUNC GetAndShowResults()
*********************************************************************
local m.lcSearchForWhat, m.lcUSearchForWhat, m.li, m.lcPage, aHeaders[8],
lcTitle

* get the value of the control from the GetParms page
m.lcSearchForWhat = alltrim (request.form('RadioType'))
m.lcUSearchForWhat = upper(m.lcSearchForWhat)

* dig the data out of the table
sele cNaRes, cNaO, cA1, ;
     cA2, cA3, cCity, cState, cZip, ;
     cVoice, cFax, cTollFree1, cTollFree2, cEmail1, ;
     cEmail2, cURL, left(mDesc,200) ;
     from RES ;
  where upper(cType) = m.lcUSearchForWhat ;
  order by cNaRes ;
  into cursor What

with Response
  if _tally = 0
    * no resources for this Type
        .HTMLHeader([Search Results for ] + m.lcSearchForWhat)
    .WriteLn([UpperSrchForWhat = ] + m.lcUSearchForWhat  + [!])
    .WriteLn([No Records for ] + m.lcSearchForWhat)
  else
```

```
    * Write a grid that shows the result
        * Format an array to hold the column headings
        * (ShowCursor will use the field names as column
        * headings if you don't)
        aHeaders[1] = "Resource"
        aHeaders[2] = "Company"
        aHeaders[3] = "Address"
        aHeaders[4] = ""
        aHeaders[5] = ""
        aHeaders[6] = "City"
        aHeaders[7] = "State"
        aHeaders[8] = "Zip"
        * Define a string to use as the page title
        lcTitle = "Results for " + m.lcSearchForWhat
    .ShowCursor(aHeaders, lcTitle,,,)

    * Button for bottom
    .WriteLn([<form method="POST" name="form1" id="form1" ;
       action="/wr04/wc.dll?wr04process~GetParms">])
    .WriteLn([<p><center><input type="submit" name="btnSubmit" value="Done">])
  endif
  .HTMLFooter()
endwith

ENDFUNC
```

The ShowCursor function uses the currently selected alias, so we must send the results of the query to a cursor instead of an array. There are several parameters available for ShowCursor that increases its flexibility. I recommend you read the “wwResponse::ShowCursor” topic in the documentation that comes with Web Connection for some additional ideas.

Updating your database

While getting data out of your database is all well and good, it’s pretty likely that you’ll want to put data into it as well—rather, let your users put data into it. I won’t go into any detail here in this part, because it’s all just Visual FoxPro database stuff that you already know how to do.

For example, in order to add a record, just take the data the user has entered into the controls on a form, using the request.form functions to get the values. Then add a record to a table with the APPEND BLANK or INSERT INTO commands, and use those values from the Web page as values to put into the new record.

Editing is nearly as easy. Find an existing record by asking the user for a value that you can use to locate the matching record in the table. Use the request.form function to find out what value they entered. Then SEEK that value in a table, and either return values from a successfully found record or present a message to the user that the record was not found. Deleting works similarly—let the user find the record they want to delete, and then just get rid of it from the table like you normally would.

Of course, you can run into a number of tricky situations—that’s why we’ll cover advanced topics and a whole host of tricks and traps later in the book. But you’ve already got the fundamental plumbing for your Web Connection apps covered here.

Using script maps instead of calls to WC.DLL

Step 3 of creating your project earlier in this chapter was to identify the script map (or, as I suggested a better name would be, the script extension). I even made a quick reference to how a Web Connection script map would be used in place of the long URL containing "wc.dll" but then skipped on ahead. Now it's time to show you how to use script maps in the previous examples. I'll walk through each example and show you the comparable syntax for the script map.

Script map syntax

The first example was a simple call to the "Hello World" method in wr04process that looked like this:

```
http://localhost/wr04/wc.dll?wr04process~HelloWorld
```

The script map version of this would be:

```
http://localhost/wr04/HelloWorld.wr
```

The next example was to reference a static file, such as an image file. The call that did this was:

```
http://localhost/wr04/wc.dll?wr04Process~ShowStatic
```

The script map version would be:

```
http://localhost/wr04/ShowStatic.wr
```

Finally, the calls to present the user with a series of choices were:

```
http://localhost/wr04/wc.dll?wr04Process~GetParms
```

and then inside the GetParms method to present the results:

```
m.lcPage ;
  = [<form action="/wr04/wc.dll?wr04process~GetAndShowResults" ;
    method="post">]
```

The corresponding script map versions would be:

```
http://localhost/wr04/GetParms.wr
```

and

```
m.lcPage ;
  = [<form action="GetAndShowResults.wr" ;
    method="post">]
```

Benefits to script maps

As you can see, there's not a lot of mystery to using simple script maps like those in the previous few examples. They can get more complex, as you'll see in Chapter 12. Still, you may be asking, why bother? There's a lot to learn—what's in it for you to learn yet something else?

First of all, they're shorter to type, and that means that you're less likely to make a mistake. And the fewer mistakes you make, well, that's got to be good, right? Second, using a script map means that the reference to WC.DLL, as well as the internals of your application, are hidden from the user. This means that the user may not be aware that you're running a Web Connection application, or how your application is structured, and the more information about the internals of your Web site that you can keep hidden from the outside world, the better.

Deploying your application to a live Web server

Tell me your eyes aren't lighting up right now. This is, of course, the ultimate goal—to deploy your application to a live Web server.

Chapter 12 will cover the development of a full application from start to finish, including deploying to your live server. You may want to get a taste of how this works before then, so I'll briefly walk you through the steps required to configure and deploy this simple WR04 Web Connection application to a live server.

The steps involved are:

1. Setting up the directory structure on your live Web server.
2. Getting the files of your application to the server.
3. Testing.

Setting up your live Web server directory structure

For the most part, you'll need directories on your live server that match the ones on your development box. You won't have your development project, Visual FoxPro, or a copy of Web Connection on your live server, but everything else on your development box will have a match.

The first directory you'll need to have set up is where your Web site will be located—when you're starting out, that would be \inetpub\wwwroot, just like on your development box. Since your live server may have a different number of drives than your development environment, the drive designation may be different.

In any case, you'll set up your Web server software (IIS) just like you did on your development box, together with identifying the Web server's home directory and default document as shown in Figures 13 and 14 in Chapter 3.

You'll then need to create application-specific directories underneath your Web server's home directory, like so:

```
\inetpub\wwwroot\wr04
```

This directory is where the "WR04" Web site will reside. Your home page, all other static HTML files, graphics, and any other files will all reside in this directory (or in a subdirectory below it, such as \images). You'll also need to create an ADMIN directory under this directory.

Note that I'm skipping the drive designation in these examples. The second directory you'll need to create is for Web Connection's temporary messaging files. Again, it's a good idea to make these the same as your development box, like so:

```
\wc_msg\wr04
```

The third directory you'll need to create is where the good stuff will be—your data and application files. As noted earlier, you don't have to keep your data and application files in the same directory—indeed, your application should never assume that your data is in the same directory. For sake of simplicity in this example, they're both going in the same place.

This directory would again be called

```
\wsdb\wr04
```

like it was in your development environment.

Getting your application's files to the server

You'll notice that I used the highly technical term "getting," instead of "copying" or "moving" or some other such highfalutin terminology. That's because we're going to employ different means to "get" different files to the server. In some cases, we'll simply copy files (you could also FTP or PCAnywhere them); in other cases, we'll use more sophisticated and complicated methods.

First, you'll copy WC.DLL and WC.INI into \inetpub\wwwroot\wr04. You'll also copy your Web site files—your default home page, your static HTML files, and all that stuff. This directory should be an identical version of \inetpub\wwwroot\wr04 on your development box.

You'll also copy ADMIN.ASP and WESTWIND.CSS to the ADMIN subdirectory. (If you don't include the WestWind cascading style sheet, ADMIN.ASP will show up in Times New Roman instead of a sans serif font.)

Next, you'll copy your data (databases, tables, indexes, memo files, and anything else needed) to \wsdb\wr04.

Finally, you'll install your application's EXE and INI file into \wsdb\wr04 as well. However, this is not simply a matter of copying those two files over to the live server. There's really no need to install the Visual FoxPro development environment on your Web server (although many developers do). And if you do, you'll need to make sure you satisfy all of the licensing issues involved.

Thus, you'll need to install the Visual FoxPro runtimes on your live Web server as well. If you're using Visual FoxPro 6.0, it's a matter of copying the runtime files along with your EXE file. If you're using Visual FoxPro 7.0, however, it's more complicated.

Essentially, you'll use InstallShield Express, a custom version of InstallShield that comes bundled with Visual FoxPro 7.0, to create a complete installation package that includes your EXE and INI file as well as the appropriate Visual FoxPro runtimes. When you're asked for the target directory during the InstallShield process, you won't use the default of "Program

Files\Your Company\Your Application"; rather, you'll specify "\wsdb\wr04" (or whatever your directory is called on your live Web server).

I've included a lengthy paper on how to use InstallShield to distribute Visual FoxPro 7.0 applications along with the other downloads for this chapter, available from the Hentzenwerke Web site.

Administering your live Web server

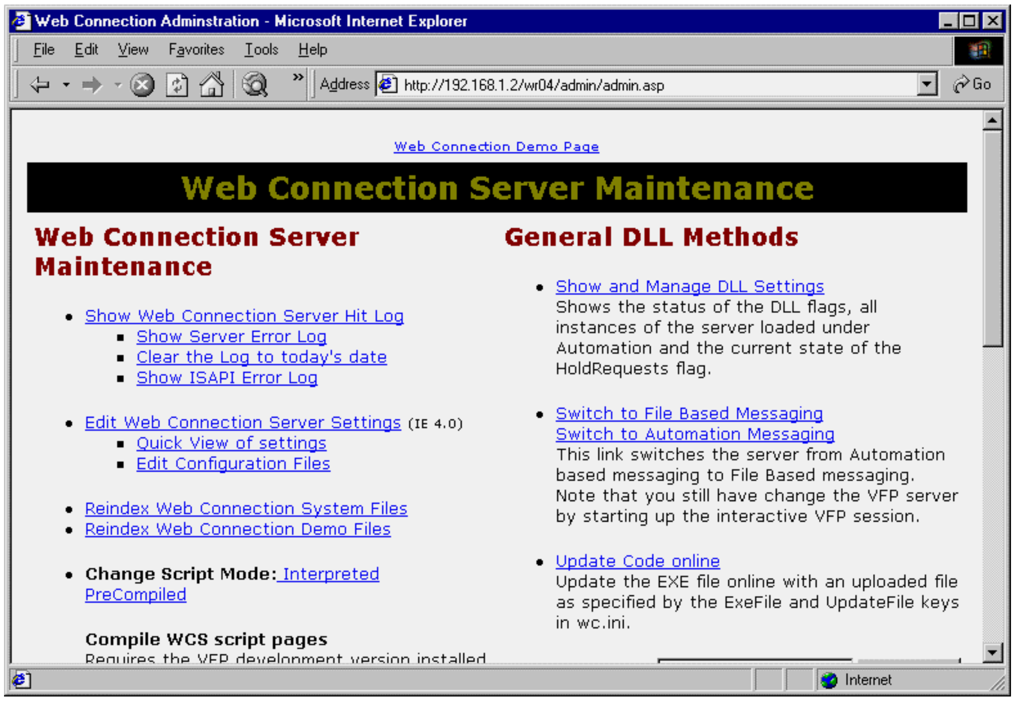
Okay, just as your eyes lit up in anticipation of the goodies in the previous section, your shoulders are probably drooping right now. "Administration"? That sounds suspiciously like "maintenance," and we all know how much developers like "maintenance." But you have to do it, and it's even more important here, because if you don't set up administration properly, outsiders can take control of your Web Connection application.

The ADMIN.ASP file in your \inetpub\wwwroot\wr04\admin directory allows you to perform a variety of maintenance tasks on your Web Connection application without having to be sitting at the server box itself.

In order to bring up the maintenance page, execute the ADMIN.ASP page in your browser, like so:

```
http://LiveServerIPOrName/wr04/admin/admin.asp
```

You'll get a page like the one shown in **Figure 24**.


Web Connection Adminstration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://192.168.1.2/wr04/admin/admin.asp
Go
Internet

Web Connection Demo Page

Web Connection Server Maintenance

Web Connection Server Maintenance

- Show Web Connection Server Hit Log
  - Show Server Error Log
  - Clear the Log to today's date
  - Show ISAPI Error Log
- Edit Web Connection Server Settings (IE 4.0)
  - Quick View of settings
  - Edit Configuration Files
- Reindex Web Connection System Files
- Reindex Web Connection Demo Files
- **Change Script Mode:** Interpreted PreCompiled

Compile WCS script pages
Requires the VFP development version installed

General DLL Methods

- Show and Manage DLL Settings
  Shows the status of the DLL flags, all instances of the server loaded under Automation and the current state of the HoldRequests flag.
- Switch to File Based Messaging
  Switch to Automation Messaging
  This link switches the server from Automation based messaging to File Based messaging. Note that you still have change the VFP server by starting up the interactive VFP session.
- Update Code online
  Update the EXE file online with an uploaded file as specified by the ExeFile and UpdateFile keys in wc.ini.

Figure 24*. The Web Connection maintenance page.*

Now, remember that the world has rights to see \inetpub\wwwroot\wr04—after all, it's your Web site, right? That also means that they have access to the ADMIN subdirectory, and thus, ADMIN.ASP… unless you stop them. You can do this through a two-step process.

First, rename ADMIN.ASP to something else, like MYADMIN.ASP or something else that you'll remember but that's not easily guessed. If you have directory browsing turned off, users won't be able to peruse through the ADMIN subdirectory to see what the file name might be.

The second step is to restrict access to running this page by making a couple of entries in WC.INI that's located in \inetpub\wwwroot\wr04. The first thing to change is the name of the admin page that Web Connect looks for. Look for the entry like this:

```
;Admin Page that is used for Backlinks from various internal pages
;Use a full server relative Web path!
AdminPage=/wconnect/Admin.asp
```

And change the value of the AdminPage entry to something like so:

```
AdminPage=/inetpub/wwwroot/wr04/admin/MyFirstAdmin.asp
```

The second entry to change is just above this one, in the section that looks like so:

```
;*** Account for Admin tasks    REQUIRED FOR ADMIN TASKS
;***       NT User Account   -  The specified user must log in
;***       Any               -  Any logged in user
;***                         -  Blank - no Authentication
AdminAccount=
```

Change the value of the AdminAccount entry to something like so:

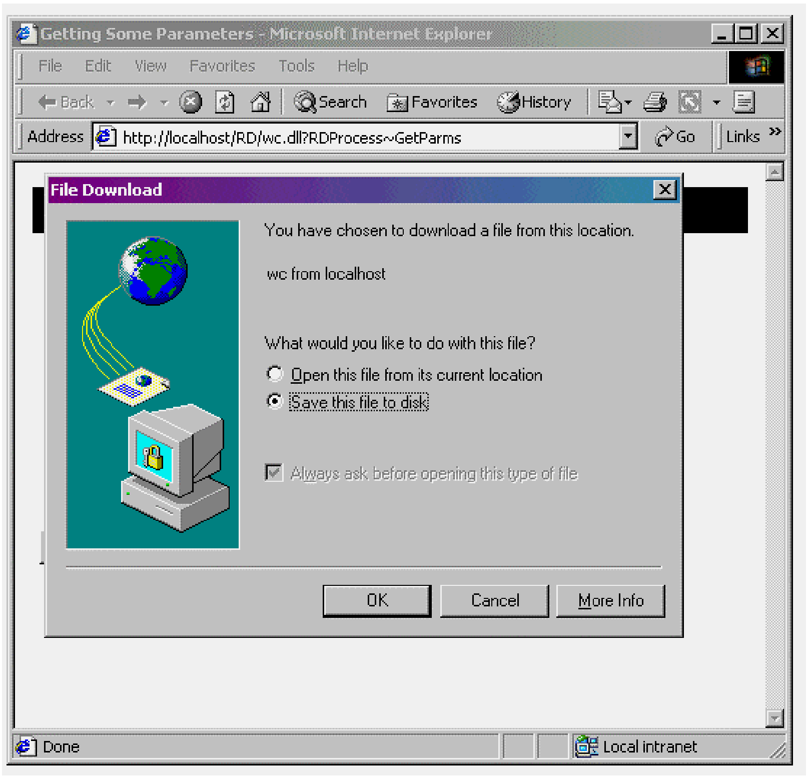
```
AdminAccount=Administrator
```

where Administrator is a Windows NT account on the server box. Once you do this, when a user opens the admin page, they'll be greeted with a Windows NT login screen and will be required to log in before getting access to the admin page.

When something goes wrong

Yes, it happens. You push the "go" button, but nothing goes. Here are some of the common mistakes developers new to Web Connection typically make, introduced by the phenomena that the user is experiencing.

The first, and possibly the most frustrating, problem is when the dialog shown in **Figure 25** appears after you make a request. I caused this error to happen by using the name of the physical directory instead of the virtual directory that WC.DLL is located in. As a result, instead of processing the WC.DLL call, the browser thinks you're trying to download the file, just as if you had typed in the name of a ZIP file contained in a hyperlink. You can also get this dialog when you don't have execute rights in the proper directories.


Getting Some Parameters - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address
Go
Links
Done
Local intranet

http://localhost/RD/wc.dll?RDProcess~GetParms

File Download

You have chosen to download a file from this location.

wc from localhost

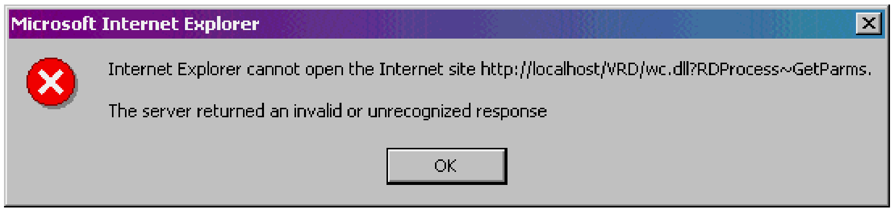
What would you like to do with this file?

( ) Open this file from its current location
(●) Save this file to disk

[✓] Always ask before opening this type of file

Figure 25. *The dialog that appears when you confuse the names of your virtual and physical directories.*

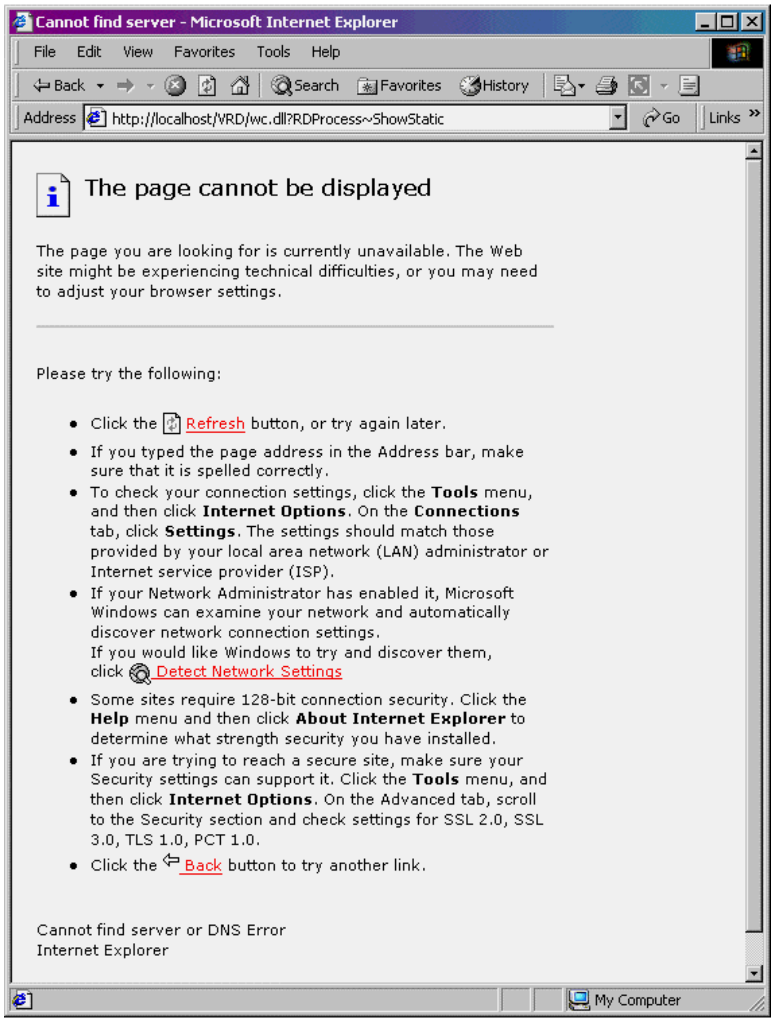
Next, you may run into the dialogs shown in **Figure 26** and **Figure 27**. These are preceded by the thermometer bar in the bottom of the browser taking its own sweet time, and then hanging up perhaps a third of way across. These errors occur when the server is not running—in other words, you forgot to click on the EXE before making the call in the browser.


Microsoft Internet Explorer

Internet Explorer cannot open the Internet site http://localhost/VRD/wc.dll?RDProcess~GetParms.

The server returned an invalid or unrecognized response

Figure 26. *One dialog that appears when the Visual FoxPro Web Connection server is not running.*


Cannot find server - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost/VRD/wc.dll?RDProcess~ShowStatic
Go Links
My Computer

The page cannot be displayed

The page you are looking for is currently unavailable. The Web site might be experiencing technical difficulties, or you may need to adjust your browser settings.

Please try the following:

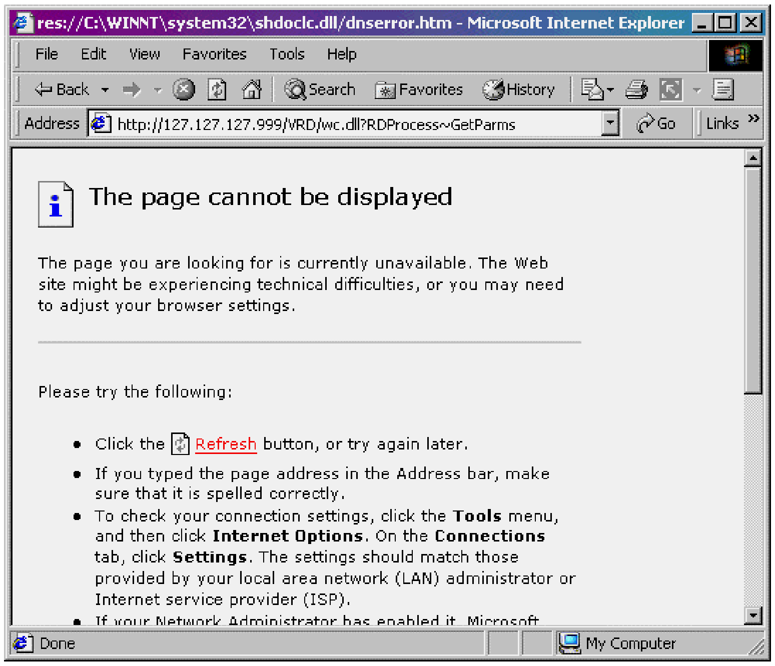
- Click the Refresh button, or try again later.
- If you typed the page address in the Address bar, make sure that it is spelled correctly.
- To check your connection settings, click the **Tools** menu, and then click **Internet Options**. On the **Connections** tab, click **Settings**. The settings should match those provided by your local area network (LAN) administrator or Internet service provider (ISP).
- If your Network Administrator has enabled it, Microsoft Windows can examine your network and automatically discover network connection settings.
  If you would like Windows to try and discover them, click Detect Network Settings
- Some sites require 128-bit connection security. Click the **Help** menu and then click **About Internet Explorer** to determine what strength security you have installed.
- If you are trying to reach a secure site, make sure your Security settings can support it. Click the **Tools** menu, and then click **Internet Options**. On the Advanced tab, scroll to the Security section and check settings for SSL 2.0, SSL 3.0, TLS 1.0, PCT 1.0.
- Click the Back button to try another link.

Cannot find server or DNS Error
Internet Explorer

Figure 27*. Another dialog that appears when the Visual FoxPro Web Connection server is not running.*

The same error can appear if you enter incorrect information into the browser’s address bar. In **Figure 28**, I’ve entered a nonexistent IP address for the server (999 is never a valid number in an IP address).

Okay, now let’s assume that the server is running, but you’re still banging your head against problems. The error page shown in **Figure 29** (the text in the band at the top of the page is yellow in color and says “Unhandled Request”) indicates that you entered the name of a process that doesn’t exist. In this case, I entered “RDPocess” (no “r”) instead of “RDProcess” in the URL.


res://C:\WINNT\system32\shdoclc.dll/dnserror.htm - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://127.127.127.999/VRD/wc.dll?RDProcess~GetParms
Go Links
Done
My Computer

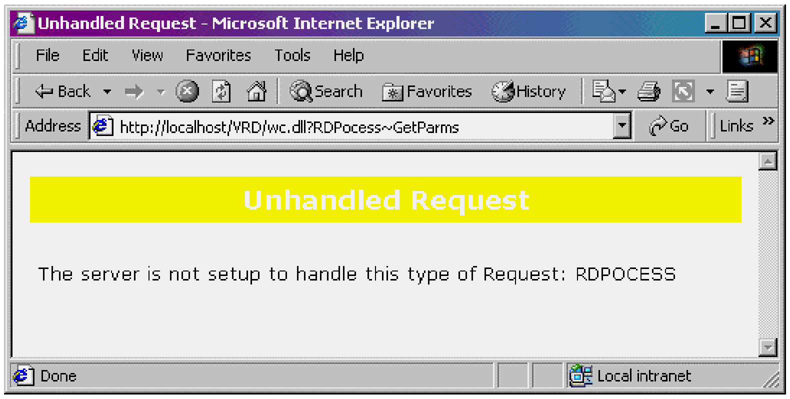
The page cannot be displayed

The page you are looking for is currently unavailable. The Web site might be experiencing technical difficulties, or you may need to adjust your browser settings.

Please try the following:

- Click the Refresh button, or try again later.
- If you typed the page address in the Address bar, make sure that it is spelled correctly.
- To check your connection settings, click the **Tools** menu, and then click **Internet Options**. On the **Connections** tab, click **Settings**. The settings should match those provided by your local area network (LAN) administrator or Internet service provider (ISP).
- If your Network Administrator has enabled it, Microsoft

Figure 28*. The dialog that appears when you enter an incorrect IP address for the server.*

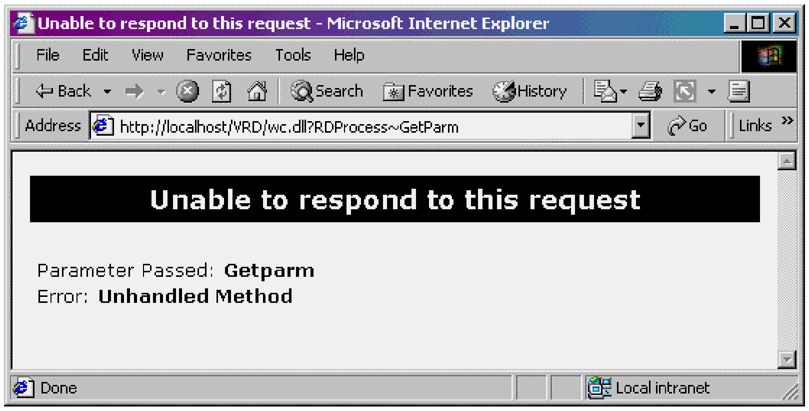

Unhandled Request - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost/VRD/wc.dll?RDPocess~GetParms
Go Links
Done
Local intranet

Unhandled Request

The server is not setup to handle this type of Request: RDPOCESS

Figure 29*. The error message that appears when you refer to a class that doesn't exist.*

Similar to the previous problem, **Figure 30** shows what happens if you make a request to a function that doesn't exist in the process method. In this case, I entered "GetParm" instead of "GetParms" (with an "s").

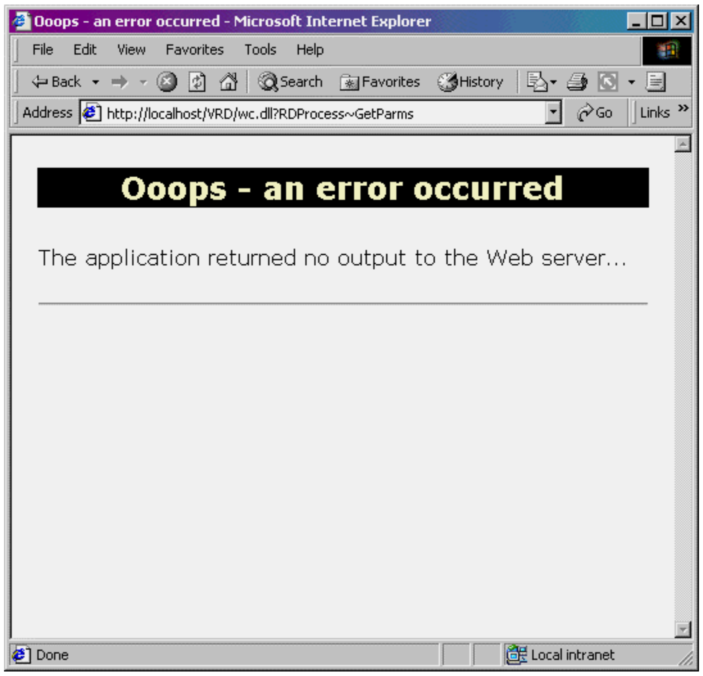

Unable to respond to this request - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost/VRD/wc.dll?RDProcess~GetParm
Go Links
Done
Local intranet

Unable to respond to this request

Parameter Passed: **Getparm**
Error: **Unhandled Method**

Figure 30*. The error message that appears when you refer to a method that doesn't exist.*

The next few errors have to do with a problem in front of the keyboard—you and your coding! **Figure 31** shows what happens when you forget to include a command in your program that returns results in an HTML page, like the StandardPage() function.

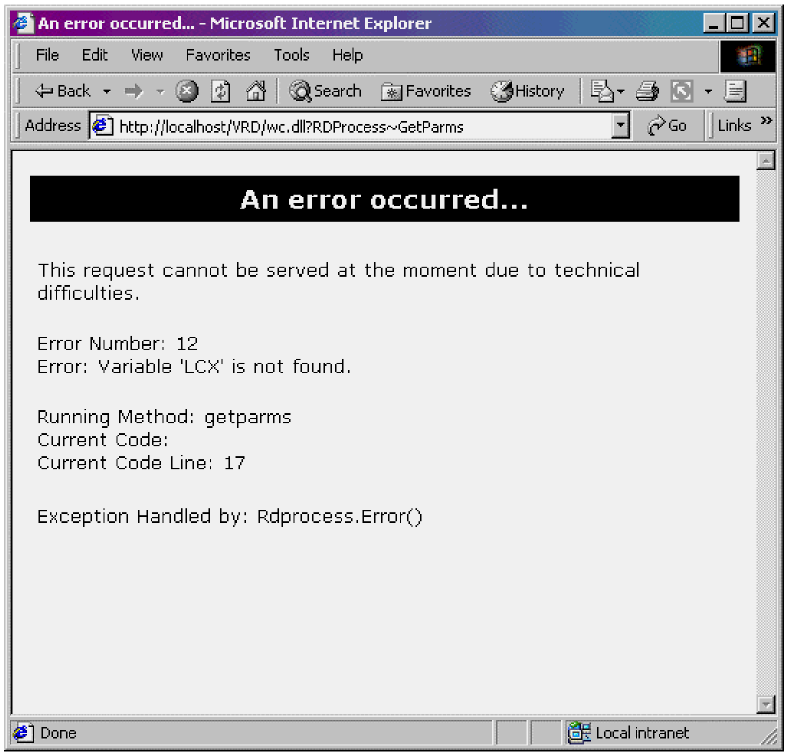

Ooops - an error occurred - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost/VRD/wc.dll?RDProcess~GetParms
Go Links
Done
Local intranet

Ooops - an error occurred

The application returned no output to the Web server...

Figure 31*. The error message that appears when you forget to send output back from your application.*

Figure 32 shows what happens when you make an error in your code that isn't detected when you compiled. For example, I included a reference to a nonexistent variable, m.clX, in the code that is used to assemble the string to be returned at the HTML page. This display will only appear if you are not running your app directly from the PRG, and Debugmode in the H file is set to .F. Otherwise, instead of this display, you'll get a VFP-style error message on the server, and eventually the browser will display the same display as shown in Figure 28.

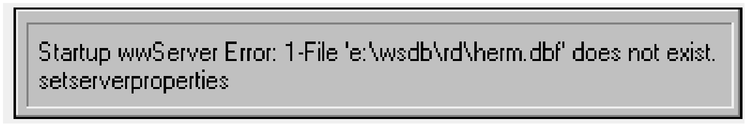

An error occurred... - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost/VRD/wc.dll?RDProcess~GetParms
Go Links
Done
Local intranet

An error occurred...

```
This request cannot be served at the moment due to technical
difficulties.

Error Number: 12
Error: Variable 'LCX' is not found.

Running Method: getparms
Current Code:
Current Code Line: 17

Exception Handled by: Rdprocess.Error()
```

***Figure 32**. The error message that appears from a programming error inside your Visual FoxPro application.*

What happens when you reference a table in your Main method, such as HERM.DBF, but HERM.DBF doesn't exist, or isn't in the place you told the method to look? While the server starts up, it'll warn you that it has run into problems. **Figure 33** shows you the message that will show up in the upper right corner of the VFP runtime window along with a not-yet-completed server window.

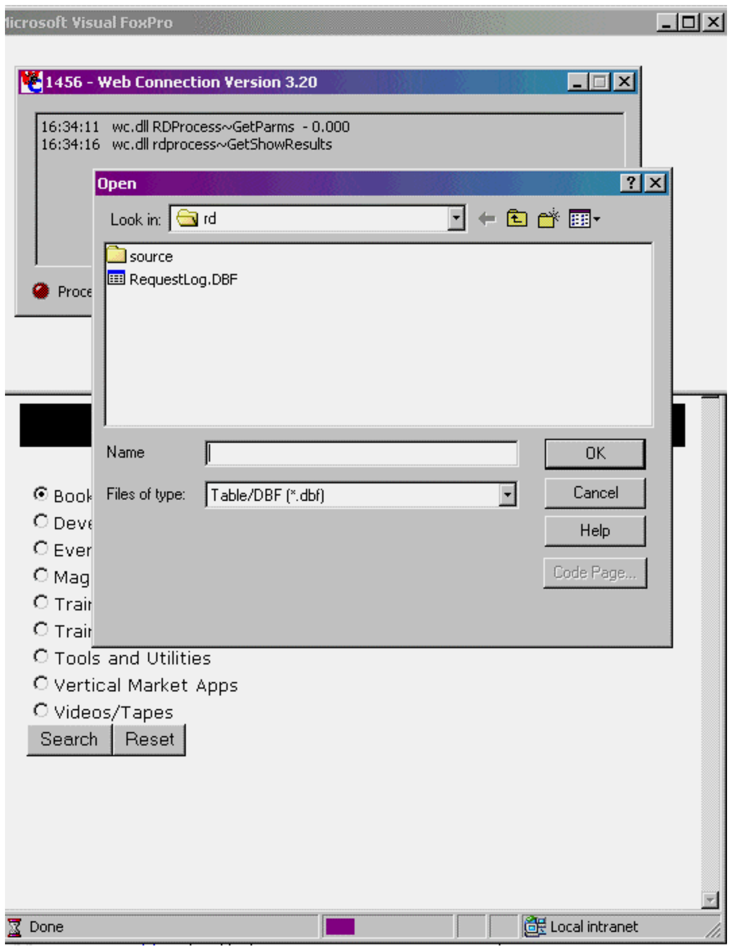


```
Startup wwServer Error: 1-File 'e:\wsdb\rd\herm.dbf' does not exist.
setserverproperties
```

***Figure 33**. One error message that appears when you reference a table that doesn't exist in the startup (MAIN) program.*

Now, suppose that you don't reference a nonexistent table in your startup code, but later, while you're trying to do some data crunching—for example, doing a SQL SELECT from a table—but that table doesn't exist (because you forgot to put it there, or because you misspelled it in your SELECT statement). What then, huh?

You'll get the mysterious File Open dialog as shown in **Figure 34**—however, it's worse than that. On your development box, where the Web server software and your application are running on the same screen as the user's browser, you'll see the File Open dialog. If you attempted to deploy this live, though, the screen on your live Web server will be where the File Open dialog displays—not on the screen where your user has their browser open. All they'll see is a Web page that eventually tells them that the request timed out, and they'll never know why.



```
16:34:11  wc.dll RDProcess~GetParms  - 0.000
16:34:16  wc.dll rdprocess~GetShowResults
```

***Figure 34**. The error message that appears when you reference a table that doesn't exist in your code.*

There are, of course, more errors than these—given the mixture of operating system, Web server software, Visual FoxPro, Web Connection, and your own programming, it's impossible to foresee every possible error. But these are the common ones. If you run into another, your best bet is to head on over to the Web Connection message board at **www.west-wind.com/wwthreads**.

Finally, I've even had a situation where the Web Connection framework "disappeared," and I started getting error messages indicating that the framework classes or files referenced by WCONNECT.H couldn't be found. In those cases, I've found that making a trivial change to WCONNECT.H (like adding a space to the end of a line), saving the file again, exiting everything, rebooting my machine, and rebuilding all files did the trick.

Another trick that sometimes helps is simply rebuilding all classes and programs in the Web Connection framework, like so:

```
cd \wconnect\classes
compile *.prg
compile classlib *.vcx
```

Conclusion

You've now built your first Web Connection application and seen it do a couple of simple things. Now it's time to get into the guts of the Internet and Web Connection. Harold will get you started in Chapter 5, "How the Internet Works," with an explanation of how the Internet works.