TRAINING & REFERENCE

murach's ASP.NET 2.0 web programming with C# 2005

(Chapter 4)

Thanks for downloading this chapter from ***Murach's ASP.NET 2.0 Web Programming with C# 2005***. We hope it will show you how easy it is to learn from any Murach book, with its paired-pages presentation, its "how-to" headings, its practical coding examples, and its clear, concise style.

To view the full table of contents for this book, you can go to our **web site**. From there, you can read more about this book, you can find out about any additional downloads that are available, and you can review our other books on .NET development.

Thanks for your interest in our books!

MIKE MURACH & ASSOCIATES, INC.
1-800-221-5528 • (559) 440-9071 • Fax: (559) 440-0963
murachbooks@murach.com • **www.murach.com**

4

How to test and debug an ASP.NET application

If you've done much programming, you know that testing and debugging are often the most difficult and time-consuming phase of program development. Fortunately, Visual Studio includes an integrated debugger that can help you locate and correct even the most obscure bugs. And ASP.NET includes a Trace feature that displays useful information as your ASP.NET pages execute.

In this chapter, you'll learn how to use both of these powerful debugging tools. But first, you'll learn how to create web sites that run under IIS so that you can test them with that web server, and you'll learn how to test an application to determine if it works properly.

How to create ASP.NET web sites that run under IIS

In chapter 2, you learned how to create a file-system web site that runs under the ASP.NET Development Server. If you have access to an IIS web server, though, you may want to create a web site that runs under IIS so you can test the web site in that environment. Or, you may want to create and test a file-system web site with the ASP.NET Development Server first, and then test it under IIS later.

How to create a local IIS web site

A local IIS web site is a web site that resides on your local computer. To create a local IIS web site, then, you must have IIS installed on your computer. Please see appendix A for information on how to install IIS.

Figure 4-1 illustrates how you create a local IIS web site. To start, you select HTTP for the location option in the New Web Site dialog box (see the next figure). Then, you typically click the Browse button to display the Choose Location dialog box shown in this figure.

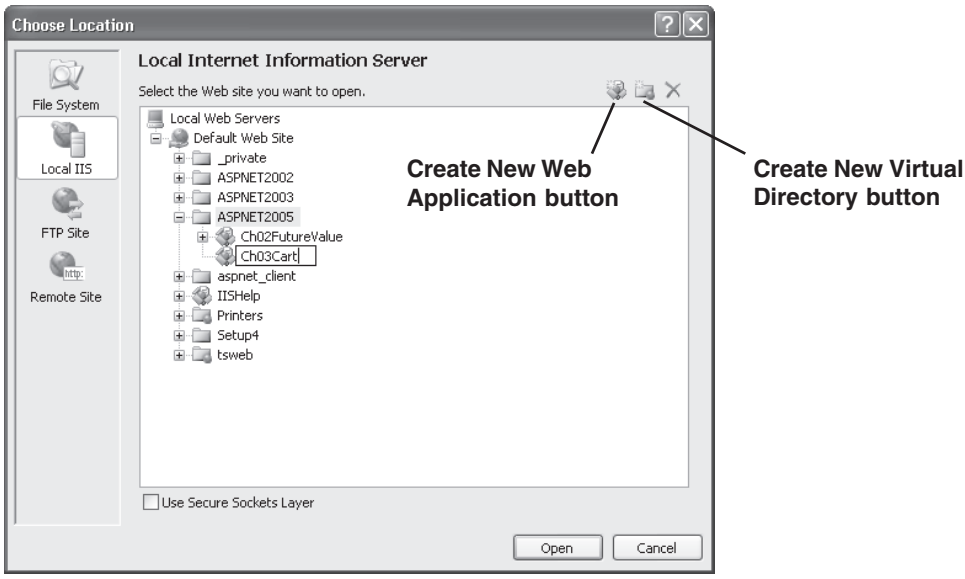
In the Choose Location dialog box, you can click the Local IIS button at the left side of the dialog box to display the IIS web server. Then, you can select the directory where you want to create your web site. In this case, I selected the ASPNET2005 directory. Then, I clicked the Create New Web Application button to create a new web site named Ch03Cart in that directory.

When you use this technique, the files for the web site are stored within the directory you create. If that's not what you want, you can create a *virtual directory* that points to the directory where the files for the web site will be stored. To do that, just click the Create New Virtual Directory button in the Choose Location dialog box. Then, a dialog box is displayed that lets you enter a name for the virtual directory and the path where the files for the web site should be stored. In the dialog box shown in this figure, for example, the virtual directory will be named Ch03Cart, and the files will be stored in a directory with the same name within the ASP.NET 2.0 Web Sites directory on the C drive.

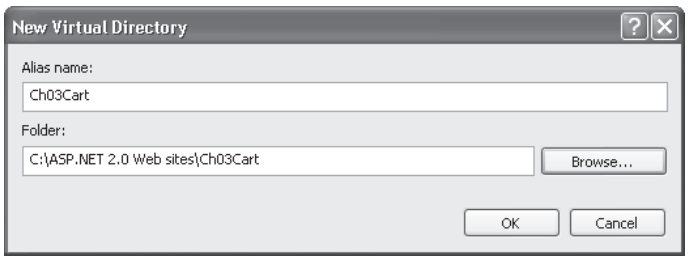
In addition to using the New Virtual Directory dialog box to create a virtual directory for a new web site, you can also use it to create a virtual directory for a file-system web site you've already created. To do that, just enter the path for the existing web site in this dialog box. Then, when you click the OK button in the New Web Site dialog box, Visual Studio will warn you that there is already a web site at the location you specified. To create a virtual directory that points to the existing web site, select the Open the Existing Web Site option.

Before I go on, you should realize that you can also create a virtual directory for a file-system web site using the IIS Management Console. If you're interested, you can learn how to do that in appendix A. Unless you need to change the default permissions for a virtual directory, though, I recommend that you create the virtual directory from within Visual Studio.

The dialog box for selecting a local IIS web site



The dialog box for creating a virtual directory



Description

- To create a web site that will run under IIS on your local computer, select HTTP for the location option in the New Web Site dialog box. Then, enter the path of the IIS directory where you want to create the web site, or click the Browse button to display the Choose Location dialog box.
- From the Choose Location dialog box, click the Local IIS button and expand the Default Web Site node. Then, select the directory where you want to create the web site from the default web site and click the Open button, or create a new directory or virtual directory.
- To create a new IIS directory for a web site, click the Create New Web Application button and then enter the name of the directory. The files for the web site will be stored in this directory.
- To create a *virtual directory* for the web site, click the Create New Virtual Directory button to display the New Virtual Directory dialog box. Enter a name for the directory and the path where you want to store the files for the web site. If the path you specify already contains a web site, you can open that web site from the virtual directory.

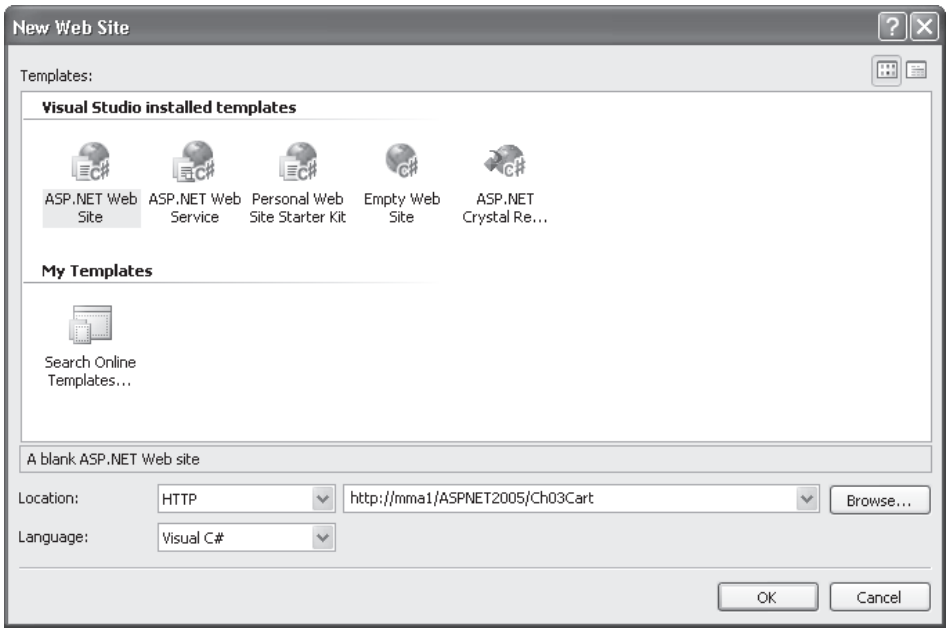
Figure 4-1 How to create a local IIS web site

How to create a remote IIS web site

A remote web site is similar to a local web site except that a remote web site resides on a computer that you have access to over a LAN. To create this type of web site, FrontPage Server Extensions must be installed on the remote computer. Then, you can just select the HTTP location option and enter the location of the web site as shown in figure 4-2. Here, a web site named Ch03Cart is being created in a directory named ASPNET2005 on a server named mma1.

Although you use the same techniques to work with a remote web site as you use to work with a local web site, you should realize that the permissions for a remote web site may not be set the way you want. For example, suppose you create a web site that writes to a text file that's stored in the App_Data folder of the site. To do that, the web site must have write permissions on that folder. By default, though, a remote web site is given only read permissions. Because of that, you'll need to have the system administrator assign the appropriate permissions to the web site.

The dialog box for creating a remote IIS web site



Description

- To create a remote web site, select HTTP from the Location drop-down list. Then, enter the URL of the web site you want to create.
- You can also create a remote web site by clicking the Browse button and then using the Choose Location dialog box that's displayed for a remote site. However, this dialog box doesn't provide any additional options.
- By default, a web application that you create on a remote server doesn't have the permissions needed to change files in the web site at runtime. If an application needs to change a file, then, you'll need to contact the system administrator about giving it the appropriate permissions.
- Visual Studio communicates with a web site on a remote server using HTTP and FrontPage Server Extensions. Because of that, FPSE must be installed on the remote server. For information on how to install FPSE, see appendix A.

Figure 4-2 How to create a remote IIS web site

How to create an FTP web site

An FTP web site is a web site that resides on a remote computer and that supports FTP file transfers. In most cases, FTP web sites are hosted on a server that you have access to over the Internet. In that case, the web site may already be set up for you. If not, you can use the Choose Location dialog box shown in figure 4-3 to create a new web site.

To display this dialog box, select FTP for the location option from the New Web Site dialog box, and click the Browse button. Then, enter the name of the server where you want to create the site and the path to the directory where the files for the web site will be stored. Note that except for the final directory, all the directories in the directory path must already exist. In this example, that means that the Murach directory must already exist. That's because this directory maps to a *virtual root* that must be set up on the server. The virtual root works much like an IIS virtual directory. However, it points to the location where files are transferred to and from the server.

In addition to the server name and directory path, you can specify the port that Visual Studio should use to send commands to the server, you can specify whether the connection to the server is established using active or passive mode, and you can specify whether you're logged in as an anonymous user or an authenticated user. In most cases, you'll leave the port set at 21. However, you may need to change the Passive Mode and Anonymous Login options.

By default, Visual Studio uses *active mode* to establish a connection with the FTP server. To understand how active mode works, you need to realize that two ports are required to use FTP: one to transmit commands and one to transmit data. In active mode, Visual Studio connects to the server using the command port and then passes the address of the data port to be used to the server. Then, the server connects to Visual Studio using the data port.

The problem with using active mode is that if the client computer is behind a firewall, the server probably won't be able to connect to it. In that case, you can connect to the server using *passive mode*. With passive mode, Visual Studio establishes the connections for both the command port and the data port. To use passive mode, just select the Passive Mode option.

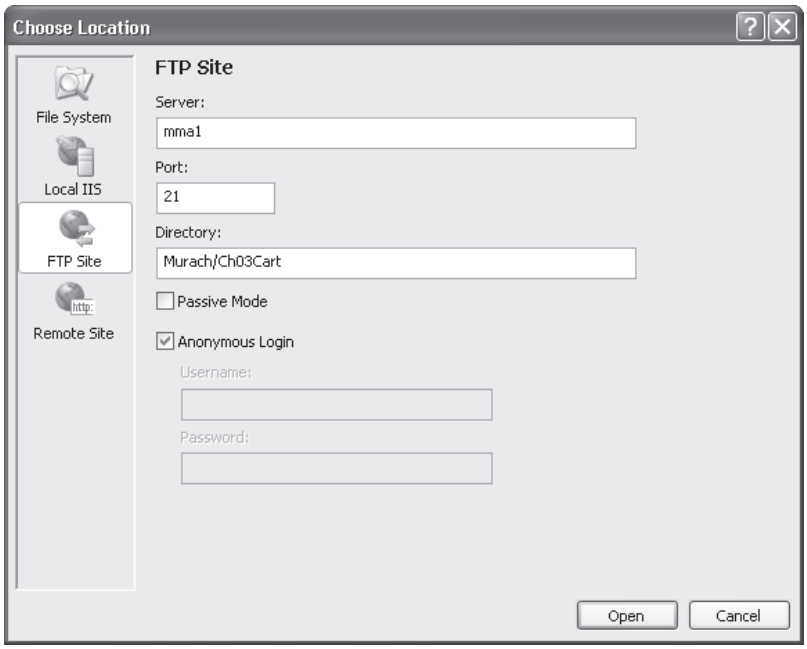
In some cases, an FTP server will require that you provide a username and password to connect to the server. Then, you'll need to remove the check mark from the Anonymous Login option and enter the required information in the Username and Password text boxes that become available. Note that because this information is saved until you end Visual Studio, you only need to enter it the first time you connect to the server during a Visual Studio session.

After you enter the required information into the Choose Location dialog box, you click the Open button to return to the New Web Site dialog box. When you do, the location will look something like this:

```
ftp://Murach/Ch03Cart
```

Then, you can just click the OK button to create the web site.

The dialog box for creating an FTP web site



Description

- To create a new FTP web site, select FTP from the Location drop-down list in the New Web Site dialog box. Then, click the Browse button to display the Choose Location dialog box shown above.
- Enter the name of the server and the directory where you want to create the web site. You can typically leave the port set at 21.
- Visual Studio can use either *active mode* or *passive mode* to establish connections to the FTP server. Active mode is the default. If the client is behind a firewall, though, you may need to use passive mode. To do that, select the Passive Mode option.
- By default, Visual Studio logs you in to the FTP server as an anonymous user. However, some FTP servers require you to provide a username and password. In that case, you can deselect the Anonymous Login option and then enter your username and password. The username and password are saved until you end the Visual Studio session.
- If you try to create a new FTP web site by entering a URL in the New Web Site dialog box, Visual Studio will display a dialog box that lets you specify whether you want to use passive mode and whether you want to log in as an anonymous user.
- IIS can be configured to act as an FTP server as well as a web server. For more information, please see appendix A.

Figure 4-3 How to create an FTP web site

How to test an ASP.NET application

To test an ASP.NET application, you typically start by running it from within Visual Studio so that you can locate and correct any errors you encounter. This initial testing uses the default browser and, if you're working with a file-system web site, the ASP.NET Development Server. Next, you test the application with other web browsers to make sure it works with them, and you test a file-system web site under IIS. Finally, you run the application from outside of Visual Studio to be sure it will work correctly in a production environment.

You'll learn the techniques for performing all of these types of testing in the topics that follow. In addition, you'll learn how to use the Exception Assistant dialog box that's displayed if an error occurs while you're testing an application.

How to test an application with the default browser

Unless you've changed it, Visual Studio uses Internet Explorer as its default browser. Figure 4-4 presents six different ways you can run a web application with the default browser. Three of these techniques start the debugger so you can use its features to debug any problems that might arise. The other three techniques do not start the debugger.

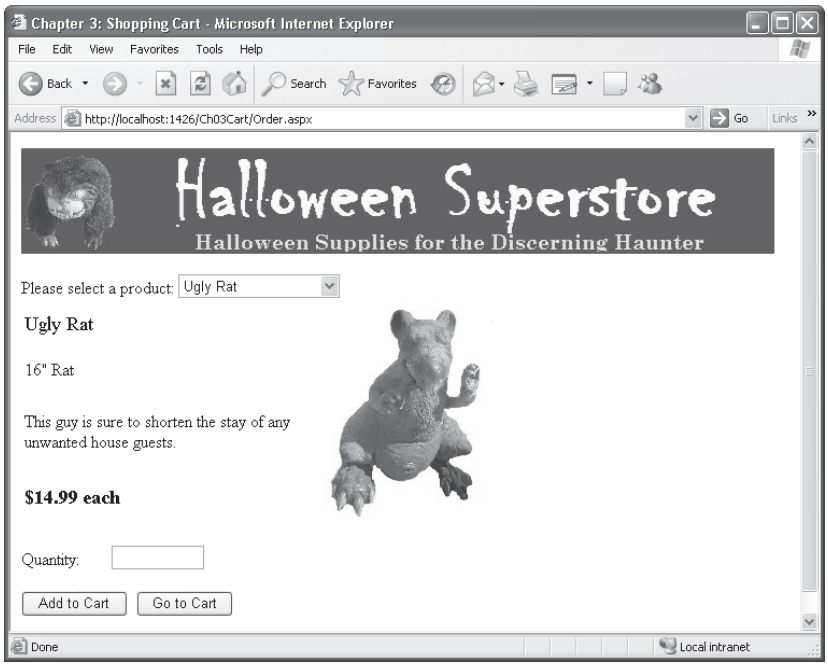
The first time you run a web application using one of the first three techniques, Visual Studio displays a dialog box indicating that debugging isn't enabled in the web.config file. From this dialog box, you can choose to add a new web.config file with debugging enabled, or you can choose to run the application without debugging. In most cases, you'll add a new web.config file with debugging enabled so that you can use the debugger with your application.

Before I go on, you should realize that before you can run an application with debugging on a remote server, a program called the *Remote Debug Monitor* must be installed and running on the server. For information on how to set up this program, see the help topic "How to: Set Up Remote Debugging."

All of the techniques listed in this figure except the View in Browser command start the application and display the application's designated start page. However, the View in Browser command displays the selected page. For example, if you right-click the Cart page and choose View in Browser, the Cart page will be displayed. This command is most appropriate for making sure that the pages of an application look the way you want them to.

At this point, you should realize that the type of web site you're developing determines the web server that's used to run the application. For example, if you're developing a file-system web site, the ASP.NET Development Server is used to run the application. In that case, the URL for the page that's displayed identifies the server as "localhost:" followed by a number that's assigned by the development server. In contrast, if you're developing an IIS web site, IIS is used to run the application. Then, the server is identified as just "localhost" for a local web site or by its actual name for a remote or FTP web site.

The Order page displayed in the default browser



Three ways to run an application with debugging

- Click the Start Debugging button in the Standard toolbar
- Press F5
- Choose the Debug➔Start command

Three ways to run an application without debugging

- Press Ctrl+F5
- Choose Debug➔Start Without Debugging
- Right-click a page in the Solution Explorer and choose View in Browser.

Three ways to stop an application that's run with debugging

- Press Shift+F5
- Click the Stop Debugging button in the Debug toolbar
- Choose Debug➔Stop Debugging

Description

- If you run an application with debugging, you can use Visual Studio's built-in debugger to find and correct program errors.
- By default, an application is run using Internet Explorer. See figure 4-5 for information on using a different browser. To end the application, close the browser.

Figure 4-4 How to test an application with the default browser

You should also realize that you can't just run an FTP application from its FTP location. That's because this location is used only to transfer files to and from the server. Instead, a *browse location* must be set up for the web site. A browse location is simply an IIS virtual directory that points to the directory that stores the files for the web site. Then, to access and run the web site, you enter the HTTP URL for the virtual directory. The first time you run an FTP web site, you will be asked to enter this URL. Because the URL is stored as part of the web site, you won't need to enter it again.

How to test an application with a browser other than the default

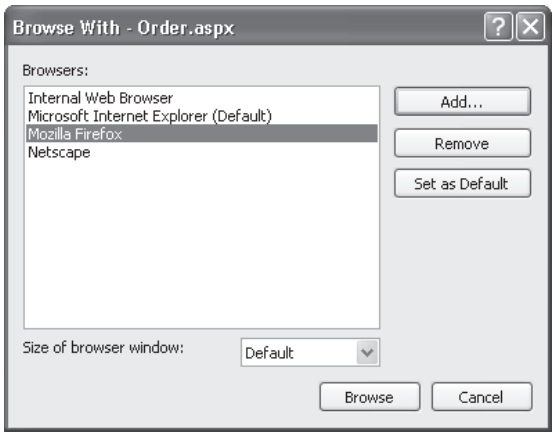
Once you've thoroughly tested your application with Internet Explorer, you'll want to test it with other browsers to make sure it works as expected. Figure 4-5 describes two ways to do that. First, you can right-click the starting page for the application in the Solution Explorer and choose the Browse With command. This displays a Browse With dialog box like the one shown in this figure. From this dialog box, you can choose the browser you want to use and then click the Browse button.

Notice that the list of browsers in the Browse With dialog box includes the Internal Web Browser. If you choose this browser, the page is displayed in a Browse window within Visual Studio. You can test the application from this browser just as you would from any other browser. Then, when you're done with the browse operation, just close the Browse window.

You can also test an application with another browser by making that browser the default browser. To do that, right-click any page in the Solution Explorer and choose Browse With. Next, select the browser you want to designate as your default browser and click the Set As Default button. Then, click the Cancel button to close the Browse With dialog box. You can then use any of the techniques listed in figure 4-4 to start the application in the browser you selected. Note that setting a browser as the default is the only way to use the debugger with the browser. That's because the Browse With command runs an application without debugging.

Sometimes, the browser you want to test an application with doesn't appear in the Browse With dialog box even though it's installed on your computer. In most cases, that's because you installed the browser after installing Visual Studio. Then, you can add the browser by clicking the Add button in the Browse With dialog box to display the Add Program dialog box. This dialog box lets you locate the executable file for the browser you want to add and enter a "friendly" name that's used for the browser in the Browse With dialog box.

The Browse With dialog box



Two ways to test an application with a browser other than the default

- Right-click the starting page for the application in the Solution Explorer and choose Browse With from the shortcut menu. In the Browse With dialog box that's displayed, select the browser you want to use and click the Browse button.
- Select the browser you want to use in the Browse With dialog box and then click the Set as Default button to make that browser the default. The next time you run the application, it will be displayed in the browser you selected.

Description

- It's important to test an ASP.NET application in any browsers that might be used to run the application.
- If a browser isn't included in the list of available browsers, you can add it to the list by clicking the Add button in the Browse With dialog box and then using the Add Program dialog box that's displayed to add the desired browser.
- If you select the Internal Web Browser option, the page is displayed in a Browse window within Visual Studio. To end the browse operation from this window, just close the window.
- You'll need to change the default browser if you want to use the debugger with another browser, since the Browse With command starts the application without debugging.

Figure 4-5 How to test an application with a browser other than the default

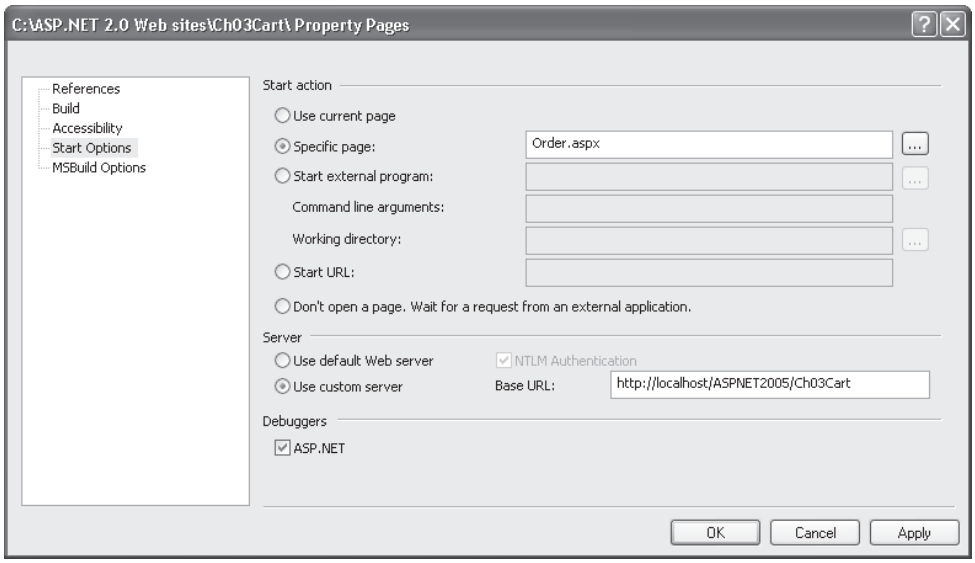
How to test a file-system web site with IIS

When you run a file-system web site, it runs under the ASP.NET Development Server by default. Because this server has limitations, however, you'll want to be sure to test a file-system web site under IIS as well as under the development server. Figure 4-6 describes how you do that.

To start, you need to create a virtual directory for the web site as described in figure 4-1. Then, if you open the web site from this virtual directory, it will automatically run under IIS. Alternatively, you can open the file-system web site directly and change its start options so the URL you specify is used to start the application. In this figure, for example, you can see that the Use Custom Server option has been selected and the URL for the web site's virtual directory has been entered in the Base URL text box.

This figure also lists the limitations of the ASP.NET Development Server. The most significant of these limitations is that it always runs under the current user's security context, but your own user account probably has stronger security privileges than the account IIS runs ASP.NET applications under. As a result, when you move the application to a production web server, you may have to contend with security issues that weren't apparent when you tested with the development server, especially if you access files or databases located in folders outside of the application's folder structure.

The dialog box for specifying a web server



How to test a file-system web site with IIS

- Create a virtual directory for the web site as described in figure 4-1.
- Open the web site from its virtual directory, or open the file-system web site and then use the Property Pages dialog box shown above to specify the URL for the virtual directory.
- Run the application using one of the techniques in figure 4-4.

Limitations of the ASP.NET Development Server

- Can serve pages only to the local computer.
- Runs in current user's security context, so it doesn't accurately test security issues.
- Does not include an SMTP server, so it can't be used to test email applications.
- Uses a randomly chosen port rather than the standard HTTP port 80.

Description

- By default, a file-system web site is run using the ASP.NET Development Server. To run a file-system web site using IIS, you can use one of the techniques above.
- To open the Property Pages dialog box, right-click the project in the Solution Explorer and select Property Pages from the shortcut menu. Then, to change the server that's used to run the web site, click the Start Options node, select the Use Custom Server option, and enter the URL of the virtual directory for the web site.

Figure 4-6 How to test a file-system web site with IIS

How to test an application from outside of Visual Studio

Once you've thoroughly tested and debugged an application from within Visual Studio, you'll want to run it from outside of Visual Studio to make sure it works properly. Figure 4-7 describes how you do that.

To start, you open the browse you want to use. In this example, I used Mozilla Firefox. Then, you enter the URL for the starting page of the application and press Enter. When you do, a request for the page is sent to the server and the resulting page is displayed in the browser.

Note that to run a local IIS web site, you use "localhost" in the URL to identify the server. The URL in this figure, for example, refers to the Order.aspx page of the Ch03Cart web site. This web site is located in the ASPNET2005 directory of the local IIS server. In contrast, to run a remote IIS web site, you specify the actual server name in the URL like this:

```
http://mma1/ASPNET2005/Ch03Cart/Order.aspx
```

And to run an FTP web site, you enter the browse location like this:

```
http://Murach/Ch03Cart/Order.aspx
```

If you haven't already tested an application from within Visual Studio in each browser that might be used to run the application, you should do that from outside Visual Studio. In addition, if the application retrieves and updates data in a database, you'll want to test it simultaneously in two browser windows. To understand why, you need to realize that after an application retrieves data from a database, it closes the connection to the database. Because of that, two or more users can retrieve the same data at the same time. This is called *concurrency*, and it can cause *concurrency errors* when the data is updated.

In section 3 of this book, you'll learn more about concurrency and how you avoid concurrency errors. For now, you should just realize that you'll need to test your applications to be sure that they handle concurrency problems.

The Order page displayed in a Mozilla Firefox browser outside of Visual Studio

Description

- You can run any IIS application from outside of Visual Studio. To do that, open the browser you want to use, enter the URL for the starting page of the application as shown above, and press Enter.
- When you run an application from outside of Visual Studio, you can run it in two or more browser windows simultaneously. That way, you can be sure that the application provides for any *concurrency errors* that can occur.
- If you want to use the debugger while testing for concurrency errors, you can run one instance of the application from within Visual Studio with debugging, and you can run another instance from outside of Visual Studio.

Figure 4-7 How to test an application from outside of Visual Studio

How to use the Exception Assistant

As you test an ASP.NET application, you may encounter runtime errors that prevent an application from executing. When that happens, an exception is thrown. In many cases, the application anticipates these exceptions and provides code to catch them and process them appropriately. If an exception is not caught, however, the application enters *break mode* and the *Exception Assistant* displays a dialog box like the one shown in figure 4-8.

As you can see, the Exception Assistant dialog box indicates the type of exception that occurred and points to the statement that caused the error. In many cases, this information is enough to determine what caused the error and what should be done to correct it. For example, the Exception Assistant dialog box in this figure indicates that an input string value was not in the correct format. The problem was encountered in this line of code for the Order page:

```
item.Quantity = Convert.ToInt32(txtQuantity.Text);
```

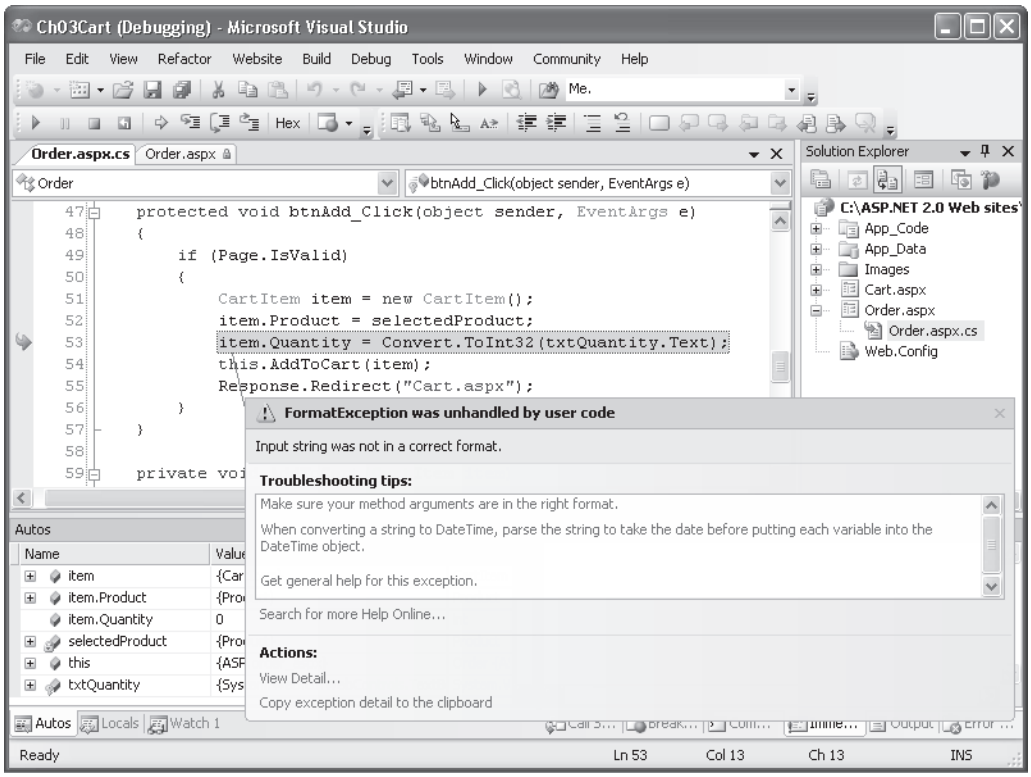
Based on that information, you can assume that the Text property of the txtQuantity control contains a value that can't be converted to an integer, since the Quantity property of the item object is declared as an integer. This could happen if the application didn't check that the user entered an integer value into this control. (To allow this error to occur, I disabled the range validator for the Quantity text box on the Order page.)

Many of the exceptions you'll encounter will be system exceptions like the one shown here. These exceptions apply to general system operations such as arithmetic operations and the execution of methods. If your applications use ADO.NET, you can also encounter ADO.NET and data provider exceptions. If the connection string for a database is invalid, for example, a data provider exception will occur. And if you try to add a row to a data table with a key that already exists, an ADO.NET error will occur.

In some cases, you won't be able to determine the cause of an error just by analyzing the information in the Exception Assistant dialog box. Then, to get more information about the possible cause of an exception, you can use the list of troubleshooting tips in this dialog box. The items in this list are links that display additional information in a Help window. You can also use the other links in this dialog box to search for more help online, to display the content of the exception object, and to copy the details of the exception to the clipboard.

If you still can't determine the cause of an error, you can use the Visual Studio debugger to help you locate the problem. You'll learn how to do that next.

An Exception Assistant dialog box



Description

- If an uncaught exception occurs in an ASP.NET application, the application enters *break mode* and the *Exception Assistant* displays a dialog box like the one shown above.
- The Exception Assistant provides the name and description of the exception, and it points to the statement in the program that caused the error. It also includes a list of troubleshooting tips. You can click on these tips to display additional information in a Help window.
- The information provided by the Exception Assistant is often all you need to determine the cause of an error. If not, you can close this window and then use the debugging techniques presented in this chapter to determine the cause.
- If you continue program execution after an exception occurs, ASP.NET terminates the application and sends a Server Error page to the browser. This page is also displayed if you run an application without debugging. It provides the name of the application, a description of the exception, and the line in the program that caused the error. It also includes a *stack trace* that indicates the processing that led up to the error.

Figure 4-8 How to use the Exception Assistant

How to use the debugger

The topics that follow introduce you to the basic techniques for using the Visual Studio *debugger* to debug an ASP.NET application. Note that these techniques are almost identical to the techniques you use to debug a Windows application. If you've debugged Windows applications, then, you shouldn't have any trouble debugging web applications.

How to use breakpoints

Figure 4-9 shows how to use *breakpoints* in an ASP.NET application. Note that you can set a breakpoint before you run an application or as an application is executing. Remember, though, that an application ends after it generates a page. So if you switch from the browser to Visual Studio to set a breakpoint, the breakpoint won't be taken until the next time the page is executed. If you want a breakpoint to be taken the first time a page is executed, then, you'll need to set the breakpoint before you run the application.

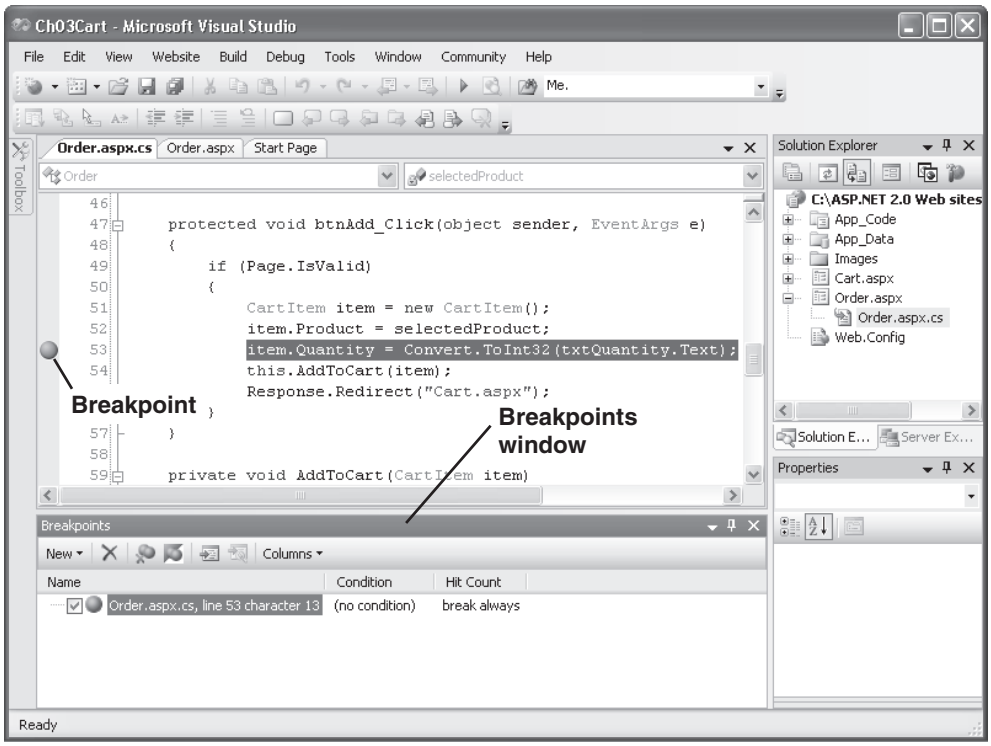
After you set a breakpoint and run the application, the application enters break mode before it executes the statement that contains the breakpoint. In this illustration, for example, the application will enter break mode before it executes the statement that caused the exception in figure 4-8 to occur. Then, you can use the debugging features described in the topics that follow to debug the application.

In some cases, you may want to set more than one breakpoint. You can do that either before you begin the execution of the application or while the application is in break mode. Then, when you run the application, it will stop at the first breakpoint. And when you continue execution, the application will execute up to the next breakpoint.

Once you set a breakpoint, it remains active until you remove it. In fact, it remains active even after you close the project. If you want to remove a breakpoint, you can use one of the techniques presented in this figure.

You can also work with breakpoints from the *Breakpoints window*. To disable a breakpoint, for example, you can remove the check mark in front of the breakpoint. Then, the breakpoint isn't taken until you enable it again. You can also move to a breakpoint in the Code Editor window by selecting the breakpoint in the Breakpoints window and then clicking on the Go To Source Code button at the top of this window, or by right-clicking the breakpoint in the Breakpoints window and choosing Go To Source Code from the shortcut menu.

The Order page with a breakpoint



How to set and clear breakpoints

- To set a *breakpoint*, click in the margin indicator bar to the left of the statement where you want the break to occur. The statement will be highlighted and a breakpoint indicator (a large dot) will appear in the margin. You can set a breakpoint before you run an application or while you're debugging the application.
- To remove a breakpoint, click the breakpoint indicator. To remove all breakpoints at once, use the Debug➔Clear All Breakpoints command.
- To temporarily disable breakpoints, use the Debug➔Disable All Breakpoints command. You can later enable the breakpoints by using Debug➔Enable All Breakpoints.

Description

- When ASP.NET encounters a breakpoint, it enters break mode before it executes the statement on which the breakpoint is set. From break mode, you can use the debugger to determine the cause of an error.
- The current breakpoints are listed in the *Breakpoints window* (Debug➔Windows➔ Breakpoints). You can use the toolbar at the top of this window to work with the breakpoints, and you can use the check box next to a breakpoint to enable or disable the breakpoint.
- You can't set breakpoints on blank lines or comments.

Figure 4-9 How to use breakpoints

How to use tracepoints

In addition to breakpoints, Visual Studio 2005 provides a new feature called *tracepoints*. A tracepoint is a special type of breakpoint that performs an action when it's encountered. Figure 4-10 shows how tracepoints work.

To set a tracepoint, you use the When Breakpoint Is Hit dialog box to indicate what you want to do when the tracepoint is encountered or "hit." In most cases, you'll use the Print a Message option to display a message in the *Output window*. As indicated in this dialog box, the message can include variable values and other expressions as well as special keywords.

For example, the message shown here will include the value of the SelectedValue property of the ddlProducts control. You can see the output from this tracepoint in the Output window in this figure. Here, the first tracepoint message was displayed the first time the page was requested. The second message was displayed when a product was selected from the drop-down list. And the third message was displayed when a quantity was entered and the Add to Cart button was clicked.

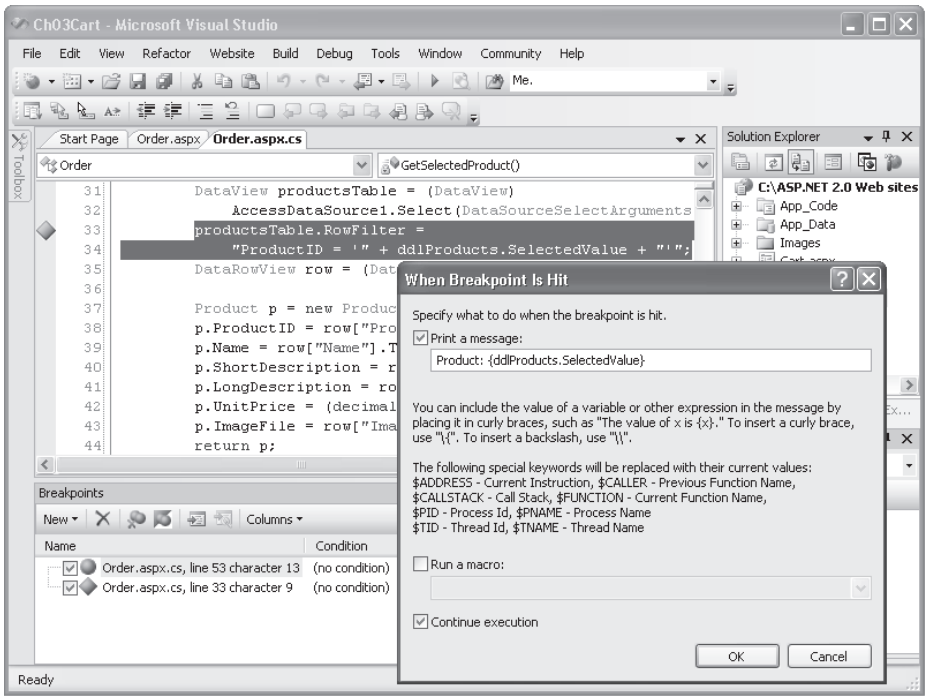
Notice that the Output window is also used to display Visual Studio messages like the first, second, and fifth messages shown in this figure. Because of that, this window is displayed automatically when you run an application. If you ever close it and want to reopen it without running the application, however, you can do that using the View➔Output command.

To run a macro when a tracepoint is encountered, you select the Run a Macro option. Then, the drop-down list becomes available and you can select the macro you want to run from this list.

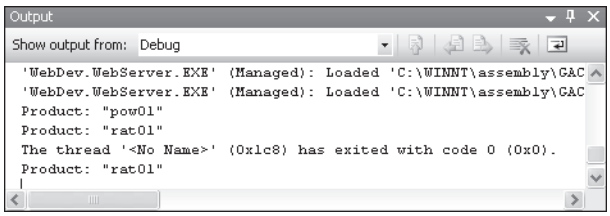
By default, program execution continues after the tracepoint action is performed. If that's not what you want, you can remove the check mark from the Continue Execution option. Then, the program will enter break mode when the tracepoint action is complete.

After you set a tracepoint on a statement, the statement will be highlighted and a breakpoint indicator will appear in the margin. If program execution will continue after the tracepoint action is performed, the indicator will appear as a large diamond. If the program will enter break mode, however, the same indicator is used as for a standard breakpoint.

The Order page with a tracepoint and the dialog box used to set it



Output from the tracepoint in the Output window



Description

- A *tracepoint* is a special type of breakpoint that lets you perform an action. When ASP.NET encounters a tracepoint, it performs the specified action and then continues execution if the Continue Execution option is selected or enters break mode if it isn't.
- You typically use tracepoints to print messages to the *Output window*. A message can include text, values, and special keywords. You can also use tracepoints to run macros.
- To set a tracepoint, right-click on the statement where you want it set and choose Breakpoint➔Insert Tracepoint. Then, complete the When Breakpoint Is Hit dialog box and click OK. You can also convert an existing breakpoint to a tracepoint by right-clicking on its indicator and choosing When Hit.
- If program execution will continue after the tracepoint action is performed, the tracepoint will be marked with a large diamond as shown above. Otherwise, it will be marked just like any other breakpoint.

Figure 4-10 How to use tracepoints

How to work in break mode

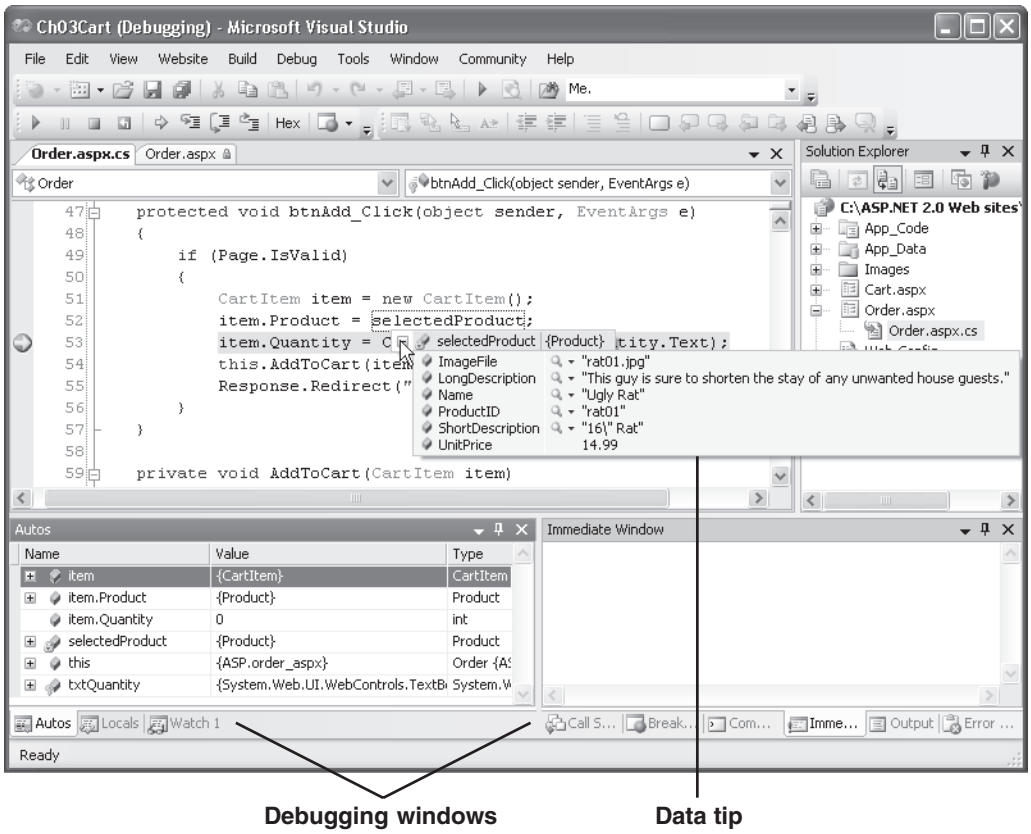
Figure 4-11 shows the Order page in break mode. In this mode, the next statement to be executed is highlighted. Then, you can use the debugging information that's available to try to determine the cause of an exception or a logical error.

For example, you can place the mouse pointer over a variable, property, or expression to display its current value in a *data tip*. You can also display the values of the members of an array, structure, or object. To do that, place the mouse pointer over the array, structure, or object to display its data tip, and then point to the plus sign in that data tip. In this figure, for example, you can see the current values of the members of the Product object named selectedProduct.

You can also use a data tip to change the value of a variable or property. To do that, just right-click the data tip and then choose Edit Value from the shortcut menu. When you do, the value that's displayed will become editable so you can enter a new value.

You can also see the values of other properties and variables in the Autos window near the bottom left of the Visual Studio window. You'll learn more about the Autos window and some of the other debugging windows in a minute.

The Shopping Cart application in break mode



Description

- When you enter break mode, the debugger highlights the next statement to be executed.
- You can use the debugging windows and the buttons in the Debug menu and toolbar to control the execution of the program and determine the cause of an exception.
- To display the value of a variable or property in a *data tip,* position the mouse pointer over the variable or property in the Code Editor window. To display a data tip for an expression, highlight the expression and then point to it. The expression must not contain a method call.
- To display the members of an array, structure, or object in a data tip, position the mouse pointer over it to display its data tip, and then point to the plus sign in the data tip.
- To change the value of a variable in a data tip, right-click the data tip, choose Edit Value, and then enter the new value.
- To continue program execution, press F5 or click the Continue button in the Standard or Debug toolbar. For more options about controlling program execution, see figure 4-12.

Figure 4-11 How to work in break mode

How to control the execution of an application

Once you're in break mode, you can use a variety of commands to control the execution of the application. These commands are summarized in figure 4-12. As you can see, most of these commands are available from the Debug menu or the Debug toolbar, but three of them are available only from the shortcut menu for the Code Editor window. You can also use shortcut keys to start some of these commands.

To execute the statements of an application one at a time, you use the Step Into command. When you use this command, the application executes the next statement, then returns to break mode so you can check the values of properties and variables and perform other debugging functions. The Step Over command is similar to the Step Into command, but it executes the statements in called methods without interruption (they are "stepped over").

The Step Out command executes the remaining statements in a method without interruption. When the method finishes, the application enters break mode before the next statement in the calling method is executed.

To skip over code that you know is working properly, you can use the Run To Cursor or Set Next Statement command. You can also use the Set Next Statement command to rerun lines of code that were executed before an exception occurred. And if you've been working in the Code Editor window and have forgotten where the next statement to be executed is, you can use the Show Next Statement command to move to it.

If your application gets caught in a processing loop so it keeps executing indefinitely without generating a page, you can force it into break mode by choosing the Debug➔Break All command. This command lets you enter break mode any time during the execution of an application.

Commands in the Debug menu and toolbar

Command	Toolbar	Keyboard	Function
Start/Continue		F5	Start or continue execution of the application.
Break All		Ctrl+Alt+Break	Stop execution and enter break mode.
Stop Debugging		Shift+F5	Stop debugging and end execution of the application.
Restart		Ctrl+Shift+F5	Restart the entire application.
Show Next Statement			Display the next statement to be executed.
Step Into		F11	Execute one statement at a time.
Step Over		F10	Execute one statement at a time except for called methods.
Step Out		Shift+F11	Execute the remaining lines in the current method.

Commands in the Code Editor window's shortcut menu

Command	Function
Run to Cursor	Execute the application until it reaches the statement that contains the insertion point.
Set Next Statement	Set the statement that contains the insertion point as the next statement to be executed.
Show Next Statement	Move the insertion point to the next statement that will be executed.

Description

- Once the application enters break mode, you can use the Step Into, Step Over, Step Out, and Run To Cursor commands to execute one or more statements and return to break mode.
- To alter the normal execution sequence of the application, you can use the Set Next Statement command. Just place the insertion point in the statement you want to execute next, issue this command, and click the Continue button to continue application execution.
- To stop an application that's caught in a loop, switch to the Visual Studio window and use the Debug➔Break All command.

Figure 4-12 How to control the execution of an application

How to use the Autos, Locals, and Watch windows to monitor variables

If you need to see the values of several application variables or properties, you can do that using the Autos, Locals, or Watch windows. By default, these windows are displayed in the lower left corner of the IDE when an application enters break mode. If they're not displayed, you can display them by selecting the appropriate command from the Debug➔Windows menu. Note that you can display up to four separate Watch windows.

The content of the Autos, Locals, and Watch windows is illustrated in figure 4-13. The difference between the Autos and Locals windows is in the amount of information they display and the scope of that information.

The *Locals window* displays information about the variables and controls within the scope of the current method. Since that includes information about the form and all of the controls on the form if the code in a form is currently executing, that information can be extensive.

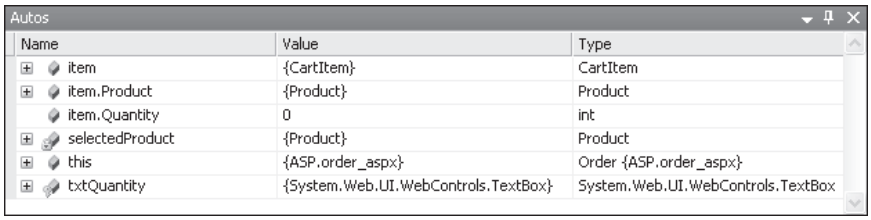
In contrast, the *Autos window* displays information about the variables, properties, and constants used in the current statement, the three statements before that statement, and the three statements after that statement. Although the information in this window is more limited than the information shown in the Locals window, the Autos window helps you focus on the variables that are relevant to the current statement.

Unlike the Autos and Locals windows, the *Watch windows* let you choose the values that are displayed. The Watch window shown in this figure, for example, displays the Text property of the txtQuantity and lblUnitPrice controls, the Quantity property of the item object, and the UnitPrice property of the Product object that's stored in the item object. The Watch windows also let you watch the values of expressions you specify. Note that an expression doesn't have to exist in the application for you to add it to a Watch window.

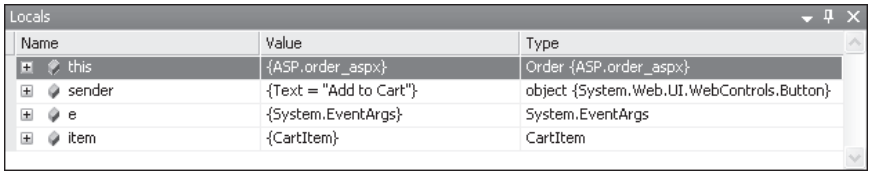
To add an item to a Watch window, you can type it directly into the Name column. Alternatively, if the item appears in the Code Editor window, you can highlight it in that window and then drag it to a Watch window. You can also highlight the item in the Code Editor or a data tip and then right-click on it and select the Add Watch command to add it to the Watch window that's currently displayed.

Besides displaying the values of variables and properties, you can use the Autos, Locals, and Watch windows to change these values. To do that, you simply double-click on the value you want to change and enter a new value. Then, you can continue debugging or continue the execution of the application.

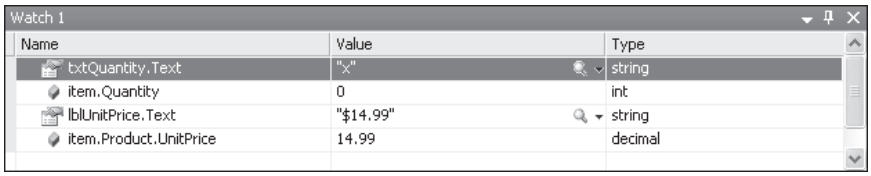
The Autos window



The Locals window



A Watch window



Description

- The *Autos window* displays information about variables, properties, and constants in the current statement and the three statements before and after the current statement.
- The *Locals window* displays information about the variables and controls within the scope of the current method.
- The *Watch windows* let you view the values of variables and expressions you specify, called *watch expressions*. You can display up to four Watch windows.
- To add a watch expression, type a variable name or expression into the Name column, highlight a variable or expression in the Code Editor window and drag it to the Watch window, or right-click on a variable or highlighted expression in the Code Editor window or a data tip and choose Add Watch.
- To delete a row from a Watch window, right-click the row and choose Delete Watch. To delete all the rows in a Watch window, right-click the window and choose Select All to select the rows, then right-click and choose Delete Watch.
- To display any of these windows, click on its tab if it's visible or select the appropriate command from the Debug➔Windows menu.
- To change the value of a property or variable from any of these windows, double-click on the value in the Value column, then type a new value and press the Enter key.

Figure 4-13 How to use the Autos, Locals, and Watch windows to monitor variables

How to use the Immediate window to work with values

The *Immediate window*, shown in figure 4-14, is useful for displaying the values of variables or properties that don't appear in the Code Editor window. To display a value, you simply type a question mark followed by the name of the variable or property. The first line of code in this figure, for example, displays the Text property of the item selected from the Products drop-down list. You can see the result in the second line of this window.

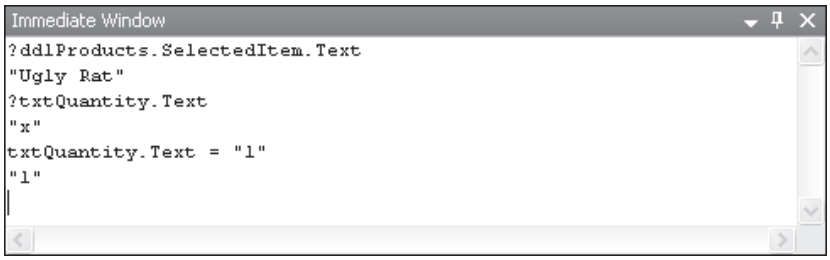
The Immediate window is also useful for executing C# statements. For example, you can execute an assignment statement to change the value of a variable or property. After I displayed the Text property of the Quantity text box, for example, I assigned a value of 1 to this property. Similarly, you can execute a user-defined method. This can be useful for testing the result of a method with different arguments. If you execute a method that returns a value, you can also preface the method name with a question mark to display the value it returns.

When you enter commands in the Immediate window, they're executed in the same context (or scope) as the application that's running. That means that you can't display the value of a variable that's out of scope. If you try to do that, the debugger displays an error message.

The commands that you enter into the Immediate window remain there until you exit from Visual Studio or explicitly delete them using the Clear All command in the shortcut menu for the window. That way, you can use standard Windows techniques to edit and reuse the same commands from one execution of an application to another without having to reenter them.

To execute a command that you've already entered in the Immediate window, just use the Up and Down arrow keys to scroll through the commands. As you scroll, the commands are displayed at the bottom of the window. Then, you can change a command if necessary and press Enter to execute it.

The Immediate window



Description

- You can use the *Immediate window* to display and assign values from a program during execution. To display this window, click on the Immediate Window tab or use the Debug➔Windows➔Immediate command.
- To display a value in the Immediate window, enter a question mark followed by the expression whose value you want to display. Then, press the Enter key.
- To assign a different value to a variable, property, or object, enter an assignment statement in the Immediate window. Then, press the Enter key.
- To execute a user-defined method from the Immediate window, enter its name and any arguments it requires. Then, press the Enter key. If you want to display the value that's returned by a method, precede the method call with a question mark.
- To reissue a command, use the Up and Down arrow keys to scroll through the commands until you find the one you want. Then, modify the command if necessary and press the Enter key to execute it.
- To remove all commands and output from the Immediate window, use the Clear All command in the shortcut menu for the window.

Figure 4-14 How to use the Immediate window to work with values

How to use the Trace feature

The *Trace feature* is an ASP.NET feature that displays some useful information that you can't get by using the debugger. Because the debugger works so well, you probably won't need to use the Trace feature very much, but you should at least be aware of the information that it can provide.

How to enable the Trace feature

To use the Trace feature, you must first enable tracing. To do that, you add a Trace attribute to the Page directive for the page, and you assign a value of True to this attribute. Then, when you run the page, trace information will be displayed at the end of the page output, as shown in figure 4-15.

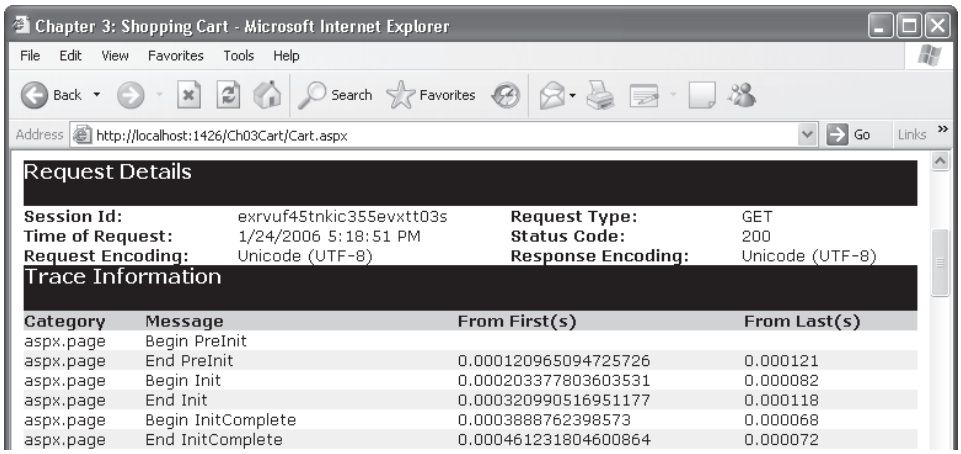
When you enable the Trace feature, it is enabled only for the current page, which is usually what you want. To enable tracing for another page, you must modify the Page directive for that page too. Once this feature has been enabled for a page, ASP.NET adds trace output to the page whenever the page is requested.

How to interpret Trace output

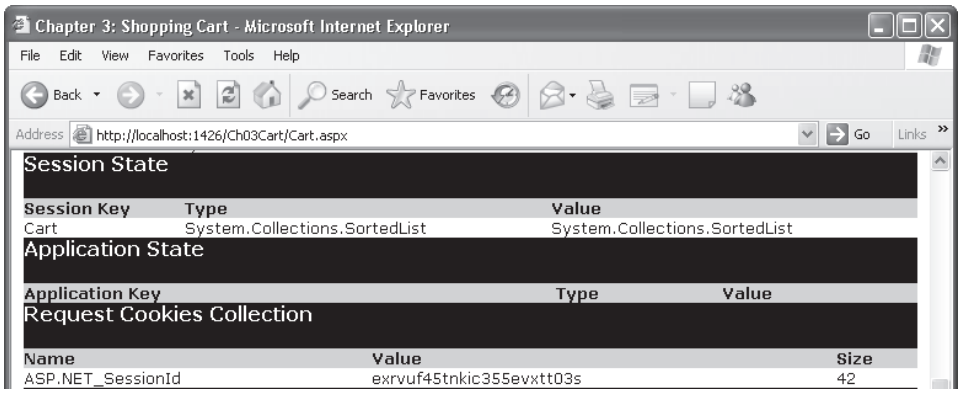
In figure 4-15, you can see the start of the output for the Cart page after the user added an item to the shopping cart. After the request details, the trace information provides a list of trace messages that are generated as the application executes. Here, ASP.NET automatically adds Begin and End messages when major page events such as PreInit, Init, and InitComplete occur. If you scroll down to see all of these trace messages, you can see the variety of events that are raised during the life cycle of a page.

After the trace messages, you'll find information about the controls used by the page, the items in the session state object, the cookies that were included with the HTTP request, the HTTP request headers, and the server variables. In this figure, for example, you can see the session state and cookies data for the Cart page of the Shopping Cart application. In this case, an item named Cart has been added to the session state object. And a cookie named ASP.NET_SessionId is used to keep track of the user's session ID so the user's session state object can be retrieved.

The beginning of the trace output for the Cart page



The session and cookies information for the Cart page



A Page directive that enables tracing for the Cart page

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Cart.aspx.cs"
Inherits="Cart" Trace="True" %>
```

Description

- The ASP.NET *Trace feature* traces the execution of a page and displays trace information and other information at the bottom of that page.
- To activate the trace feature for a page, you add a Trace attribute to the Page directive at the top of the aspx file for the page and set its value to True as shown above.
- The trace information is divided into several tables that provide specific types of trace information. For example, the Trace Information table provides information about how the page request was processed, and the Session State table provides information about the items currently stored in session state.

Figure 4-15 How to enable the Trace feature and interpret Trace output

How to create custom trace messages

In some cases, you may want to add your own messages to the trace information that's generated by the Trace feature. This can help you track the sequence in which the methods of a form are executed or the changes in the data as the methods are executed. Although you can also do this type of tracking by stepping through the methods of a form with the debugger, the trace information gives you a static listing of your messages.

Note, however, that you can also create this type of listing using tracepoints as described earlier in this chapter. The advantage to using tracepoints is that you can generate trace information without adding code to your application. In addition, this output is generated only when you run an application with debugging. In contrast, you have to add program code to add custom trace messages, and the trace output is generated whenever the Trace feature is enabled. Because of that, you may not want to create custom trace messages. But I've included it here in case you do.

To add messages to the trace information, you use the Write or Warn method of the TraceContext object. This is summarized in figure 4-16. The only difference between these two methods is that messages created with the Warn method appear in red. Notice that to refer to the TraceContext object, you use the Trace property of the page.

When you code a Write or Warn method, you can include both a category and a message or just a message. If you include just a message, the category column in the trace output is left blank. If you include a category, however, it appears as shown in this figure. In most cases, you'll include a category because it makes it easy to see the sequence in which the methods were executed.

If you want to determine whether tracing is enabled before executing a Write or Warn method, you can use the IsEnabled property of the TraceContext object as shown in the example in this figure. Normally, though, you won't check the IsEnabled property because trace statements are executed only if tracing is enabled.

Common members of the TraceContext class

Property	Description
IsEnabled	True if tracing is enabled for the page.
Method	**Description**
Write(message)	Writes a message to the trace output.
Write(category, message)	Writes a message to the trace output with the specified category.
Warn(message)	Writes a message in red type to the trace output.
Warn(category, message)	Writes a message in red type to the trace output with the specified category.

Code that writes a custom trace message

```
if (Trace.IsEnabled)
    Trace.Write("Page_Load", "Binding products drop-down list.");
```

A portion of a trace that includes a custom message

Trace Information

Category	Message	From First(s)	From Last(s)
aspx.page	Begin PreInit		
aspx.page	End PreInit	0.000114260331969566	0.000114
aspx.page	Begin Init	0.00019499685015833	0.000081
aspx.page	End Init	0.000337752423841578	0.000143
aspx.page	Begin InitComplete	0.000406476242092221	0.000069
aspx.page	End InitComplete	0.000478831806835785	0.000072
aspx.page	Begin PreLoad	0.000545879434397388	0.000067
aspx.page	End PreLoad	0.000634158810686833	0.000088
aspx.page	Begin Load	0.000704000089396837	0.000070
Page_Load	Binding products drop-down list.	0.00367448935549071	0.002970
aspx.page	End Load	0.00780099146679257	0.004127
aspx.page	Begin LoadComplete	0.00797559466356758	0.000175
aspx.page	End LoadComplete	0.00804683276785178	0.000071
aspx.page	Begin PreRender	0.00811583595121726	0.000069
aspx.page	End PreRender	0.00827116295506831	0.000155

Custom trace message

Description

- You can use the TraceContext object to write your own messages to the trace output. The TraceContext object is available through the Trace property of a page.
- Use the Write method to write a basic text message. Use the Warn method to write a message in red type.
- Trace messages are written only if tracing is enabled for the page. To determine whether tracing is enabled, you use the IsEnabled property of the TraceContext object.

Figure 4-16 How to create custom trace messages

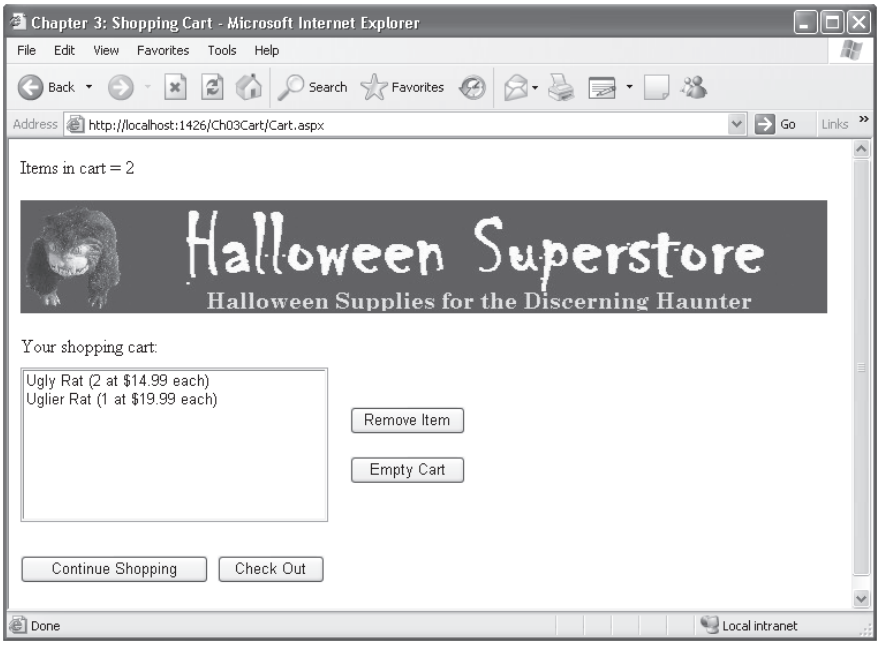
How to write information directly to the HTTP output stream

Another way to display information as a program executes is to write it directly to the HTTP output stream. To do that, you use the Write method of the HTTP Response object as shown in figure 4-17. When you use this technique, you'll want to be sure to remove any statements you've added when you finish testing your application.

At the top of this figure, you can see a Cart page that includes output that indicates the number of items that are currently in the shopping cart. To generate this output, I added a Response.Write method to the Page_Load event handler of the page. As you can see, this event handler uses the Count property of the Cart object to determine the number of items that are in the cart.

Notice that the text you include on the Write method can include HTML tags. For example, the Write method shown here includes a
 tag so that the item count is followed by a blank line. Also note that the output you write to the output stream is always added to the beginning of the stream. Because of that, it always appears at the top of the browser window.

The Cart page with output generated by Response.Write



A Page_Load event handler that writes to the HTTP output stream

```
protected void Page_Load(object sender, EventArgs e)
{
    this.GetCart();
    if (!IsPostBack)
    {
        this.DisplayCart();
        Response.Write("Items in cart = " + cart.Count + "<br />");
    }
}
```

Code that writes HTML output from another class

```
HttpContext.Current.Response.Write("Now updating file.<br />");
```

Description

- The Write method of the HttpResponse object provides a convenient way to write data directly to the HTTP output stream. The output can include any valid HTML.
- To access the HttpResponse object from the code-behind file for a web page, you use the Response property of the page. To access this object from a class that doesn't inherit the Page class, you use the Response property of the HttpContext object for the current request. To access this object, you use the Current property of the HttpContext class.
- The HTML output you add to the HTTP output stream using the Write method of the HttpResponse object appears at the beginning of the output stream. As a result, the output from the Write method always appears at the top of the page in the browser window.

Figure 4-17 How to write information directly to the HTTP output stream

Perspective

As you can now appreciate, Visual Studio provides a powerful set of tools for debugging ASP.NET applications. For simple applications, you can usually get the debugging done just by using breakpoints, data tips, and the Autos window. You may also need to step through critical portions of code from time to time. For complex applications, though, you may discover the need for some of the other features that are presented in this chapter. With tools like these, a difficult debugging job becomes manageable.

Terms

virtual directory
virtual root
active mode
passive mode
Remote Debug Monitor
browse location
concurrency
concurrency error
break mode
Exception Assistant
stack trace
debugger
breakpoint
Breakpoints window
tracepoint
Output window
data tip
Immediate window
Autos window
Locals window
Watch window
watch expression
Trace feature