ASP.NET 2.0 Revealed

PATRICK A. LORENZ

Apress™

ISBN (pbk): 1-59059-337-5

Printed and bound in the United States of America 12345678910

Technical Reviewer: Marc Höppner

Editorial Board: Dan Appleman, Craig Berry, Gary Cornell, Tony Davis, Steven Rycroft, Julian Skinner, Martin Streicher, Jim Sumser, Karen Watterson, Gavin Wray, John Zukowski

Assistant Publisher: Grace Wong

Project Manager: Kylie Johnston

Copy Editors: Ami Knox, Nicole LeClerc

Production Manager: Kari Brooks

Production Editor: Laura Cheu

Proofreaders: Lori Bring, Linda Seifert

Compositor: Diana Van Winkle, Van Winkle Design Group

Indexer: Nancy A. Guenther

Cover Designer: Kurt Krames

Manufacturing Manager: Tom Debolski

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 175 Fifth Avenue, New York, NY 10010 and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.

In the United States: phone 1-800-SPRINGER, email orders@springer-ny.com, or visit http://www.springer-ny.com. Outside the United States: fax +49 6221 345229, email orders@springer.de, or visit http://www.springer.de.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, email info@apress.com, or visit http://www.apress.com.

The source code for this book is available to readers at http://www.apress.com in the Downloads section. You will need to answer questions pertaining to this book in order to successfully download the code.

CHAPTER 2

Introducing VS .NET for Web Developers

Do you already know Visual Studio .NET (VS .NET)? Sure, you do! And what about Web Matrix? The new version of Visual Studio .NET for Web Developers now combines the best features of Web Matrix and good old VS .NET. Moreover, it extends the set of functions with a bunch of features that make development of web applications (called "web sites" in the new terminology) as simple as possible.

So why a completely updated development environment? There are several good reasons. Neither Web Matrix nor the current version of VS .NET has been really optimized for web site development. VS .NET was targeted to desktop development. Support of web projects was available, but especially in large projects it was annoying. For example, the Microsoft HTML editors' bad habit of defacing your code was perfectly integrated here. And although Web Matrix offered some smart new features, it wasn't appropriate for larger projects at all, because some important functions such as IntelliSense, source control, or debugging weren't available.

The new version of VS .NET fills in these gaps and provides web developers with an environment especially optimized for their needs. Here's a list of some of the features offered by the new program:

- With "project-less" development, project files are no longer necessary. The integrated development environment (IDE) gets all the required information directly out of the directory.

- Besides real IIS projects, you're now able to store data directly in the file system and access the projects via FTP and FrontPage Server Extensions.

- Thanks to the integrated web server (have a look at "Cassini"), Internet Information Services (IIS) is no longer required during development, but it's still supported, of course.

- VS .NET can handle inline source code within the ASPX file, and it offers a revised code-behind model called code-beside.

- Regardless of where editing takes place, full support of IntelliSense and debugging is available.

- The new editor makes sure that manually entered HTML source code will remain unchanged; custom formatting and designs won't be lost.

- An exchangeable validation engine allows developers to check the HTML source code against standards such as Extensible Hypertext Markup Language (XHTML). Failures are entered into the Task List.

- In the final release, the output of all server controls will be clean XHTML.

Figure 2-1 shows an integrated sample web site project being edited within VS .NET. At first sight, the differences between the old and new versions seem to be small, but after a few minutes working with the IDE you'll discover that the new environment is easier and more manageable, and offers better support than the old version.

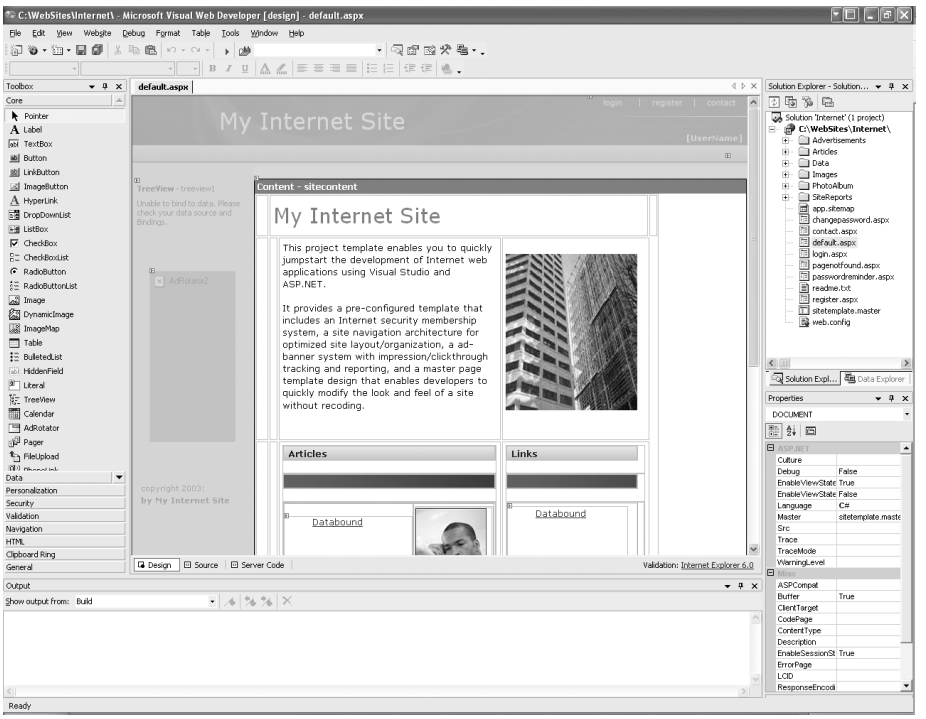

Figure 2-1. Here's the new Visual Studio .NET at work.

Installing VS .NET

VS .NET's installation process is self-explanatory. Please note that all existing IIS applications are mapped to new Internet Server Application Programming Interface (ISAPI) extension during installation of the current Alpha. As far as I know, this will be fixed in the Beta. Existing installations of an older framework, VS .NET 2002, or VS .NET 2003 don't cause any problems. Currently I use all three versions on one machine.

After you install VS .NET, a new group, Microsoft Visual Studio Whidbey, is available in your Start menu. Start it by clicking the Microsoft Visual Studio codename Whidbey link. The IDE's documentation is accessible via the Microsoft Visual Studio 2003 Documentation link. The IDE's documentation describes the improvements in ASP.NET and introduces the new VS .NET version. If you've installed the new .NET Framework SDK, you're able to start the voluminous reference separately via a link in the newly created Microsoft .NET Framework SDK v1.2 group.

As you can see in the group description and in the file system, the .NET Framework version is 1.2. There are historical reasons for using this version number, but the next version number will be 2.0 in the RTM version (it's probably 2.0 already in the Beta version).

NOTE *If you're using Windows Server 2003, you'll need to enable the ASP.NET 1.2 (2.0) ISAPI extension manually. Start the Internet Service Manager, choose Web Service Extensions, activate ASP.NET v1.2 ISAPI, and click Enable.*

Creating and Opening Web Sites

In this section, I cover how to create a new web site and open existing ones with the new version of VS .NET. Also, I show you how to migrate applications created with the older version of the IDE.

Creating a new web site within VS .NET has become quite easy. To do so, you can use the local IIS 5.0, 5.1, or 6.0. Alternatively, storing the web site directly in your local file system and accessing it via FrontPage Server Extensions and even via FTP is possible.

Creating a New Web Site

The New Web Site dialog box (which you access through the File menu) offers all opportunities mentioned previously, as shown in Figure 2-2. When you click the Browse button, the Choose Location dialog box appears (see Figure 2-3), from which you can select the site's storage location. If you choose File System, you can create new directories. If you choose IIS Local, you can define new applications and virtual directories at the local IIS level.

VS .NET includes several sample projects. Some of these projects are pretty extensive, and you can use them as starting points for your own web sites or to discover the new features of ASP.NET v2.0:

- The project template ASP.NET Internet Site provides a web site along with a secured area, including registration and login. It demonstrates the new data source concept in action. The required data is provided by static XML files and by an included Access database. Access also acts as the storage location for user management.

- The project template ASP.NET Intranet Site is, as its name implies, targeted to intranet applications. The template's main goal is to reveal the possibilities of personalization. The example shows the visual customization of the displayed information based on the authenticated user called Web Parts.

- The templates ASP.NET Web Site and Empty Web Site are very similar at this point. The first one creates a single page automatically; the second one doesn't.

- Last but not least, the project template ASP.NET Web Services is used to create a new web services web site. Of course, you can also add web services to any other existing project.

All of the previously described templates are available in C# and Visual Basic .NET (VB .NET) versions. This isn't a big deal, however, because the templates don't require much source code thanks to the new zero-code scenarios. Samples in J# aren't available yet.

Something different from previous approaches is that there's no special project file generated at the time of project creation. From now on, every file within the selected directories belongs to the project automatically. This approach simplifies adding new files, including references to the bin directory, and moving projects. Distributing a complete project is now a job for `XCopy`. If you work in a team, you'll love this new approach. Besides the project files, VS .NET still offers the old solution files.

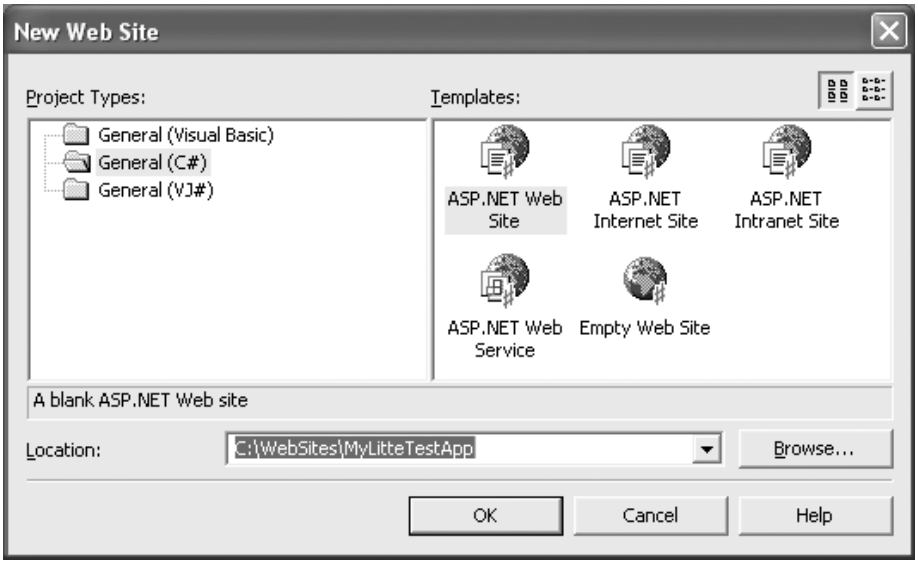

Figure 2-2. Creating a new web site is quite simple nowadays.

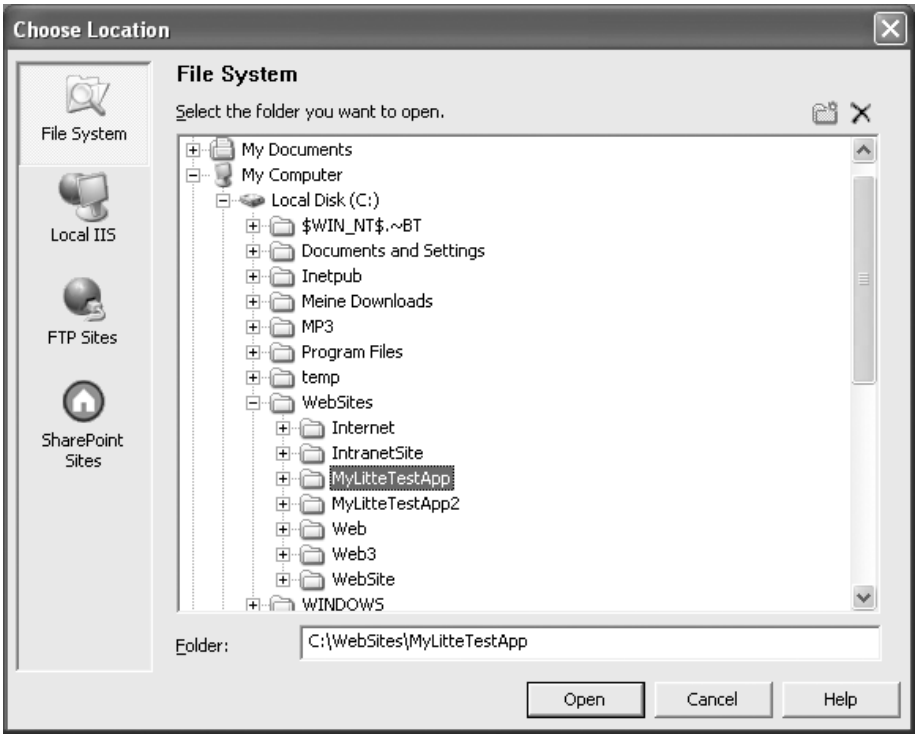

Figure 2-3. In the Choose Location dialog box, you choose among the File System, IIS Local, FTP Sites, or SharePoint Sites options.

Opening an Existing Web Site

Opening a web site is very easy too through VS .NET's Open Web Site dialog box. The Open Web Site dialog box shown in Figure 2-4 is similar to the New Web Site dialog box shown in Figure 2-1. It offers direct access to the file system, the local IIS, an FTP server, and any server offering FrontPage Server Extensions (mentioned as SharePoint Sites in the dialog box). You can also handle projects managed by Visual SourceSafe.

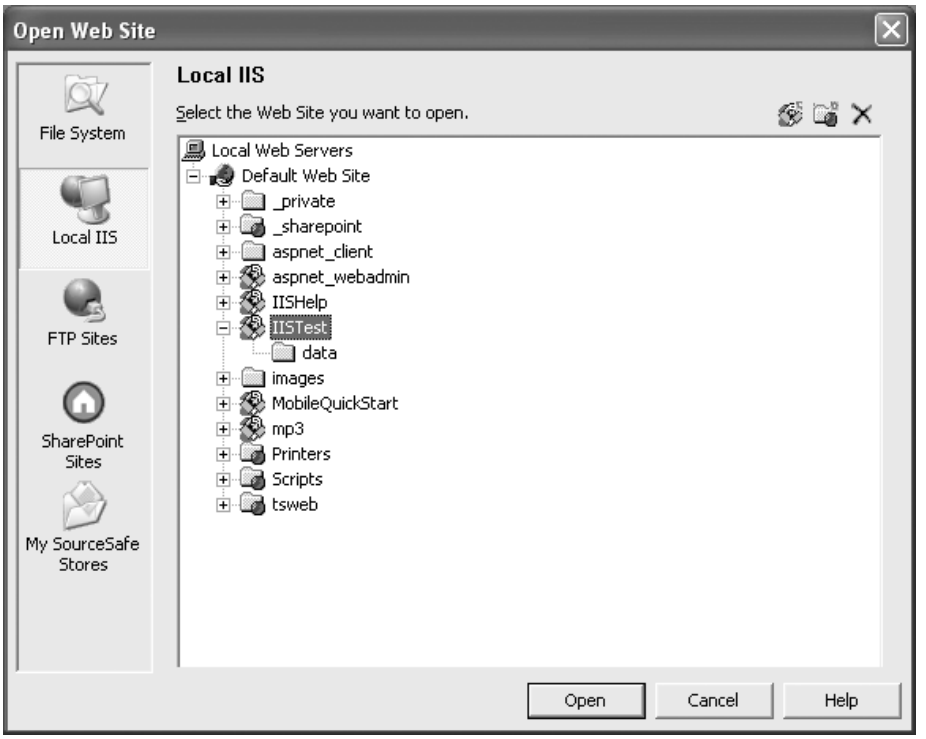

Figure 2-4. To open a web site, just point to the corresponding directory in the Open Web Site dialog box.

Migrating an Existing VS .NET 2002/2003 Web Application

Changes in the project management features are among the reasons that you need to migrate VS .NET 2002 and 2003 applications before you use them in the new version. After you confirm your intention to migrate, VS .NET performs the whole conversion without requiring further input from you. All changes are logged in the `UpgradeReport.txt` file. I didn't encounter any problems during my tests.

> **CAUTION** *Keep in mind that migration works in one direction only, and the old VS .NET versions can't handle a migrated project. Make sure to create backup before you start the migration process—VS .NET doesn't create one automatically for you!*

Editing Web Sites

Now it's time to play with the new VS .NET version. Let's go! Start with a new ASP.NET Web Site project. The first thing you'll see is an almost empty ASP.NET page—only the HTML body is in there.

When you take a look at the generated page, you'll realize that the new VS .NET works without code-behind. This shouldn't upset you too much—code-behind is still supported in an updated and slightly different way. The IDE offers several views of a single file, a concept you may already be familiar with from Web Matrix. Three tab buttons allow you to switch between the different views:

- The Design button shows the visual editor.
- The Source button provides the whole file, including the server-side source code.
- The Server Code button extracts the first `<script>` block of the file and displays it.

By default, the window appears in Source mode.

To create a new page, just right-click the project node in Solution Explorer and then select Add new Item. Alternatively, you can choose the same command from the Website menu. By the way, instead of naming the newly created pages webform?.aspx, VS .NET names them default?.aspx. Makes sense, doesn't it?

Placing Server Controls

The editor now supports adding server controls in Design view as well as in Source view. Just drag the control out of the Toolbox and drop it at the desired location. The current cursor position and selection (if one exists) remain unaffected when you change the view. In both views you can rely on the Properties window to manipulate a control. Don't be surprised if you edit a tag in Source view. There is an extended version of IntelliSense at your service.

The new version of VS .NET answers the prayers of a lot of developers: It doesn't pick your manually entered HTML code to pieces as the previous versions did. Changes will be made in the particular context only—for example, if a new control is inserted. Even if you switch between the views, there will be no changes. Now go and celebrate!

If a row in the source view is modified by the environment or by the developer, it's marked in yellow next to the row number, as shown in Figure 2-5. As changes are saved, the mark changes from yellow to green.

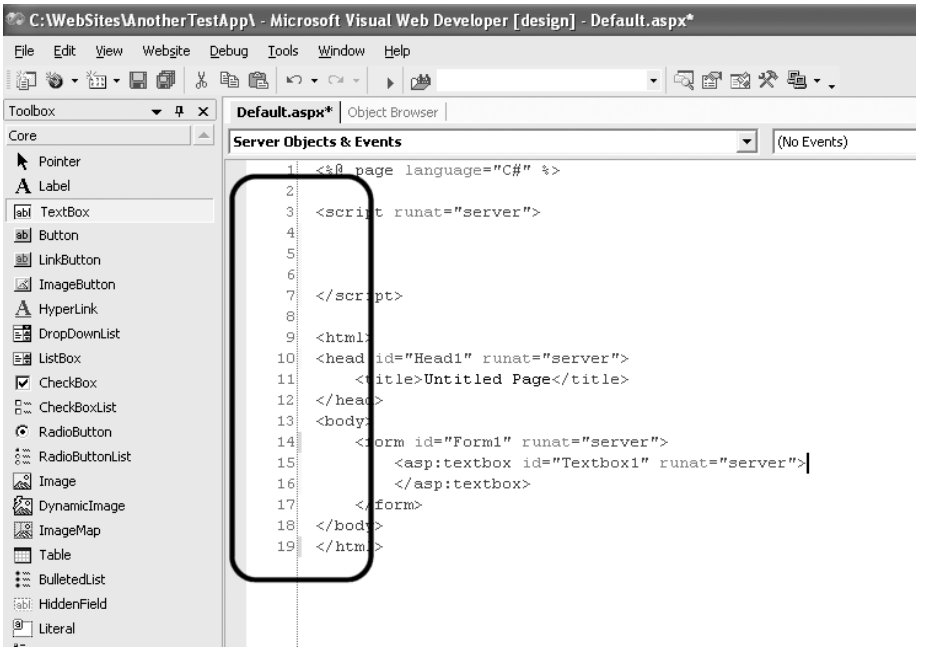

Figure 2-5. VS .NET now fully preserves your HTML code.

You'll find something new in the Toolbox, too: structure. Because the number of controls has increased dramatically, the Toolbox is split into several categories: Core, Data, Personalization, Security, Validation, Navigation, and HTML.

Editing Controls

So what about editing controls in VS .NET? It's hard to believe, but this has become much more convenient, too. With few exceptions, you don't need to edit control parameters in the HTML source code. You can change almost everything in Design view.

Smart Tags and Data Binding

Another new VS .NET feature is smart tags, which are available for all controls with assigned design-time actions—the so-called verbs. Clicking a smart tag opens a context menu with all the provided options, as shown in Figure 2-6. By the way, insiders call this *task-based editing*.

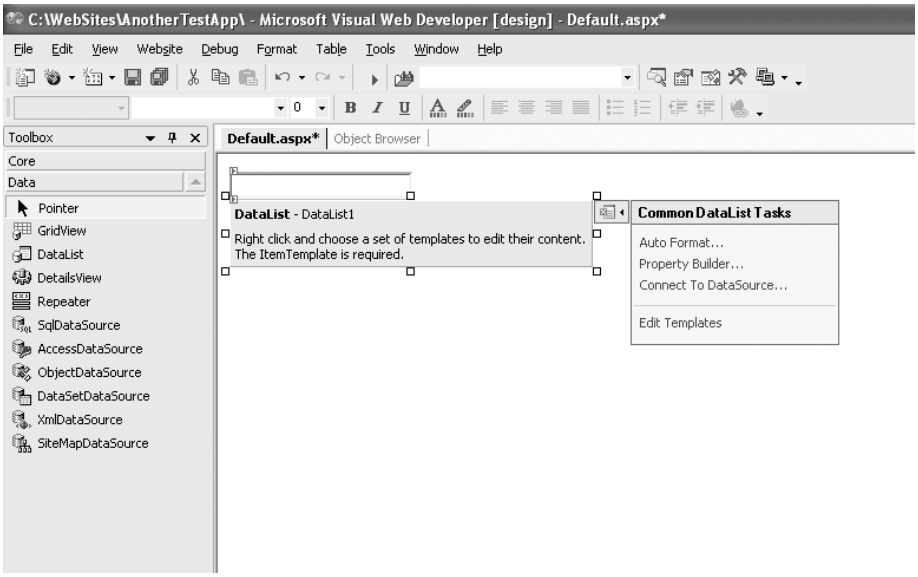

Figure 2-6. VS .NET now supports smart tags.

Smart tags are a pretty cool feature now that you can edit virtually any template in Design view. In the smart tag menu of the data control, you can choose Edit Templates. The list allows you to select one template or even a whole group of templates. Just fill the templates with some text and controls as usual. Figure 2-7 shows the new feature in action.

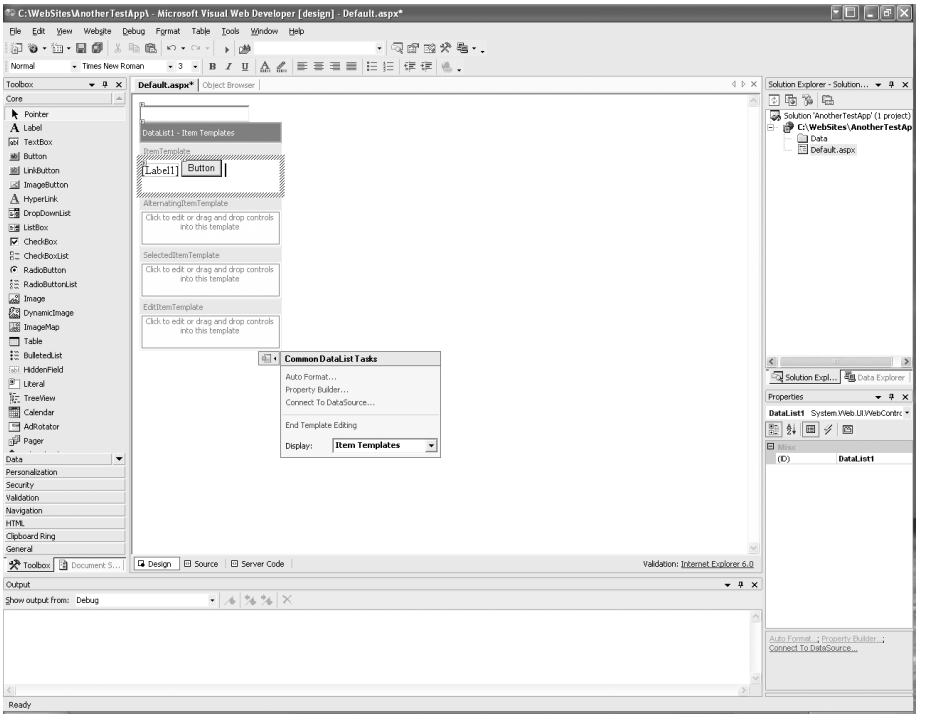

Figure 2-7. Editing templates is much easier now.

Don't worry about specifying the data-binding expression in the HTML source code, which can be challenging sometimes. Each and every control in a template features an extended smart tag menu. Select Edit DataBindings and a dialog box pops up that allows you to enter or edit the binding. As Figure 2-8 shows, you can either bind selected properties to a data source or enter an expression to be evaluated as usual.

If you look closely at Figure 2-8, you should be able to figure out which expression I've used for data binding. The syntax has been significantly shortened and therefore simplified compared to the old versions. Until now the syntax looked like this:

```
<%# DataBinder.Eval(Container.DataItem, "MyField") %>
```

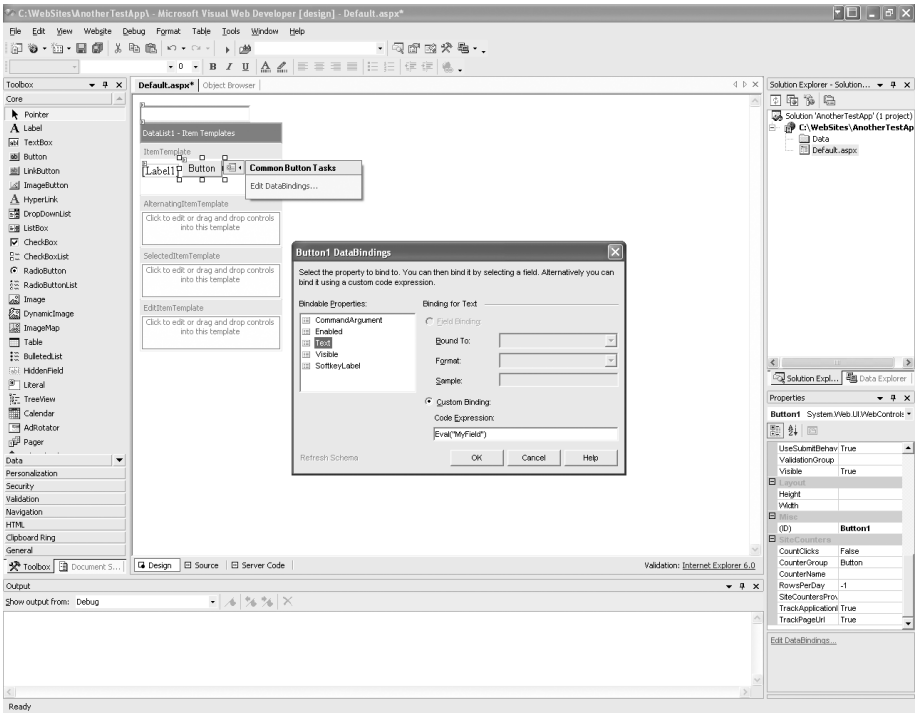

Figure 2-8. You can assign data bindings easily using this dialog box.

Using the new syntax, the same result is achieved with fewer characters:

```
<%# Eval("MyField") %>
```

This was made possible through the integration of a new protected method called `Eval` in the `Page` class. It's nice to know that you can still use an overloaded implementation of this method by passing an additional format string:

```
<%# Eval("MyDateField", "{0:d}") %>
```

Note that in the preceding dialog box, you aren't required to put the data binding in brackets.

In addition to using `Eval`, you may also use the new method `XPath`, as long as the bound data is XML:

```
<%# XPath ("orders/order/customer/@id") %>
```

You can, of course, also pass a format instruction as a second parameter.

Enhanced Table-Editing Support

Improved support for HTML tables is yet another enhancement in VS .NET. You add a table using the Table menu, and though the subsequent editing of single cells hasn't changed in a major way, it has become easier to manage, as shown in Figure 2-9. Here are some of the improvements:

- A border is displayed around the current row, making it easier to see the row you're editing.
- You can use the Tab key to switch between cells.
- You can select rows, columns, and cells by holding down the Ctrl key, even if the elements you're selecting aren't connected.
- You can change the properties of all selected items in one shot.
- You can connect cells.
- You can move tables more easily using the mouse.

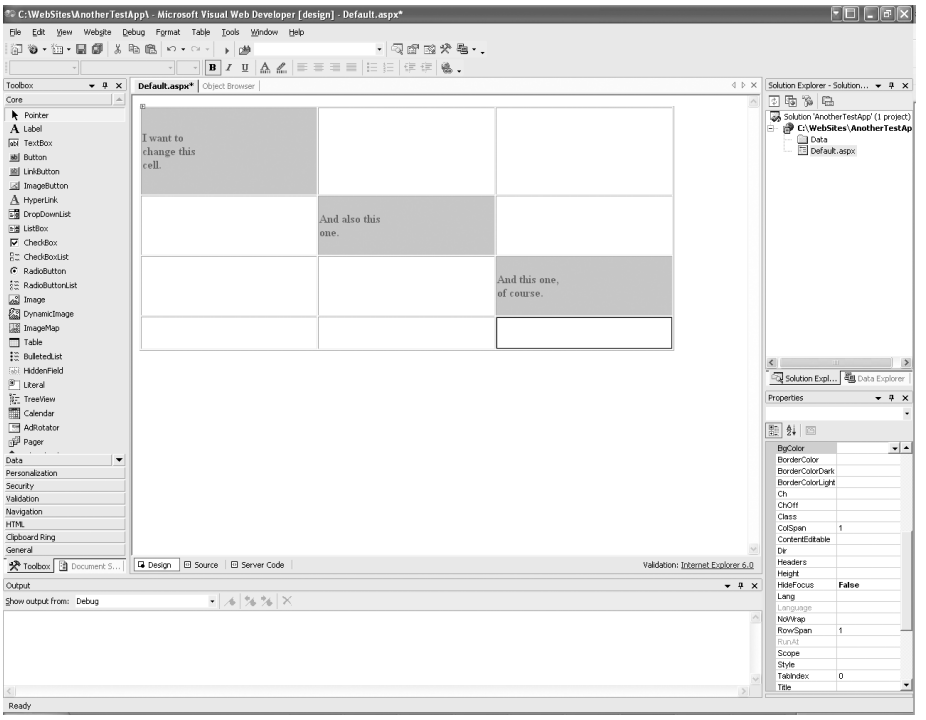

Figure 2-9. The table editing features have been slightly improved.

User Controls

If you like the way Web Matrix (since version 0.6) supports user controls, you'll be pleasantly surprised to discover that VS .NET supports them in the same way. Instead of displaying a meaningless gray box, VS .NET now shows the content of a control. Starting with the Beta version, the VS .NET IDE will display the content of user controls while you edit a page. The current Alpha version, however, still uses the gray box. And more bad news about the Alpha version: Although it provides a Properties window to edit properties, you can't use it because it's a still a bit buggy at the moment. But both will be improved in the Beta version.

With these two extensions, user controls have become more attractive, haven't they?

Enhanced Support for Control Developers

The next improvement relates to custom controls. At least this is the plan of the ASP.NET team. Some of the possible improvements are enrichment of task-based editing, addition of in-place editing, and access to the development environment, including an active source document, directives, and so on. For now, these are only plans—none has been implemented in the current Alpha version.

Validating HTML Source Code

VS .NET comes with some new features related to HTML source code validation. The current validation schema for each ASPX page is displayed at the lower right of the status bar. Just click the schema and the Options dialog box will appear, in which you can choose among several alternatives, for example, XHTML 1.0 Strict, XHTML 1.0 Transitional, Internet Explorer 6.0, and so on. Errors appear in the Task List provided by VS .NET and are also marked directly in the source code. A tool tip shows a description of the problem.

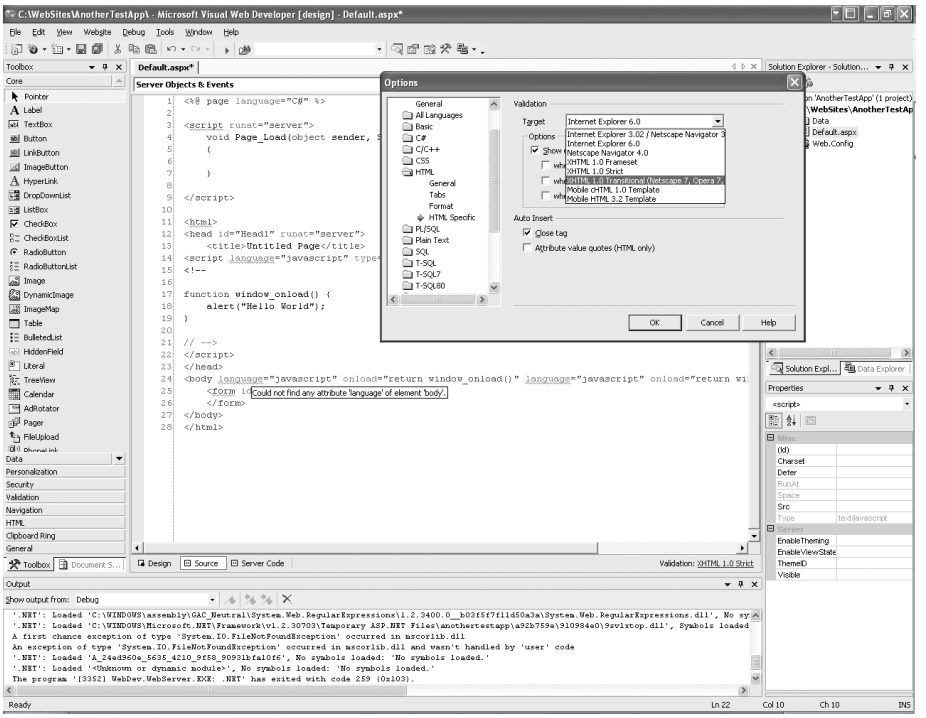

Figure 2-10. Select the validation schema you want to target.

Creating Events Handlers

So what has happened to the event-handling method creation process? It has been simplified. Though the automatic generation of the code body of a standard event by double-click onto the desired control in Design view isn't new (but it's still a nice feature, of course!), VB .NET developers can now use the Properties window to create an event-handling method. C# developers know how to do this already.

All languages provide two drop-down lists in HTML view and Source view, as shown in Figure 2-11. In one list you can select the control, and in the other you can choose the event. If you do so, the method body is automatically generated.

As you can see, everything is fine on the server side. Now what about on the client side? Support is available there too. If you select Source view, the left list contains the group Client Objects & Events. As the group's name suggests, you get access to objects such as `window` and `document` that are offered by a client-side script language (JavaScript, for example). After you select an object, the supported events are displayed in the right list. Creating the related event-handling method within a client-side script block is a matter of one mouse-click. Figure 2-12 shows this with `window_onload`.

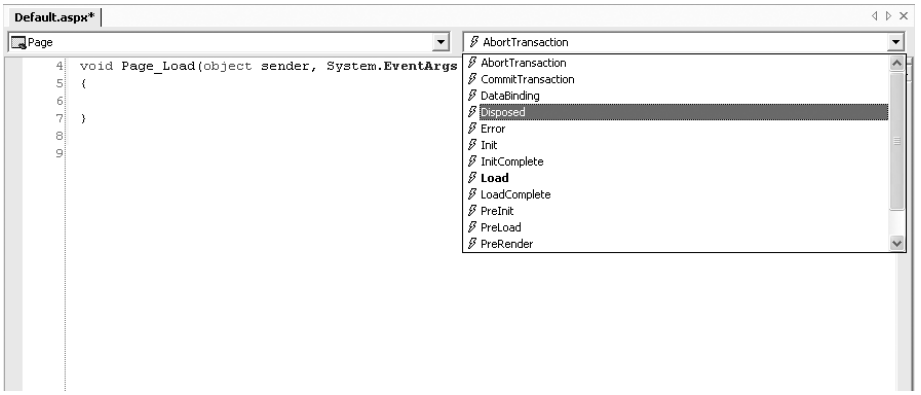

Figure 2-11. Just select the event you need to handle.

> **TIP** *The objects and events listed in Source view may vary depending on the selected validation target.*

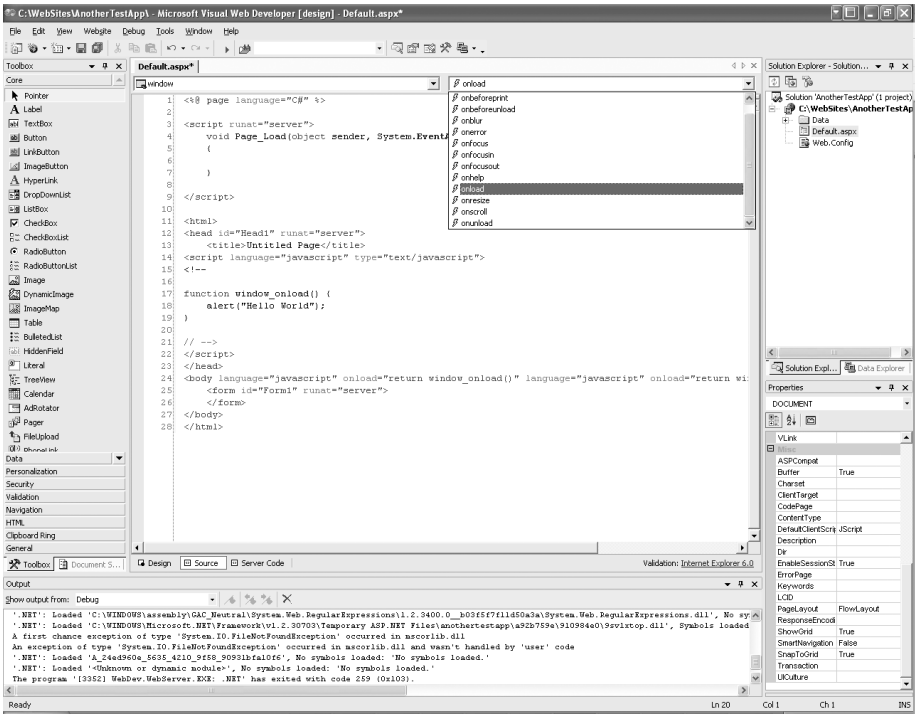

Figure 2-12. You can now apply even client-side event handlers easily.

Using IntelliSense

Compared to previous versions, in the new version of VS .NET the IntelliSense capabilities are highly enhanced. In the final release, the helpful window will be always at your service:

- On editing server code in HTML view
- On editing server code in Source view
- On editing client-side code in HTML view
- On editing server controls, HTML tags, CSS, and directives
- On editing C# or Visual Basic source code files
- On editing ASMX files (from the Beta version on)
- On editing ASHX files (from the Beta version on)

As in VS .NET 2003, preselection of frequently used class members and support for VB .NET while implementing interfaces are in the function set.

I really like IntelliSense's support for client-side scripts in the new version of VS .NET (see Figure 2-13). According to the selected validation schema (see the "Validating HTML Source Code" section for more information), all available objects and their members are listed. Sound good? Then enjoy creating client-side functions!

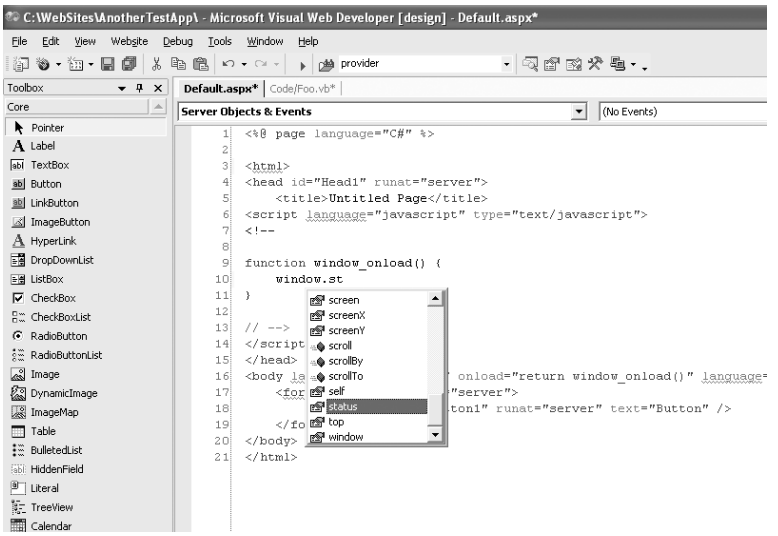

Figure 2-13. IntelliSense now supports client-side scripting.

Testing and Debugging

If you built your web application with VS .NET, you can test and debug it right inside the development environment. In edit mode press the F5 key to start your application in debug mode and press Ctrl+F5 if you don't want to use the debugger. The current page will be displayed within the standard browser.

Any application stored at the local IIS level will be used as server. Whenever you save a project in the local file system, the integrated web server, which looks somewhat like the familiar Cassini server, will be started. The final VS .NET release will likely offer you the choice to select one or both directly.

> **TIP** *So far, web projects in VS .NET aren't compiled explicitly but implicitly. The main benefit of implicit compilation is that it's possible to have a browser window open in parallel to the VS .NET window. You can edit the code, save it, and then just refresh your browser window. As I said, no explicit compilation is required—that makes development a lot easier, doesn't it?*

Please note that any page of the project, including dependent files, will be compiled at the first request dynamically (see Figure 2-14). Failures may be detected by calling each page. Another method is to compile the whole application at once by using this URL: `http://<Host>:<Port>/<App>/precompile.axd`. I cover precompilation further later on in the chapter.

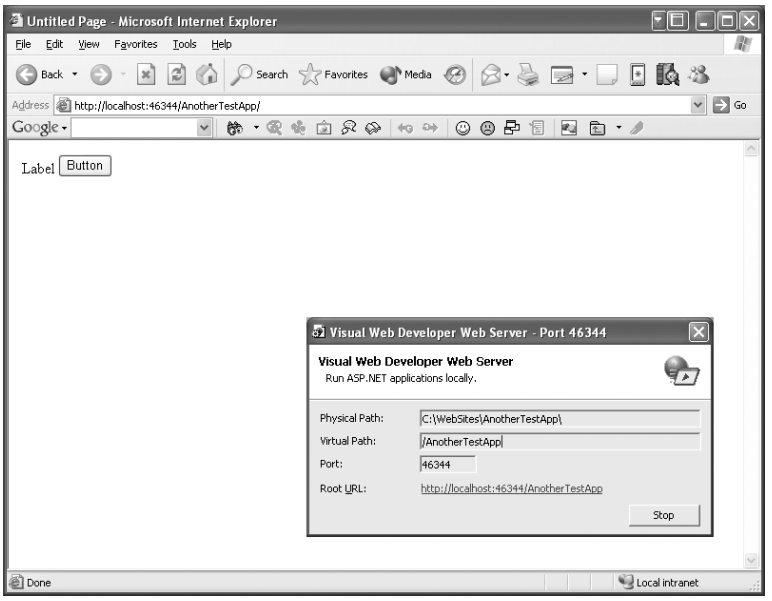

Figure 2-14. You no longer need IIS to run your project—just click and go for it.

Comfortable debugging doesn't depend on the chosen server. As usual, pressing F9 or clicking the gray area next to the row number will set or delete a breakpoint. Pressing Ctrl+Shift+F9 will remove all breakpoints at once after confirmation. Additionally, the following functions are available:

- F10: Step over
- F11: Step into
- Ctrl+F10: Run to cursor

> **TIP** *When you start a project by pressing F5, the IDE generates a* `web.config` *file automatically after a confirmation if one doesn't already exist. This* `web.config` *file will contain the known* `compilation` *tag.*

Debugging in VS .NET is quite smart and intuitive. Regardless of whether the code lives in the source code file or in the ASPX file, the compiler will jump to the right position. Another nice feature is the option to change the value of a variable right in the tool tip while debugging, as shown in Figure 2-15.

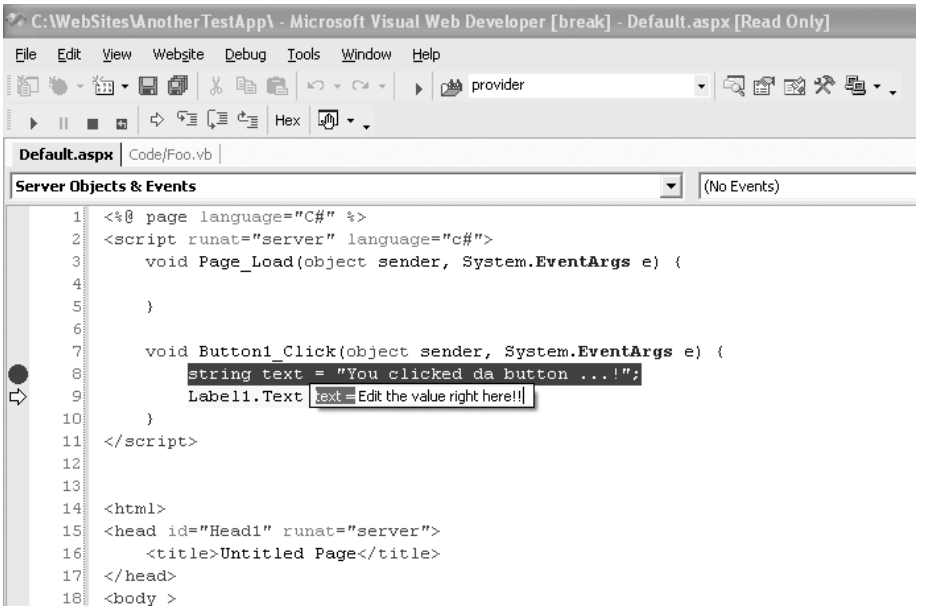

Figure 2-15. You can change the value of a variable in the tool tip while you debug.

Now I'd like to introduce the Object Inspector in VB .NET to you (see Figure 2-16). An enhanced tool tip shows all properties of an object, and you can open the sub-objects step by step. Stop dreaming: Changing the source code at run time as you can in client applications still doesn't work in web applications. But it is possible (in all languages) to edit the source code while you're debugging. The changes will take effect on the next request or postback.

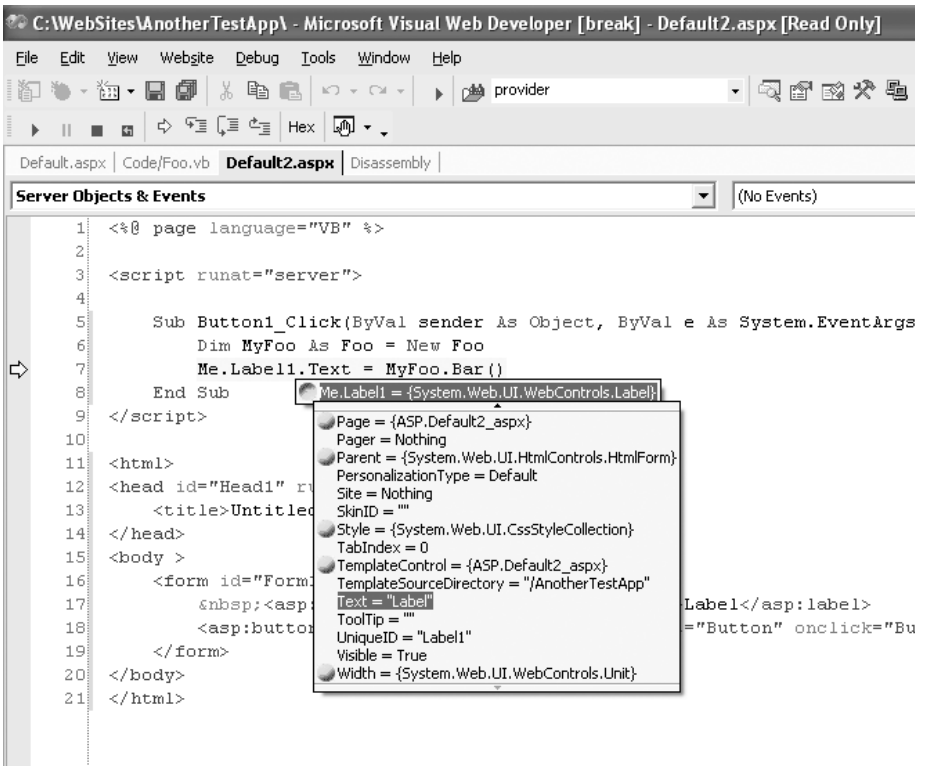

Figure 2-16. VB .NET allows a detailed view into every object.

Code-Beside/Code Separation

Code-behind was a very useful and comfortable feature for web developers. But I have some bad news: Code-behind is dead. Hey, don't worry—you'll use the new code-beside model instead.

Code-behind was smart, but from an object-based view it was far from perfect. The designed ASPX page was derived from a class that was generated in parts by the development environment. You had to declare controls in both files, and manual changes often resulted in problems. There are a lot of good reasons to move away from code-behind.

The solution is *code-beside,* which is supposed to make everything better, brighter, and of course more object oriented. This is what the ASP.NET team promises. Instead of using the class derivation as it was up to now, the design file and the source code file are created as partial classes that will be merged by the compiler (I describe partial classes in Chapter 1). This approach means that there's no automatically generated source code *at all.* Both files work hand in hand, and you can assign events directly in the server control tags.

To add a new code-beside page, right-click the project in Solution Explorer and choose Add New Item. Then select Web Form Using Code Separation in the following dialog box and click Open. Now the project contains two new files, one with the design and one with the source code. The two files are nothing new, but the source code file doesn't include any directives generated by the IDE. VS .NET has just built an empty partial class.

Figure 2-17 shows an example of code-beside. The page contains a Label control and a Button control. Clicking the Button control will assign new text to the Label control.

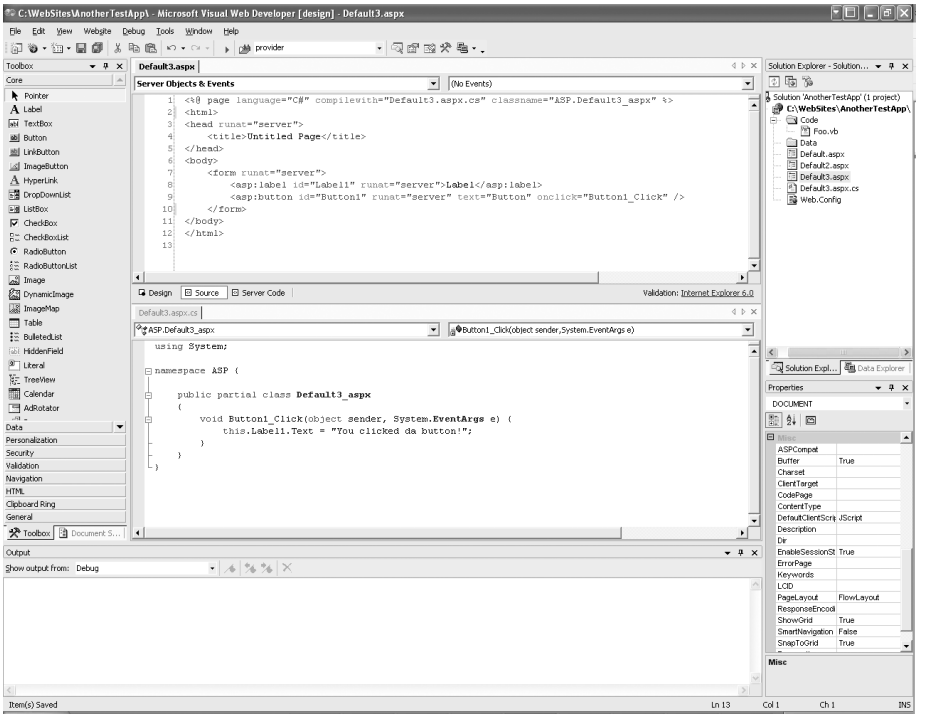

Figure 2-17. Code-beside keeps your files clean.

The old code-behind model isn't supported in the current Alpha version, but it will be supported in the Beta version. So don't worry about migrating your projects.

TIP *You can use code-beside pages and single file pages within one project without any problems. VS .NET handles the files in an appropriate manner automatically, and it takes care that you get the right information while debugging. Currently, code-beside is available for pages as well as for user controls and web services.*

TIP *Put source code in the code-beside file* and *in the ASPX file, and then try to debug the page. What happens? Yeah, the compiler always jumps into the correct file—well done!*

Code Directory

Earlier in the chapter I mentioned that an explicit compilation isn't required anymore. But what about your business objects, your application layer, and your data layer? Well, that's why you have the new code directory.

The code directory looks like nothing more than an ordinary folder, but it's actually quite special. It lives inside of the application root. Any source code files such as `*.vb` or `*.cs` files, but also resource files and even Web Services Description Language (WSDL) files, are saved here. Every supported file is compiled dynamically and is available instantly after you save it.

Using the Code Directory

To experience the code directory in action, you must first create a new folder with the given name "code" by right-clicking the project and choosing New Folder from the context menu. The folder shows its special meaning at once. The second step is to add a C# source code file, such as the `Foo.cs` file in Listing 2-1, to the folder, attach the method `Bar`, and save the new file.

Listing 2-1. Adding the `Foo` *Class to a Code Directory*

```
using System;

/// <summary>
/// Summary description for Foo
/// </summary>
public class Foo
{
   public void Bar()
   {
   }
}
```

You can use the class in your project without compilation. Even the IntelliSense window knows the class, as shown in Figure 2-18. In addition, full debugging support is available for the file. This means that you're able to step into every single method.

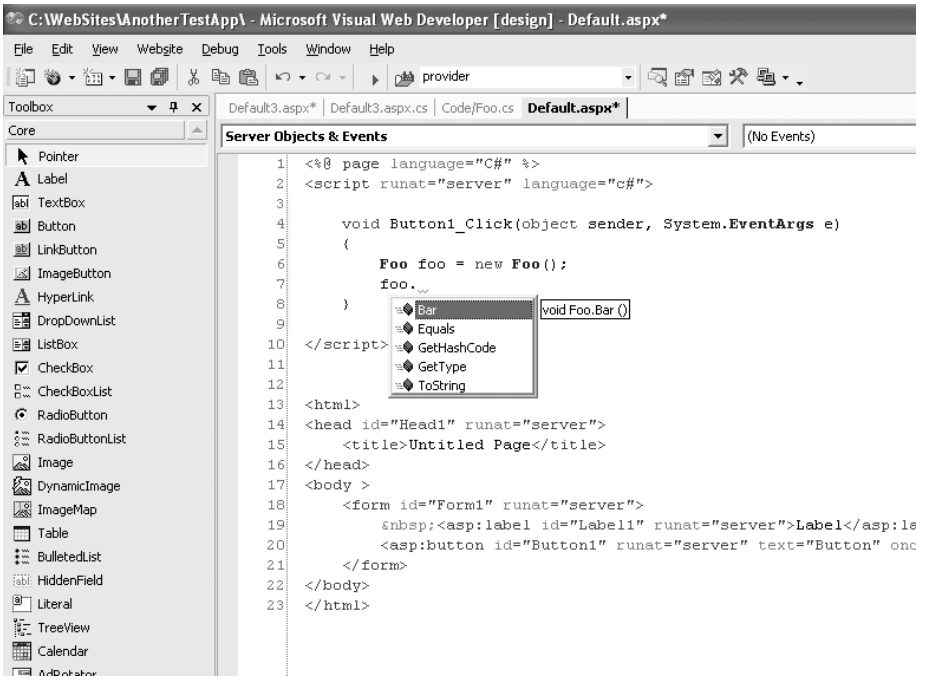

Figure 2-18. Any known file types placed within the code directory are dynamically compiled.

> **TIP** *For better structuring of your source code, it's advisable to spread the classes in subdirectories of the code folder, especially if you work with large projects. The files in subfolders will be dynamically compiled as well.*

In comparison with the previous approach, the code directory offers enhanced possibilities of teamwork, which is a significant advantage. In this new model, a team member no longer has to check out a project file to add or remove files. Changes become effective on a global basis very easily.

Multilanguage and Multitype Support

With ASP.NET you're no longer limited to one language per web site. You can now mix different .NET languages whenever you want. Does it make sense to write one page in C#, a second in VB .NET, and a third in J#? I don't know, but at least it's possible!

Another option is to mix different languages using the code directory. The `Foo` class written in C# and shown in Listing 2-1 works fine with VB .NET pages. This includes full IntelliSense support. But be aware that debugging isn't fully supported in such a scenario, so you can't jump from your VB .NET page to the C# source code. This is currently a major strike against using different languages. However, this will presumably be fixed within the Beta version.

As I mentioned previously, you can store many different file types in the code directory. Besides the programming languages' source code files, the following types are supported:

- RESX
- RESOURCES
- WSDL
- XSD

This list can be individually extended by a provider model. Corresponding abstract base classes are available in the `System.Web.Compilation` namespace. Just let your custom provider inherit from a base class and then assign the desired file extensions to it in the `machine.config` or `web.config` file.

What Happened to the Bin Directory?

Not much, it's still alive. But I wouldn't mention it if there hadn't been some changes. Because web projects are no longer compiled explicitly, there is no single project dynamic link library (DLL) anymore besides the already mentioned precompilation DLL. But external components are referenced in the bin directory further on.

To include a DLL in VS .NET, you just have to copy the bin directory. As a result, the contained classes are automatically available, with IntelliSense support, of course.

Precompilation

Usually an ASP.NET page, including all dependent source code files, is automatically compiled on the first request. This once-compiled version is used until the page or one of its dependencies changes. Besides this approach, version 2.0 offers precompilation. In this approach, all the pages and all related files are compiled explicitly and at once.

There are two different kinds of precompiling: in-place precompilation and precompilation for deployment. The following sections describe these precompilation types in detail.

In-Place Precompilation

You execute *in-place precompilation* by calling the following URL: `http://<Host>:<Port>/<App>/precompile.axd`. Direct integration of this functionality in VS .NET is being planned. So far, so good, but what is it worth? It has two advantages:

- The first request of each page gets accelerated because it's completely compiled already. From the second request on, there's no gain in speed.

- You can detect and fix all compilation errors at once.

Precompilation does more for you than previous versions of VS .NET did. The ASPX files are converted to classes and stored as DLLs in the cache directory together with any code-beside. Only static pages and (graphic) resources are taken out of the directory of the web site afterward.

> **NOTE** *After the first in-place precompilation, any changes are included incrementally. This kind of compilation makes sense even if there are only small changes.*

Precompilation for Deployment

The other type of precompilation is targeted for a different purpose. It allows you to distribute an entire compiled web site compiled as DLLs. In contrast to previous versions of VS. NET, the compiled version includes ASPX pages, ASCX files, and so on with the design, because they've been converted to classes and compiled too.

You can copy the result of this compilation to your server by using `XCopy` or you can ship it to your customers. After that, it isn't possible for the customer to change the application or any of the pages. (A perfect model for software licensing!)

Start the compilation with a new command-line application called `spnet_compiler.exe`. You'll find it in the `<windir>\Microsoft.NET\Framework\<FrameworkVersion>` directory.

To compile a web site stored in the file system, just pass the path of the project and a target folder in which the compiled version of the site will be stored:

```
aspnet_compiler –v / -p <source> <target>
```

In later versions, the parameter `–v /` may not be required. It's just a workaround in the current Alpha. You can use it to define a virtual directory that will be used for absolute links.

Have a look at the result of the compilation and you'll see that it has almost the same structure as your web site (see Figure 2-19). Here you can recognize your ASPX files among other things. But they're used as marks only, and they contain just a message and no design. The documentation consists of an overview of different files and file types. Files with the extension `.compile` are important; they're necessary to establish the allocation between pages and the generated assemblies.

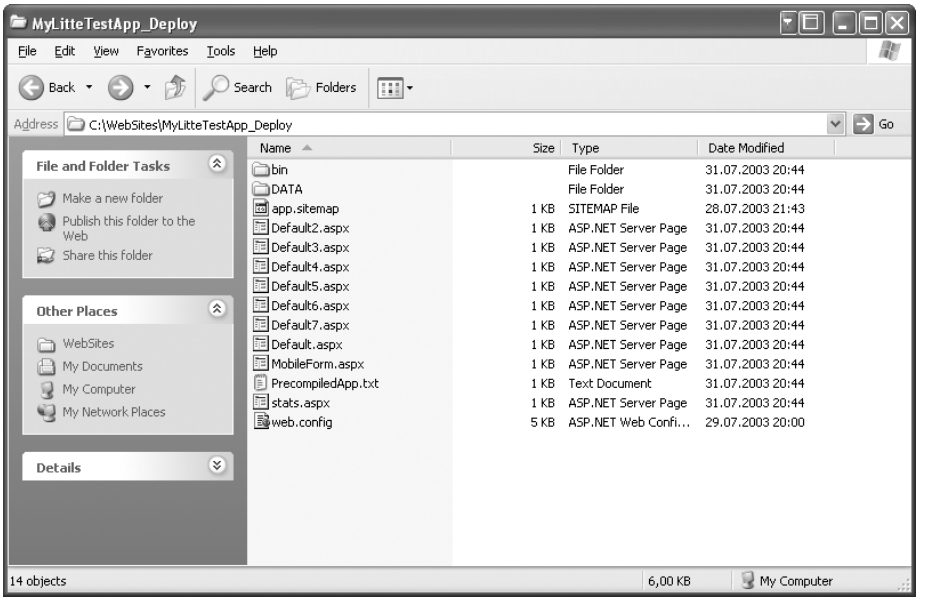

Figure 2-19. The precompilation tool creates a marker file for each page of your site.

To transfer the web site to a server, copy the whole folder. Please don't delete any of the files—they're all required for execution.

Deploying Web Sites

Deployment is much easier with the help of the previously described precompilation. A new tool named Web Copy, starting with the Beta version, will help you to easily copy and synchronize projects, for example with FTP. The corresponding command Publish has already been integrated in the Website menu, but it isn't functioning yet.

Customizing the IDE

The new version of VS .NET has a highly customizable development environment. In particular, you can configure the text editors to fit your individual needs. Figure 2-20 shows a part of the Options dialog box. It allows you, for example, to specify in detail if a line break is inserted before an angle bracket in C#, how parameters should be arranged, and much more.

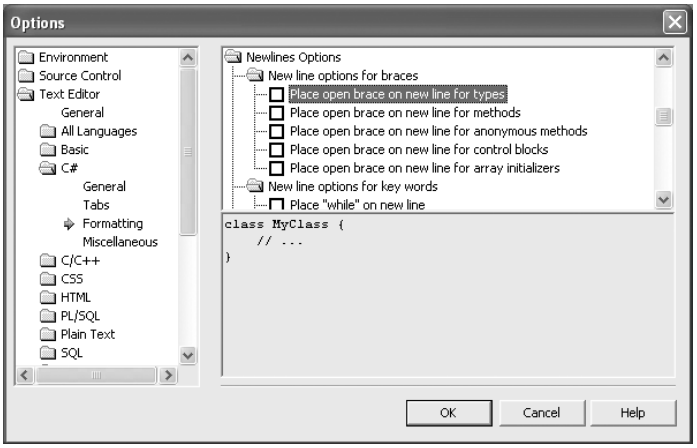

Figure 2-20. You can customize virtually any editor behavior.

For your convenience, you can save your complete settings as an XML file and copy it to a different system. This way, it's possible to enforce companywide coding guidelines. Just select Import/Export Settings from the Tools menu. Figure 2-21 shows the Import/Export Settings dialog box.

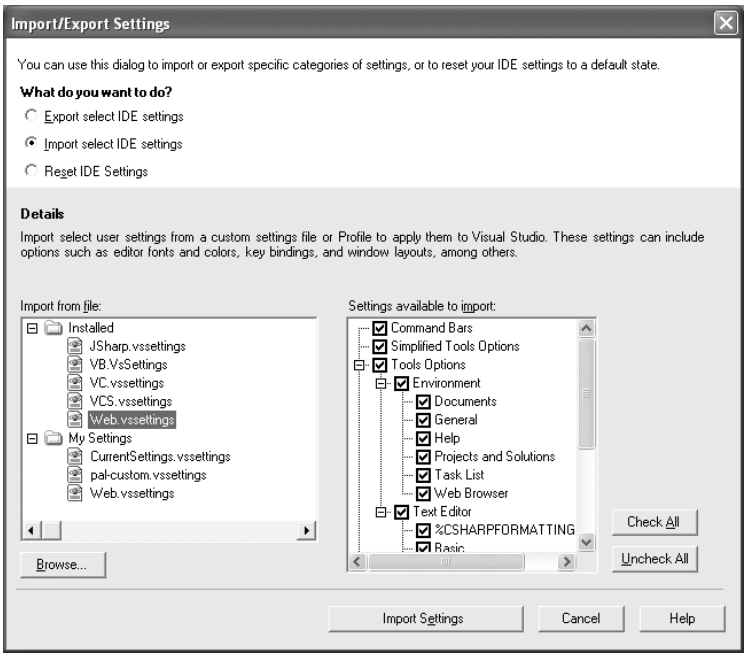

Figure 2-21. In the Import/Export Settings dialog box you back up, share, and reset your individual IDE settings.

Summary

This chapter covered a lot of enhancements in VS .NET, but even more are planned for the final release. In particular, the community features from Web Matrix will surely be taken into consideration more than in the current Alpha version. I hope you've enjoyed this little guided tour through the new development environment. The VS .NET IDE offers a number of interesting improvements, such as better integration of teamwork-related features, and it simplifies the creation of simple and even complex web sites.