Beginning ASP.NET 2.0 E-Commerce in C# 2005

From Novice to Professional

Cristian Darie and Karli Watson

Apress®

Beginning ASP.NET 2.0 E-Commerce in C# 2005: From Novice to Professional

ISBN (pbk): 1-59059-468-1

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Ewan Buckingham
Technical Reviewer: Paul Sarknas
Editorial Board: Steve Anglin, Dan Appleman, Ewan Buckingham, Gary Cornell, Tony Davis, Jason Gilmore, Jonathan Hassell, Chris Mills, Dominic Shakeshaft, Jim Sumser
Project Manager: Kylie Johnston
Copy Edit Manager: Nicole LeClerc
Copy Editor: Julie McNamee
Assistant Production Director: Kari Brooks-Copony
Production Editor: Linda Marousek
Compositor: Susan Glinert Stevens
Proofreader: Nancy Sixsmith
Indexer: Broccoli Information Management
Artist: Kinetic Publishing Services, LLC
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

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, e-mail `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 Source Code section.

CHAPTER 2

Laying Out the Foundations

Now that you've convinced the client that you can create a cool web site to complement the client's store activity, it's time to stop celebrating and start thinking about how to put into practice all the promises made to the client. As usual, when you lay down on paper the technical requirements you must meet, everything starts to seem a bit more complicated than initially anticipated.

Note It is strongly recommended to consistently follow an efficient project-management methodology to maximize the chances of the project's success, on budget and on time. Most project-management theories imply that an initial requirements/specifications document containing the details of the project you're about to create has been signed by you and the client. You can use this document as a guide while creating the solution, and it also allows you to charge extra in case the client brings new requirements or requests changes after development has started. See Appendix B for more details.

To ensure this project's success, you need to come up with a smart way to implement what you've signed the contract for. You want to make your life easy and develop the project smoothly and quickly, but the ultimate goal is to make sure the client is satisfied with your work. Consequently, you should aim to provide your site's increasing number of visitors with a pleasant web experience by creating a nice, functional, and responsive web site by implementing each one of the three development phases described in the first chapter.

The requirements are high, but this is normal for an e-commerce site today. To maximize the chances of success, we'll try to analyze and anticipate as many of the technical requirements as possible, and implement the solution in way that supports changes and additions with minimal effort.

In this chapter, we'll lay down the foundations for the future BalloonShop web site. We'll talk about what technologies and tools you'll use, and even more important, how you'll use them. Let's consider a quick summary of the goals for this chapter before moving on:

- Analyze the project from a technical point of view.
- Analyze and choose an architecture for your application.
- Decide which technologies, programming languages, and tools to use.

- Discuss naming and coding conventions.
- Create the basic structure of the web site and set up the database.

Designing for Growth

The word "design" in the context of a Web Application can mean many things. Its most popular usage probably refers to the visual and user interface (UI) design of a web site.

This aspect is crucial because, let's face it, the visitor is often more impressed with how a site looks and how easy it is to use than about which technologies and techniques are used behind the scenes, or what operating system the web server is running. If the site is hard to use and easy to forget, it just doesn't matter what rocket science was used to create it.

Unfortunately, this truth makes many inexperienced programmers underestimate the importance of the way the invisible part of the site is implemented—the code, the database, and so on. The visual part of a site gets visitors interested to begin with, but its functionality makes them come back. A web site can sometimes be implemented very quickly based on certain initial requirements, but if not properly architected, it can become difficult, if not impossible, to change.

For any project of any size, some preparation must be done before starting to code. Still, no matter how much planning and design work is done, the unexpected does happen and hidden catches, new requirements, and changing rules always seem to work against deadlines. Even without these unexpected factors, site designers are often asked to change or add new functionality after the project is finished and deployed. This also will be the case for BalloonShop, which you'll implement in three separate stages, as discussed in Chapter 1.

You'll learn how to create the web site so that the site (or you) will not fall apart when functionality is extended or updates are made. Because this is a programming book, it doesn't address important aspects of e-commerce, such as designing the UI, marketing techniques, or legal issues. You'll need additional material to cover that ground. Instead, in this book, we'll pay close attention to constructing the code that makes the site work.

The phrase "designing the code" can have different meanings; for example, we'll need to have a short talk about naming conventions. Still, the most important aspect that we need to look at is the architecture to use when writing the code. The architecture refers to the way you split the code for a simple piece of functionality (for example, the product search feature) into smaller, interconnected components. Although it might be easier to implement that functionality as quickly and as simply as possible, in a single component, you gain great long-term advantages by creating more components that work together to achieve the desired result.

Before considering the architecture itself, you must determine what you want from this architecture.

Meeting Long-Term Requirements with Minimal Effort

Apart from the fact that you want a fast web site, each of the phases of development we talked about in Chapter 1 brings new requirements that must be met.

Every time you proceed to a new stage, you want to **reuse** most of the already existing solution. It would be very inefficient to redesign the site (not just the visual part, but the code as well!) just because you need to add a new feature. You can make it easier to reuse the solution

by planning ahead so that any new functionality that needs to be added can slot in with ease, rather than each change causing a new headache.

When building the web site, implementing a **flexible architecture** composed of pluggable components allows you to add new features—such as the shopping cart, the departments list, or the product search feature—by coding them as separate components and plugging them into the existing application. Achieving a good level of flexibility is one of the goals regarding the application's architecture, and this chapter shows how you can put this into practice. You'll see that the level of flexibility is proportional to the amount of time required to design and implement it, so we'll try to find a compromise that provides the best gains without complicating the code too much.

Another major requirement that is common to all online applications is to have a **scalable architecture**. Scalability is defined as the capability to increase resources to yield a linear increase in service capacity. In other words, in a scalable system, the ratio (proportion) between the number of client requests and the hardware resources required to handle those requests is constant, even when the number of clients increases (ideally). An unscalable system can't deal with an increasing number of clients, no matter how many hardware resources are provided. Because we're optimistic about the number of customers, we must be sure that the site will be able to deliver its functionality to a large number of clients without throwing out errors or performing sluggishly.

Reliability is also a critical aspect for an e-commerce application. With the help of a coherent error-handling strategy and a powerful relational database, you can ensure data integrity and ensure that noncritical errors are properly handled without bringing the site to its knees.

The Magic of the Three-Tier Architecture

Generally, the architecture refers to splitting each piece of the application's functionality into separate components based on what they do and grouping each kind of component into a single logical tier.

The three-tier architecture has become popular today because it answers most of the problems discussed so far by splitting an application's functionality unit into three logical tiers:

- The presentation tier
- The business tier
- The data tier

The **presentation tier** contains the UI elements of the site, and includes all the logic that manages the interaction between the visitor and the client's business. This tier makes the whole site feel alive, and the way you design it is crucially important to the site's success. Because your application is a web site, its presentation tier is composed of dynamic web pages.

The **business tier** (also called the *middle tier*) receives requests from the presentation tier and returns a result to the presentation tier depending on the business logic it contains. Almost any event that happens in the presentation tier results in the business tier being called (except events that can be handled locally by the presentation tier, such as simple input data validation). For example, if the visitor is doing a product search, the presentation tier calls the business tier and says, "Please send me back the products that match this search criterion." Almost always,

the business tier needs to call the data tier for information to respond to the presentation tier's request.

The **data tier** (sometimes referred to as the *database tier*) is responsible for storing the application's data and sending it to the business tier when requested. For the BalloonShop e-commerce site, you'll need to store data about products (including their categories and their departments), users, shopping carts, and so on. Almost every client request finally results in the data tier being interrogated for information (except when previously retrieved data has been cached at the business tier or presentation tier levels), so it's important to have a fast database system. In Chapters 3 and 4, you'll learn how to design the database for optimum performance.

These tiers are purely logical—there is no constraint on the physical location of each tier. You're free to place all the application, and implicitly all its tiers, on a single server machine. Alternatively, you can place each tier on a separate machine or even split the components of a single tier over multiple machines. Your choice depends on the particular performance requirements of the application. This kind of flexibility allows you to achieve many benefits, as you'll soon see.

An important constraint in the three-layered architecture model is that information must flow in sequential order between tiers. The presentation tier is only allowed to access the business tier and never directly the data tier. The business tier is the "brain" in the middle that communicates with the other tiers and processes and coordinates all the information flow. If the presentation tier directly accessed the data tier, the rules of three-tier architecture programming would be broken. When you implement a three-tier architecture, you must be consistent and obey its rules to reap the benefits.

Figure 2-1 is a simple representation of how data is passed in an application that implements the three-tier architecture.

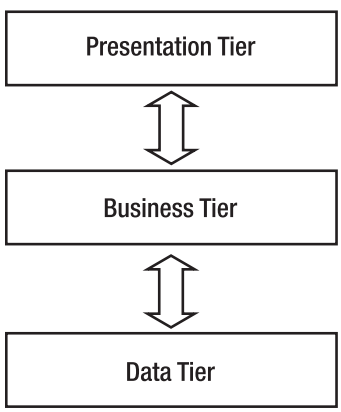

Figure 2-1. *Simple representation of the three-tier architecture*

A Simple Scenario

It's easier to understand how data is passed and transformed between tiers if you take a closer look at a simple example. To make the example even more relevant to the project, let's analyze a situation that will actually happen in BalloonShop. This scenario is typical for three-tier applications.

Like most e-commerce sites, BalloonShop will have a shopping cart, which we'll discuss later in the book. For now, it's enough to know that the visitor will add products to the shopping cart by clicking an Add to Cart button. Figure 2-2 shows how the information flows through the application when that button is clicked.

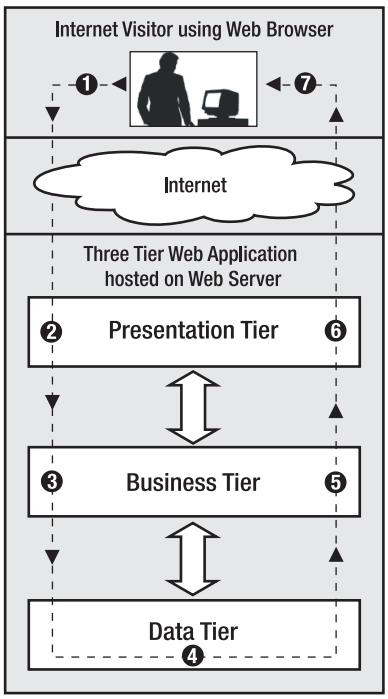

Figure 2-2. *Internet visitor interacting with a three-tier application*

When the user clicks the Add to Cart button for a specific product (Step 1), the presentation tier (which contains the button) forwards the request to the business tier—"Hey, I want this product added to the visitor's shopping cart!" (Step 2). The business tier receives the request, understands that the user wants a specific product added to the shopping cart, and handles the request by telling the data tier to update the visitor's shopping cart by adding the selected product (Step 3). The data tier needs to be called because it stores and manages the entire web site's data, including users' shopping cart information.

The data tier updates the database (Step 4) and eventually returns a success code to the business tier. The business tier (Step 5) handles the return code and any errors that might have occurred in the data tier while updating the database and then returns the output to the presentation tier.

Finally, the presentation tier generates an updated view of the shopping cart (Step 6). The results of the execution are wrapped up by generating an HTML (Hypertext Markup Language) web page that is returned to the visitor (Step 7), where the updated shopping cart can be seen in the visitor's favorite web browser.

Note that in this simple example, the business tier doesn't do a lot of processing, and its business logic isn't very complex. However, if new business rules appear for your application, you would change the business tier. If, for example, the business logic specified that a product

could only be added to the shopping cart if its quantity in stock were greater than zero, an additional data tier call would have been made to determine the quantity. The data tier would only be requested to update the shopping cart if products were in stock. In any case, the presentation tier is informed about the status and provides human-readable feedback to the visitor.

What's in a Number?

It's interesting to note how each tier interprets the same piece of information differently. For the data tier, the numbers and information it stores have no significance because this tier is an engine that saves, manages, and retrieves numbers, strings, or other data types—not product quantities or product names. In the context of the previous example, a product quantity of 0 represents a simple, plain number without any meaning to the data tier (it is simply 0, a 32-bit integer).

The data gains significance when the business tier reads it. When the business tier asks the data tier for a product quantity and gets a "0" result, this is interpreted by the business tier as "Hey, no products in stock!" This data is finally wrapped in a nice, visual form by the presentation tier, for example, a label reading, "Sorry, at the moment the product cannot be ordered."

Even if it's unlikely that you want to forbid a customer from adding a product to the shopping cart if the product isn't in stock, the example (described in Figure 2-3) is good enough to present in yet another way how each of the three tiers has a different purpose.

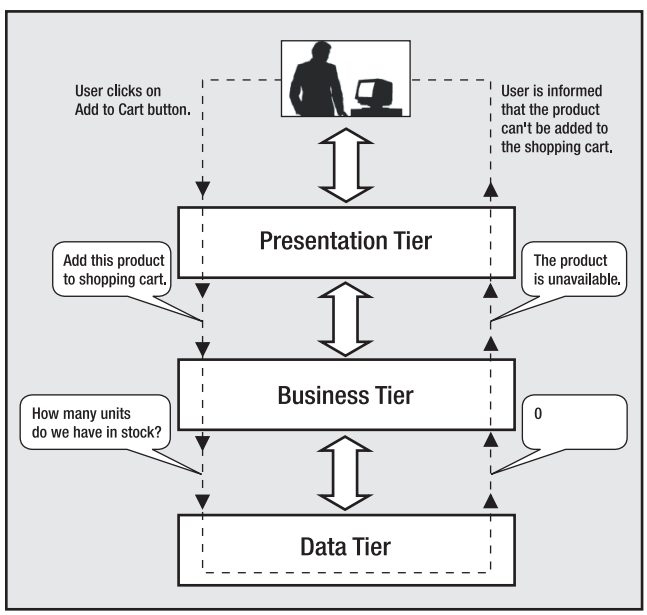

Figure 2-3. *Internet visitor interacting with a three-tier application*

The Right Logic for the Right Tier

Because each layer contains its own logic, sometimes it can be tricky to decide where exactly to draw the line between the tiers. In the previous scenario, instead of reading the product's

quantity in the business tier and deciding whether the product is available based on that number (resulting in two data tier, and implicitly database, calls), you could have a single data tier method named `AddProductIfAvailable` that adds the product to the shopping cart only if it's available in stock.

In this scenario, some logic is transferred from the business tier to the data tier. In many other circumstances, you might have the option to place the same logic in one tier or another, or maybe in both. In most cases, there is no single best way to implement the three-tier architecture, and you'll need to make a compromise or a choice based on personal preference or external constraints.

Occasionally, even though you know the right way (in respect to the architecture) to implement something, you might choose to break the rules to get a performance gain. As a general rule, if performance can be improved this way, it's okay to break the strict limits between tiers *just a little bit* (for example, add some of the business rules to the data tier or vice versa), if these rules are not likely to change in time. Otherwise, keeping all the business rules in the middle tier is preferable because it generates a "cleaner" application that is easier to maintain.

Finally, don't be tempted to access the data tier directly from the presentation tier. This is a common mistake that is the shortest path to a complicated, hard-to-maintain, and inflexible system. In many data access tutorials or introductory materials, you'll be shown how to perform simple database operations using a simple UI application. In these kinds of programs, all the logic is probably written in a short, single file instead of separate tiers. Although the materials might be very good, keep in mind that most of these texts are meant to teach you how to do different individual tasks (for example, access a database) and not how to correctly create a flexible and scalable application.

A Three-Tier Architecture for BalloonShop

Implementing a three-tiered architecture for the BalloonShop web site will help you achieve the goals listed at the beginning of the chapter. The coding discipline imposed by a system that might seem rigid at first sight allows for excellent levels of flexibility and extensibility in the long run.

Splitting major parts of the application into separate, smaller components also encourages reusability. More than once when adding new features to the site you'll see that you can reuse some of the already existing bits. Adding a new feature without needing to change much of what already exists is, in itself, a good example of reusability. Also, smaller pieces of code placed in their correct places are easier to document and analyze later.

Another advantage of the three-tiered architecture is that, if properly implemented, the overall system is resistant to changes. When bits in one of the tiers change, the other tiers usually remain unaffected, sometimes even in extreme cases. For example, if for some reason the backend database system is changed (say, the manager decides to use Oracle instead of SQL Server), you only need to update the data tier. The existing business tier should work the same with the new database.

Why Not Use More Tiers?

The three-tier architecture we've been talking about so far is a particular (and the most popular) version of the *n*-Tier Architecture, which is a commonly used buzzword these days. *n*-Tier architecture refers to splitting the solution into a number (*n*) of logical tiers. In complex projects, sometimes it makes sense to split the business layer into more than one layer, thus resulting in

an architecture with more than three layers. However, for this web site, it makes most sense to stick with the three-layered design, which offers most of the benefits while not requiring too many hours of design or a complex hierarchy of framework code to support the architecture.

Maybe with a more involved and complex architecture, you would achieve even higher levels of flexibility and scalability for the application, but you would need much more time for design before starting to implement anything. As with any programming project, you must find a fair balance between the time required to design the architecture and the time spent to implement it. The three-tier architecture is best suited to projects with average complexity, like the BalloonShop web site.

You also might be asking the opposite question, "Why not use fewer tiers?" A two-tier architecture, also called client-server architecture, can be appropriate for less-complex projects. In short, a two-tier architecture requires less time for planning and allows quicker development in the beginning, although it generates an application that's harder to maintain and extend in the long run. Because we're expecting to extend the application in the future, the client-server architecture isn't appropriate for this application, so it won't be discussed further in this book.

Now that you know the general architecture, let's see what technologies and tools you'll use to implement it. After a brief discussion of the technologies, you'll create the foundation of the presentation and data tiers by creating the first page of the site and the backend database. You'll start implementing some real functionality in each of the three tiers in Chapter 3 when you start creating the web site's product catalog.

Choosing Technologies and Tools

No matter which architecture is chosen, a major question that arises in every development project is which technologies, programming languages, and tools are going to be used, bearing in mind that external requirements can seriously limit your options.

■ **Note** In this book, we're creating a web site using Microsoft technologies. Keep in mind, however, that when it comes to technology, problems often have more than one solution, and rarely is there only a single best way to solve the problem. Although we really like Microsoft's technologies as presented in this book, it doesn't necessarily mean they're the best choice for any kind of project, in any circumstances. Additionally, in many situations, you must use specific technologies because of client requirements or other external constraints. The *System Requirements* and *Software Requirements* stages in the software development process will determine which technologies you must use for creating the application. See Appendix B for more details.

This book is about programming e-commerce web sites with **ASP.NET 2.0** (Active Server Pages .NET 2.0) and **C#**. The tools you'll use are **Visual Web Developer 2005 Express Edition** and **SQL Server 2005 Express Edition**, which are freely available from Microsoft's web site. See Appendix A for installation instructions. Although the book assumes a little previous experience with each of these, we'll take a quick look at them and see how they fit into the project and into the three-tier architecture.

■ **Note** This book builds on *Beginning ASP.NET 1.1 E-Commerce: From Novice to Professional* (Apress, 2004), which used ASP.NET 1.1, Visual Studio .NET 2003, and SQL Server 2000. If you're an open source fan, you might also want to check out *Beginning PHP 5 and MySQL E-Commerce: From Novice to Professional* (Apress, 2004).

Using ASP.NET 2.0

ASP.NET 2.0 is Microsoft's latest technology set for building dynamic, interactive web content. Compared to its previous versions, ASP.NET 2.0 includes many new features aimed at increasing the web developer's productivity in building web applications.

Because this book is targeted at both existing ASP.NET 1.1 and existing ASP.NET 2.0 developers, we'll highlight a number of ASP.NET 2.0-specific techniques along the way and try to provide useful tips and tricks that increase your coding efficiency by making the most of this technology. However, do keep in mind that while building your e-commerce web site with this book, we only cover a subset of the vast number of features ASP.NET 2.0 has to offer. Therefore, you still need additional ASP.NET 2.0 books (or other resources) to use as a reference and to complete your knowledge on theory issues that didn't make it into this book. In the Apress technology tree, reading this book comes naturally after *Beginning ASP.NET 2.0 in C#: From Novice to Professional* (Apress, 2005), but you can always use the beginners' books of your choice instead.

ASP.NET is not the only server-side technology around for creating professional e-commerce web sites. Among its most popular competitors are PHP (Hypertext Preprocessor), JSP (JavaServer Pages), ColdFusion, and even the outdated ASP 3.0 and CGI (Common Gateway Interface). Among these technologies are many differences, but also some fundamental similarities. For example, pages written with any of these technologies are composed of basic HTML, which draws the static part of the page (the template), and code that generates the dynamic part.

Web Clients and Web Servers

You probably already know the general principles about how dynamic web pages work. However, as a short recap, Figure 2-4 shows what happens to an ASP.NET web page from the moment the client browser (no matter if it's Internet Explorer, Mozilla Firefox, or any other web browser) requests it to the moment the browser actually receives it.

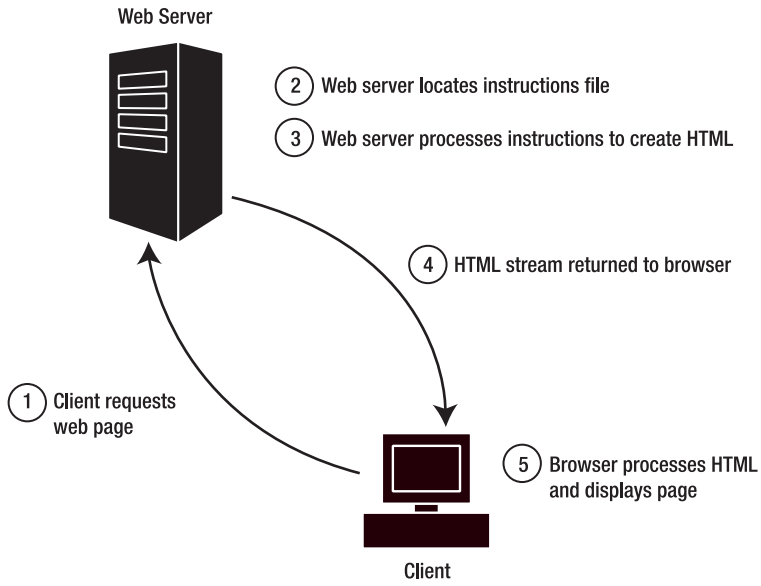

Figure 2-4. *Web server processing client requests*

After the request, the page is first processed at the server before being returned to the client (this is the reason ASP.NET and the other mentioned technologies are called server-side technologies). When an ASP.NET page is requested, its underlying code is first executed on the server. After the final page is composed, the resulting HTML is returned to the visitor's browser.

The returned HTML can optionally contain client-side script code, which is directly interpreted by the browser. The most popular client-side scripting technologies are JavaScript and VBScript. JavaScript is usually the better choice because it has wider acceptance, whereas only Internet Explorer recognizes VBScript. Other important client-side technologies are Macromedia Flash and Java applets, but these are somewhat different because the web browser does not directly parse them—Flash requires a specialized plug-in and Java applets require a JVM (Java Virtual Machine). Internet Explorer also supports ActiveX controls and .NET assemblies.

The Code Behind the Page

From its first version, ASP.NET encouraged (and helped) developers to keep the code of a web page physically separated from the HTML layout of that page. Keeping the code that gives life to a web page in a separate file from the HTML layout of the page was an important improvement over other server-side web-development technologies whose mix of code and HTML in the same file often led to long and complicated source files that were hard to document, change, and maintain. Also, a file containing both code and HTML is the subject of both programmers' and designers' work, which makes team collaboration unnecessarily complicated and increases the chances of the designer creating bugs in the code logic while working on cosmetic changes.

ASP.NET 1.0 introduced a **code-behind** model, used to separate the HTML layout of a web page from the code that gives life to that page. Although it was possible to write the code and HTML in the same file, Visual Studio .NET 2002 and Visual Studio .NET 2003 always automatically generated two separate files for a Web Form: the HTML layout resided in the .ASPX file and the code resided in the code-behind file. Because ASP.NET allowed the developer to write the code in the programming language of his choice (such as C# or VB .NET), the code-behind file's extension depended on the language it was written in (such as `.ASPX.CS` or `.ASPX.VB`).

ASP.NET 2.0 uses a refined code-behind model. Although the new model is more powerful, the general principles (to help separate the page's looks from its brain) are still the same. We'll look over the differences a bit later, especially for existing ASP.NET 1.x developers migrating to ASP.NET 2.0.

Before moving on, let's summarize the most important general features of ASP.NET:

- The server-side code can be written in the .NET language of your choice. By default, you can choose from C#, VB .NET, and J#, but the whole infrastructure is designed to support additional languages. These languages are powerful and fully object oriented.
- The server-side code of ASP.NET pages is fully compiled and executed—as opposed to being interpreted line by line—which results in optimal performance and offers the possibility to detect a number of errors at compile-time instead of runtime.
- The concept of code-behind files helps separate the visual part of the page from the (server-side) logic behind it. This is an advantage over other technologies, in which both the HTML and the server-side code reside in the same file (often resulting in the popular "spaghetti code").
- Visual Web Developer 2005 is an excellent and complete visual editor that represents a good weapon in the ASP.NET programmer's arsenal (although you don't need it to create ASP.NET Web Applications). Visual Web Developer 2005 Express Edition is free, and you can use it to develop the examples in this book.

ASP.NET Web Forms, Web User Controls, and Master Pages

ASP.NET web sites are developed around ASP.NET **Web Forms**. ASP.NET Web Forms have the `.aspx` extension and are the standard way to provide web functionality to clients. A request to an ASPX resource, such as `http://web.cristiandarie.ro/BalloonShop/default.aspx`, results in the `default.aspx` file being executed on the server (together with its code-behind file) and the results being composed as an HTML page that is sent back to the client. Usually, the `.aspx` file has an associated code-behind file, which is also considered part of the Web Form.

Web User Controls and **Master Pages** are similar to Web Forms in that they are also composed of HTML and code (they also support the code-behind model), but they can't be directly accessed by clients. Instead, they are used to compose the content of the Web Forms.

Web User Controls are files with the `.ascx` extension that can be included in Web Forms, with the parent Web Form becoming the container of the control. Web User Controls allow you to easily reuse pieces of functionality in a number of Web Forms.

Master Pages are a new feature of ASP.NET 2.0. A Master Page is a template that can be applied to a number of Web Forms in a site to ensure a consistent visual appearance and functionality throughout the various pages of the site. Updating the Master Page has an immediate effect on every Web Form built on top of that Master Page.

Web User Controls, Web Server Controls, and HTML Server Controls It's worth taking a second look at Web User Controls from another perspective. Web User Controls are a particular type of server-side control. Server-side controls generically refer to three kinds of controls: Web User Controls, Web Server Controls, and HTML Server Controls. All these kinds of controls can be used to reuse pieces of functionality inside Web Forms.

As stated in the previous section, Web User Controls are files with the `.ascx` extension that have a structure similar to the structure of Web Forms, but they can't be requested directly by a client web browser; instead, they are meant to be included in Web Forms or other Web User Controls.

Web Server Controls are compiled .NET classes that, when executed, generate HTML output (eventually including client-side script). You can use them in Web Forms or in Web User Controls. The .NET Framework ships with a large number of Web Server Controls (many of which are new to version 2.0 of the framework), including simple controls such as `Label`, `TextBox`, or `Button`, and more complex controls, such as validation controls, data controls, the famous `GridView` control (which is meant to replace the old `DataGrid`), and so on. Web Server Controls are powerful, but they are more complicated to code because all their functionality must be implemented manually. Among other features, you can programmatically declare and access their properties, make these properties accessible through the Visual Web Developer designer, and add the controls to the toolbox, just as in Windows Forms applications or old VB6 programs.

HTML Server Controls allow you to programmatically access HTML elements of the page from code (such as from the code-behind file). You transform an HTML control to an HTML Server Control by adding the `runat="server"` attribute to it. Most HTML Server Controls are doubled by Web Server Controls (such as labels, buttons, and so on). For consistency, we'll stick with Web Server Controls most of the time, but you'll need to use HTML Server Controls in some cases.

For the BalloonShop project, you'll use all kinds of controls, and you'll create a number of Web User Controls.

Because you can develop Web User Controls independently of the main web site and then just plug them in when they're ready, having a site structure based on Web User Controls provides an excellent level of flexibility and reusability.

ASP.NET and the Three-Tier Architecture

The collection of Web Forms, Web User Controls, and Master Pages form the presentation tier of the application. They are the part that creates the HTML code loaded by the visitor's browser.

The logic of the UI is stored in the code-behind files of the Web Forms, Web User Controls, and Master Pages. Note that although you don't need to use code-behind files with ASP.NET (you're free to mix code and HTML just as you did with ASP), we'll exclusively use the code-behind model for the presentation-tier logic.

In the context of a three-tier application, the logic in the presentation tier usually refers to the various event handlers, such as `Page_Load` and `someButton_Click`. As you learned earlier,

these event handlers should call business-tier methods to get their jobs done (and never call the data tier directly).

Using C# and VB .NET

C# and VB .NET are languages that can be used to code the Web Forms' code-behind files. In this book, we're using C#; in a separate version of this book called *Beginning ASP.NET E-Commerce in VB .NET: From Novice to Professional*, we'll present the same functionality using VB .NET. Unlike its previous version (VB6), VB .NET is a fully object-oriented language and takes advantage of all the features provided by the .NET Framework.

ASP.NET 2.0 even allows you to write the code for various elements inside a project in different languages, but we won't use this feature in this book. Separate projects written in different .NET languages can freely interoperate, as long as you follow some basic rules. For more information about how the .NET Framework works, you should read a general-purpose .NET book.

■ **Note** Just because you *can* use multiple languages in a single language, doesn't mean you *should* overuse that feature, if you have a choice. Being consistent is more important than playing with diversity if you care for long-term ease of maintenance and prefer to avoid unnecessary headaches (which is something that most programmers do).

In this book, apart from using C# for the code-behind files, you'll use the same language to code the middle tier classes. You'll create the first classes in Chapter 3 when building the product catalog, and you'll learn more details there, including a number of new features that come with .NET 2.0.

Using Visual Studio 2005 and Visual Web Developer 2005 Express Edition

Visual Studio 2005 is by far the most powerful tool you can find to develop .NET applications. Visual Studio is a complete programming environment capable of working with many types of projects and files, including Windows and Web Forms projects, setup and deployment projects, and many others. Visual Studio also can be used as an interface to the database to create tables and stored procedures, implement table relationships, and so on.

Visual Web Developer 2005 Express Edition is a free version of Visual Studio 2005, focused on developing Web Applications with ASP.NET 2.0. Because the code in this book can be built with any of these products, we'll use the terms *Visual Web Developer* and *Visual Studio* interchangeably.

A significant new feature in Visual Studio .NET 2005 and Visual Web Developer 2005 compared to previous versions of Visual Studio is the presence of an integrated web server, which permits you to execute your ASP.NET Web Applications even if you don't have IIS (Internet Information Services) installed on your machine. This is good news for Windows XP Home Edition users, who can't install IIS on their machines because it isn't supported by the operating system.

Although we'll use Visual Web Developer 2005 Express Edition for writing the BalloonShop project, it's important to know that you don't have to. ASP.NET and the C# and VB .NET compilers are available as free downloads at `http://www.microsoft.com` as part of the .NET Framework SDK (Software Developers Kit), and a simple editor such as Notepad is enough to create any kind of web page.

■ **Tip** In the ASP.NET 1.x days when there were no free versions of Visual Studio, many developers preferred to use a neat program called Web Matrix—a free ASP.NET development tool whose installer (which can still be downloaded at `http://www.asp.net/webmatrix`) was only 1.3MB. Development of Web Matrix has been discontinued though, because Visual Web Developer 2005 Express Edition is both powerful and free.

Visual Studio 2005 and Visual Web Developer 2005 come with many new features compared to its earlier versions, and we'll study a part of them while creating the BalloonShop project.

Using SQL Server 2005

Along with .NET Framework 2.0 and Visual Studio 2005, Microsoft also released a new version of its player in the Relational Database Management Systems (RDBMS) field—SQL Server 2005. This complex software program's purpose is to store, manage, and retrieve data as quickly and reliably as possible. You'll use SQL Server to store all the information regarding your web site, which will be dynamically placed on the web page by the application logic. Simply said, all data regarding the products, departments, users, shopping carts, and so on will be stored and managed by SQL Server.

The good news is that a lightweight version of SQL Server 2005, named SQL Server 2005 Express Edition, is freely available. Unlike the commercial versions, SQL Server 2005 Express Edition doesn't ship by default with any visual-management utilities. However, a very nice tool called SQL Server Express Manager is also freely available. Appendix A contains details for installing both SQL Server 2005 Express Edition and SQL Server Express Manager.

■ **Tip** To learn more about the differences between SQL Server 2005 Express Edition and the other versions, you can check `http://www.microsoft.com/sql/2005/productinfo/sql2005features.asp`.

The first steps in interacting with SQL Server come a bit later in this chapter when you create the BalloonShop database.

SQL Server and the Three-Tier Architecture

It should be clear by now that SQL Server is somehow related to the data tier. However, if you haven't worked with databases until now, it might be less than obvious that SQL Server is more than a simple store of data. Apart from the actual data stored inside, SQL Server is also capable of storing logic in the form of stored procedures, maintaining table relationships, ensuring that various data integrity rules are obeyed, and so on.

You can communicate with SQL Server through a language called T-SQL (Transact-SQL), which is the SQL dialect recognized by SQL Server. SQL, or Structured Query Language, is the language used to interact with the database. SQL is used to transmit to the database instructions such as "Send me the last 10 orders" or "Delete product #123."

Although it's possible to compose T-SQL statements in your C# code and then submit them for execution, this is generally a *bad practice*, because it incurs security, consistency, and performance penalties. In our solution, we'll store all data tier logic using **stored procedures**. Historically, stored procedures were programs that were stored internally in the database and were written in T-SQL. This still stands true with SQL Server 2005, which also brings the notion of *managed stored procedures* that can be written in a .NET language such as C# and VB.NET and are, as a result, compiled instead of interpreted.

■ **Note** Writing stored procedures in C#, also called *managed stored procedures*, doesn't just sound interesting, it actually is. However, managed stored procedures are very powerful weapons, and as with any weapon, only particular circumstances justify using them. Typically it makes sense to use managed stored procedures when you need to perform complex mathematical operations or complex logic that can't be easily implemented with T-SQL. However, learning how to do these tasks the right way requires a good deal of research, which is outside the scope of this book. Moreover, the data logic in this book didn't justify adding any managed stored procedures, and as a result you won't see any here. Learning how to program managed stored procedures takes quite a bit of time, and you might want to check out one of the books that are dedicated to writing managed code under SQL Server.

The stored procedures are stored internally in the database and can be called from external programs. In your architecture, the stored procedures will be called from the business tier. The stored procedures in turn manipulate or access the data store, get the results, and return them to the business tier (or perform the necessary operations).

Figure 2-5 shows the technologies associated with every tier in the three-tier architecture. SQL Server contains the data tier of the application (stored procedures that contain the logic to access and manipulate data) and also the actual data store.

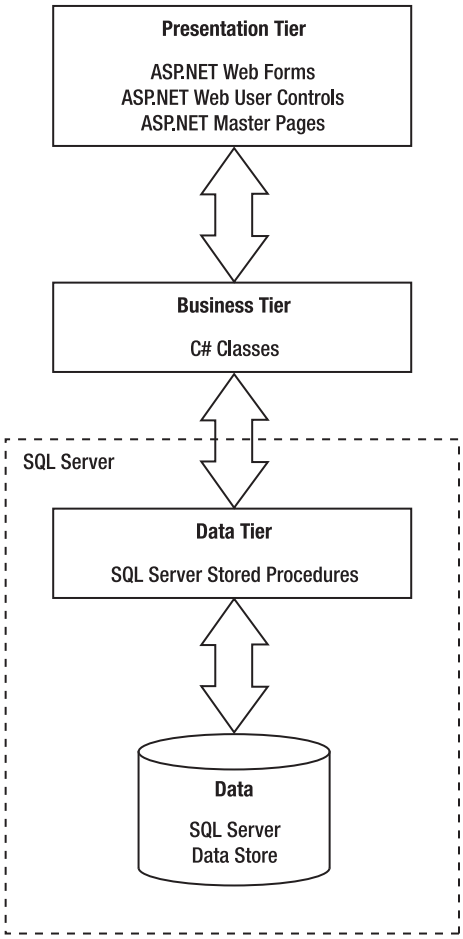

Figure 2-5. *Using Microsoft technologies and the three-tier architecture*

Following Coding Standards

Although coding and naming standards might not seem that important at first, they definitely shouldn't be overlooked. Not following a set of rules for your code almost always results in code that's hard to read, understand, and maintain. On the other hand, when you follow a consistent way of coding, you can say your code is already half documented, which is an important contribution toward the project's maintainability, especially when many people are working at the same project at the same time.

■ **Tip** Some companies have their own policies regarding coding and naming standards, whereas in other cases you'll have the flexibility to use your own preferences. In either case, the golden rule to follow is *be consistent in the way you code*. Check out Microsoft's suggested naming conventions at `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/cpconnamingguidelines.asp`.

Naming conventions refer to many elements within a project, simply because almost all of a project's elements have names: the project itself, namespaces, Web Forms, Web User Controls, instances of Web User Controls and other interface elements, classes, variables, methods, method parameters, database tables, database columns, stored procedures, and so on. Without some discipline when naming all those elements, after a week of coding, you won't understand a line of what you've written.

This book tries to stick to Microsoft's recommendations regarding naming conventions. Now the philosophy is that a variable name should express what the object does and not its data type. We'll talk more about naming conventions while building the site. Right now, it's time to play.

Creating the Visual Web Developer Project

Our favorite toy is, of course, Visual Web Developer. It allows you to create all kinds of projects, including Web Site projects (formerly known as Web Application projects). The other necessary toy is SQL Server, which will hold your web site's data. We'll deal with the database a bit later in this chapter.

■ **Note** At this point, we assume you have Visual Web Developer 2005 Express Edition and SQL Server 2005 Express Edition installed on your computer. It's okay if you use the commercial versions of Visual Studio 2005 or SQL Server 2005, in which case the exercise steps you need to take might be a bit different from what is presented in the book. Consult Appendix A for more details about the installation work.

The first step toward building the BalloonShop site is to open Visual Web Developer and create a new ASP.NET Web Site project. If with previous versions of Visual Studio you needed to have IIS installed, due to the integrated web server of Visual Studio .NET 2005 (named Cassini), you can run the ASP.NET Web Application from any physical folder on your disk. As a result, when creating the Web Site project, you can specify for destination either a web location (such as `http://localhost/BalloonShop`) or a physical folder on your disk (such as `C:\BalloonShop`).

If you have a choice, usually the preferred solution is still to use IIS because of its better performance and because it guarantees that the pages will display the same as the deployed solution. Cassini (the integrated web server) does an excellent job of simulating IIS, but it still shouldn't be your first option. For this book, you can use either option, although our final tests and screenshots were done using IIS.

You'll create the BalloonShop project step-by-step in the exercises that follow. To ensure that you always have the code we expect you to have and to eliminate any possible frustrations or misunderstandings, we'll always include the steps you must follow to build your project in separate Exercise sections. We know it's very annoying when a book tells you something, but the computer's monitor shows you another thing, so we did our best to eliminate this kind of problems.

Let's go.

Exercise: Creating the BalloonShop Project

Follow the steps in this exercise to create the ASP.NET Web Site project.

1. Start Visual Web Developer 2005 Express Edition, choose **File ➤ New Web Site**. In the dialog box that opens, select **ASP.NET Web Site** from the Templates panel, and **Visual C#** for the Language.

2. In the first Location combo box, you can choose from File System, HTTP, and FTP, which determine how your project is executed. If you choose to install the project on the File System, you need to choose a physical location on your disk, such as `C:\BalloonShop\`. In this case, the Web Application is executed using Visual Web Developer's integrated web server (Cassini). If you choose an HTTP location (such as `http://localhost/BalloonShop`), the Web Application will be executed through IIS.

   Make a choice that fits you best. If you go the HTTP way and you're developing on your local machine, make sure that your machine has IIS installed (see Appendix A). For the purpose of this exercise, we're creating the project in the `http://localhost/BalloonShop` location, as shown in Figure 2-6.

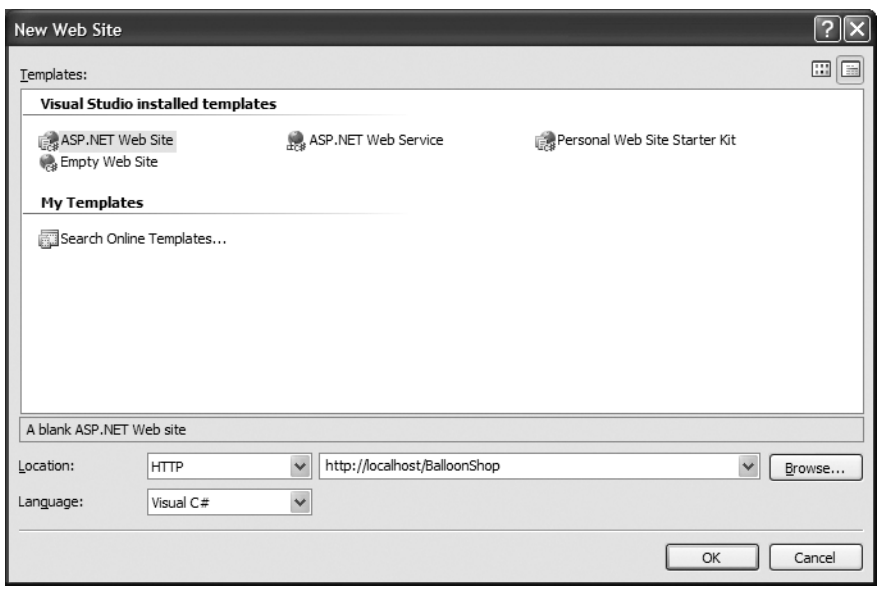

Figure 2-6. *Creating the Visual Studio .NET project*

■ **Note** When creating the project on an HTTP location with the local IIS server, the project is physically created, by default, under the `\InetPub\wwwroot` folder. If you prefer to use another folder, use the Internet Information Services applet by choosing Control Panel ➤ Administrative Tools to create a virtual folder pointing to the physical folder of your choice prior to creating the Web Application with Visual Web Developer. If this note doesn't make much sense to you, ignore it for now.

3. Click **OK**. Visual Studio now creates the new project in the `BalloonShop` folder you specified.

   In the new project, a new Web Form called `Default.aspx` is created by default, as shown in Figure 2-7.

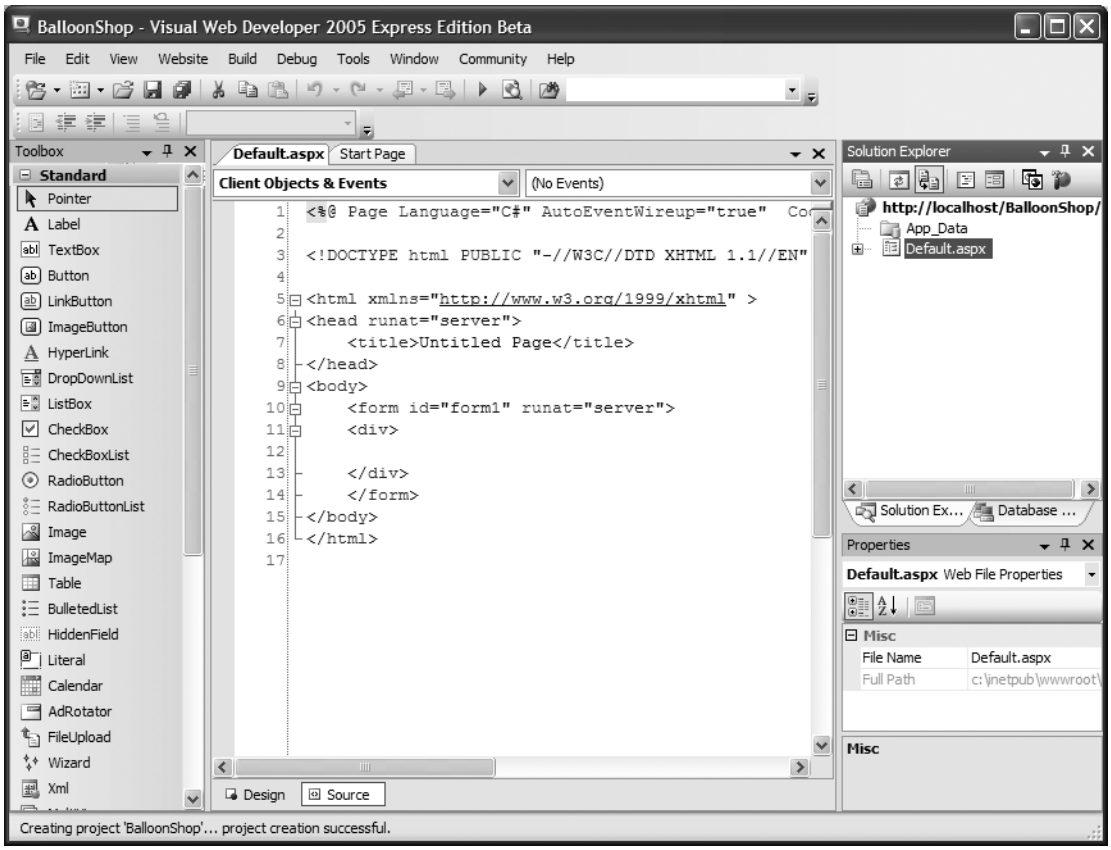

Figure 2-7. *The BalloonShop project in Visual Web Developer 2005 Express Edition*

4. Execute the project in debug mode by pressing **F5**. At this point, Visual Web Developer will complain (as shown in Figure 2-8) that it can't debug the project as long as debugging is not enabled in `web.config` (actually, at this point, the `web.config` file doesn't even exist). Click **OK** to allow Visual Studio to enable debug mode for you. Feel free to look at the newly created `web.config` file to see what has been done for you.

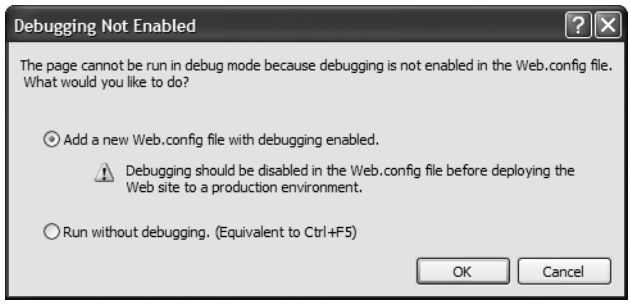

Figure 2-8. *Debugging must be enabled in* `web.config`

5. When executing the project, a new and empty Internet Explorer should open. Closing the window will also stop the project from executing (the Break and Stop Debugging symbols disappear from the Visual Web Developer toolbar, and the project becomes fully editable again).

■ **Note** When executing the project, the web site is loaded in your system's default web browser. For the purposes of debugging your code, we recommend configuring Visual Web Developer to use Internet Explorer by default, even if your system's preferred browser is (for example) Mozilla Firefox. The reason is that Internet Explorer integration seems to work better. For example, Visual Web Developer knows when you close the Internet Explorer window and automatically stops the project from debug mode so you can continue development work normally; however, with other browsers, you may need to manually Stop Debugging (click the Stop square button in the toolbar, or press Shift+F5 by default). To change the default browser to be used by Visual Web Developer, right-click the root node in Solution Explorer, choose Browse With, select a browser from the Browsers tab, and click Set as Default.

How It Works: Your Visual Web Developer Project

Congratulations! You have just completed the first step in creating your e-commerce store!

Unlike with previous versions of ASP.NET, you don't need an IIS virtual directory (or IIS at all, for that matter) to run a Web Application, because you can create the ASP.NET Web Site project in a physical location on your drive. Now it's up to you where and how you want to debug and execute your Web Application!

When not using IIS and executing the project, you'll be pointed to an address like `http://localhost:5392/BalloonShop/Default.aspx`, which corresponds to the location of the integrated web server.

At this moment your project contains three files:

- `Default.aspx` is your Web Form.
- `Default.aspx.cs` is the code-behind file of the Web Form.
- `web.config` is the project's configuration file.

We'll have a closer look at these files later.

Implementing the Site Skeleton

The visual design of the site is usually agreed upon after a discussion with the client and in collaboration with a professional web designer. Alternatively, you can buy a web site template from one of the many companies that offer this kind of service for a reasonable price.

Because this is a programming book, we won't discuss web design issues. Furthermore, we want a simple design that allows you to focus on the technical details of the site. A simplistic design will also make your life easier if you'll need to apply your layout on top of the one we're creating here.

All pages in BalloonShop, including the first page, will have the structure shown in Figure 2-9. In later chapters, you'll add more components to the scheme (such as the login box or shopping cart summary box), but for now, these are the pieces we're looking to implement in the next few chapters.

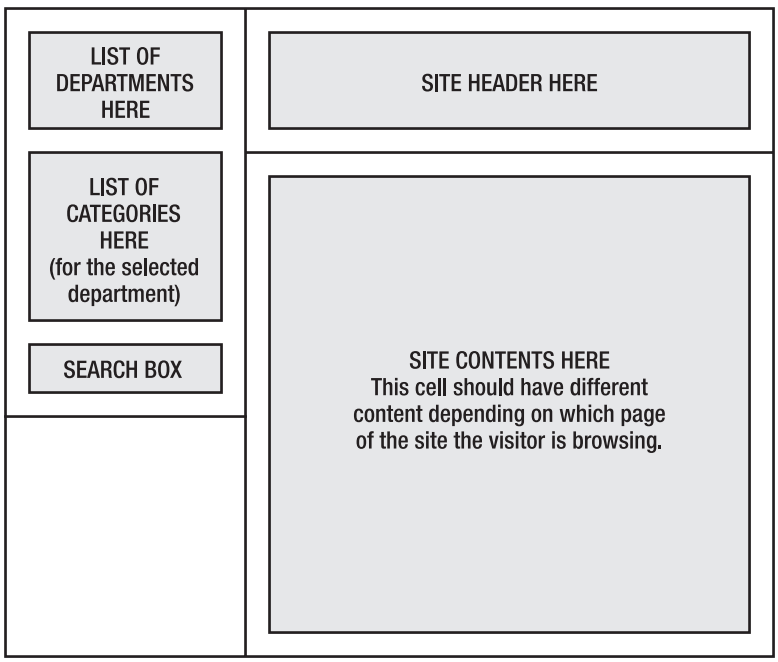

Figure 2-9. *Structure of web pages in BalloonShop*

Although the detailed structure of the product catalog is covered in the next chapter, right now you know that the main list of departments needs to be displayed on every page of the site. You also want the site header to be visible in any page the visitor browses.

You'll implement this structure by creating the following:

- A Master Page containing the general structure of all the web site's pages, as shown in Figure 2-9
- A number of Web Forms that use the Master Page to implement the various locations of the web site, such as the main page, the department pages, the search results page, and so on
- A number of Web User Controls to simplify reusing specific pieces of functionality (such as the departments list box, the categories list box, the search box, the header, and so on)

Figure 2-10 shows a few of the Web User Controls you'll create while developing BalloonShop.

Figure 2-10. *Using Web User Controls to generate content*

Using Web User Controls to implement different pieces of functionality has many long-term advantages. Logically separating different, unrelated pieces of functionality from one another gives you the flexibility to modify them independently and even reuse them in other pages without having to write HTML code and the supporting code-behind file again. It's also extremely easy to extend the functionality or change the place of a feature implemented as a user control in the parent web page; changing the location of a Web User Control is anything but a complicated and lengthy process.

In the remainder of this chapter, we'll write the Master Page of the site, a Web Form for the first page that uses the Master Page, and the Header Web User Control. We'll deal with the other user controls in the following chapters. Finally, at the end of the chapter, you'll create the `BalloonShop` database, which is the last step in laying the foundations of the project.

Building the First Page

At the moment, you have a single Web Form in the site, `Default.aspx`, which Visual Web Developer automatically created when you created the project. By default, Visual Web Developer didn't generate a Master Page for you, so you'll do this in the following exercise.

Exercise: Creating the Main Web Page

1. Click **Website ➤ Add New Item** (or press **Ctrl**+**Shift**+**A**). In the dialog box that opens, choose **Master Page** from the Visual Studio Installed Templates list.
2. Choose **Visual C#** for the language, check the **Place code in a separate file** check box, and change the page name to **`BalloonShop.master`** (the default name `MasterPage.master` isn't particularly expressive). The Add New Item dialog box should now look like Figure 2-11.

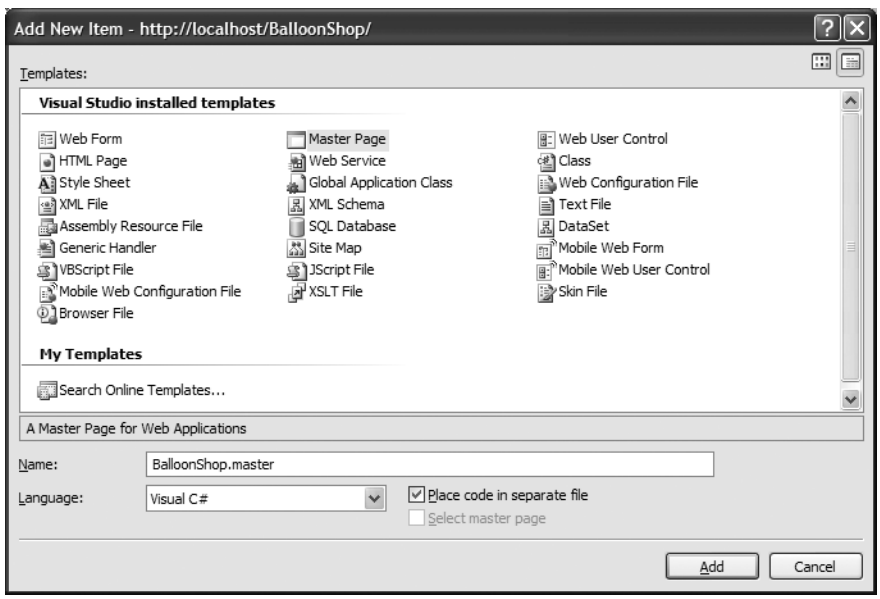

Figure 2-11. *Adding a new Master Page to the project*

3. Click **Add** to add the new Master Page to the project. The new Master Page will be opened with some default code in Source View. If you switch to Design View, you'll see the `ContentPlaceHolder` object that it contains. While in Source View, update its code like this:

```
<%@ Master Language="C#" AutoEventWireup="true"
CodeFile="BalloonShop.master.cs" Inherits="BalloonShop" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>BalloonShop</title>
</head>
<body>
  <form id="Form1" runat="server">
    <table cellspacing="0" cellpadding="0" width="770" border="0">
      <tr>
        <td width="220" valign="top">
          List of Departments
          <br />
          List of Categories
          <br />
        </td>
        <td valign="top">
          Header
          <asp:ContentPlaceHolder ID="contentPlaceHolder" runat="server">
          </asp:ContentPlaceHolder>
        </td>
      </tr>
    </table>
  </form>
</body>
</html>
```

4. Now switch again to Design View; you should see something like Figure 2-12. If you haven't changed the default behavior of Visual Web Developer, you'll see that the `ContentPlaceHolder` object is marked with a little green arrow sign (which is probably hardly visible in the figure). This indicates that the control is marked to be executed at server-side (on the server). All server-side controls (including `Label`s, `TextBox`es, and so on) on the page will be marked with the same green symbol. If you look at the HTML code of the `ContentPlaceHolder`, you'll see the `runat="server"` clause:

```
<asp:contentplaceholder id="contentPlaceHolder" runat="server">
</asp:contentplaceholder>
```

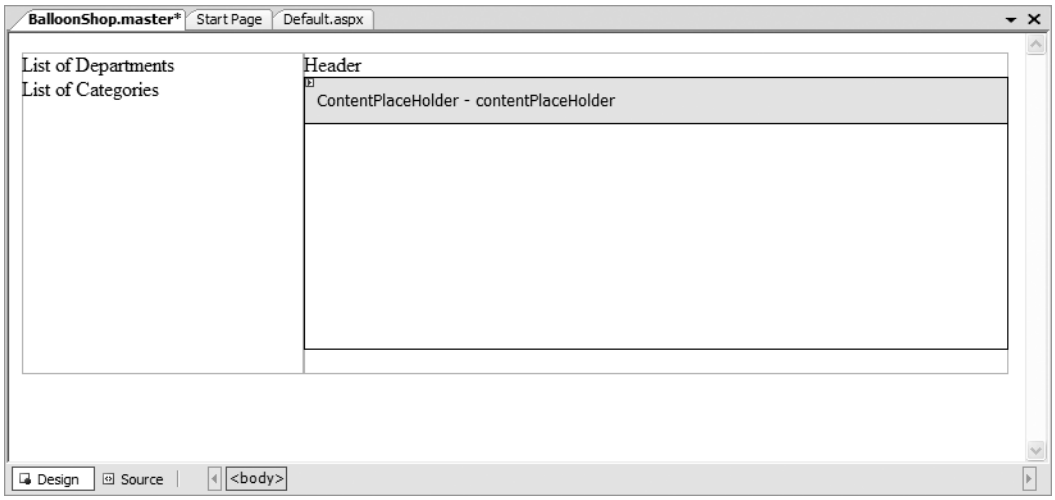

Figure 2-12. *Your New Master Page in Design View*

Master Pages are not meant to be accessed directly by clients, but to be implemented in Web Forms. You'll use the Master Page you've just created to establish the template of the `Default.aspx` Web Form. Because the `Default.aspx` page that Visual Web Developer created for you was not meant to be used with Master Pages (it contains code that should be inherited from the Master Page), it's easier to delete and re-create the file.

5. Right-click **`Default.aspx`** in Solution Explorer and choose **Delete**. Confirm the deletion.

6. Right-click the project root in Solution Explorer and select **Add New Item**. Choose the Web Form template, leave its name as `Default.aspx`, make sure both check boxes **Place code in separate file** and **Select Master Page** are checked, verify that the language is **Visual C#**, and click **Add**. When asked for a Master Page file, choose **`BalloonShop.master`** and click **OK**. Your new page will be created with just a few lines of code, all the rest being inherited from the Master Page:

```
<%@ Page Language="C#" MasterPageFile="~/BalloonShop.master"
AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default"
Title="Untitled Page" %>
<asp:Content ID="Content1" ContentPlaceHolderID="contentPlaceHolder"
Runat="Server">
</asp:Content>
```

When you switch to Design View, `Default.aspx` will look like Figure 2-13.

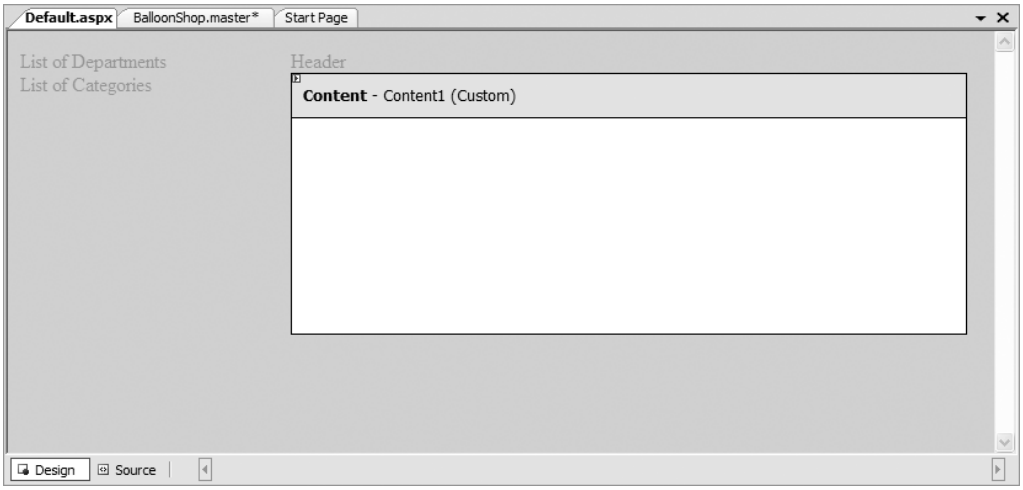

Figure 2-13. *Default.aspx in Design View*

7. Change the title of the page from "Untitled Page" to "Welcome to BalloonShop!" by either using the Properties window in Design View (see Figure 2-14) or by editing the code in Source View like this:

```
<%@ Page Language="C#" MasterPageFile="~/BalloonShop.master"
AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default"
Title="Welcome to BalloonShop!" %>
```

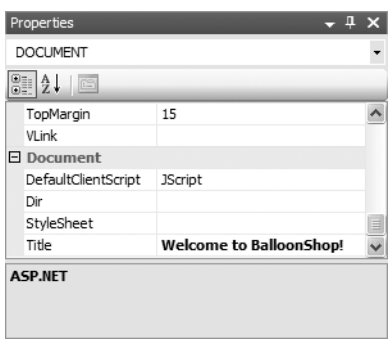

Figure 2-14. *Changing the form name using the Properties window*

8. Press **F5** to execute the project. You should get a page similar to the one in Figure 2-15.

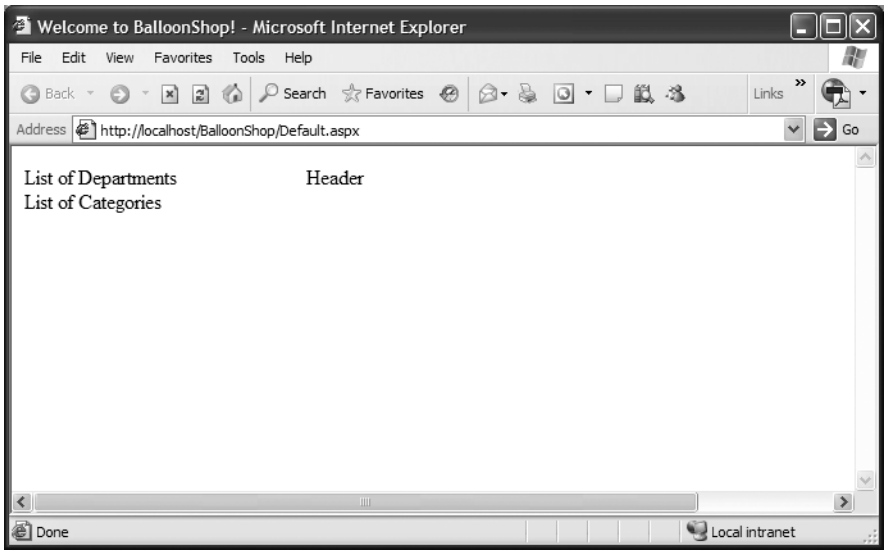

Figure 2-15. *Default.aspx in action*

■ **Note** You need to close the browser window or manually stop the project from running before you can use the Designer window to edit your forms at full power again.

How It Works: The Main Web Page

Right now you have the skeleton of the first BalloonShop page in place. Perhaps it's not apparent right now, but working with Master Pages will save you a lot of headaches later on when you extend the site.

The Master Page establishes the layout of the pages that implement it, and these pages have the freedom to update the contents of the `ContentPlaceHolder` elements. In our case, the header, the list of departments, and the list of categories are standard elements that will appear in every page of the web site (although the list of categories will have blank output in some cases and will appear only when a department is selected—you'll see more about this in the next chapter). For this reason, we included these elements directly in the Master Page, and they are not editable when you're designing `Default.aspx`. The actual contents of every section of the web site (such as the search results page, the department and category pages, and so on) will be generated by separate Web Forms that will be differentiated by the code in the `ContentPlaceHolder` object.

■ **Note** A Master Page can contain more than one `ContentPlaceHolder` object.

The list of departments and the list of categories will be implemented as Web User Controls that generate their output based on data read from the database. You'll implement this functionality in Chapters 3 and 4.

Adding the Header to the Main Page

After so much theory about how useful Web User Controls are, you finally get to create one. The `Header` control will populate the upper-right part of the main page and will look like Figure 2-16.

Figure 2-16. *The BalloonShop logo*

To keep your site's folder organized, you'll create a separate folder for all the user controls. Having them in a centralized location is helpful, especially when the project grows and contains a lot of files.

Exercise: Creating the Header Web User Control

Follow these steps to create the Web User Control and add it to the Master Page:

1. Download the code for this book from the Source Code area at `http://www.apress.com`, unzip it somewhere on your disk, and copy the `ImageFolders\Images` folder to your project's directory (which will be `\Inetpub\wwwroot\BalloonShop\` if you used the default options when creating the project). The `Images` folder contains, among other files, a file named `BalloonShopLogo.png`, which is the logo of your web site. Now, if you save, close, and reload your solution, the `Images` folder will show up in Solution Explorer.

2. Make sure that the project isn't currently running (if it is, the editing capabilities are limited), and that the Solution Explorer window is visible (if it isn't, choose **View ➤ Solution Explorer** or use the default **Ctrl**+**Alt**+**L** shortcut). Right-click the root entry and select **Add Folder ➤ Regular Folder**.

3. Enter **UserControls** as the name of the new folder, as shown in Figure 2-17.

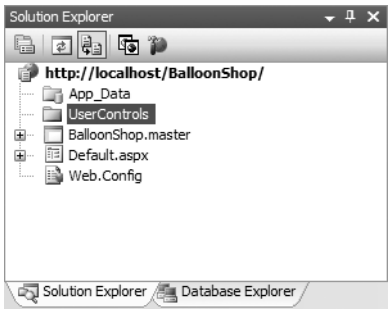

Figure 2-17. *Adding a new folder to the BalloonShop project*

4. Create the `Header.ascx` user control in the `UserControls` folder. Right-click **UserControls** in Solution Explorer and click **Add New Item**. In the form that appears, choose the **Web User Control** template and change the default name to **Header.ascx**. Leave the other options in place (as shown in Figure 2-18), and click **Add**.

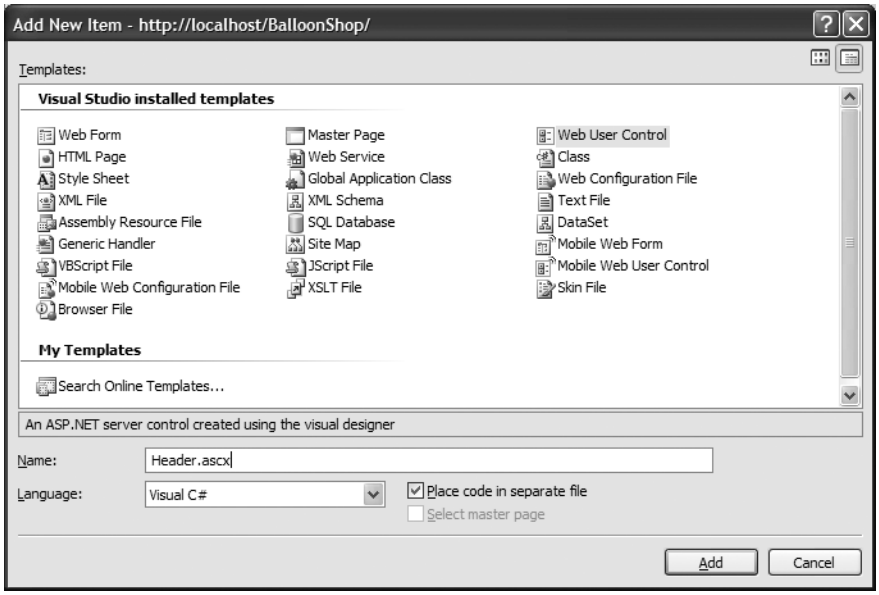

Figure 2-18. *Creating the* `Header.ascx` *Web User Control*

5. The `Header` Web User Control automatically opens in Source View. Modify the HTML code like this:

```
<%@ Control Language="C#" AutoEventWireup="true" CodeFile="Header.ascx.cs"
Inherits="Header" %>
<p align="center">
  <a href="Default.aspx">
    <img src="Images/BalloonShopLogo.png" border="0">
  </a>
</p>
```

■ **Note** If you switch the control to Design View right now, you won't see the image because the relative path to the `Images` folder points to a different absolute path at designtime than at runtime. At runtime, the control is included and run from within `BalloonShop.master`, not from its current location (the `UserControls` folder).

---

6. Open `BalloonShop.master` in Design View, drag `Header.ascx` from Solution Explorer, drop it near the "Header" text, and then delete the "Header" text from the cell. The Design view of `BalloonShop.master` should now look like Figure 2-19.

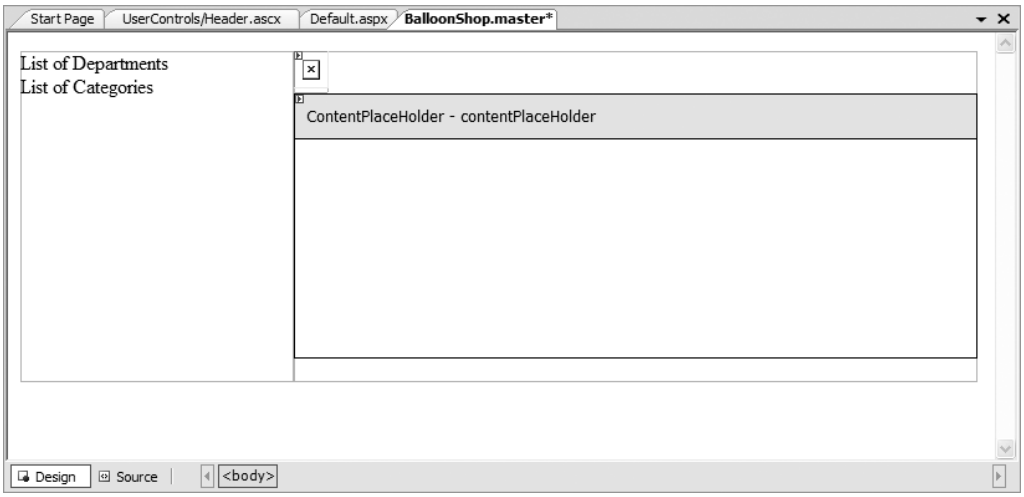

Figure 2-19. *Adding* `Header.ascx` *to the Master Page*

7. Click **Debug ➤ Start** (**F5** by default) to execute the project. The web page will look like Figure 2-20.

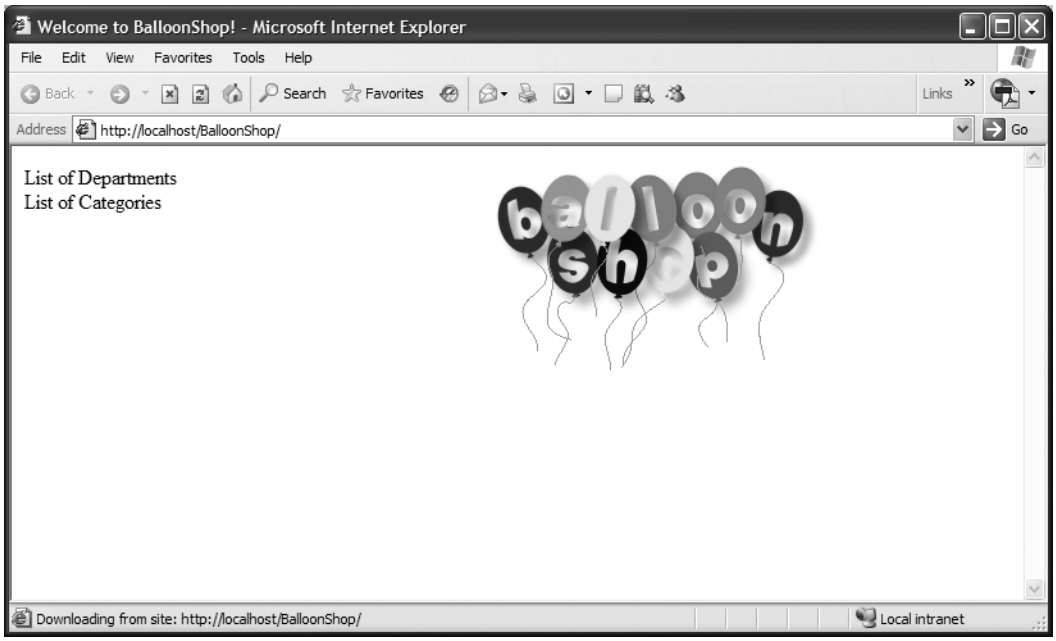

Figure 2-20. *BalloonShop in action*

How It Works: The Header Web User Control

Congratulations once again! Your web site has a perfectly working header! If you don't find it all that exciting, then get ready for the next chapter, where you'll get to write some real code and show the visitor dynamically generated pages with data extracted from the database. The final step you'll make in this chapter is to create the `BalloonShop` database (in the next exercise), so everything will be set for creating your product catalog!

Until that point, make sure you clearly understand what happens in the project you have at hand. The Web User Control you just created is included in the `BalloonShop.master` Master Page, so it applies to all pages that use this Master Page, such as `Default.aspx`. Having that bit of HTML written as a separate control will make your life just a little bit easier when you need to reuse the header in other parts of the site. If at any point the company decides to change the logo, changing it in one place (the `Header.ascx` control) will affect all pages that use it.

This time you created the control by directly editing its HTML source, but it's always possible to use the Design View and create the control visually. The HTML tab of the Toolbox window in Visual Studio contains all the basic HTML elements, including `Image`, which generates an `img` HTML element.

Let's move on.

Creating the SQL Server Database

The final step in this chapter is to create the SQL Server database, although you won't get to effectively use it until the next chapter. SQL Server 2005 Express Edition, the free version of SQL

Server, doesn't ship with the SQL Server Management Studio (formerly known as the Enterprise Manager). However, now you can also create databases using Visual Web Developer's features.

All the information that needs to be stored for your site, such as data about products, customers, and so on, will be stored in a database named, unsurprisingly, BalloonShop.

Exercise: Creating a New SQL Server Database

The following steps show how to create the BalloonShop database using Visual Studio. However, feel free to use the tool of your choice.

1. In your Visual Web Developer project, make sure the Database Explorer window is open. If it isn't, you can either select **View ➤ Database Explorer** or press the default shortcut keys **Ctrl**+**Alt**+**S**.

2. In Server Explorer, right-click the **Data Connections** entry, and select **Create New SQL Server Database**. In the window that appears (see Figure 2-21), enter the name of the SQL Server instance where you want to create the database (note that you can use `(local)` instead of the local computer's name), the login information, and the name of the new database. If you installed SQL Server Express using the default options as shown in Appendix A, then your server name should be `(local)\SqlExpress`; in the installation process you are provided with the necessary data to connect to your database. Enter **BalloonShop** for the name of the new database.

Figure 2-21. *Creating a new SQL Server database using Visual Web Developer*

■ **Note** Using Windows Authentication, your local Windows account will be used to log in to SQL Server. If you installed SQL Server yourself, you'll have full privileges to SQL Server, and everything will run smoothly; otherwise, you'll need to make sure you're provided with administrative privileges on the SQL Server instance. With SQL Server Authentication, you need to provide a username and password, but note that this authentication mode is disabled by default in SQL Server.

How It Works: The SQL Server Database

That's it! You've just created a new SQL Server database! The Server Explorer window in Visual Studio allows you to control many details of the SQL Server instance. After creating the BalloonShop database, it appears under the Data Connections node in Server Explorer. Expanding that node reveals a wide area of functionality you can access directly from Visual Web Developer (see Figure 2-22). Because the database you just created is (obviously) empty, its subnodes are also empty, but you'll take care of this detail in the following chapters.

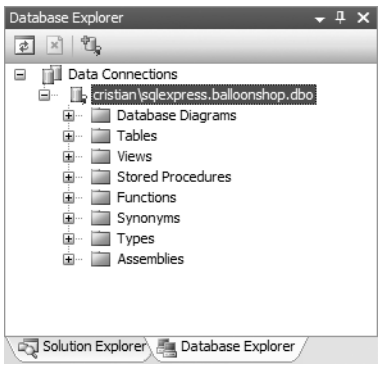

Figure 2-22. *Accessing the BalloonShop database from the Database Explorer*

Downloading the Code

The code you have just written is available in the Source Code area of the Apress web site at `http://www.apress.com` or at the author's web site at `http://www.CristianDarie.ro`. It should be easy for you to read through this book and build your solution as you go; however, if you want to check something from our working version, you can. Instructions on loading the chapters are available in the `Welcome.html` document in the download. You can also view the online version of BalloonShop at `http://web.cristiandarie.ro/BalloonShop`.

Summary

We covered a lot of ground in this chapter, didn't we? We talked about the three-tier architecture and how it helps you create powerful flexible and scalable applications. You also saw how each of the technologies used in this book fits into the three-tier architecture.

So far you have a very flexible and scalable application because it only has a main web page formed from a Web Form, a Master Page, and the `Header` Web User Control, but you'll feel the real advantages of using a disciplined way of coding in the next chapters. In this chapter, you have only coded the basic, static part of the presentation tier and created the BalloonShop database, which is the support for the data tier. In the next chapter, you'll start implementing the product catalog and learn a lot about how to dynamically generate visual content using data stored in the database with the help of the middle tier and with smart and fast presentation tier controls and components.