Deploying .NET Applications
Learning MSBuild and ClickOnce

Sayed Y. Hashimi and Sayed Ibrahim Hashimi
To my parents, Sayed A. and Sohayla Hashimi,  
and to my wife and daughter, Farishta and Fairoza.  
—Sayed Y. Hashimi

To my parents, Sayed A. and Sohayla Hashimi,  
because this would have not been possible without their support and guidance.  
—Sayed Ibrahim Hashimi
Contents at a Glance

About the Authors ................................................................. xi
About the Technical Reviewer ............................................. xiii
Acknowledgments ............................................................... xv
Introduction ........................................................................ xvii

CHAPTER 1 Deployment Prerequisites ......................................... 1
CHAPTER 2 The Unified Build Engine: MSBuild .......................... 21
CHAPTER 3 MSBuild: By Example ........................................... 45
CHAPTER 4 Extending MSBuild .............................................. 75
CHAPTER 5 Introducing Team Foundation Server and Team Build .... 107
CHAPTER 6 Deploying Smart Clients with ClickOnce ................. 137
CHAPTER 7 ClickOnce Updates, Security, and the Bootstrapper .... 161
CHAPTER 8 The ClickOnce Data Directory and Deploying Prerequisites ... 185
CHAPTER 9 ClickOnce Tools and Scenarios ............................... 219

INDEX ................................................................................ 253
# Contents

About the Authors ................................................................. xi
About the Technical Reviewer ............................................... xiii
Acknowledgments .................................................................... xv
Introduction .............................................................................. xvii

## CHAPTER 1  Deployment Prerequisites .................................. 1

- Types of Applications ......................................................... 1
  - Windows Forms (Smart Client) Applications ........................ 2
  - Web Applications ............................................................ 6
  - Web Services .................................................................. 9
  - Smart Device Applications ............................................. 10
  - Windows Services .......................................................... 11
  - Console Applications ...................................................... 13
  - Hosted Applications ....................................................... 13
- Application Architectures ..................................................... 14
  - Client-Server Architecture ............................................ 14
  - N-Tier Architecture ....................................................... 15
  - Service-Oriented Architecture ........................................ 16
- Strategies for Deploying the .NET Framework ....................... 17
  - Where Do You Need the .NET Runtime? .......................... 17
  - Typical Deployment of a Smart Client ............................... 18
  - Typical Deployment of a Thin Client ................................. 18
- Summary .............................................................................. 19

## CHAPTER 2  The Unified Build Engine: MSBuild .................. 21

- Introducing Build Tools and Systems ................................... 22
  - Make-Style Build Tools .................................................. 22
  - Ant/NAnt .................................................................. 23
- Introducing MSBuild ........................................................... 24
  - Properties .................................................................. 26
  - Targets .................................................................... 27
  - Tasks ....................................................................... 37
- Summary ............................................................................ 43
## CHAPTER 3  MSBuild: By Example

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introducing Well-Known Metadata</td>
<td>45</td>
</tr>
<tr>
<td>Formatting Your Output</td>
<td>48</td>
</tr>
<tr>
<td>Editing MSBuild Files with IntelliSense</td>
<td>54</td>
</tr>
<tr>
<td>Integrating MSBuild into Visual Studio</td>
<td>55</td>
</tr>
<tr>
<td>Introducing Custom Metadata</td>
<td>57</td>
</tr>
<tr>
<td>Understanding the Difference Between <code>@</code> and <code>%</code></td>
<td>59</td>
</tr>
<tr>
<td>Using Environment Variables in Your Project</td>
<td>63</td>
</tr>
<tr>
<td>Reusing MSBuild Project Elements</td>
<td>63</td>
</tr>
<tr>
<td>Dealing with MSBuild Errors</td>
<td>69</td>
</tr>
<tr>
<td>Summary</td>
<td>74</td>
</tr>
</tbody>
</table>

## CHAPTER 4  Extending MSBuild

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logging with MSBuild</td>
<td>75</td>
</tr>
<tr>
<td>Writing a Logger</td>
<td>77</td>
</tr>
<tr>
<td>Using NUnit and MSBuild</td>
<td>85</td>
</tr>
<tr>
<td>Summary</td>
<td>106</td>
</tr>
</tbody>
</table>

## CHAPTER 5  Introducing Team Foundation Server and Team Build

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introducing Visual Studio Team System (VSTS)</td>
<td>107</td>
</tr>
<tr>
<td>Introducing Team Build</td>
<td>108</td>
</tr>
<tr>
<td>Introducing the Team Foundation Build Architecture</td>
<td>108</td>
</tr>
<tr>
<td>Team Foundation Build Client</td>
<td>109</td>
</tr>
<tr>
<td>Application Tier</td>
<td>110</td>
</tr>
<tr>
<td>Data Tier</td>
<td>110</td>
</tr>
<tr>
<td>Build Machine</td>
<td>110</td>
</tr>
<tr>
<td>Drop Location</td>
<td>110</td>
</tr>
<tr>
<td>Using Team Foundation Build</td>
<td>110</td>
</tr>
<tr>
<td>Creating a New Team Project</td>
<td>111</td>
</tr>
<tr>
<td>Understanding the Team Project Fundamentals</td>
<td>112</td>
</tr>
<tr>
<td>Work Items</td>
<td>113</td>
</tr>
<tr>
<td>Documents</td>
<td>113</td>
</tr>
<tr>
<td>Reports</td>
<td>114</td>
</tr>
<tr>
<td>Team Builds</td>
<td>114</td>
</tr>
<tr>
<td>Source Control</td>
<td>114</td>
</tr>
<tr>
<td>Placing Code in Source Control</td>
<td>114</td>
</tr>
</tbody>
</table>
CHAPTER 6  Deploying Smart Clients with ClickOnce  

Introducing Side-by-Side Deployment  
Looking at the Previous Approaches of Deploying Windows Forms Applications  
MSI Deployment  
No-Touch Deployment  
The Updater Application Block  
Introducing ClickOnce  
Introducing the ClickOnce Deployment Methods  
Introducing the ClickOnce Architecture  
Seeing ClickOnce in Action  
Updating and Versioning with ClickOnce  
Introducing ClickOnce Security  
Customizing Deployment with the ClickOnce API  
Understanding the Bootstrapper  
Summary

CHAPTER 7  ClickOnce Updates, Security, and the Bootstrapper  

Understanding the ClickOnce Manifest Files  
Offline vs. Online Applications  
Performing ClickOnce Updates  
Configuring Update Notification  
Configuring Application Update Policy  
Understanding ClickOnce Security  
Using Trusted Publishers in ClickOnce  
Seeing a Trusted Publisher in Action  
Introducing Code Access Security (CAS)  
Introducing Partially Trusted Applications with ClickOnce  
Full Trust vs. Partial Trust  
Available Permissions vs. Actual Permissions  
Deploying Prerequisites with ClickOnce  
Using the Bootstrapper to Install Prerequisites  
Summary
About the Authors

SAYED Y. HASHIMI was born in Kabul, Afghanistan, and now resides in Jacksonville, Florida. Sayed has expertise in the areas of healthcare, banking, logistics, scientific computing, and civil/structural engineering. In his professional career, Sayed has developed large-scale distributed applications with a variety of programming languages and platforms, including C++, Java, and .NET. He has published articles in major software journals and is the principal author of *Pro Service-Oriented Smart Clients with .NET 2.0* (Apress, 2005). Sayed has a master’s degree in engineering from the University of Florida. You can reach Sayed by visiting [http://www.sayedhashimi.com](http://www.sayedhashimi.com).

SAYED IBRAHIM HASHIMI has a computer engineering degree from the University of Florida. He works in Jacksonville, Florida, as a developer and architect. He is an expert in the financial, education, and collection industries. His primary focus is working with .NET, but he also has extensive industrial experience with Java-based technologies. Sayed’s research interests include a wide range of topics including computer graphics, peer-to-peer technologies, and lucid dreaming. You can read Sayed’s blog at [http://www.sedodream.com](http://www.sedodream.com). When he’s not busy creating software or dreaming, you’re likely to find him at the local coffee shop.
BART DE SMET was born on February 11, 1983, in Belgium and has a master’s degree in computer science from Ghent University. Since early 2000, Bart has been involved in the wonderful world of .NET and is also a Visual C# MVP. While keeping his brain busy with further university studies, he focuses on C#, the CLR, SQL Server 2005, and WinFX. Regularly you can find Bart speaking at various European Microsoft events, and if time permits, he writes articles for the local MSDN Web site. To read about his adventures in the .NET galaxy, check out Bart’s blog at http://blogs.bartdesmet.net/bart.
Writing this book took effort from not only the authors but also from some of the very talented staff at Apress. Therefore, we would like to thank Jonathan Hassell, Richard Dal Porto, Ellie Fountain, and Kim Wimpsett. We would also like to acknowledge the technical reviewer, Bart De Smet, for taking the time to review the book. His corrections and commentary were invaluable.
This book covers two important aspects of the software life cycle: build and deployment. The coverage of these crucial topics is only half the attraction of this book, though. The other half is the technologies covered: MSBuild and ClickOnce.

What Is MSBuild?

Previously the build process that Visual Studio followed was basically a black box and was difficult to customize. With the arrival of Visual Studio 2005 and .NET 2.0 comes the arrival of the Microsoft Build Engine, otherwise known as MSBuild. MSBuild is the utility that Visual Studio uses to build your managed (C#, VB .NET, and J#) projects. MSBuild is an XML-based build engine and a tool that has been developed with customizability and extensibility in mind from its conception. By using MSBuild, you can change how your projects are built, creating customizations to fit your needs.

With the advent of MSBuild, you no longer have to rely on third-party tools to handle the custom aspects of building your application. In addition, not only is this approach supported by Microsoft, but it is completely integrated with Visual Studio. If you need to tweak the settings for the C# compiler or how resources are generated, you now have this ability. The entire build process is open, and you can customize it in any way necessary. With other third-party tools this is simply unachievable.

In this book, we will start with MSBuild concepts that you need to know in order to get started, and then we will cover some advanced topics. Over the course of a few chapters we will take you from an MSBuild newbie to an MSBuild expert! For example, we will discuss how to inject custom steps into the build process, how to create custom tasks, and much more. We will cover all of the necessary topics that you need to know in order to use MSBuild in all the great ways that it was intended.

What Is ClickOnce?

For more than a decade now, technology decision makers have implemented business processes using “the disconnected Web” simply because Web applications are easy to deploy. If you perform a feature-by-feature comparison of a Web application versus a desktop application (such as a Windows Forms application), you’ll be amazed to see what you bypass just to have something easily deployed (see Table 1).
Table 1. *Web Application vs. Desktop Application*

<table>
<thead>
<tr>
<th>Feature</th>
<th>Desktop Application</th>
<th>Web Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Has interactive and stateful user interface?</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Offers offline support?</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Uses desktop resources?</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Is easy to deploy?</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Organizations have repeatedly given up interactive and stateful applications just so they can easily deploy them. In addition, organizations have repeatedly given up all the benefits of having access to a workstation’s local resources so they can easily deploy applications. Finally, organizations have repeatedly given up the benefits of having applications function without a server connection so they can easily deploy them. You don’t have to do this anymore. ClickOnce, finally, solves the complicated problem of “easily deploying a desktop application” and gives the desktop back to you.

ClickOnce enables you to deploy Windows Forms applications just like you deploy Web-based applications. In addition, ClickOnce provides automatic updates and traditional features found in a Windows Installer, without the disadvantages. For example, the ClickOnce technology can add entries to the user’s Start menu and provides an icon in Add/Remove Programs for the user to uninstall the application. It does this without requiring users to be administrators on their workstation. ClickOnce provides all of this out of the box.

With ClickOnce you get the ease of deployment of a Web application along with the following additional benefits inherent in a Windows Forms application:

- Web-based installation
- Automatic and configurable updates via a URL
- Installation without administrator privileges (users don’t have to be admins to install ClickOnce applications)
- Automatic rollback facilities and traditional desktop installations (such as a menu item under the user’s Start menu)

These features, as a whole, have not been available to thick client applications in the past. With ClickOnce, organizations can return to offering dynamic applications that interact with the user’s desktop (for example, with Microsoft Office, a printer, a network, and so on) while providing easy installation and automatic updates.

**Who Should Read This Book?**

This book was written for developers and deployment engineers working with .NET 2.0 on the Windows platform. Developers will benefit from reading this book because build and deployment are fundamental aspects of writing and testing software. Deployment engineers will benefit from reading this book because ClickOnce is now the recommended deployment model for Windows Forms applications and because MSBuild is now the unified build engine for the Windows platform.
What’s in This Book?

This book covers build and deployment using MSBuild and ClickOnce. The book is broken up into two parts; the first part (Chapters 2–5) covers MSBuild, and the second part (Chapters 6–9) covers ClickOnce. Here is a breakdown of each chapter:

Chapter 1, “Deployment Prerequisites”: Most large organizations have a team dedicated to build and deployment. Individuals on a team like this are not developers. In this chapter, we’ll assume you are not a developer and give you the proper background required to do build and deployment. We’ll talk about .NET, application architecture, and various types of applications. The goal of this chapter is to help you to understand what you can expect to build and deploy.

Chapter 2, “The Unified Build Engine: MSBuild”: In previous versions of Visual Studio, the build process was mostly a black box; because of this, performing customizations to the build process was not very easy. With the new versions of Visual Studio and the .NET Framework, the build process is fully exposed and documented. It is easy to fine-tune the steps that will be followed when your projects are built. MSBuild is an XML-based build system; in this chapter, we’ll introduce MSBuild and its fundamental concepts.

Chapter 3, “MSBuild: By Example”: In Chapter 2, we’ll outline the fundamentals of MSBuild. Like with many other technologies, it is easier to get a feel for MSBuild when you see it in use in different scenarios. The aim of this chapter is to provide real examples that will provide a concrete foundation to your MSBuild knowledge. Topics vary from how to use MSBuild item metadata to the difference between the @ syntax and the % notation.

Chapter 4, “Extending MSBuild”: MSBuild is a system with extensibility as a focal point from its conception. Two aspects that MSBuild provides are flexible and powerful extensions: custom tasks and custom loggers. In this chapter, we present a real-world custom task from the ground up. This task and its accompanying targets file are responsible for executing any NUnit tasks that are contained in the built assemblies. As a sample of a real-world logger, we’ll show how to create a custom XML logger.

Chapter 5, “Introducing Team Foundation Server and Team Build”: With this version of Visual Studio, Microsoft has made some other tools available. One of these tools is the Team Foundation Server (TFS). A part of TFS is a new source control management tool. When using TFS, you can also use Team Build, which is a utility that can create, maintain, and execute public builds. For enterprise organizations, creating and verifying a public build is a critical component of projects. With TFS and Team Build, you can achieve this. In this chapter, we’ll introduce the necessary concepts to use TFS and Team Build to create and customize your public build.

Chapter 6, “Deploying Smart Clients with ClickOnce”: This chapter opens the second part of the book—deploying Windows Forms applications with ClickOnce. This chapter is an overview of what ClickOnce is. We’ll start by building the case for why ClickOnce is important. We’ll talk about some of the technologies that tried to do the same thing but failed. We’ll give short introductions to how ClickOnce supports automatic updates. We’ll also cover how to handle the sensitive issue of giving an application the proper privileges to do what it needs on the client.
Chapter 7, “ClickOnce Updates, Security, and the Bootstrapper”: This chapter tells you everything you need to know about ClickOnce updates, security, and the generic bootstrapper. We’ll start by dissecting the deployment and application manifest files. We’ll then jump into how an application is configured for updates and when and how an application is updated in the background. After you understand the details of ClickOnce updates, we’ll talk about ClickOnce security. Historically, thick client applications that have a client-side footprint have always been restricted to a security sandbox. This sandbox either was not configurable at all or was configurable in a way that was not practical. After reading this chapter, you’ll see how ClickOnce solves this problem in a practical manner. The last topic we’ll talk about in this chapter concerns getting application prerequisites deployed with your ClickOnce applications, which will prep you for the next chapter.

Chapter 8, “The ClickOnce Data Directory and Deploying Prerequisites”: Nontrivial business applications today need a way to store application data. Storing application state is not something new and is easily accomplished if you have a connection to your database. But what if your application is a smart client and has to support offline capabilities? In other words, where do you store application state if you don’t have a connection to your database on the network? ClickOnce provides the ClickOnce data directory for you to store application state. The ClickOnce data directory is something special and is managed as you move from one version of your application to the next. This chapter talks about the data directory, offline support, and how to migrate data as your application gets updated. The second portion of this chapter is about deploying custom prerequisites. Visual Studio 2005 comes with a short list of popular packages that you can deploy with your application, but what if you have your own prerequisites that you built or one that is not in the list? How do you deploy your own prerequisites? This chapter will tell you how to do that.

Chapter 9, “ClickOnce Tools and Scenarios”: This chapter will talk about three tools that will help in deploying ClickOnce applications and some common ClickOnce scenarios. The tools discussed include the Bootstrapper Manifest Generator (BMG), the Manifest Generation and Editing (MAGE) tool, and MSBuild. The BMG is a Windows Forms application that provides a user interface for building the package and product manifest files that are required to deploy a custom prerequisite. The MAGE tool is a Windows Forms application that helps you build the deployment and application manifest files for your ClickOnce applications. We’ll also talk about how you can automate a ClickOnce deployment—of course, you do this using MSBuild tasks. Finally, we’ll present some common ClickOnce scenarios, covering practical ClickOnce problems and offering possible solutions.

After reading this book, you’ll have a good understanding of the fundamentals of MSBuild and ClickOnce. You’ll also understand how to use the two technologies to establish a build and deployment process in your organization.
Build and deployment are engineering problems, and in most big organizations, these are delegated to an entire team. The members of these teams aren’t necessarily developers, yet they are experts on build and deployment processes and engineering. That is, you do not need to know how to code or know how something was created to be able to build and/or deploy it. And you should not have to know this!

Having said that, as a build and deployment engineer, you should have a fundamental understanding of the various types of applications and application architectures. You should, for example, know the types of components in a Web application versus a Windows Forms application. Similarly, you should know the differences between client-server architecture and n-tier architecture. Why is this important? It is important for a deployment and build engineer to be familiar with the various types of applications and application architectures for two reasons. The first and obvious reason is that in order to build an application or deploy it, you need to know what components it contains. For example, with a Web application, it helps to know it has a configuration file and you may have to modify this file when you write a build script for it. With a client-server application, it helps to know it has a client-side deployment and a server-side deployment. Having this knowledge helps you do your job better. The second important reason is that often deployment engineers have to perform a basic level of testing after deploying an application. Having some basic knowledge about the type of application and its architecture can go along way to resolving some fundamental problems.

In this chapter, we will define the various types of applications and briefly describe commonly used application architectures. We will also define the .NET Framework and describe methods of deploying the .NET runtime.

**Types of Applications**

With the .NET Framework, you can build Windows Forms applications, Web applications, Web services, smart device applications, Windows services, console applications, and hosted applications:

*Windows Forms applications:* Windows Forms applications are applications with a graphical user interface (GUI) front end, and they run on desktops. Examples of this type of application include Microsoft Word, Microsoft Excel, and so on.
**Web applications:** Web applications are applications built with ASP.NET. These applications have a server-side component and a client-side component. The server-side contains the business logic, and the client-side contains the view (GUI) that is displayed in a browser. Web applications are accessed via a uniform resource locator (URL)—for example, http://www.sayedhashimi.com.

**Web services:** Web services are standards-based systems accessible over a network such as the Internet. Web services are generally employed to connect disparate systems. Web services are sometimes called XML Web Services.

**Smart device applications:** Smart device applications are applications that target mobile devices (for example, Smartphone devices). Smart device applications are built with the .NET Compact Framework, a subset of the .NET Framework.

**Windows services:** Windows services are executables that run in the background. The special feature of Windows services is that they don't require an interactive user. That is, Windows services can run while no one is logged on to the system. An example of a Windows service is a device driver or an application that performs background tasks based on a timer.

**Console applications:** Console applications are executables that are run from the Windows command prompt. Functionally, console applications are similar to Windows Forms applications; the difference is that console applications don't have a Windows Forms user interface and are text oriented.

**Hosted applications:** Last but not least, you can build hosted applications with the .NET Framework. Hosted applications are applications that allow the hosting of managed code inside an application. Hosted applications provide the facility for you to allow your customers (clients) to extend your application. Hosted applications are built with something called Visual Studio Tools for Applications (VSTA). Microsoft also has a variation of VSTA for the Microsoft Office suite called Visual Studio Tools for the Microsoft Office System (VSTO). The idea behind VSTO is to leverage the power of Office, Visual Studio, and managed code to build more feature-rich applications. Historically, VSTO came before VSTA; Microsoft extended the idea in VSTA to allow third parties to benefit from managed code extensibility in its own products.

That's the quick, five-minute tour. We'll now cover these types of applications individually so you can better understand them and the components they contain.

---

**Windows Forms (Smart Client) Applications**

Windows Forms applications are desktop applications that have rich user interfaces. For example, the Visual Studio integrated development environment (IDE) is an example of a Windows Forms application.1 Windows Forms applications are built using the .NET Framework class libraries and have the following features:

---

1. Visual Studio 2005 was not actually written using Windows Forms, although part of it is managed code. The user interface is an example of a Windows Forms application.
• They possess a dynamic user interface with rich controls (for example, DataGrids).

• Users generally have the ability to do sophisticated actions quickly (for example, dragging and dropping).

• The application is installed on desktops and thus uses desktop resources. For example, the application can use the printer, the hard drive, and so on. The application can also communicate with running applications on the machine or spawn new processes and threads.

Recently, Microsoft decided to label Windows Forms applications as *smart clients* (see Figure 1-1).

![Figure 1-1. Features of a thick² client, a thin client, and a smart client](image)

Figure 1-1. *Features of a thick² client, a thin client, and a smart client*

Smart client applications are Windows Forms applications with several additional features to those listed previously. For instance, a smart client has the following features:

• It supports offline capabilities. That is, the application doesn't require a network connection and is intelligent about detecting network connectivity automatically. So, for example, if you run a smart client application on your laptop and decide to go talk to a client in an area where you can't access your network (or the Internet), the smart client will still work even though your database is not accessible.

• It is easy to deploy and update.

2. *Thick client, rich client, and fat client* are synonyms.
In Visual Studio 2005, you create Windows Forms/smart client applications by choosing Windows Application in the New Project dialog box, under Visual C# or Visual Basic (see Figure 1-2).

Windows Forms applications comprise an executable, zero or more dependent assemblies, resource files, and an application configuration file. The executable has an .exe extension, and the dependent assemblies typically have a .dll extension. In .NET, these DLLs are called assemblies. The dependent assemblies are generally placed in a folder named bin or directly next to the executable. Often the application will also use shared

---

3. You can also embed resource files, such as images, data files, and so on, within assemblies.
4. Strictly speaking, putting dependent assemblies within the bin directory is a Visual Studio convention. The common language runtime (CLR) assembly loader uses a concept known as probing to locate assemblies. The bin directory happens to be one of the directories that is “probed” when the CLR looks to load an assembly. For more details about this, see http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconassemblies.asp.
assemblies from the global assembly cache (GAC). The GAC contains assemblies that are shared among the applications installed on the machine. You can see the contents of the GAC by going to %windir%\assembly.

A deployment of a Windows Forms application looks like Figure 1-3.

![Application Root Folder](image)

Figure 1-3. Typical Windows Forms deployment

You can deploy Windows Forms applications using ClickOnce. You can use ClickOnce to deploy a rich client application using a Web-based deployment model. That is, you can deploy Windows Forms applications over the Web.

Other deployment options exist, such as Windows Installer (MSI), but ClickOnce is the new recommended method of deploying Windows Forms applications. Visual Studio 2005 has built-in support for deploying Windows Forms applications. Figure 1-4 shows the Publish dialog box used to configure the deployment of a Windows Forms application. We will talk about this in great detail in later chapters.

---

5. ClickOnce is new deployment technology built into the .NET runtime 1.0.
Web Applications

A Web application is an application that is targeted to render in a browser. With the .NET Framework, you build Web applications using ASP.NET. To build a new Web application using Visual Studio 2005, you choose File ➤ New ➤ Web Site.

Figure 1-5 shows the New Web Site dialog box in Visual Studio 2005. An ASP.NET application consists of dynamic pages, static pages, configuration files, resources, and dependent assemblies. From a deployment perspective, it is important to know that a Web application’s dependent assemblies are located in a folder named bin. Moreover, the configuration of a Web application is stored in a file called web.config. The web.config file is an Extensible Markup Language (XML) file. Application authors generally put environment-specific settings (for example, a database connection string) in this file. Therefore, during deployment, the file will likely need to be modified to reflect the environment in which the application is being deployed. In the past, this task was either done by hand or done by an automated script. With Visual Studio 2005, you can use a Web-based administration console to modify the web.config file of an application (see Figure 1-6).

---

6. ASP.NET requires Web applications to have a bin folder; with Windows Forms applications, the bin folder is just one place where the probing process looks for assemblies.
Figure 1-5.  *New Web Site* dialog box in Visual Studio 2005

Figure 1-6.  *Web Site Administration Tool* console
You can access the administration console from the Website ➤ ASP.NET Configuration menu item in Visual Studio 2005. As shown in Figure 1-6, the administration console is a Web-based tool. From the URL, you can conclude that when Visual Studio was installed, it created an application called asp.netwebadminfiles whose default.aspx file takes the path to a Web application. With this path, the application knows which application’s configuration file to display in the administration console. Note that you can also get to the administration console from Visual Studio 2005 by clicking the ASP.NET Configuration button in the Solution Explorer (see Figure 1-7).

![Figure 1-7. ASP.NET configuration via the Solution Explorer in Visual Studio 2005](image)

A typical deployment of a Web application looks like Figure 1-8.

![Figure 1-8. A typical deployment of a Web application](image)
Web applications are deployed simply by copying files to the Web server. Visual Studio 2005 has a built-in Web deployment tool that helps with this (see Figure 1-9). You can access this tool via Web Site ➤ Copy Web Site menu item.

![Image of Web site deployment tool in Visual Studio 2005]

**Figure 1-9. Web site deployment tool in Visual Studio 2005**

With the deployment tool in Visual Studio 2005, you can deploy your Web applications to a Web server, to a File Transfer Protocol (FTP) site, or to a folder somewhere.

**Web Services**

Web services in .NET have an .asmx file extension. The .asmx file provides the means for clients to call Web services over Hypertext Transfer Protocol (HTTP). The actual Web service implementation is embedded within an assembly. On a Windows platform, Web services are typically hosted under Internet Information Services (IIS). Therefore, from a deployment perspective, deploying Web services is no different than deploying Web applications. A Web service, in fact, can reside by itself under its own Web application, or it can reside under a Web application that has the usual ASPX and HTML files. Because Web services are packaged as part of a Web application, Web services have a web.config file that they use for storing and retrieving application configuration.

---

7. This is commonly called *xcopy* deployment.
8. You can also host Web services outside of IIS. For example, you can host Web services outside of IIS using Web Services Enhancements (WSE).
Smart Device Applications

Smart device applications run on smart devices. Visual Studio 2005 has project templates that target three types of smart devices: Pocket PC 2003, Smartphone 2003, and Windows CE 5.0. The Pocket PC and Smartphone projects target these specific devices; the Windows CE project type does not target any specific device. (In other words, it does not reference any device-specific functionality.) Figure 1-10 shows the New Project dialog box for smart device applications. The dialog box allows you to create graphical applications, console applications, and support assemblies that all target smart devices.

![New Project dialog box](image)

**Figure 1-10. New smart device application in Visual Studio 2005**

Smart device applications are built on top of the .NET Compact Framework (.NET CF). The .NET CF is a subset of the .NET Framework. This means the .NET CF doesn't have all the functionality that is available in the .NET Framework.

You can build Web applications, console applications, Windows Forms applications, and so on, that target devices that are not smart devices (that is, desktops). Similarly, you can build a Web application or a Windows Forms application that targets smart devices. Therefore, deploying Web applications that target smart devices is no different because Web pages are still deployed to an actual server and rendered to the smart device. Note that you cannot use a smart device as a Web server, however. With Windows Forms and console applications, you have to install the applications on the smart device. This turns out to be different from what you do for desktop applications. That is, you use ClickOnce to deploy Windows Forms applications to desktops. You don't, however, have this luxury to deploy to smart devices. Instead, you have to package these applications using CAB files.
Windows Services

Windows services are executables that run in the background and have no user interface. You can create a Windows Service using Visual Studio 2005 by selecting Windows Service in the New Project dialog box (see Figure 1-11). Windows services are supported on Windows NT, Windows 2000, Windows XP, Windows Server 2003, and future versions of the operating system. These versions of Windows are multiuser systems. This means multiple users can be logged on to the system simultaneously. In addition, generally there is no one logged on to the main console (the server itself). Therefore, having a UI for these services is a bit useless.

**Note** You can manage and configure Windows services through a console called Service Control Manager (SCM).\(^9\)


Having said that, you can still have a service that has a UI. For almost all cases, Windows services don't have UIs, but Windows still allows you to have one if you need one. Services that have a UI need a special flag enabled. You can set the special flag by right-clicking the service from the Services list in the Microsoft Management Console (MMC) and choosing Properties ➤ Log On tab (see Figure 1-12). Then check the Allow Service to Interact with Desktop box.

**Figure 1-11.** New Windows service project and service configuration
CHAPTER 1  ▫ DEPLOYMENT PREREQUISITES

10. This feature is likely to be dropped in future versions of Windows.


12. You can view the services on Windows XP by right-clicking My Computer and then selecting Manage. From there, choose Services under Services and Applications.

Figure 1-12. Configuring a Windows service to interact with the desktop

Windows services comprise an executable, zero or more dependent assemblies, resources, and a configuration file. The executable contains service-level event methods (for example, OnStart), which are fired when a service is started, stopped, and so on, through the SCM (see Figure 1-13).

Figure 1-13. Start-up configuration of a Windows service

Copyright © 2003 Microsoft Corporation. All rights reserved.