CHAPTER 1

Introducing the XNA Framework and XNA Game Studio Express

IN THIS CHAPTER

- What Is the XNA Framework?
- Installing Visual C# Express
- Installing the DirectX Runtime
- Installing XNA Game Studio Express
- Creating Spacewar Windows Project
- Compiling and Running Spacewar

Most developers I know decided to enter the computer field and specifically programming because of computer games. Game development can be one of the most challenging disciplines of software engineering—it can also be the most rewarding!

Never before has it been possible for the masses to create games for a game console, much less a next generation game console. We are coming in on the ground floor of a technology that is going to experience tremendous growth. Microsoft is leading the way into how content will be created for game consoles. Soon other game console manufacturers will be jumping at a way to allow the public to create content for their machines. The great news for the Xbox 360 is that Microsoft has spent so much time over the years creating productive and stable development environments for programmers. We will be installing one of Microsoft's latest integrated development environments (IDEs) in this chapter. Before we get to that, let's take a look at the technology we discuss in this book—XNA.

What Is the XNA Framework?

You have probably heard the statement, "To know where you are going, you need to know where you have been." I am uncertain if that is entirely true, but I do believe it applies here. Before we dig into exactly what XNA is and what it can do for us, let's take a moment to look at DirectX because that is what the XNA Framework is built on.

The Foundation of the XNA Framework

Let's take a journey back to the days of DOS on the PC. When programming games, graphic demos, and the like in DOS, programmers typically had to write low-level code to talk directly to the sound card, graphics cards, and input devices. This was tedious and the resulting code was error prone because different manufacturers would handle different BIOS interrupts, IO ports, and memory banks—well, differently, so the code would work on one system and not another.

Later, Microsoft released the Windows 95 operating system. Many game programmers were skeptical at writing games for Windows—and rightly so—because there was no way to get down to hardware level to do things that required a lot of speed. Windows 95 had a protected memory model that kept developers from directly accessing the low-level interrupts of the hardware.

To solve this problem, Microsoft created a technology called DirectX. It was actually called Windows Game SDK to begin with, but quickly switched names after a reporter poked fun at the API names DirectDraw, DirectSound, and DirectPlay, calling the SDK Direct "X." Microsoft ran with the name and DirectX 1.0 was born a few months after Windows 95 was released. I remember working with DirectDraw for a couple of demos back when this technology first came out.

Because of DirectX, developers had a way to write games with one source that would work on all PCs regardless of their hardware. Hardware vendors were eager to work with Microsoft on standardizing an interface to access their hardware. They created device drivers to which DirectX would map its API, so all of the work that previously had to be done by game programmers was taken care of, and programmers could then spend their time doing what they wanted to—write games! Vendors called this a Hardware Abstraction Layer (HAL). They also developed a Hardware Emulation Layer (HEL), which emulates hardware through software in case hardware isn't present. Of course, this is slower but it allowed certain games to be run on machines with no special hardware.

After a couple of years Microsoft released DirectX 3.0, which ran on Windows NT 4 as well as Windows 95. As part of those upgrades, they introduced Direct3D. This allowed developers to create 3D objects inside of 3D worlds. DirectX 4 was never released, but DirectX 5 was released in 1997 and later had some upgrades to work under Windows 98.

When DirectX 8 came on the scene in 2000, some of the newly available graphics hardware had vertex and pixel shaders. As a result, Microsoft added in a way to pass custom program code to the hardware. Through assembly code, the game developer could manipulate the data the main game passed to the graphics card. This assembly code was consumed directly by the graphics hardware.

When there was no graphics hardware, games were slow, but they were very flexible. Later, as hardware rendering became prominent, the games were faster, but they were not very flexible in that all of the games really started to look the same. Now with shaders, the speed of the hardware is combined with the flexibility for each game to render and light its 3D content differently.

This brings us to present-day DirectX: We are up to DirectX 9 and 10. Before I talk about DirectX 9, I spend some time talking about DirectX 10. DirectX 10 was released at the same time as Microsoft Windows Vista. In fact, DirectX 10 only works on Vista. This is largely due to the fact that Microsoft has made major changes in the driver model for this operating system. DirectX 10 also requires Shader Model 4.0 hardware.

The Xbox 360 runs on DirectX 9 plus some additional partial support for Shader Model 3.0 functionality. DirectX 9 is the foundation for Managed DirectX, an API that exposed the core DirectX functionality to .NET Framework developers. There was a lot of concern about whether this "wrapper" could be as fast as the C++ counterparts. Fortunately, it was almost as fast—about 98 percent was the benchmark touted. I experienced these benchmark speeds firsthand while on the beta team for this technology. I fell in love with Managed DirectX.

The XNA Framework used the lessons learned from Managed DirectX and used that foundation as a launching pad. To be clear, XNA was built from the ground up and was not built on top of Managed DirectX. It didn't use the same namespaces as Managed DirectX and is not simply pointing to the Managed DirectX methods in the background. Although XNA utilizes DirectX 9 in the background, there are no references to DirectX's API like there were in Managed DirectX.

XNA Today

XNA is actually a generic term much like the term .NET. XNA really refers to anything that Microsoft produces that relates to game developers. The XNA Framework is the API we are discussing. The final piece to XNA is the XNA Game Studio Express application, which we discuss in detail later. This is the IDE we use to develop our XNA games.

TIP

In this book, whenever we use the term XNA, we are really referring to the XNA Framework unless otherwise noted.

XNA allows us to do a lot of things. We have easy access to the input devices (keyboard, game pad or controller, mouse). XNA gives us easy access to the graphics hardware. We are able to easily control audio through XNA. XNA provides the ability for us to store information like high scores and even saved games. XNA does not currently have any networking capability. Microsoft wants to use the Xbox Live technology for adding network support to XNA. However, there is more work to be done to make sure Microsoft can provide multiplayer functionality in a secure manner.

To get started using XNA we have to install some software. We need to install the latest version of DirectX 9 as well as have a graphics card that supports DirectX 9.0c and Shader Model 1.1. (You should get a card that supports Shader Model 2.0 as some of the examples, including the starter kit we use in this chapter and the next one, will not run without it.) We also need to install Visual C# Express, the DirectX 9 runtime, and finally XNA Game Studio Express. Fortunately, all of the software is free! If you don't have

graphics hardware that can support Shader Model 2.0 you can pick up a card relatively inexpensively for about $35 USD. If possible, you should purchase a graphics card that can support Shader Model 3.0, as a couple of examples at the end of the book require it.

At this point, games cannot be created for commercial use on the Xbox 360 but Microsoft has mentioned they are interested in supporting commercial games in future versions. Fortunately, we can create community games for the Xbox 360 with the Express versions.

XNA Game Studio Express is great for the game hobbyist, a student, or someone just getting started because you do not have to shell out a lot of (any!) money to get up and running. One exception to this is if you actually want to deploy your games on your Xbox 360. To do that, you will need to subscribe to the XNA Creators Club for $99 USD a year (or $49 USD for four months). Remember, writing games for the PC using XNA is totally free!

Oh, in case you are wondering what XNA stands for, **X**NA's **N**ot **A**cronymed (or so Microsoft says in the XNA FAQ).

Installing Visual C# Express

To get started, we must have the software installed. Let's start by installing Visual C# Express. Visual C# Express is the IDE that is required to run XNA Game Studio Express. XNA requires C# due to how the Content Pipeline is used. There are some people who have successfully created demos using other languages such as VB.NET and even F#. However, this is not supported by Microsoft currently and won't be discussed in this book. This book assumes you have a good understanding of C#. If you know C++, Java, or VB.NET, you should be able to pick up C# pretty quickly.

I am going to be detailed in the steps to make sure that anyone who has not worked with Visual C# Express will be able to get it installed with no issues. Feel free to skip this section if you already have Visual C# Express installed.

To install Visual C# Express, follow these steps:

1. You will need to be connected to the Internet to install the application. The application can be downloaded by browsing to http://msdn.microsoft.com/vstudio/express/downloads/ and clicking the Visual C# Express Go button to download the vcssetup.exe setup program.

2. Optional. On the Welcome to Setup screen select the check box to send data about your setup experience to Microsoft. This way if something goes awry, Microsoft can get the data and try to make the experience better the next time around. This screen is shown in Figure 1.1.

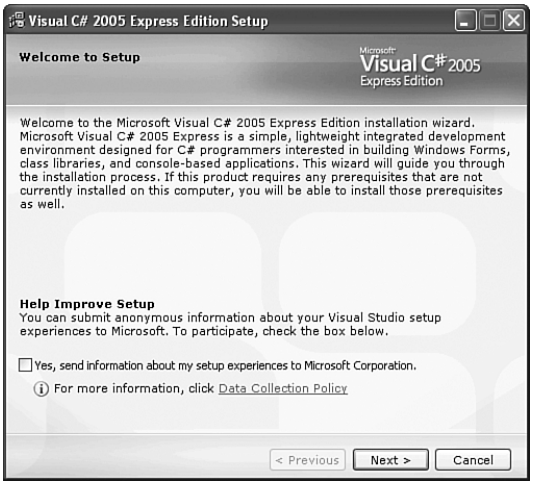

FIGURE 1.1 Select the check box if you want the system to provide feedback to Microsoft about your installation experience.

3. Click Next to continue.
4. The next screen is the End-User License Agreement. If you accept the terms, select the check box and click Next.
5. The following screen, shown in Figure 1.2, has two installation options you can check. Neither of these options is required to utilize XNA.

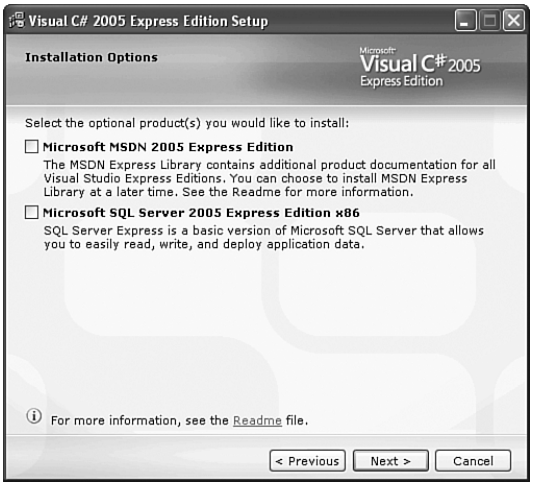

FIGURE 1.2 Neither of these options is required to utilize XNA.

6. Click Next to continue.
7. The next screen, shown in Figure 1.3, asks where we would like to install Visual C# Express. It is going to install other required applications including Microsoft .NET Framework 2.0. This is required, as C# runs on the .NET Framework. You will also notice it requires more than 300MB of space.
8. Click Next to continue.

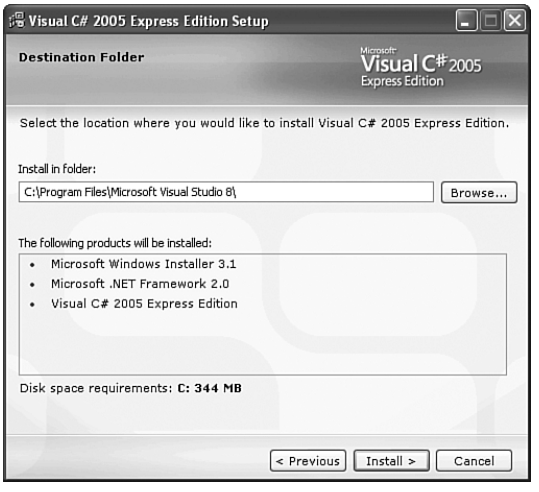

FIGURE 1.3 Specify which directory you want Visual C# Express to be installed in.

9. Now we are looking at the Installation Progress screen where we will be able to monitor the progress of the installation.
10. Finally, on the Setup Complete screen we can see the Windows Update link we can click on to get any of the latest service packs for Visual C# Express.
11. Click Exit to complete the installation.

We have successfully installed the first piece of the pie to start creating excellent games with XNA! Before we continue to the next piece of software, we need to open up Visual C# Express. It might take a couple of minutes to launch the first time the application is loaded. Once the Visual C# Express is loaded we should see the Start Page as shown in Figure 1.4.

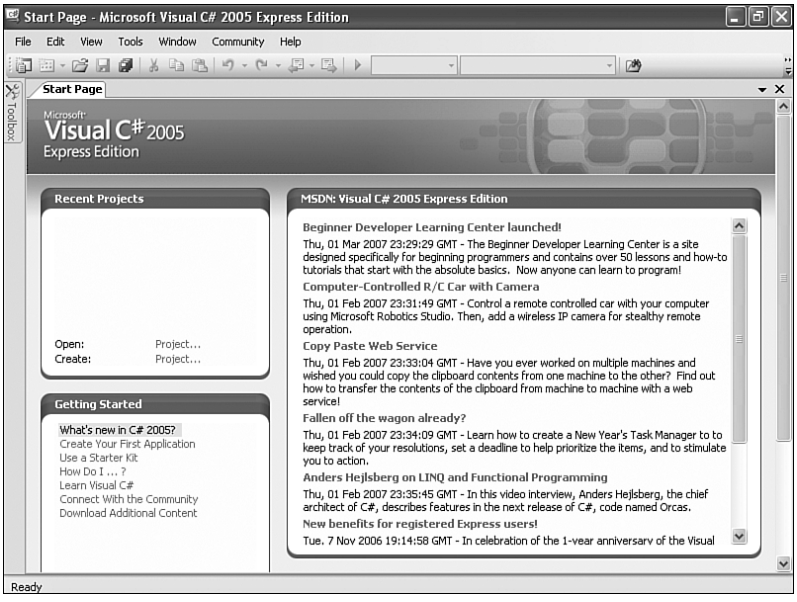

FIGURE 1.4 This is the Start Page inside of Visual C# Express.

The following procedure is optional, but it does ensure that everything is working correctly on our machine.

1. In the Recent Projects section, find Create Project and click the link. You can also create a new project under the File menu.
2. Visual C# Express installed several default templates that we can choose from. Select the Windows Application template as displayed in Figure 1.5.
3. You can leave the name set to WindowsApplication1 as we will just be discarding this project when we are done.

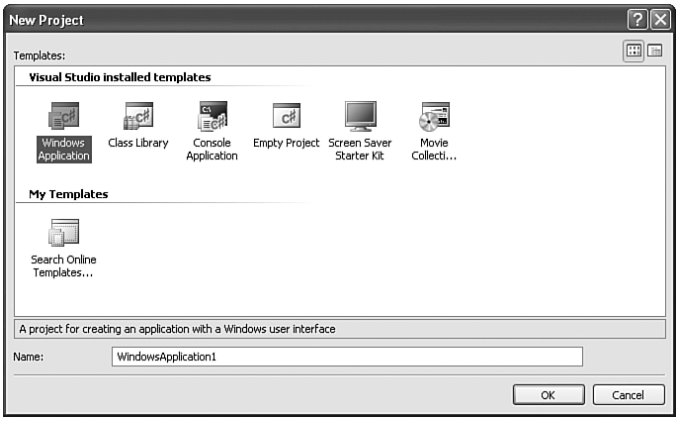

FIGURE 1.5 The New Project dialog box allows you to choose from the default templates to create an application.

4. Click OK to create the application.

At this point a new project should have been created and we should be looking at a blank Windows Form called Form1.

5. Press Ctrl+F5 or click Start Without Debugging on the Debug menu.

If everything compiled correctly, the form we just saw in design mode should actually be running. Granted, it doesn't do anything, but it does prove that we can compile and run C# through Visual C# Express. The end result can be seen in Figure 1.6. Let's close down the application we just created as well as Visual C# Express. Feel free to discard the application.

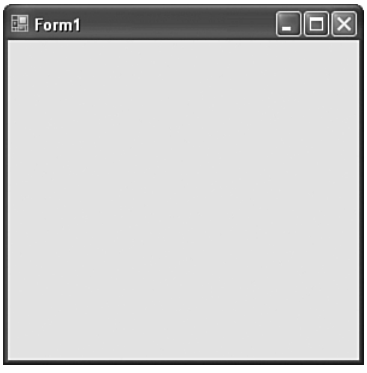

FIGURE 1.6 This is a C# Windows Form application after compiling and running the default template.

Installing the DirectX Runtime

We also need the DirectX 9 runtime if it isn't already on the machine. To get started, follow these steps:

1. Run the dxwebsetup.exe file from Microsoft's website. This can be found by clicking on the DirectX Runtime Web Installer link at the bottom of the Creator's Club Resources—Essentials web page http://creators.xna.com/Resources/Essentials.aspx. This file contains the redistribution package of the February 2007 DirectX 9. You will need to be connected to the Internet so it can completely install the application.
2. We are greeted with the End-User License Agreement. Handle with care.
3. The next screen is a dialog box asking where we would like the installation files to be stored. We can pick any directory we want as long as we remember it so we can actually install the runtime—we are simply extracting the files needed to install the runtime.
4. Click OK to continue.

5. We will be prompted to create that directory if the directory entered doesn't exist. Click Yes to continue.
6. Wait for the dialog box with the progress bar to finish unpacking the files.

Now we can actually install the runtime by following these steps:

1. Browse to the folder where we installed the files and run the dxsetup.exe file to actually install DirectX 9 onto the machine.
2. The welcome screen we see includes the End-User License Agreement. Select the appropriate radio button to continue.
3. Following the agreement is a screen stating that it will install DirectX—click Next.
4. Once it finishes installing (a progress bar will be visible while it is installing the files) we will be presented with the Installation Complete screen.
5. Simply click Finish to exit the setup.

Now, we can move on to installing XNA Game Studio Express.

Installing XNA Game Studio Express

To use XNA Game Studio Express we must use Visual C# Express. We cannot use Visual Studio .NET Professional nor can we use any other IDE. Although there are people who have successfully been able to run XNA and even get the Content Pipeline (which we talk about in Part III of the book) to work in Visual Studio .NET Professional, it is not officially supported by Microsoft and is not covered in this book.

WARNING

You must run the Visual C# Express IDE at least one time before installing XNA Game Studio Express. If this is not done, not all of the functionality will be installed. If XNA Game Studio Express was installed prematurely, you will need to uninstall XNA Game Studio Express and run Visual C# Express and then exit the IDE. Then you will be able to reinstall XNA Game Studio Express.

To get started complete the following steps:

1. Run the xnagse_setup.msi file from Microsoft's website. The file can be downloaded by clicking on the top link of the Creator's Club Resources—Essentials web site http://creators.xna.com/Resources/Essentials.aspx.
2. Click Next to get past the setup welcome screen.
3. The next screen is the End-User License Agreement. If you accept the terms, select the check box and click Next.
4. This will open up a notification dialog box that explains that the Windows Firewall will have a rule added to it to allow communication between the computer and the Xbox 360. This can be seen in Figure 1.7.

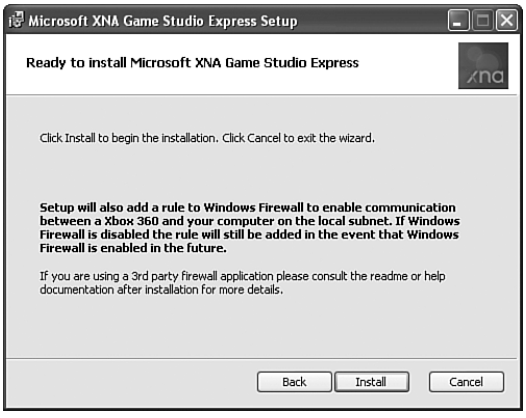

FIGURE 1.7 XNA Game Studio Express modifies the Windows Firewall so an Xbox 360 and the PC can talk to each other.

5. Click Install to continue. The next screen shows the progress of the installation.
6. Once it has completed installing all of the required files we will be presented with the completion dialog box. Simply click Finish to exit the setup.

After we have installed XNA Game Studio Express, we can go to the Start menu and see it added a few more items than those contained in the IDE. Make sure to take time and read through some of the XNA Game Studio Express documentation. There is also a Tools folder that contains a couple of tools we will be looking at later. We will be discussing the XACT tool in Chapter 6, "Loading and Texturing 3D Objects," and the XNA Framework Remote Performance Monitor for Xbox 360 application in Chapter 3, "Performance Considerations." Go ahead and open the IDE by clicking XNA Game Studio Express on the Start menu.

Hmm, this looks identical to the Visual C# Express IDE. There is a good reason for this—it is the same application! When we installed XNA Game Studio Express it added properties to Visual C# Express to allow it to behave differently under certain circumstances. Mainly it added some templates, which we will look at shortly, and it added the ability for Visual C# Express to handle content via the XNA Content Pipeline. It also added a way for us to send data to our Xbox 360, as we will see in the next chapter.

TIP

If you ever uninstall XNA Game Studio Express, you will need to uninstall Visual C# Express first. Otherwise, XNA Game Studio Express will not properly uninstall.

Creating Spacewar Windows Project

With XNA Game Studio Express opened, create a new project and we should see a screen similar to Figure 1.8. Select the Spacewar Windows Starter Kit template and feel free to change the name of the project. Click OK to create the project.

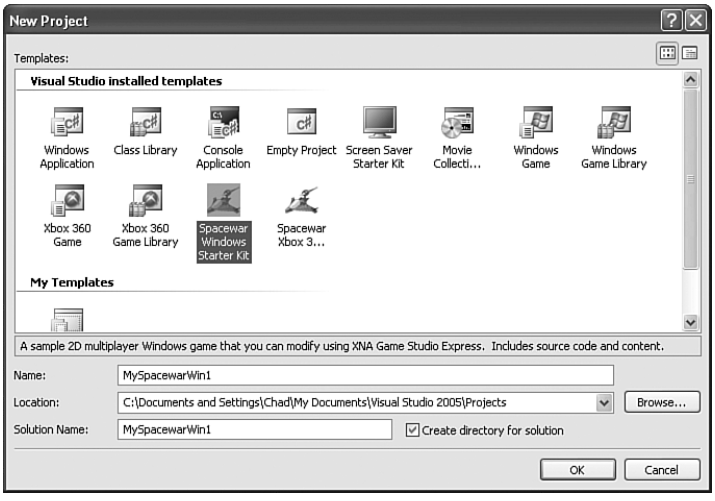

FIGURE 1.8 We can see that installing XNA Game Studio Express added six more templates to Visual C# Express.

Compiling and Running Spacewar

At this point we have our software installed and have even created a starter template that was created by Microsoft that we can take for a spin. We need to make sure we can compile the code. To just compile without running we can press Ctrl+Shift+B or press F6 or click Build Solution on the Build menu. It should have compiled without any issues. You can now press Ctrl+F5 to actually run the game. Have some fun playing the game. Feel free to look around the code and tweak it. Fortunately, you can always re-create the template if something gets really messed up!

Summary

In this chapter we laid the groundwork by getting the all of the software required installed to actually create games on our PC. We even compiled a game and played it. After getting a game session fix, join me in the next chapter where we will get this project up and running on the Xbox 360!