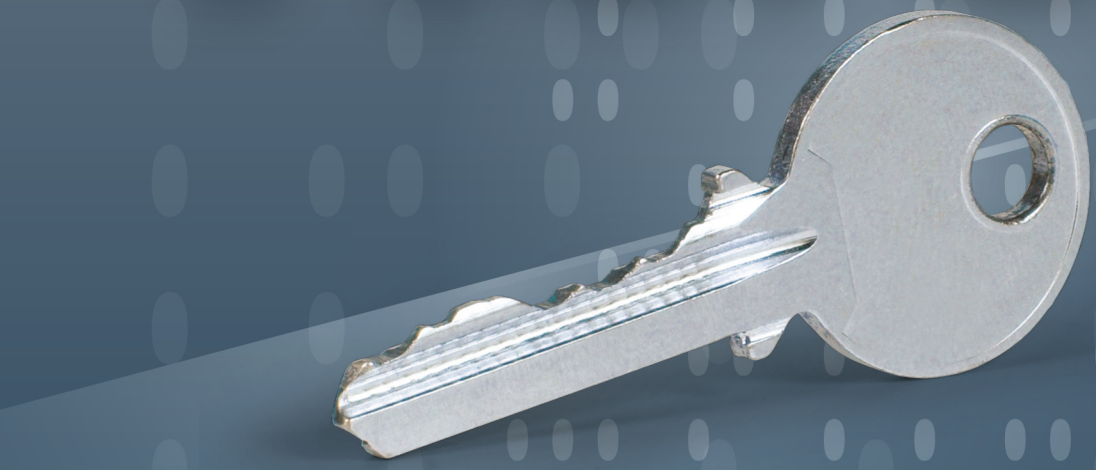
Microsoft

Start Here!™



Learn Microsoft® Visual Basic® 2012

Michael Halvorson

Microsoft

Start Here!™

Learn Microsoft®

Visual Basic® 2012

Michael Halvorson

ISBN: 978-0-7356-7298-7

2 3 4 5 6 7 8 9 10 LSI 8 7 6 5 4 3

Printed and bound in the United States of America.

Microsoft Press books are available through booksellers and distributors worldwide. If you need support related to this book, email Microsoft Press Book Support at *mspinput@microsoft.com*. Please tell us what you think of this book at http://www.microsoft.com/learning/booksurvey.

Acquisitions and Developmental Editor: Russell Jones

Production Editor: Holly Bauer

Editorial Production: Zyg Group, LLC

Technical Reviewer: Tim Patrick

Copyeditor: Zyg Group, LLC

Indexer: Zyg Group, LLC

Cover Design: Jake Rae

Cover Composition: Zyg Group, LLC

Illustrator: Rebecca Demarest

[2013-03-12]

For my brother, Jon Halvorson

Contents at a Glance

Contents

Introduction

Microsoft Windows 8 is a powerful and visually compelling operating system designed to dramatically enhance consumer productivity and offer access to a wide range of web-based products and services. A rich user experience is at the heart of Windows 8, where the new look and feel of Windows 8 applications provide rapid access to music, photos, contacts, and user settings in the Internet "cloud", and the Windows Store provides immediate access to exciting consumer applications. Windows 8 has been designed to operate on a broad spectrum of devices, from touch-enabled tablets, to laptops, to traditional desktop computers. As customers immerse themselves in hundreds of vibrant Windows 8 applications, they are given the freedom to focus on the task at hand, rather than the commands or features of the operating system.

From the perspective of the software developer, Windows 8 presents amazing opportunities; it's fast, secure, and robust, and will be installed on millions of computers worldwide, including the Microsoft Surface tablets. Windows 8 applications are exciting and easy to use, and they offer customers an interface that is content-rich and runs equally well on touch-based devices or desktop PCs. Most significantly, the Windows Store allows developers to sell their Windows 8 applications directly to the global marketplace, providing new sources of revenue and streamlining installation procedures.

This book will show you how to create compelling Windows 8 applications with Microsoft Visual Studio 2012, the newest version of Microsoft's bestselling software development suite. You will learn how to download a free version of the Visual Studio 2012 Express software (that's right—*free*!), and how to create interesting Windows 8 apps by using several of the tools and technologies within Visual Studio, including the Visual Basic programming language. By the end of this book you will have learned how to create the core features of a Windows 8 application; how to work productively in the Visual Studio Integrated Development Environment (IDE); how to design a user interface with XAML markup and Blend for Visual Studio; how to write efficient Visual Basic program code; and how to sell your own applications in the Windows Store.

One of the coolest features of this book, of course, is that every programming tool that it teaches and describes is *free*! Microsoft is offering complementary access to the Express edition of Visual Studio because it hopes that you will enjoy learning how to program with it, and that you will one day become a professional Visual Basic programmer who will build and sell great Windows applications. All you need is Windows 8 installed on a compatible computer with an Internet connection, and the desire to write Visual Basic programs.

In fact, the outlook for professional Visual Basic programmers has never been brighter. You just need to *Start Here*!

Who Should Read This Book

This is a hands-on programming tutorial for readers who enjoy learning to do new things by actually doing them. *Start Here! Learn Microsoft Visual Basic 2012* assumes no prior knowledge of Visual Studio or Visual Basic, and it focuses entirely on introductory programming concepts and procedures. You will be surprised at how much you can accomplish as a beginning programmer with Visual Studio, and you will be building your own projects in no time. I assume only that you are an intelligent student, hobbyist, or IT professional who is interested in learning how to program, and that you have no prior experience with Visual Basic or the Visual Studio software suite.

This book's content will provide you with concrete Visual Basic coding techniques as well as a broad overview of programming strategies. In addition, you will learn about the capabilities of the Windows 8 operating system, and the specific design guidelines that Microsoft recommends for Windows 8 applications, an exciting new way of creating software. The Windows 8 user interface design principles are sleek and empowering, and they encourage developers to put information-rich, web-aware applications at the center of the computing experience. Windows 8 applications present new ways of collaborating with others, as well as exciting opportunities for working with new input devices, such as built-in cameras, touchpads, accelerometers, gyros, compasses, GPS controls, and ambient light sensors.

The overall goal of *Start Here! Learn Microsoft Visual Basic 2012* is to get you to the point where you can comfortably use the development tools in Visual Studio, create your own basic Windows 8 applications, and then be ready to follow a more comprehensive Visual Basic programming book, such as my own *Microsoft Visual Basic 2012 Step by Step* (Microsoft Press, 2013).

Assumptions

This book is designed to teach readers with no programming experience how to use the Visual Basic programming language. As part of that process, readers will also learn how to use the Visual Studio 2012 Express software, which they can download for free. Chapter 1, "Getting to Know Visual Basic 2012," shows you how to download and install Visual Studio 2012 Express on your system.

The book assumes that you have purchased and are running the Windows 8 operating system, and that you want to learn how to create applications for Windows 8. These applications are simply programs that run under Windows 8, follow basic guidelines about how the user interface works, and are (or should be) designed to take advantage of the numerous resources and connections available on the web. Windows 8 applications are deeply interactive, and are designed to be downloaded by customers from the Windows Store.

To make the most of your programming practice, you will need to know a little about how to perform common tasks in Windows 8, how to work with information on the web, how to customize the Start page and user interface, and how to adjust basic system settings. If you also have Windows 8 installed on a tablet or touchpad device, all the better, because a fundamental design emphasis of Windows 8 is to make touch and gestures a natural way to manipulate content. You can build your applications on a laptop or desktop running Visual Studio 2012 and Windows 8, and then test them on your tablet or touchpad.

If you happen to be using one of the full retail versions of Visual Studio 2012, you will be able to create a wider range of application types than I describe in this book—Visual Studio Express 2012 for Windows 8 software restricts the application types you can create to just Windows 8–style applications. A more advanced book such as *Microsoft Visual Basic 2012 Step by Step* will show you how to create HTML applications for the Web, how to create console applications, how to develop software specifically for Windows Phone, and how to create desktop applications (Windows Forms projects) for Windows 8 and Windows 7.

Who Should Not Read This Book

You're going to be disappointed with this book if you're an advanced programmer and interested in learning Visual Basic as a second language. The examples in this book are relatively basic, and the explanations are kept simple. You may also be disappointed if you already have significant Visual Basic programming experience, and just want to know the new features of Visual Studio 2012. However, if you have not programmed before, or if it has been some time since you wrote programs, you will probably appreciate the thorough introduction to Visual Studio 2012 and the coverage of the fundamentals of writing Windows 8 programs with Visual Basic, tasks that involve a number of tools and methods that may be unfamiliar.

Developers who have a lot of experience will feel that I'm exploring the obvious—but what is obvious to experienced programmers often isn't obvious at all to someone who is just learning to write code. If programming is a new concept for you, this is the place to start.

Organization of This Book

Start Here! Learn Microsoft Visual Basic 2012 uses a hands-on approach to learning, in which readers actually build Windows 8 applications from scratch, one step at a time. Each chapter introduces a new tool or technique, and the book has been designed to be read sequentially, so that what you learn in one chapter is carried forward to the next. Although the core of this book involves teaching Visual Basic coding techniques, you will also learn how to use the interesting tools and features in the Visual Studio IDE, including the Toolbox, the Code Editor, XAML controls, Solution Explorer, and the debugger. You will also learn how to use Blend for Visual Studio 2012, a separate design application distributed with Visual Studio.

Collectively, the twelve chapters in this book offer you a complete introductory programming course that you can complete at your own pace. You might try to finish one or two chapters a day for a few days, and then take some time off to practice building applications on your own before moving on. Reading about new techniques, trying out what you have learned, and then pushing a bit further on your own is the best way to acquire many new skills, including how to program.

This book offers the following topics:

- **Chapter 1: Getting to Know Visual Basic 2012** What types of applications can Visual Basic programmers actually create, and how should they go about doing it? This introductory chapter answers these fundamental questions, and then introduces the Visual Studio IDE, an electronic workshop where Visual Basic applications are built from the ground up. You'll learn how to download the Visual Studio Express 2012 for Windows 8 software, how to start it, and how to get going with the Visual Studio programming tools.
- **Chapter 2: Creating Your First Windows 8 Application** In this chapter you learn how to build your first Windows 8 application, a web browser that allows you to explore web sites and record the locations that you have visited. You'll learn more about the programming tools in Visual Studio, and you'll learn what it means to test an application and prepare it for distribution to others.
- **Chapter 3: Using Controls** The controls that you use to receive input, display output, and help the user navigate your application represent a fundamental element of the user interface. In this chapter, you'll learn how to create several useful XAML controls, including *Ellipse*, *TextBlock*, *CheckBox*, *RadioButton*, and *MediaElement*.
- **Chapter 4: Designing Windows 8 Applications with Blend for Visual Studio** Your Visual Studio 2012 Express software installation includes a separate program called Blend for Visual Studio, which provides easy-to-use design tools for creating the user interface of a Windows 8 application. You'll use Blend in this chapter to construct a user interface that displays digital photographs and uses storyboards and animation effects. You'll also learn how to switch from Blend to Visual Studio, where you can write Visual Basic program code.
- **Chapter 5: Working with XAML** Windows 8 applications use the XAML markup language to define how the user interface appears on the screen, and how it presents information to the user. This chapter explores in detail the structure of XAML markup, and explains how you can customize a program's look and feel by working with XAML markup in the Visual Studio Code Editor.
- **Chapter 6: Visual Basic Language Elements** Visual Basic is an advanced programming language that allows you to control how a Windows application operates. When you create a Windows 8 application, you use Visual Basic code to define how the application manages all types of information, such as input received from the user and the results of mathematical calculations. In this chapter, you will learn the syntax and format of Visual Basic program statements, how to use variables to store information, how to use fundamental data types and constants, and how to work with formulas and operators in a program.

- **Chapter 7: Controlling Application Design, Layout, and Program Flow** Windows 8 applications should feature compelling content and pages prepared for rich user interaction. This chapter digs deeper into Windows 8 design principles by focusing on tile-based layout and user input with the *Image* and *ListBox* controls. To help you control execution and program flow, you'll learn how to write effective decision structures, loops, and exception handlers in your applications.
- **Chapter 8: Using the .NET Framework** As you write more sophisticated programs, you'll need to manipulate graphics, display text files, perform calculations, process strings, and retrieve information from the web. These capabilities and much more are supplied to you via the .NET Framework, an underlying programming interface that is part of the Windows operating system. This chapter explains how to learn more about .NET Framework classes using the Visual Studio Object Browser, how to use Framework methods to process strings and calculate formulas, and how to save development time by inserting ready-made code snippets into your project.
- **Chapter 9: Debugging Applications** The complex nature of Window programming means that you'll run into syntax errors and other logic problems from time to time as you build your applications. This chapter introduces the programming tools in the Visual Studio IDE that help you locate and correct programming mistakes, and how to anticipate operating errors that your users may encounter in the future.
- **Chapter 10: Managing Data with Arrays and LINQ** Because there is so much data in the world—employee records at the office, price and product information online, confidential patient records at the clinic—it makes sense that software developers are spending a lot of time thinking about how data is managed in their programs. In Visual Studio, an important technology used for accessing and managing data is known as Language Integrated Query (LINQ), and you will learn the basics of using LINQ in this chapter. You'll learn how to store information in temporary locations called arrays, how to write LINQ query expressions to retrieve data from arrays, and how to use the data in XML documents as a source for LINQ queries.

- **Chapter 11: Design Focus: Five Great Features for a Windows 8 Application** This chapter returns to the user interface of Windows 8 applications, and offers additional instruction about how programs can be designed so that they comply with Microsoft's design guidelines for Windows 8 applications. You'll learn how to create a tile for your app on the Windows Start page, how to create a splash screen for your project, how to control application permissions and capabilities, how to use ready-made project templates, and how to add support for touch input and gestures.
- **Chapter 12: Future Development Opportunities and the Windows Store** This last chapter provides a summary of the Visual Basic programming techniques that you have learned, and presents future development opportunities for those interested in careers in Visual Studio programming. The chapter also presents a detailed look at the final testing and packaging of applications, including a discussion of the Windows Store, an exciting new distribution point for Windows 8 applications. Also included in this chapter are web resources and books that you can use to continue your learning.

Free eBook Reference

When you purchase this title, you also get the companion reference, *Start Here!™ Fundamentals of Microsoft® .NET Programming*, for free. To obtain your copy, please see the instruction page at the back of this book.

The *Fundamentals* book contains information that applies to any programming language, plus some specific material for beginning .NET developers.

As you read through this book, you'll find references to the *Fundamentals* book that look like this:

For more information, see <*topic*> in the accompanying *Start Here! Fundamentals of Microsoft .NET Programming* book.

When you see a reference like this, if you're not already familiar with the topic, you should read that section in the *Fundamentals* book. In addition, the *Fundamentals* book contains an extensive glossary of key programming terms.

Conventions and Features in This Book

This book presents information using conventions designed to make the information readable and easy to follow:

- Step-by-step instructions help you create Visual Basic applications. Each set of instructions is listed in a separate section and describes precisely what you'll accomplish by following the steps that it contains.
- Screen illustrations show you exactly what is happening as you complete the step-by-step instructions. I have used the default colors and settings for Windows 8 to create these illustrations, and configured my screen resolution at a low setting to make the illustrations as readable as possible.
- Boxed elements with labels such as "Note" provide additional information or alternative methods for completing a step successfully. Make sure that you pay special attention to warnings because they contain helpful information for avoiding problems and errors.
- Text that you type (apart from code blocks) appears in **bold**.
- A plus sign (+) between two key names means that you must press those keys at the same time. For example, "Press Alt+Tab" means that you hold down the Alt key while you press the Tab key.
- A vertical bar between two or more menu items (such as File | Close), means that you should select the first menu or menu item, then the next, and so on.

System Requirements

You will need the following hardware and software to work through the examples in this book:

- The Windows 8 operating system. Depending on your Windows configuration, you might require Local Administrator rights to install or configure Visual Studio 2012 Express.
- An Internet connection to download Visual Studio, try out the Windows Store, and download this book's sample files.
- Visual Studio 2012 Express for Windows 8 (see Chapter 1 for installation instructions).
- A computer with 1.6 GHz or faster processor.

- 1 GB RAM (32-bit) or 2 GB RAM (64-bit).
- 16 GB available hard disk space (32-bit) or 20 GB (64-bit) for Windows 8.
- 4 GB of available hard disk space for Visual Studio 2012 Express.
- DirectX 9 graphics device with WDDM 1.0 or higher driver.
- 1024 × 768 minimum screen resolution.
- If you want to use touch for user input, you'll need a multitouch-capable laptop, tablet, or display. Windows 8 supports at least five simultaneous touch points, although not all tablets or displays do. A multitouch-capable device is optional for the exercises in this book, although one is useful if you want to understand what such devices are capable of as a software developer. Typically a programmer will develop software on a desktop or laptop computer, and then test multitouch functionality on a multitouch-capable device.

Code Samples

Most of the chapters in this book include exercises that let you interactively try out new material learned in the main text. All sample projects, in both their pre-exercise and post-exercise formats, can be downloaded from the following page:

http://www.microsoftpressstore.com/title/9780735672987

Follow the instructions to download the *9780735672987-files.zip* file.

Installing the Code Samples

Follow these steps to install the code samples on your computer so that you can use them with the exercises in this book:

1. Unzip the *9780735672987-files.zip* file that you downloaded from the book's website. (Name a specific directory along with directions to create it, if necessary.)
2. If prompted, review the displayed end-user license agreement. If you accept the terms, select the accept option, and then click Next.

Using the Code Samples

The code samples .zip file for this book creates a folder named "Start Here! Programming in Visual Basic" that contains 11 subfolders—one for each of the chapters in the book (except the last chapter). To find the examples associated with a particular chapter, open the appropriate chapter folder. You'll find the examples for that chapter in separate subfolders. The subfolder names have the same names as the examples in the book. For example, you'll find an example called "Web List" in the *My Documents**Start Here! Programming in Visual Basic**Chapter 02* folder on your hard drive. If your system is configured to display file extensions of the Visual Basic project files, look for *.sln* as the file extension. Depending on how your system is configured, you may see a "Documents" folder rather than a "My Documents" folder.

Acknowledgments

The planning for this book began well before the release of Windows 8 and the first Visual Studio 2012 test releases. In early conversations with Microsoft Press and O'Reilly Media, we all realized that Windows 8 and Visual Studio 2012 presented truly revolutionary opportunities for Visual Basic programmers. The question was: how could we prepare the right learning materials for new and existing software developers so that they could get up-to-speed quickly and begin exciting Windows applications as soon as possible?

The solution we came up with was to create two original books with information about the Visual Studio software release—the book that you are holding now, my *Start Here! Learn Microsoft Visual Basic 2012*, and a second book designed for more experienced developers, my *Microsoft Visual Basic 2012 Step by Step*. These two books work together to provide a comprehensive course on Windows 8 programming with Visual Basic 2012.

Although I have written over a dozen books on Visual Basic programming in my career as a writer and software developer, this experience was one of the most rewarding and exciting, as two back-to-back book projects required significant coordination among publishing team members at both Microsoft Press and O'Reilly Media. I hope that you enjoy the results and are able to use the books to explore deeply these amazing new products. Very quickly, you'll be learning to program in Visual Basic 2012, and preparing applications for distribution in the Windows Store.

At Microsoft Press, I would like to thank Devon Musgrave for his early enthusiasm for the books, and for connecting me to team members in the Visual Studio product group. At O'Reilly Media, I would like to thank first and foremost Russell Jones, for our many conversations about Visual Basic programming, and our hope that these books will provide a complete path for new and experienced programmers to unlock the secrets of Visual Basic 2012. Tim Patrick, a talented author and Visual Basic developer in his own right, provided a thorough review of the *Start Here!* manuscript, and answered many practical questions about Visual Studio for me. Within the editorial group, I would like to thank Holly Bauer, for scheduling the editorial review and answering questions about content; and Damon Larson, for his skillful copy editing and managing the style issues that arose. Also within O'Reilly Media, I would like to thank Kristin Borg and Rebecca Demarest, and at Zyg Group, I'd like to thank Linda Weidemann, Kim Burton-Weisman, and Kevin Broccoli for their important editorial, technical, and artistic contributions.

I am also most grateful to the Microsoft Visual Studio 2012 development team for providing me with the beta and release candidate software to work with. In addition, I would like to thank the Microsoft Windows 8 team for their support, and offer my special thanks to the many MSDN forum contributors who asked and answered questions about Visual Basic and Windows 8 programming.

As always, I offer my deepest gratitude and affection to my family for their continued support of my writing projects and various academic pursuits. In particular, Henry Halvorson created impressive electronic music, electronic artwork, and a video file for Chapters 3, 7, and 11. I am so thankful for your efforts, son.

Errata & Book Support

We've made every effort to ensure the accuracy of this book and its companion content. Any errors that have been reported since this book was published are listed on our Microsoft Press site:

http://www.microsoftpressstore.com/title/9780735672987.

If you find an error that is not already listed, you can report it to us through the same page. If you need additional support, email Microsoft Press Book Support at *mspinput@microsoft.com*.

Please note that product support for Microsoft software is not offered through the addresses above.

We Want to Hear from You

At Microsoft Press, your satisfaction is our top priority, and your feedback our most valuable asset. Please tell us what you think of this book at:

http://www.microsoft.com/learning/booksurvey

The survey is short, and we read every one of your comments and ideas. Thanks in advance for your input!

Stay in Touch

Let's keep the conversation going! We're on Twitter: *http://twitter.com/MicrosoftPress*.

You can also learn more about Michael Halvorson's books and ideas at *http://michaelhalvorsonbooks.com*.

Chapter 1

Getting to Know Visual Basic 2012

After completing this chapter, you'll be able to

- Describe various development opportunities for Visual Basic programmers.
- Download and install Visual Studio Express 2012 for Windows 8.
- Start Visual Studio Express and explore the Visual Studio IDE.
- Open and run a Visual Basic program.
- Use Visual Studio programming tools and windows.
- Customize the Visual Studio IDE.
- Save changes and exit Visual Studio.

THIS CHAPTER INTRODUCES YOU to Microsoft Visual Basic programming and gives you the skills you need to get up and running with the Microsoft Visual Studio Express 2012 for Windows 8 Integrated Development Environment (IDE). The Visual Studio IDE is the application you use to build and run Visual Basic programs. The Visual Studio IDE is a busy place with numerous menu options, buttons, tool windows, code editors, and output windows. However, you'll discover that any general experience you may have had with Windows applications will help you a lot as you learn how to use the IDE, and you'll find that some of the tools and features are more important than others. The important thing to remember, faced with the IDE's extensive capabilities, is that you don't need to learn everything at once.

This chapter also provides an overview of the types of programs, called *applications*, that you can create with Visual Basic 2012. This was once a rather straightforward subject, because choosing to write programs in Visual Basic meant you could create great Windows desktop applications but not much more. As you'll see, however, Visual Studio now allows Visual Basic programmers to create applications in a variety of formats for many different uses. Although this book focuses on creating Windows 8 apps, it will be helpful for you to learn just how capable Visual Studio is, especially in one of the full retail versions.

Development Opportunities for Visual Basic Programmers

Visual Basic is an object-oriented computer programming language that has roots in earlier development tools such as BASIC and QuickBASIC—that is, logical and practical (though somewhat quirky) programming languages from the 1960s, '70s, and '80s.

In 1991, Microsoft released Visual Basic 1.0, which innovatively combined a sophisticated Visual Basic language compiler with an IDE that allowed developers to build Windows applications by visually arranging controls on a Windows form and then customizing the controls with property settings and Visual Basic code. Over the past two decades, Visual Basic has grown into an extremely powerful development tool, capable of creating fast and efficient Windows applications that can run on a variety of hardware platforms.

The term *Visual Basic* has come to have two meanings over the past 10 years or so. In the narrower sense, *Visual Basic* is the name of a programming language with specific syntax rules and logical procedures that must be followed when a developer creates code to control some aspect of an application's functionality. However, *Visual Basic* is also used in a more comprehensive product-related sense to describe the collection of tools and techniques that developers use to a build Windows applications with a particular software suite. In the past, developers could purchase a stand-alone version of Visual Basic, such as Microsoft Visual Basic .NET 2003 Professional Edition, but these days Visual Basic is sold only as a component within the Visual Studio software suite, which also includes Microsoft Visual C#, Microsoft Visual C++, Microsoft Visual Web Developer, and other development tools.

More Info For more information about object-oriented computing, see Chapter 10, "Object-Oriented Programming," in the free companion volume, *Start Here! Fundamentals of .NET Programming* (Microsoft Press, 2011).

The Visual Studio 2012 development suite is distributed in several different product configurations, including Test Professional, Ultimate, Premium, Professional, Express for Windows 8, and Express for Web. Express for Windows 8 and Express for Web are currently the free editions that you can use to test-drive the software. (Express for Windows 8 is the product that you will be using in this book.) The full retail versions of Visual Studio 2012 have different prices and feature sets, with Ultimate being the most comprehensive (and expensive) development package. The Visual Studio web site (*http://www.microsoft.com/visualstudio*) explains the differences between all these versions.

You have purchased this book because you want to learn how to program in Visual Basic. This is an excellent choice; there are over 3 million Visual Basic programmers in the world developing innovative solutions, blogging on the web, and shopping for add-ons and training materials.

The programming language that you decide to learn is a matter of choice, often related to your past experiences and the requirements of the companies that you work for. Because different organizations have spent considerable time and capital building up their code bases, you'll find that they have specific language and software requirements for the teams that they employ. You may have encountered such requirements listed in hiring advertisements for programmers. Often they require

that developers know more than one programming language, in addition to specific skills related to database or web development.

Microsoft has tried to satisfy a wide range of programming audiences by bundling many different software development technologies into Visual Studio, including Visual Basic, Visual C#, Visual Web Developer, and JavaScript. Visual Studio also contains some core tools that all developers use, no matter which programming language they choose. These include the various toolbox controls, the Project and Properties windows, the code editors, the debugger, the Blend Designer, various management tools, and the .NET Framework—a library of coded solutions designed to be used by applications that run on the Windows operating system.

New Development Platforms

So, what can you actually *do* with Visual Basic and Visual Studio?

In the early 2000s, Visual Basic programmers were concerned primarily with creating Windows applications that helped businesses manage data effectively. Visual Basic's ability to graphically display information and provide access to it with powerful user interface controls gained many supporters for the product. Over the past decade, the leading Visual Basic applications have been database front-ends, inventory management systems, web applications and utilities, purchasing tools, CAD programs, scientific applications, and games.

In the 2010s, however, the explosion of Internet connectivity and online commerce has dramatically changed the landscape for software developers. In the past, most Windows applications ran on a server or a desktop PC. Today, laptops, tablet devices, and smartphones are everywhere, and often the same person owns all three device types. Consumers need to move applications and information seamlessly across devices, and software developers need the tools that will allow them to create applications that work on multiple platforms, or can at least be ported easily from one device to the next.

The Visual Studio 2012 product team took the challenge of coding for diverse platforms seriously, and they have created a software suite that allows developers to leverage their existing work while also allowing developers to create a variety of different application types. The following list highlights the major development platforms and opportunities for Visual Basic programmers (some of which are supported only by the full retail versions of Visual Studio 2012):

- **Windows 8** Visual Basic developers can create Windows 8 applications and traditional desktop applications for a wide range of Windows 8 devices, including Microsoft Surface tablets, and sell them on the Windows Store.
- **Windows 7 and earlier** Visual Basic developers can create applications for earlier versions of Windows and distribute them in a variety of ways. The Visual Basic and Extensible Application Markup Language (XAML) programming techniques you learn in this book will be closest to writing Visual Basic and XAML programs for Windows Presentation Foundation (WPF).

- **Windows Phone** Using Visual Studio and the Windows Phone software development kit (SDK), Visual Basic programmers can create applications that run on Windows Phone and take advantage of its unique features.
- **Web development** Developers can use Visual Basic, HTML5, CSS3, or JavaScript to create applications that will run on the web and look great in a variety of browsers. A technology known as ASP.NET allows Visual Basic programmers to build web sites, web applications, and web services quickly without knowing all the details about how the information will be stored on the web.
- **Device drivers and console applications** Visual Basic programmers can write applications that work primarily with the internal components of the operating system or run in command-line mode (the MS-DOS shell).
- **Office applications** Visual Basic programmers can build macros and other tools that enhance the functionality of Microsoft Office applications, such as Excel, Word, Access, and PowerPoint.
- **Xbox 360** Visual Basic programmers can write games for the Xbox using Visual Studio and Microsoft XNA Game Studio.
- **Windows Azure applications for web servers and the cloud** Visual Basic is powerful enough to write applications that will be used on sophisticated web servers, distributed data centers, and a version of Windows designed for cloud computing known as Windows Azure.

This is an amazing list of application types! Although this list might seem daunting at first, the good news is that the fundamental Visual Basic programming skills that you will explore here remain the same from platform to platform, and there are numerous tools and techniques that help you to port work easily between them. This book provides a solid introduction to many of the core skills that you will use; you can then learn specific programming techniques related to any particular platform when you are ready.

Obtaining, Installing, and Starting Visual Studio Express 2012 for Windows 8

Before you can begin programming in Visual Basic, you need to install the Visual Studio software. If you already have Windows 8 and one of the retail versions of Visual Studio 2012, you are all set already; the teaching in this book will apply to Windows 8 and the Visual Studio software that you have. If you don't already have a version of Visual Studio, you can download a free copy of Visual Studio Express 2012 for Windows 8 directly from Microsoft. After you install that product, you'll be able to use the examples in this book to create your own Windows 8 apps.

Note This book requires that you are running Windows 8 and that you have a version of Visual Studio 2012 installed on your system. Although you can download Visual Studio Express 2012 for Windows 8 for free, you will also need a valid, installed version of Windows 8 to create the applications.

Downloading the Product

Microsoft produces a number of Express products that you can download from *http://www.microsoft.com/express/Downloads/*, but for the purposes of this book you need to download only this one:

- **Visual Studio Express 2012 for Windows 8** This product provides the Visual Studio IDE and tools that allow you to create Visual Basic, Visual C#, Visual C++, or JavaScript applications for Windows 8. You also have to have Windows 8 installed on your computer—Windows 8 does not come with Visual Studio Express 2012.

You must have an Internet connection to install the product. The setup files for Visual Studio Express 2012 can either be installed either directly from the web or downloaded to your hard drive and then opened and installed later.

Installing Visual Studio Express 2012 for Windows 8

To download Visual Studio Express 2012 for Windows 8, complete the following steps:

1. Open a web browser (Internet Explorer or other) and go to the following web site: *http://www.microsoft.com/express/Downloads*.
2. Click Visual Studio 2012 to see a list of the Express products available for Visual Studio 2012.

Note You must have Window 8 installed on your computer *before* you install Visual Studio Express 2012 for Windows 8.

3. Click Express for Windows 8, and follow the instructions to download and install Visual Studio Express 2012 for Windows 8.

   Specify a web installation or download the product files first and then install them. You will also have an opportunity to specify the language that you will be using when using Visual Studio. (For this book, the recommended language is English.) When the Express installation is complete, you're ready to start working with Visual Studio!

Starting Visual Studio Express 2012

To start Visual Studio Express and begin working with the Visual Studio IDE, complete the following steps.

Start Visual Studio Express 2012

1. On the Windows Start screen, click VS Express for Windows 8.

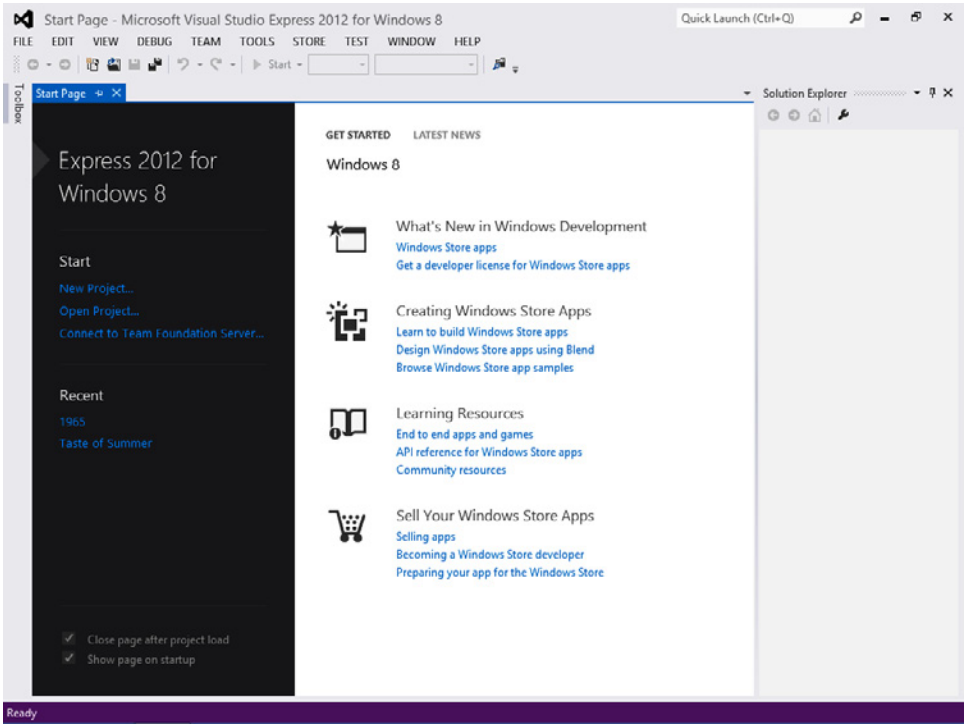
   If this is the first time you are starting Visual Studio, the program will take a few moments to configure the environment. You may be prompted to get a developer license for Windows 8, which typically requires that you create a Windows Live account or enter existing account information. During the beta testing for Visual Studio 2012, developer licenses were free and valid for about a month before they needed to be renewed. You will likely encounter a similar registration scenario.

2. If you are prompted to identify your programming preferences, select Visual Basic Development Settings.

   When Visual Studio starts, you see the IDE on the screen with its many menus, tools, and component windows. (These windows are sometimes called tool windows.) You also should see a Start page containing a set of tabs with links, learning resources, news, and project options. The Start page is a comprehensive source of information about your project, as well as resources within the Visual Basic development community.

   The screen shown following offers a typical Visual Studio setup. I captured the screen at a resolution of 1024x768, which may be smaller than you are using on your computer, but I wanted you to see the content as clearly as possible. (Larger resolutions are often great to work with on screens that support them, but they don't reproduce well in books.)

   I have also chosen to use the Light color theme for the screen illustrations in this book. When you first open Visual Studio, however, you may see the Dark color theme, which displays white text on a dark background. Although the Dark color theme is restful and emphasizes the code and user interface elements of your program, it doesn't appear well in books. If you see the Dark color theme now, change to the Light theme by choosing the Options command on the Tools menu, clicking General in the Environment category, selecting Light from the Color Theme drop-down list box, and clicking OK. The following screen illustration shows the Light theme:



After starting Visual Studio, you're ready to explore the Visual Studio IDE.

Note The following section describes how to open and run a Visual Basic program in the IDE. If you haven't downloaded this book's sample files yet, you should do so now, because you'll be asked to open a specific program on your hard disk. (For sample code installation instructions, see the "Code Samples" section in the Introduction.) Return to this point after you have installed the code samples.

The Visual Studio Development Environment

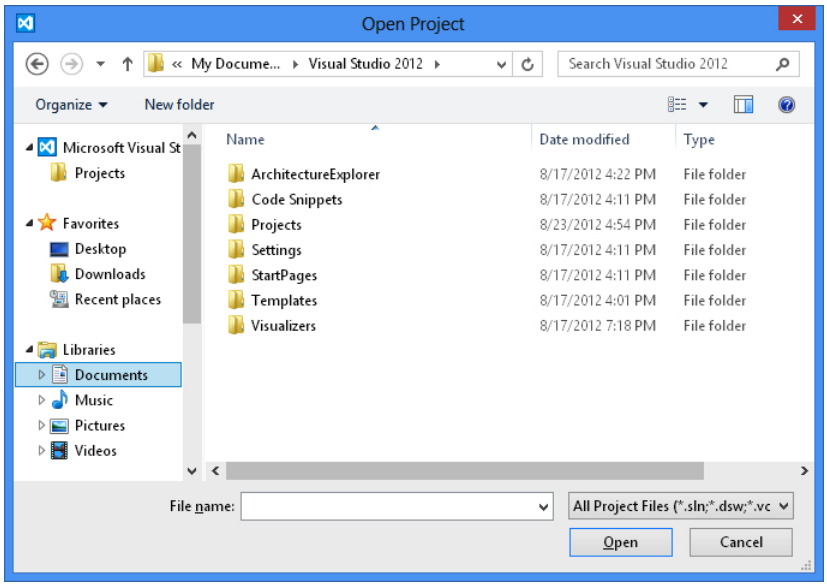
In the Visual Studio IDE, you can open a new or existing Visual Studio project, or you can explore the many online resources available to you for Visual Basic programming.

Right now, let's open an existing Visual Studio project that I created for you, entitled World Capitals, which displays the capital of Peru in a text box.

Open a Visual Basic project

1. On the Start page, on the left side of the screen, click the Open Project link.

   You'll see the Open Project dialog box shown in the following illustration. (You can also display this dialog box by clicking the Open Project command on the File menu or by pressing Ctrl+Shift+O.) Even if you haven't used Visual Studio before, the Open Project dialog box will seem straightforward because it resembles the familiar Open dialog box in many other Windows applications.



Tip In the Open Project dialog box, you see a number of storage locations along the left side of the window. The Projects folder under Microsoft Visual Studio Express 2012 for Windows 8 is particularly useful. By default, Visual Studio saves your programming projects in this Projects folder, giving each project its own subfolder. However, this book uses a different folder to organize your programming coursework, as you'll discover following. Additional locations such as Favorites and Libraries are also made available to you through this dialog box, depending on how your computer and operating system has been configured.

2. Browse to the *My Documents\Start Here! Programming in VB 2012* folder on your hard disk.

   This folder is the default location for the book's extensive sample file collection, and you'll find the files there if you followed the instructions in "Code Samples" in the Introduction. If you didn't copy the sample files, close this dialog box and copy them now.

3. Open the *Chapter 01\World Capitals* folder, and then double-click the *WorldCapitals* solution file. (If your system shows file name extensions, this file will end with *.sln*.)

   Visual Studio loads the *WorldCapitals* page, properties, and program code for the solution. Solution Explorer, a tool window on the right side of the screen, lists some of the files in the solution.

Visual Studio provides a special option named Always Show Solution to control several options related to solutions within the IDE. The option's check box is located on the Projects And Solutions | General tab of the Options dialog box, which you open by clicking the Options command on the Tools menu. If the check box is selected (the default position), a subfolder is created for each new solution, placing the project and its files in a separate folder beneath the solution. Also, if you keep the default selection for Always Show Solution, a few options related to solutions appear in the IDE, such as commands on the File menu and a solution entry in Solution Explorer. If you like the ideas of creating separate folders for solutions and seeing solution-related commands and settings, I suggest that you keep the default (selected) option for this check box. You'll learn more about these options at the end of the chapter.

Projects and Solutions

In Visual Studio, programs under development are typically called projects or solutions because they contain many individual components, not just one file. Visual Basic 2012 programs include a project file (*.vbproj*), a solution file (*.sln*), one or more markup files (*.xaml*), and several supporting files organized into various subfolders.

A *project* contains files and other information specific to a single programming undertaking. A *solution* contains all the information for one or more projects. Solutions are therefore useful mechanisms to manage multiple related projects. The samples included with this book typically have a single project for each solution, so opening the project file (*.vbproj*) has the same effect as opening the solution file (*.sln*). But for a multiproject solution, you will want to open the solution file.

The Visual Studio Tools

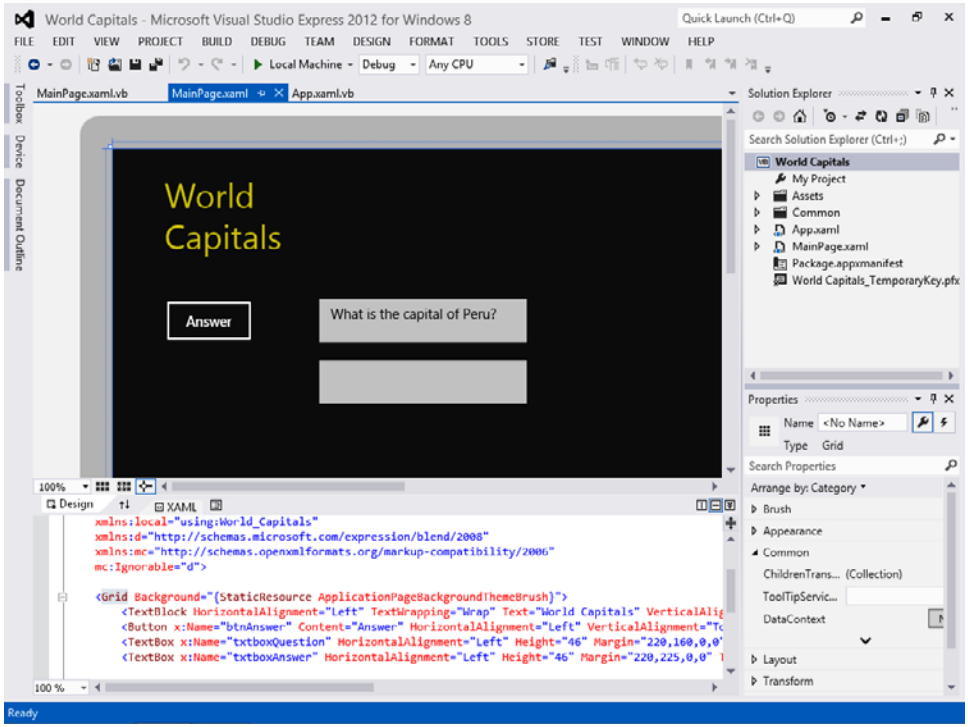
At this point, you should take a few moments to study the Visual Studio IDE and identify some of the programming tools and windows that you'll be using as you complete this book. If you've written Visual Basic programs before, you'll recognize many (but perhaps not all) of the programming tools. Collectively, these features are the components that you use to construct, organize, and test your Visual Basic programs. A few of the programming tools also help you learn more about the resources on your system, including the larger world of databases and web site connections available to you. There are also several powerful help tools.

The menu bar provides access to most of the commands that control the development environment. Menus and commands work as they do in all Windows-based programs, and you can access them by using the keyboard or the mouse. Located below the menu bar is the Standard toolbar, a collection of buttons that serve as shortcuts for executing commands and controlling the Visual Studio IDE. My assumption is that you've used Word, Excel, or some other Windows application enough to know quite a bit about toolbars, and how to use familiar toolbar commands, such as Open, Save, Cut, and Paste. But you'll probably be impressed with the number and range of toolbars provided by Visual Studio for programming tasks. In this book, you'll learn to use several toolbars; you can see the full list of toolbars at any time by right-clicking any toolbar in the IDE.

Along the bottom of the screen you may see the Windows taskbar. You can use the taskbar to switch between various Visual Studio components and to activate other Windows-based programs. You might also see taskbar icons for Windows Internet Explorer, antivirus utilities, and other programs installed on your system. In most of my screen shots, I'll hide the taskbar to show more of the IDE.

The following illustration shows some of the tools and windows in the Visual Studio IDE. Don't worry that this illustration looks different from your current development environment view. You'll learn more about these elements (and how you adjust your views) as you work through the chapter.

The main tools visible in this Visual Studio IDE are the Designer, Solution Explorer, the Properties window, and the XAML tab of the Code Editor. You should locate these tools and remember their names now, as you'll be using them often. You might also see more specialized tools such as the Toolbox, Document Outline window, Device window, Server Explorer, and Object Browser; alternatively, these tools may appear as tabs within the IDE. Because no two developers' preferences are exactly alike, it is difficult to predict what you'll see if your Visual Studio software has already been used. (What I show is essentially the fresh-download (or out-of-the-box) view, with the Designer displaying the World Capitals user interface contained in *MainPage.xaml*.)



If a tool isn't visible and you want to see it, click the View menu and then select the tool. Because the View menu has expanded steadily over the years, Microsoft has moved some of the less frequently used View tools to a submenu called Other Windows. Check there if you don't see what you need.

The reason I said your IDE view probably doesn't match the preceding image is because the exact size and shape of the tools and windows in the IDE depend on how your particular development environment has been configured. With Visual Studio, you can align and attach, or *dock*, windows to make visible only the elements that you want see. You can also partially conceal tools as tabbed documents along the edge of the development environment and then switch back and forth between documents quickly. For example, if you click the Toolbox label on the left side of the screen, the Toolbox panel will fly out, ready for use. If you click another tool or window in the IDE, the Toolbox panel will return to its concealed position.

Trying to sort out which tools are important to you now and which you can learn about later is a difficult early challenge when you're learning the busy Visual Studio interface. Your development environment will probably look best if you set your monitor and Windows desktop settings so that they maximize your screen space, but even then things can get a little crowded. (In fact, some experienced Visual Studio programmers use two monitors to display different views of the software.)

The purpose of all this tool complexity is to add many new and useful features to the IDE while providing clever mechanisms for managing the clutter. These mechanisms include features such as docking, autohiding, floating, and a few other window states that I'll describe later. Visual Studio 2012 also hides rarely used IDE features until you begin to use them, which has also helped to clean up the IDE workspace.

If you're just starting out with Visual Studio, the best way to deal with feature overload is to hide the tools that you don't plan to use often to make room for the important ones. The crucial tools for beginning Visual Basic programming—the ones you'll start using right away in this book—are the Designer window, the Properties window, Solution Explorer, and the Toolbox. You won't use the Document Outline, Server Explorer, Class View, Object Browser, Device, or Debug windows until later in the book, so feel free to hide them by clicking the Close button on the title bar of any windows that you don't want to see.

In the following exercises, you'll start experimenting with the crucial tools in the Visual Studio IDE. You'll also learn how to display a web browser within Visual Studio and more about hiding the tools that you won't use for a while.

The Designer Window

If you completed the previous exercise ("Open a Visual Basic project"), the World Capitals project is loaded in the Visual Studio development environment. However, the user interface, or page, for the project might not yet be visible in Visual Studio. (More sophisticated projects might contain several pages, but this first example program needs only one.) To make the page of the World Capitals project visible in the IDE, you display it by using Solution Explorer.

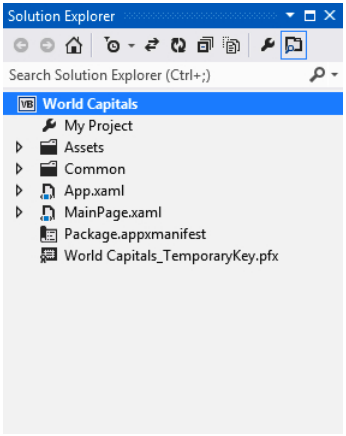
Note If you don't currently have the World Capitals project loaded, go back to and complete the exercise in this chapter titled "Open a Visual Basic project."

Display the Designer window

1. Locate the Solution Explorer window near the upper-right corner of the Visual Studio development environment. If you don't see Solution Explorer (if it is hidden as a tab in a location that you cannot see or isn't currently visible), click the View menu and then select Solution Explorer to display it.

Note From here on in this book, you'll sometimes see a shorter method for describing menu choices. For example, "Choose View | Solution Explorer" means "Click the View menu and then select Solution Explorer."

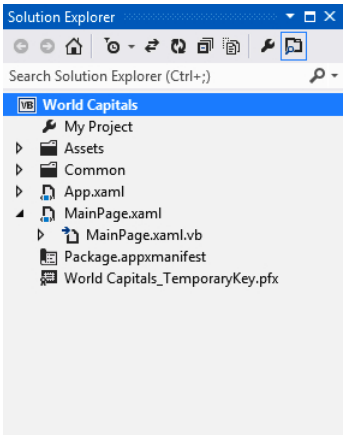
When the World Capitals project is loaded, Solution Explorer looks like this:



Like most basic Windows 8 applications, this Visual Basic solution contains an *App.xaml* file that holds global project settings and resources; an *Assets* folder that contains any splash screen and logo files for the project; a *Common* folder, which contains common classes and XAML styles that simplify your development tasks; a *deployment package manifest*, containing build and distribution settings for your file; and one or more user interface windows, or *pages*, which you can identify because they have the extension *.xaml*.

2. Click the expansion arrow to the left of the *MainPage.xaml* file in the Solution Explorer window.

With the *MainPage.xaml* file expanded, Solution Explorer looks like this:



In this project, the main page of the World Capitals program is defined by the *MainPage.xaml* file. (*MainPage.xaml* is the default name for the main page when you create a new application without a specific template.)

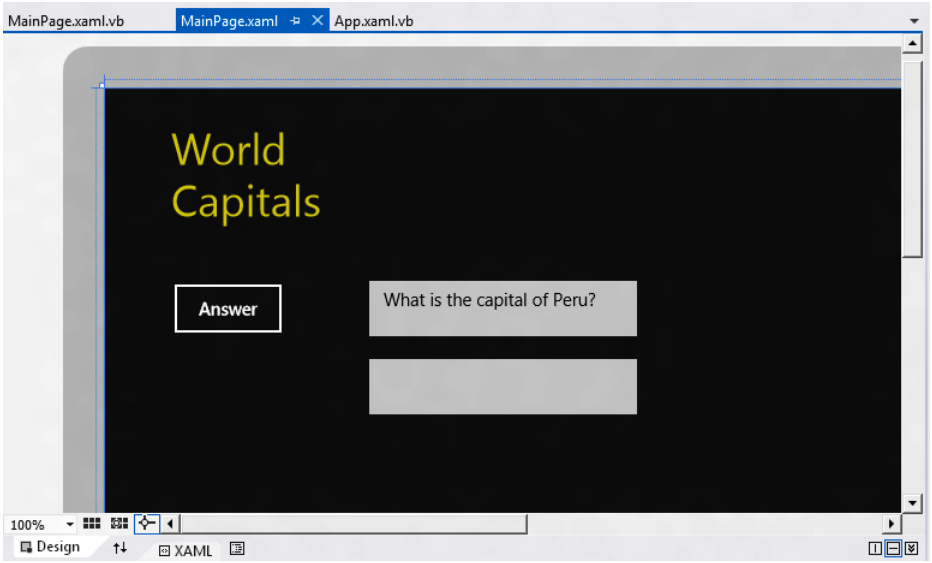
You can open *MainPage.xaml* in Design view so that you can examine and modify the user interface with graphical design tools, or you can open the file in the Code Editor, where you can modify the user interface with XAML, a special user interface definition language designed for Windows applications and other computer programs.

Below the *MainPage.xaml* file, you will see a second file, named *MainPage.xaml.vb*. This file is also associated with the user interface of the World Capitals project. *MainPage.xaml.vb* is called a *code-behind file* because it contains a listing of the Visual Basic program code connected to the user interface defined by *MainPage.xaml*. As you learn how to program in Visual Basic, you'll become very adept at customizing this file.

Solution Explorer is the gateway to working with the various files in your project—it is an essential tool. When you double-click a file in Solution Explorer, it opens the file in an appropriate editor, if direct editing of the file is allowed.

3. Double-click the *MainPage.xaml* file in Solution Explorer to display the project's user interface in the Designer window, if it is not already visible. Use the vertical scroll bar if necessary to adjust your view of the user interface.

   The World Capitals page is displayed in the Designer, as shown here:



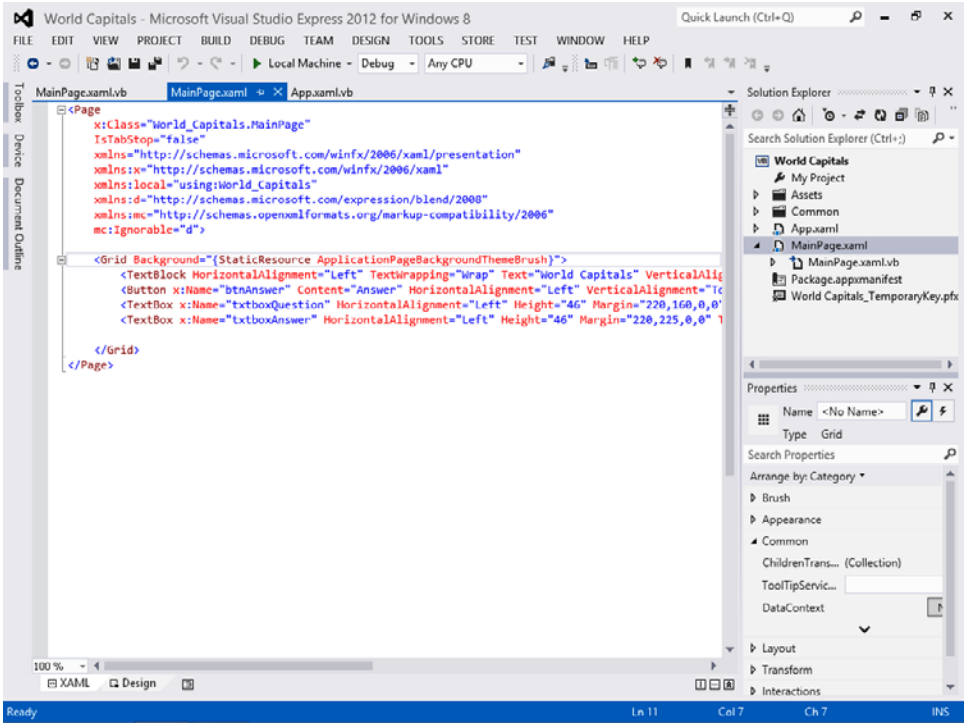
   Notice that a tab with the file name *MainPage.xaml* is visible near the top of the Designer window, along with additional tab names. You can click a tab at any time to display the contents of the various files, even if the windows become covered.

   As noted previously, the *MainPage.xaml* file is the visual representation of the program's user interface. However, you can readily examine the XAML markup used to define the user interface by double-clicking the XAML tab of the Code Editor at the bottom of the Designer window. Or, if the XAML tab is already open in the Code Editor, you can examine the XAML

markup for the user interface and use the window's scroll bars to view any part of the markup that is not currently visible.

4. Double-click the XAML tab to display the XAML markup for the page in the Code Editor, and scroll to the top of the window to see the entire document.

   You'll see the following:



The XAML contents of *MainPage.xaml* appear in the Code Editor, and it is this structured information that controls how Visual Studio and Windows will display the application's user interface and graphics. If you know some HTML, this should look somewhat familiar. XAML contains *markup*—instructions whose primary purpose is to tell a program how to display things on the screen. The XAML markup shown here is displayed between *<Page* and *</Page>* tags, and is further indented to make the information readable. The first seven lines below *<Page* define the resources used to create the user interface. Below these lines, the section between *<Grid* and *</Grid>* defines the objects in the user interface. This XAML content defines one button, two text boxes, and one text block. If you look at the screen illustration of the Designer again, you can see how these elements appear visually. You can even see specific property settings for the objects being assigned through individual property names (like *HorizontalAlignment*) and values (like *Left*).

You'll learn a lot more about XAML markup in later chapters. For now, you should know that the Designer window allows you to see both a *preview* of the user interface and the XAML markup that defines the specific characteristics of objects that appear on the preview page.

Visual Studio programmers often want to see both panes of information side by side as they work on a program. In fact, if you've built an HTML application in the past for the web, this whole concept might seem a little familiar, as a number of web design tools also display page layout at the top of the screen, while showing HTML code at the bottom.

Tip There are some handy buttons along the bottom of the Designer window and Code Editor that allow you significant control over the split-screen behavior of these elements. At the bottom left of the Designer window are XAML and Design tabs, as well as a handy Document Outline button, which opens a separate window to display the objects within the user interface organized by type. At the bottom right of the Designer window are Vertical Split, Horizontal Split, and Expand Pane/Collapse Pane buttons, which control how the Designer window and Code Editor are arranged. Expand Pane/Collapse Pane is especially useful; it is a toggle that allows you to view the windows one at a time or side by side.

5. Click the Design tab to display the project's main page in the Designer window again.
6. Click the Expand Pane button to display the XAML markup that renders the page in a window below the Designer window.

Now you'll try running this simple program within Visual Studio.

Running a Visual Basic Program

World Capitals is a simple Visual Basic program designed to familiarize you with the programming tools in Visual Studio. The page you see now has been customized with four objects (one button, a text block, and two text boxes), and I've added one line of program code to a code-behind file to make the program ask a simple question and display the appropriate answer. You'll learn more about creating objects like these and adding Visual Basic code to a code-behind file in Chapter 2, "Creating Your First Windows 8 Application." For now, try running the program in the Visual Studio IDE.

Run the World Capitals program

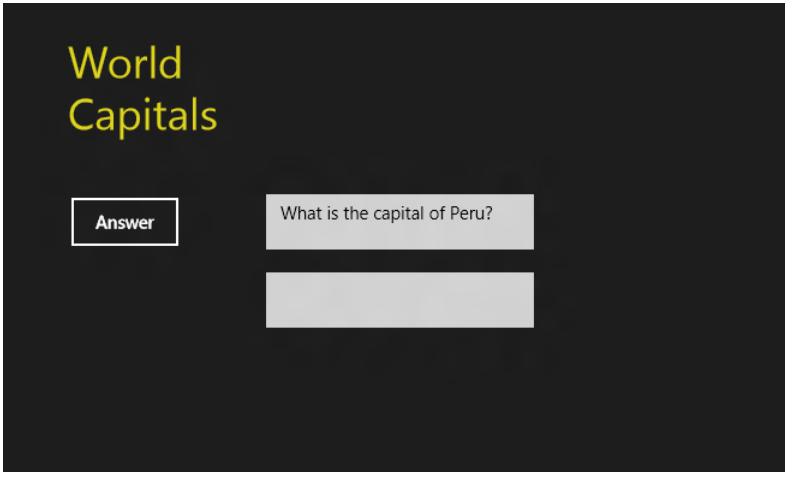
1. Click the Start button (the right-facing arrow next to the words *Local Machine*) on the Standard toolbar to run the World Capitals program in Visual Studio.

Tip You can also press F5 or click the Start Debugging command on the Debug menu to run a program in the Visual Studio IDE.

Visual Studio loads and compiles the project into an *assembly*, an EXE file that contains data and code in a form that can be used by the computer. This particular assembly also contains information that is useful for testing, or *debugging*, which is a fundamental part of the

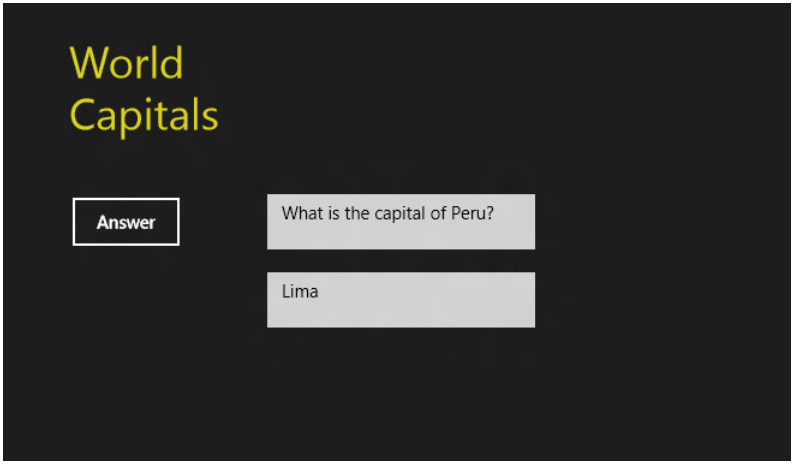
software development process. If the compilation is successful, Visual Studio runs the program in the IDE. (This is known as running the program on a *local machine*, as opposed to running on a remote computer somewhere on the web or in a software simulator of some kind.)

While the program is running, an icon for the program appears on the Windows taskbar. After a moment, you will see the World Capitals user interface running as any application would under Windows 8. The program looks just like the preview version did within the Visual Studio Designer:



World Capitals now asks you a question: What is the capital of Peru?

2. Click the Answer button to reveal the solution to the question, and the program should display the answer (Lima) below the question.



3. Close the application by dragging the title bar (or top portion of the screen) to the bottom of the screen (or however you normally terminate a Windows 8 application).

When you move the mouse cursor to the top edge of the screen, it changes to a hand, which provides some visual feedback as you drag the title bar to the bottom of the screen to terminate the program. After the application closes, you can press the Windows key or click the Visual Studio program icon on the desktop to activate the IDE again.

The World Capitals application may continue to run for a moment or two as the Visual Studio IDE catches up with the terminate-program request that you just issued. (For example, you may see the phrase *Running* in the Visual Studio title bar, which indicates that a program in the IDE is still executing.) You can force an immediate stop to any running application in the Visual Studio IDE by clicking the Stop Debugging button on the toolbar.

After the program has stopped running, you will notice a few changes in the IDE. For example, you will likely see an Output window at the bottom of the IDE with information about how the assemblies in the application were compiled and executed. This is the expected behavior within Visual Studio after a program has been compiled and run. The Output window provides a fairly detailed listing of what happened during compilation, a process that involves several stages and the loading of a number of files and resources called *libraries*. This record of the process is especially valuable when the compilation fails due to an unforeseen programming mistake or error. (Something that you will certainly experience, though not yet!)

4. After you've reviewed the content of the Output window, click its Close button to hide it.

   I won't emphasize the Output window much in the early chapters of this book, but if you encounter an inadvertent error as you write your own programs, you'll find this tool useful. Most of the time, though, you'll can simply close the window to allow more room for examining your code.

It's time to learn about another useful development tool.

The Properties Window

In the IDE, you can use a tool known as the Properties window to change the characteristics, or property settings, of one or more user interface elements on a page. A *property setting* is a quality of one of the objects in your program, such as its position on the screen, its size, the text displayed on it, and so on. For example, you can modify the question about capitals that the program asks to appear in a different font or font size by adjusting property settings. (With Visual Studio, you can display text in any font installed on your system, just as you can in Excel or Word.)

The Properties window contains a list of the properties for the object that is currently selected in the Designer window. For example, if a button object is selected in the Designer, the properties for the button object will be visible in the Properties window. The first property listed at the top of the Properties window is the *Name* property, and you will use this property to name your objects if you plan to customize them using Visual Basic code. (By default, new XAML objects are unnamed.) Although there are a lot of properties for each object on a page, Visual Studio assigns default values for most of them, and you can quickly find the properties that you want to set by arranging them using the Arrange By drop-down box at the top of the Properties window.

You can change property settings from the Properties window while you are working on a page, you can change a property setting by editing the XAML markup for a page, and you can add Visual Basic code to a page's code-behind file to instruct Windows to change one or more property settings while a program is running.

As you'll learn later, you can also customize the event handlers for objects on a page by using the Event Handlers button (which looks like a lightning bolt) near the top of the Properties window. *Event handlers* are custom Visual Basic routines that run when the user interacts with the objects on a page by clicking, tapping, dragging, and other actions.

To get some practice setting properties, you'll edit the text in a button object, and you'll change the font weight and style of a text block object to bold and italic, respectively.

Change properties

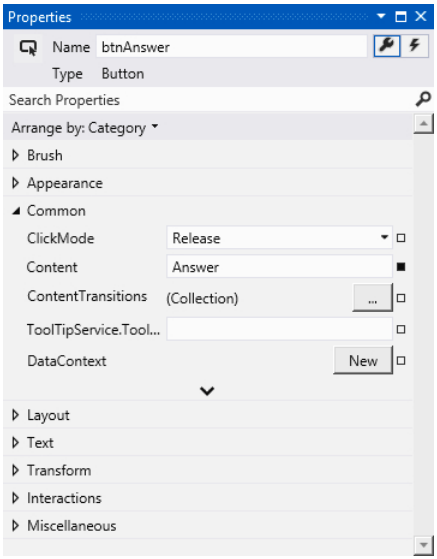
1. Click the Answer button on the page that is currently loaded in the Designer window.

   To work with an object on a page, you must first select the object. When you select an object, resize handles appear around it, and the property settings for the object are displayed in the Properties window. You'll also see numbers indicating the distance (in pixels) to the top and left edges of the window.

2. Press Alt+Enter to display the Properties window, if it is not currently visible.

   The Properties window might or might not be visible in Visual Studio, depending on how it has been configured and used on your system. It usually appears below Solution Explorer on the right side of the IDE. (If it is visible but not active, you might click the Properties window to activate it.)

You'll see a window similar to this one:



The Properties window lists all the property settings for the selected button object, named *btnAnswer*. Property names are listed in nested groups, and the default view displays the properties alphabetically by category. (*Brush* is first, *Appearance* is second, *Common* is third, and so on.) When you expand the property groups, the property names are generally listed on the left side, and the property settings are listed on the right. Some property settings, like Brush, are made by selecting color values with a design tool, so there are a variety of ways to set properties—not just entering text via the keyboard.

3. In the Common property group (containing the most typical properties for a button object), see that the *Content* property is set to "Answer."

   "Answer" is the text that currently appears on the page's main button, and you can change it to whatever you would like using the Properties window. Add an exclamation point now to the current value to add a little more emphasis to the button.

4. Click after "Answer" in the *Content* text box, type an exclamation mark (**!**), and then press Enter.

   The *Content* property setting is changed to "Answer!" in three places: within the Properties window, on the page in the Designer window, and within the XAML markup in the Code Editor.

   Note that instead of pressing the Enter key to change a property setting, you can simply click another location in the Properties window. (Click in another text box, for example.) Just don't inadvertently adjust another property setting by clicking around.

Now you'll change the font style of the text block object to bold and italic. The text block object currently contains the text "World Capitals."

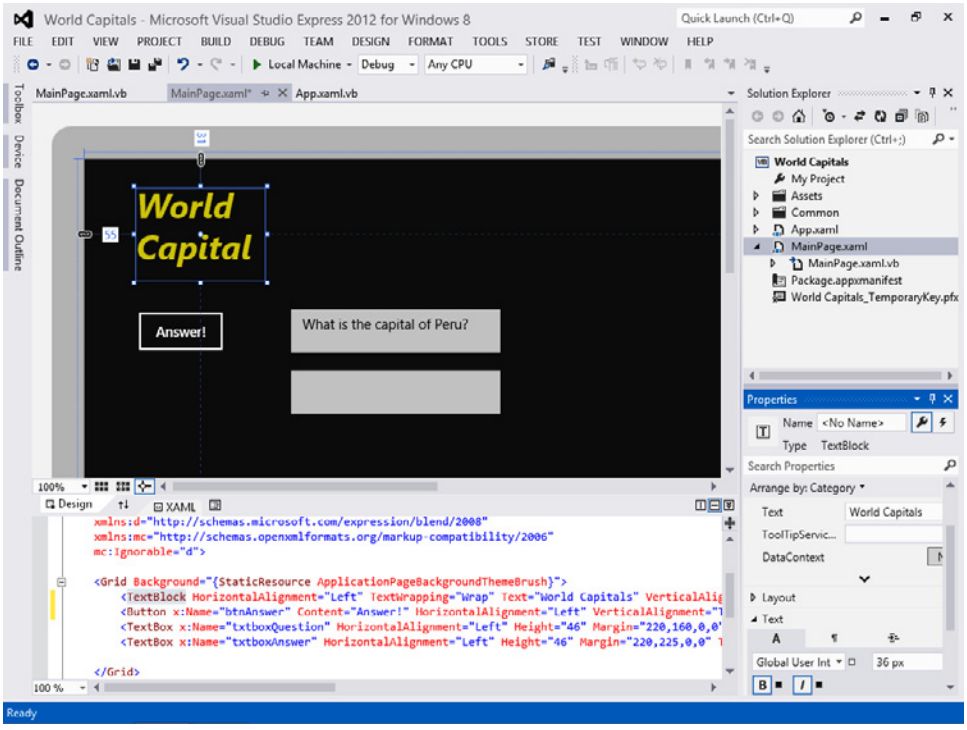
5. Click the "World Capitals" text block object on the page. A text block object is an excellent way to display descriptive text on a page.

6. In the Properties window, click the Text property group (not the *Text* property in the Common group that is currently visible).

7. Click the Bold button to change the font weight to bold.

8. Click the Italic button to change the font style to italic.

Visual Studio records your changes and adjusts the property settings accordingly. Your screen should look like this:



Congratulations—you've just updated three properties! As you design your programs, you'll have numerous font, color, and style options to choose from. And you've just learned how to use the Properties window—one of the important skills in becoming a Visual Basic programmer.

Thinking About Properties

In Visual Basic, each user interface element in a program (including the page itself) has a set of definable properties. You can set properties at design time by using the Properties window, or by editing properties in the XAML markup for the page that defines one part the program's user interface. Properties can also be set or referenced in Visual Basic code to make changes to program elements while the application runs. (User interface elements that receive input often use properties to receive information into the program.) At first, you might find properties a difficult concept to grasp. Viewing them in terms of something from everyday life can help.

Consider this bicycle analogy: a bicycle is an object you use to ride from one place to another. Because a bicycle is a physical object, it has several inherent characteristics. It has a brand name, a color, gears, brakes, and wheels, and it's built in a particular style. (It might be a road bike, a mountain bike, or a tandem bike.) In Visual Basic terminology, these characteristics are properties of the bicycle object. Most of the bicycle's properties were defined when the bicycle was built. But others (tires, travel speed, and options such as reflectors and mirrors) are properties that change while the bicycle is used. The bike might even have intangible (that is, invisible) properties, such as manufacture date, current owner, value, or rental status. And to add a little more complexity, a company or shop might own one bicycle or (the more likely scenario) an entire fleet of bicycles, all with different properties. As you work with Visual Basic, you'll set the properties of a variety of objects, and you'll organize them in very useful ways. Working with properties is a fundamental task in object-oriented programming.

Moving and Resizing the Programming Tools

With numerous programming tools to contend with on the screen, the Visual Studio IDE can become a pretty busy place. To give you complete control over the shape and size of the elements in the development environment, Visual Studio lets you move, resize, dock, and autohide most of the interface elements that you use to build programs.

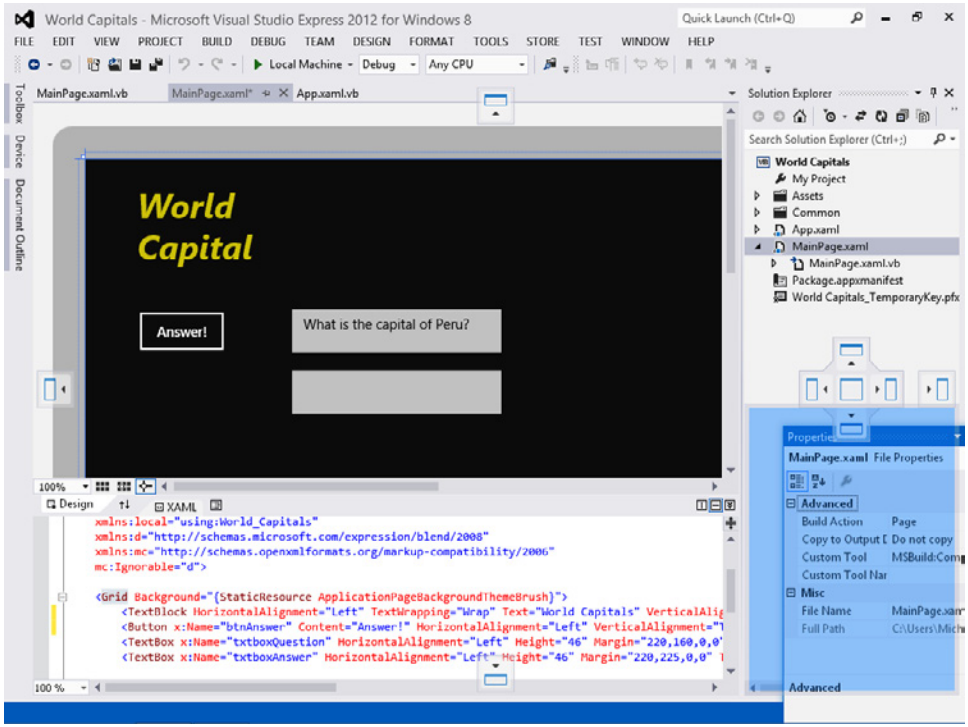
To move one of the tool windows in Visual Studio, simply click its title bar and drag the window to a new location. If you position the window somewhere in the middle of the IDE and let go, it will *float* over the surface of Visual Studio, unattached to other tool windows. If you drag a window along the edge of another window, it attaches to that window, or *docks* itself. Dockable windows are advantageous because they always remain visible. (They don't become hidden behind other windows.) If you want to see more of a docked window, simply drag one of its borders to view more content.

If you want to completely close a window, click the Close button in the upper-right corner of the window. You can always open the window again later by clicking the appropriate command on the View menu.

If you want an option somewhere between docking and closing a window, you might try *autohiding* a tool window at the side, top, or bottom of the Visual Studio IDE by clicking the tiny Auto Hide pushpin button on the right side of the tool's title bar. This action removes the window from the docked position and places the title of the tool at the edge of the development environment on an unobtrusive tab. When you autohide a window, you'll notice that the tool window remains visible as long as you keep the mouse pointer in the area of the window. When you click another part of the IDE, the window slides out of view.

To restore a window that you have autohidden, click the tool tab at the edge of the development environment. (You can recognize a window that is autohidden because the pushpin in its title bar is pointing sideways.) By clicking the tool tab repeatedly at the edge of the IDE, you can use the tools in what I call peekaboo mode—in other words, to quickly display an autohidden window, click its tab, check or set the information you need, and then click its tab again to make it disappear. If you ever need the tool displayed permanently, click the Auto Hide pushpin button again so that the point of the pushpin faces down, and the window then remains visible.

A useful capability of Visual Studio is also the ability to dock windows as tabbed documents (windows with tab handles that partially hide behind other windows). You can also manually dock tool windows where you would like by dragging the windows and using the docking guides that appear as tiny squares on the perimeter of the IDE. A centrally located *guide diamond* will also help you manually dock tool windows by giving you a preview of where the windows will go.



The docking guides are changeable icons that appear on the surface of the IDE when you move a window or tool from a docked position to a new location. Because the docking guides are associated with shaded, rectangular areas of the IDE, you can preview the results of your docking maneuver before you actually make it. Your window orientation changes will not stick until you release the mouse button.

Since docking and auto-hiding techniques take some practice to master, I recommend that you use the following exercises to experiment with the window-management features of the IDE. After you complete the exercises here, feel free to configure the Visual Studio tools in a way that seems comfortable for you.

Moving and Resizing Tool Windows

To move and resize one of the programming tool windows in Visual Studio, follow these steps. This exercise demonstrates how to manipulate the Properties window, but you can work with a different tool window if you want to.

Move and resize the Properties window

1. If the Properties window isn't visible in the IDE, click Properties Window on the View menu.

   This will activate the Properties window is in the IDE and highlight its title bar.

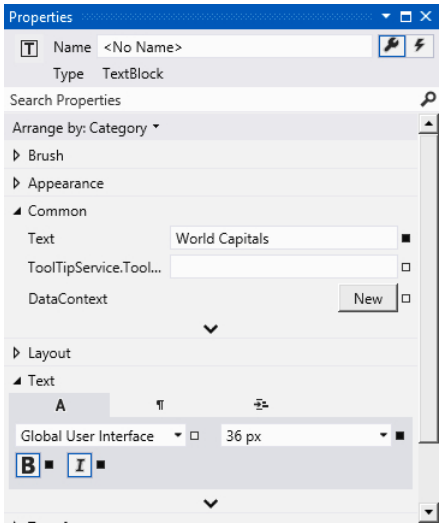
2. Click the Window Position toolbar menu button (the small arrow icon) on the Properties window title bar, and then click Float to display the window as a floating (undocked) window.

3. Using the Properties window title bar, drag the window to a new location in the development environment, but don't dock it.

   Moving windows around the Visual Studio IDE gives you some flexibility with the tools and the look of your development environment.

   Now you'll resize the Properties window to see more object property settings at once.

4. Point to the lower-right corner of the Properties window until the pointer changes to a double-headed arrow (the resizing pointer). Then drag the lower-right border of the window down and to the right to enlarge the window.



You can work more quickly and with more clarity of purpose in a bigger window. Feel free to move or resize a window when you need to see more of its contents.

Docking Tool Windows

If a tool window is floating over the development environment, you can move it to a new place by dragging the window to the handy docking guides. You can also maximize a floating tool window by double-clicking its title bar. (If you double-click the title bar again, it will return to its original shape.) Try docking the Properties window in a different location now.

Dock the Properties window

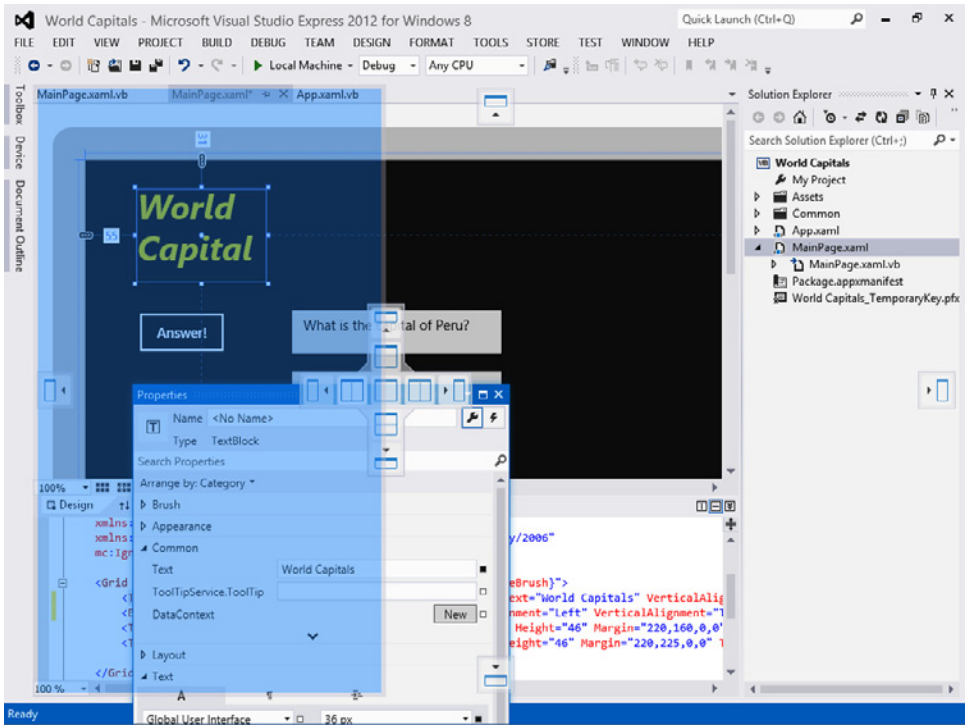
1. Verify that the Properties window (or another tool that you want to dock) is floating over the Visual Studio IDE in an undocked position.

   If you completed the previous exercise, the Properties window should be undocked now.

2. Drag the title bar of the Properties window to the top, bottom, right, or left edge of the development environment (your choice), taking care to drag the mouse pointer over one of the docking guides (small squares with arrows) on the perimeter of the Visual Studio IDE, or one of the centrally located rectangles in a tight pattern, called collectively a *guide diamond*.

   As you move the mouse over a docking guide, a blue, shaded rectangle representing the Properties window snaps into place, giving you an indication of how your window will appear when you release the mouse button. Note that there are several valid docking locations for

tool windows in Visual Studio, so you might want to try two or three different spots until you find one that looks right to you. (A window should be located in a place that's handy and not in the way of other needed tools.)



3. Release the mouse button to dock the Properties window, and the window should snap into place in its new home.

Tip To switch between dockable, tabbed document, hidden, and floating styles, right-click the window's title bar (or tab, if it is a tabbed document), and then click the option you want. Although the Properties window works very well as a dockable window, you'll probably find that larger windows work best as tabbed document windows.

4. Try docking the Properties window several more times in different places to get the feel of how docking works.

I guarantee that although a few of these window procedures may seem confusing at first, after a while they'll become routine for you. In general, you want to create window spaces that have enough room for the information you need to see and use while you work on more important tasks in the Designer window and in the Code Editor.

Hiding Tool Windows

To hide a tool window, click the Auto Hide pushpin button on the right side of the title bar to conceal the window beneath a tool tab on the edge of the IDE, and click it again to restore the window to its docked position. You can also use the Auto Hide command on the Window menu (or right-click a title bar and select Auto Hide) to autohide a tool window. Give it a try now.

Use the Auto Hide feature

1. Locate the Auto Hide pushpin button on the title bar of the Properties window.

   The pushpin is currently in the *down*, or pushed-in, position, meaning that the Properties window is "pinned" open and autohide is disabled.

2. Click the Auto Hide button on the Properties window title bar.

   The Properties window will slide off the screen and be replaced by a small tab named Properties.

   The benefit of enabling autohide, of course, is that it frees up additional work space in Visual Studio while keeping the hidden window quickly accessible.

3. Click the Properties tab.

   The Properties window should immediately reappear.

4. Click the mouse elsewhere within the IDE.

   The window disappears again.

5. Finally, display the Properties window again, and then click the pushpin button on the Properties window title bar.

   The Properties window returns to its familiar docked position, and you can use it without worrying about it sliding away.

   Spend some time moving, resizing, docking, and autohiding tool windows in Visual Studio now to create your version of the perfect work environment. As you work through this book, you'll want to adjust your window settings periodically to adapt your work area to the new tools you're using.

Tip Visual Studio lets you save your window and programming environment settings and copy them to a second computer or share them with members of your programming team. To experiment with this feature, click the Import And Export Settings command on the Tools menu and follow the wizard instructions to export (save) or import (load) settings from a file.

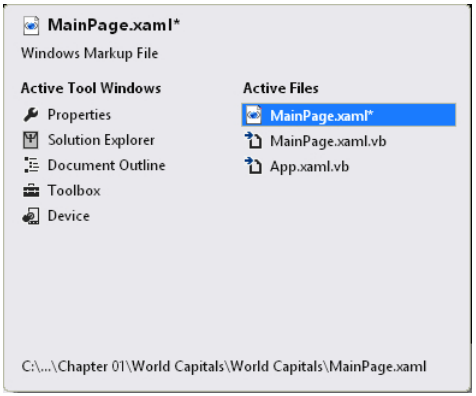
Switching among Open Files and Tools Using the IDE Navigator

Visual Studio has a feature that makes it even easier to switch among open files and programming tools in the development environment. This feature is called the *IDE Navigator*, and it lets you cycle through open files and tools by using key combinations, in much the same way that you cycle through open programs on the Windows taskbar. This feature is especially useful when you have a lot of files open in the IDE. Give it a try now.

Use the IDE Navigator

1. Hold down the Ctrl key and press Tab to open the IDE Navigator.

   The IDE Navigator opens and displays the active (open) files and tools in the IDE. Your screen will look similar to the following:



2. While holding down the Ctrl key, press Tab repeatedly to cycle through the active files until the file you want is highlighted.

   To cycle through the files in the reverse direction, hold down Ctrl+Shift and press Tab.

3. While holding down the Ctrl key, use the arrow keys to cycle through both the active files and the active tools. You can also select an active file (or tool) by clicking its name.

4. When you're finished with the IDE Navigator, release the Ctrl key.

   The last selected item in the IDE Navigator will become active.

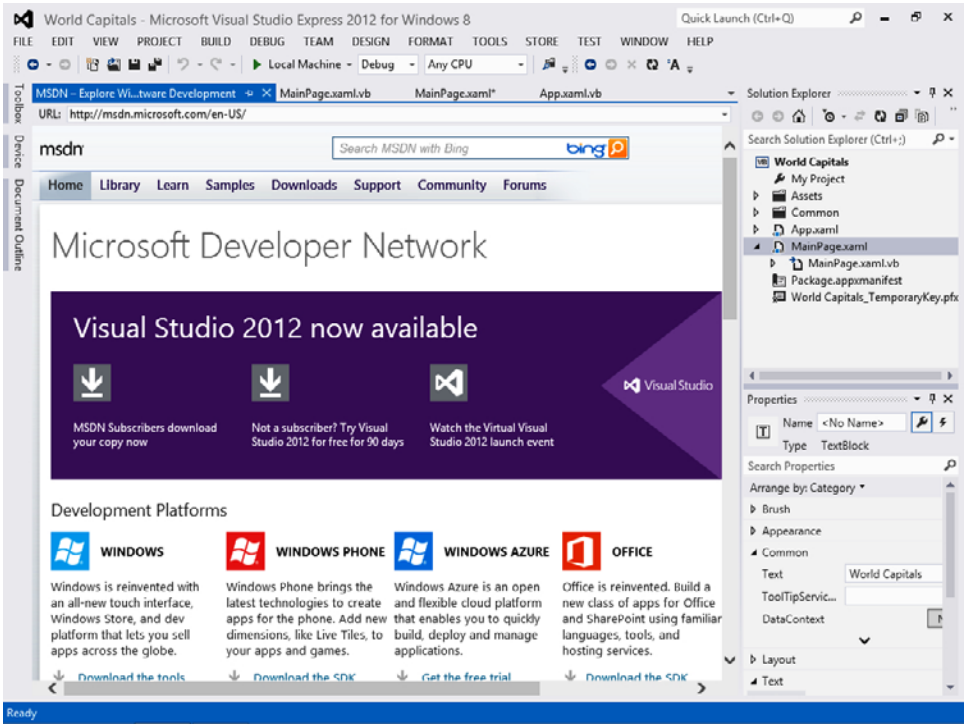
Opening a Web Browser Within Visual Studio

A handy feature in Visual Studio is the ability to open a simple web browser within the development environment. The browser appears as a tabbed document window in the IDE, so it takes up little space but can be opened immediately when needed. You could open a stand-alone web browser (such as Internet Explorer) and keep it nearby on the Windows taskbar, but running a web browser within Visual Studio makes examining web sites and copying data into Visual Studio even easier. Try using the Visual Studio web browser now.

Open the Visual Studio web browser

1. Click the Other Windows submenu on the View menu and then click the Web Browser command.

   The Web Browser window appears, as shown here:



   The browser is a tabbed document window by default, but you can change it to the float position by right-clicking the window title bar and then clicking the Float command.

Tip You can change the default page that appears in the Web Browser window by changing the setting in the Options dialog box. Open the Options dialog box by clicking Options on the Tools menu. Expand Environment, and then click Web Browser. Change the Home Page setting to a URL you want for the default page.

2. Experiment with the browser and how it functions within the IDE.

   Although the browser is more basic than Internet Explorer or another full-featured browser, you may find it a useful addition to the Visual Studio tool collection. You can also open and run Internet Explorer (or another browser) directly from the Windows taskbar.

3. When you're finished, click the Close button on the right side of the web browser title bar to close the window.

Customizing IDE Settings to Match This Book's Exercises

Like the tool windows and other environment settings within the IDE, the compiler and personal settings within Visual Studio are highly customizable. It is important to review a few of these settings now so that your version of Visual Studio is configured in a way that is compatible with the step-by-step programming exercises that follow. You will also learn how to customize Visual Studio generally so that as you gain programming experience, you can set up Visual Studio in the way that is most productive for you.

Checking Project and Compiler Settings

If you just installed Visual Studio, you are ready to start this book's programming exercises. But if your installation of Visual Studio has been on your machine for a while, or if your computer is a shared resource used by other programmers who might have modified the default settings (perhaps in a college computer lab), complete the following steps to verify that your settings related to projects, solutions, and the compiler match those that I use in the book.

Check project and compiler settings

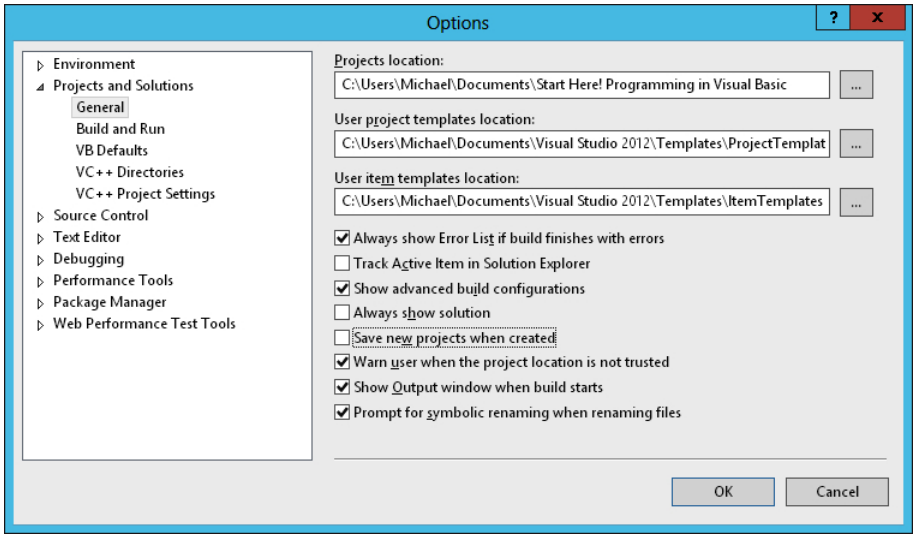
1. Click the Options command on the Tools menu to display the Options dialog box.

   The Options dialog box is your window to many of the customizable settings within Visual Studio. To assist you in finding the settings that you want to change, Visual Studio organizes the settings by category.

2. Expand the Projects And Solutions category and then click the General item within it.

   This group of check boxes and options configures the Visual Studio project and solution settings.

3. So that your software matches the settings used in this book, adjust your settings to match those shown in the following dialog box:



In particular, I recommend that you clear the check marks (if you see them) from the Always Show Solution and Save New Projects When Created check boxes. The first option shows additional solution commands in the IDE, which is not necessary for solutions that contain only one project (the situation for most programs in this book). The second option causes Visual Studio to postpone saving your project until you click the Save All command on the File menu and provide a location for saving the file. This delayed-save feature allows you to create a test program, compile and debug the program, and even run it without actually saving the project on disk—a useful feature when you want to create a quick test program that you might want to discard instead of saving. (An equivalent situation in word-processing terms is when you open a new Word document, enter an address for a mailing label, print the address, and then exit Word without saving the file.) With this default setting, the exercises in this book prompt you to save your projects after you create them, although you can also save your projects in advance by selecting the Save New Projects When Created check box.

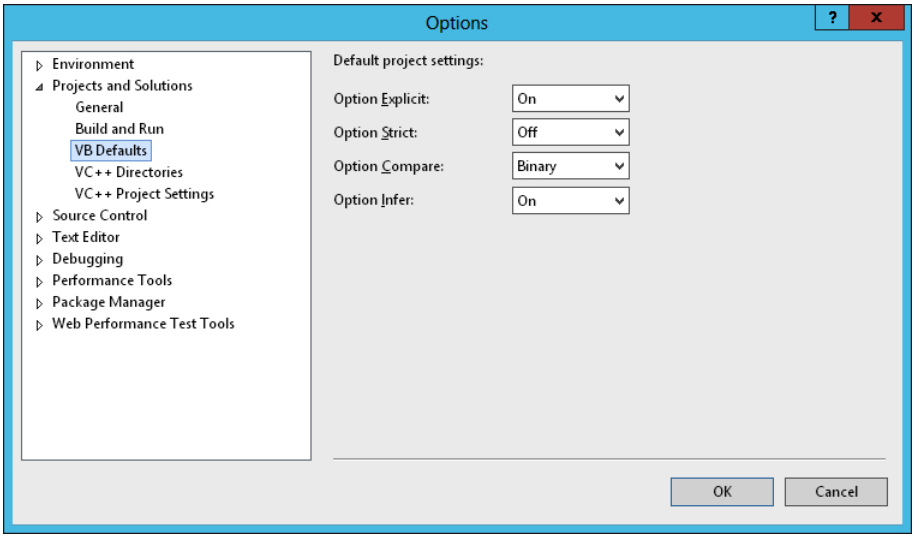
You'll also notice that I have browsed to the location of the book's sample files (Start Here! Programming in Visual Basic) in the top text box on the form to indicate the default location for this book's sample files. Most of the projects that you create will be stored in this folder,

and they will have a "My" prefix to distinguish them from the completed project I provide for you to examine. (Be sure to change this path to the location of the book's sample files on your computer.)

After you have adjusted these settings, you're ready to check the Visual Basic compiler settings.

4. Click the VB Defaults item in the expanded Projects And Solutions section.

Visual Studio displays a list of four compiler settings: Option Explicit, Option Strict, Option Compare, and Option Infer. Your screen looks like this:



Although a detailed description of these settings is beyond the scope of this chapter, you'll want to verify that Option Explicit is set to On and Option Strict is set to Off—the default settings for Visual Basic programming within Visual Studio. Option Explicit On is a setting that requires you to declare a variable before using it in a program—a very good programming practice that I want to encourage. Option Strict Off allows variables and objects of different types to be combined under certain circumstances without generating a compiler error. (For example, a number can be assigned to a text box object without error.) Although this is a potentially worrisome programming practice, Option Strict Off is a useful setting for certain types of demonstration programs.

Option Compare determines the comparison method when different text strings are compared and sorted. For more information about comparing strings and sorting text, see Chapter 10, "Managing Data with Arrays and LINQ."

Option Infer was a new setting in Visual Basic 2008. When you set Option Strict to Off and Option Infer to On, you can declare variables without explicitly stating a data type. Or rather, if you make such a declaration, the Visual Basic compiler will infer (or take an educated guess about) the data type based on the initial assignment you made for the variable. You'll learn more about the feature in Chapter 6, "Visual Basic Language Elements."

As a general rule, I recommend that you set Option Infer to Off to avoid unexpected results in how variables are used in your programs. I have set Option Infer to Off in most of the sample projects included in the sample files.

Feel free to examine additional settings in the Options dialog box related to your programming environment and Visual Studio. When you're finished, click OK to close the Options dialog box.

You're ready to exit Visual Studio and start programming.

Exiting Visual Studio

When you're finished using Visual Studio for the day, save any projects that are open and close the development environment. Give it a try.

Exit Visual Studio

1. Save any changes you've made to your program by clicking the Save All button on the Standard toolbar.

   You've made a few changes to your project, so you should save your changes now.

2. On the File menu, click the Exit command.

   The Visual Studio program closes. You are now ready to create your first program from scratch in Chapter 2.

Summary

Each chapter in this book concludes with a Summary section that offers a review of what the chapter has presented. You can use these sections to quickly recap what you have learned in each chapter before you move on to the one that follows.

This chapter introduced you to Visual Studio 2012 Express for Windows 8 and the IDE that you use to open and run Visual Basic programs. You can create Windows 8 apps by opening a new or existing solution in Visual Studio, and then adding to the solution with the assorted programming tools provided by the product. In this chapter, you learned how to display the user interface of a Visual Basic program, how to examine the objects on a page, and how to change property settings with the Properties window.

As you toured the Visual Studio IDE, you learned how to use menu commands, how to open and run a program, how to examine XAML markup for a page in the Code Editor, and how to move important tool windows around the IDE. You also learned how to customize settings in Visual Studio by using the Options command on the Tools menu.

The Visual Studio IDE is a busy place, with many more commands and features than this chapter covered. However, as you continue reading this book, you will be introduced to the most important programming tools and techniques one step at a time. Although you worked with a very simple Windows 8 application in this chapter, Visual Studio is capable of creating a variety of powerful programs for a number of different hardware environments or platforms. These platforms include Windows 8 and earlier operating systems, Windows Phone, web applications for numerous browsers, Xbox system games, and Windows Azure (cloud-computing) web servers. By learning Visual Basic 2012, you will be well positioned to benefit from many of the most exciting technologies in the marketplace.

Chapter 2

Creating Your First Windows 8 Application

After completing this chapter, you'll be able to

- Create the user interface for a Windows 8 application.
- Add XAML controls to a page.
- Move and resize objects on a page.
- Set the properties for objects in the user interface.
- Write Visual Basic program code for an event handler.
- Save, run, and test a program.
- Build an executable file and deploy the application.

CHAPTER 1, "GETTING TO KNOW VISUAL BASIC 2012," introduced you to Microsoft Visual Studio 2012 and the development tools that you can use to build Microsoft Visual Basic applications. In this chapter, you'll learn how to create your first Windows 8 application from scratch. The program will be a simple Internet-browsing tool that displays active webpage content in a window and also keeps track of the web addresses that you use as you run the program.

Although the project itself is quite basic, it will teach you essential programming techniques that you will use each time that you write a program. You will create a basic user interface with XAML controls, adjust property settings for the controls, and add Visual Basic program code to create an event handler. Along the way, you'll learn how to use the Visual Studio Toolbox, how to use the *Button*, *TextBox*, and *WebView* controls, and how to use the Code Editor to create a Visual Basic code-behind file for the user interface. You'll also learn how to run and test a program, how to save your changes, and how to deploy an application so that it is ready to use on your computer.

Web List: Your First Visual Basic Program

The Windows 8 application that you are going to create is Web List, a program that displays live webpage content and also keeps track of the web sites that you visit in a text box. The tool allows you to rapidly examine and record a list of favorite web sites, and it also shows you how to view web site information directly on the home page of an application. Here's what the Web List program will look like when it's finished:

The Web List user interface contains one button, two text boxes, and a web browser window that allows you to examine the live contents of any web site you wish. I produced these elements by creating four objects on the Web List application page and then changing several properties for each object. After I designed the user interface, I used Visual Basic program code to create an event handler for the Visit Web Site button, which executes when the user types a web address in the first text box and clicks the Visit Web Site button. To re-create Web List, you'll follow three essential programming steps in Visual Studio: creating the user interface, setting the properties, and writing the Visual Basic program code. The following list outlines the development process for Web List:

1. Create the user interface (requires four objects).
2. Set the properties (requires nine properties).
3. Write the program code (requires one object).

Creating the User Interface

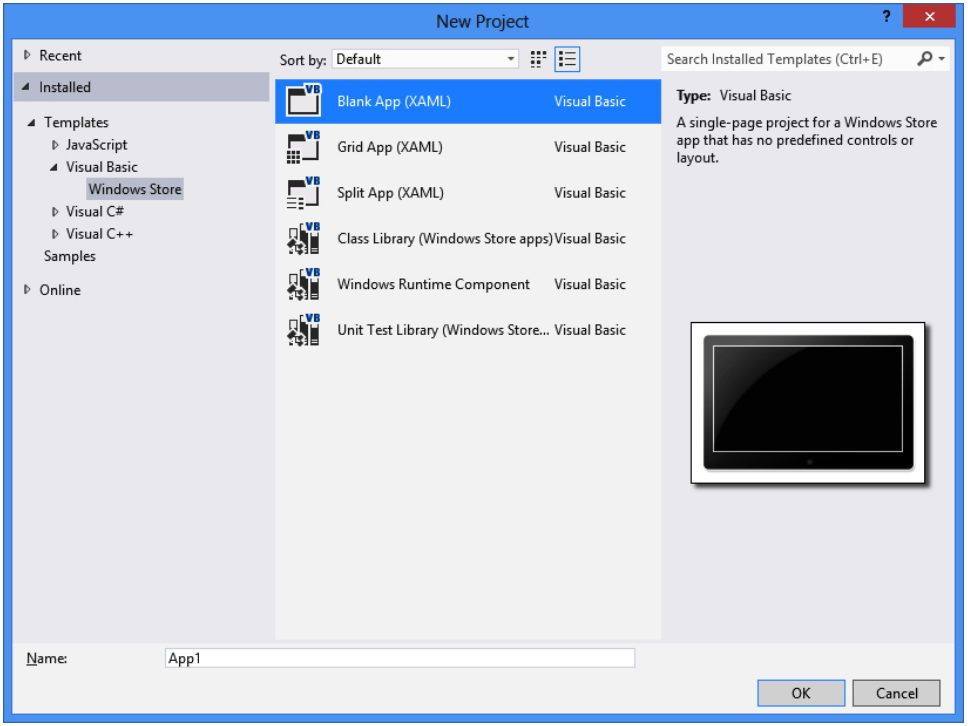
In this exercise, you'll start building Web List by creating a new project and then using XAML Toolbox controls to construct the user interface.

Create a new project

1. Start Visual Studio 2012.
2. On the Visual Studio File menu, click New Project.

Tip You can also start a new programming project by clicking the New Project link on the Visual Studio Start page. (The link is formatted in blue in most configurations.)

The New Project dialog box opens, as shown here:



The New Project dialog box provides access to the major project types available for writing Windows 8 applications, which are also called Windows Store apps. Since the most recent selection I made in this dialog box was Visual Basic, the Visual Basic templates are currently visible, but other programming templates and resources are also available, including those for Microsoft Visual C#, Microsoft Visual C++, and JavaScript.

3. In the Visual Basic template group, click the Blank App (XAML) project.

   Visual Studio prepares the development environment for a basic Windows 8 application with no predefined controls or layout.

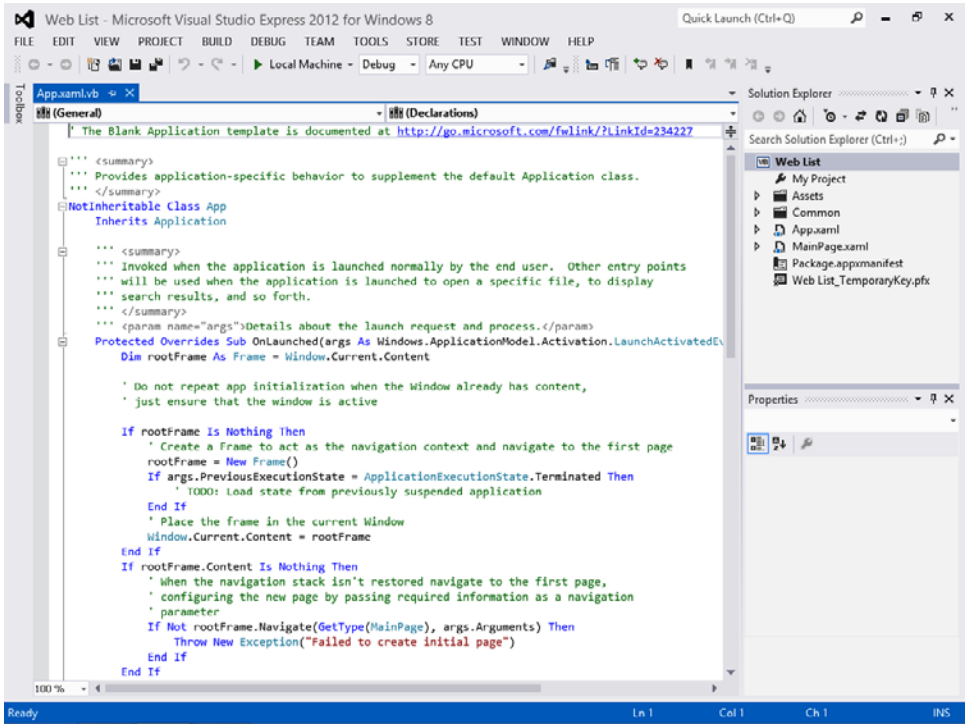
4. In the Name text box, type **My Web List**.

   Visual Studio assigns the name My Web List to your project. (You'll specify a folder location for the project later.) I'm recommending the "My" prefix here so you don't confuse your new application with the Web List project I've created for you on disk.

Tip If your New Project dialog box contains Location and Solution Name text boxes, you need to specify a folder location and solution name for your new programming project now. The presence of these text boxes is controlled by a check box in the Project And Solutions category of the Options dialog box, but it may not be the default setting. (You display this dialog box by clicking the Options command on the Tools menu.) Throughout this book, you will be instructed to save your projects (or discard them) after you have completed the programming exercise. For more information about this delayed-saving feature and default settings, see the section entitled "Customizing IDE Settings to Match this Book's Exercises" in Chapter 1.

5. Click OK to create the new project in Visual Studio.

   Visual Studio cleans the slate for a new programming project and displays Visual Basic code associated with the blank application template in the IDE. Your screen will look like this:

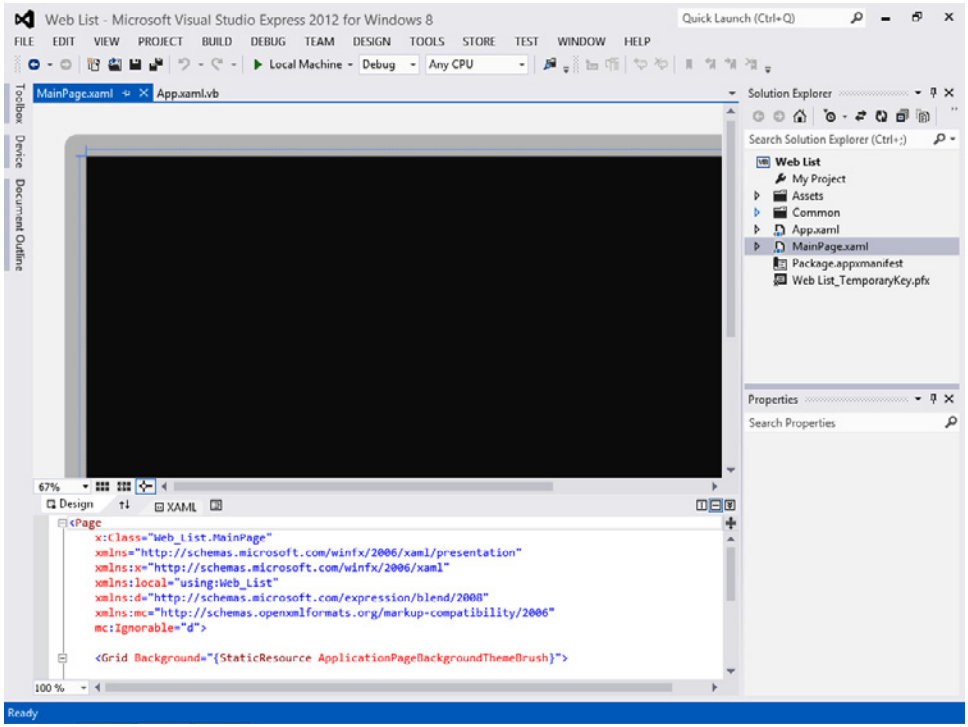


You won't spend too much time with this code right now. What you see is standard start-up code for a Windows 8 application created within Visual Studio, and it is stored in the file *App.xaml.vb* within the project. Although each project contains an *App.xaml* file, your work today will begin in the application's user interface, which is stored in the *MainPage.xaml* file. You'll display that user interface now and enhance it with Toolbox controls.

Create a user interface

1. Open Solution Explorer if it is not currently visible, and then double-click the file *MainPage.xaml*.

   Visual Studio opens *MainPage.xaml* in a Designer window and shows the upper-left corner of the application's main page. Below this page, you'll see the Code Editor with several lines of XAML markup associated with the user interface page in the Designer window. As you add controls to the application page in the Designer window, the Code Editor reflects the changes by displaying the XAML statements that will create the user interface. Your screen should look like this:



Now let's get to know the Designer window a bit better.

2. Click the scroll box in the Designer window's vertical scroll bar and drag it down.

   When you drag a scroll box in the Designer window, you can see more of the user interface you are working on.

3. Click the scroll box on the Designer window's horizontal scroll bar and drag it right. (Likewise, when you drag a horizontal scroll box, you can see hidden parts of the user interface.)

   Near the lower-left corner of the Designer window, you'll see a Zoom tool, which allows you to zoom in on the current application page (to see more detail) or zoom out (to see more of the page). The current value of the Zoom tool is 100%. You can select a different value by clicking the Zoom tool's drop-down button.

4. Click the Zoom drop-down button and then click Fit All.

   The entire application page now fits within the Designer window. Depending on your screen resolution and the amount of screen space you have designated for the other IDE tools, you'll see a somewhat smaller version of the page.

Tip If your mouse has a mouse wheel, you can move quickly from one zoom setting to the next by holding down the Ctrl key and rotating the mouse wheel. This feature works whenever the Designer window is active.

It is important to be able to quickly view different parts of the application page in different sizes while you build it. Sometimes you want to see the entire page to consider the layout of controls or other elements, and sometimes you need to view portions of the page up close. Now return to the original setting.

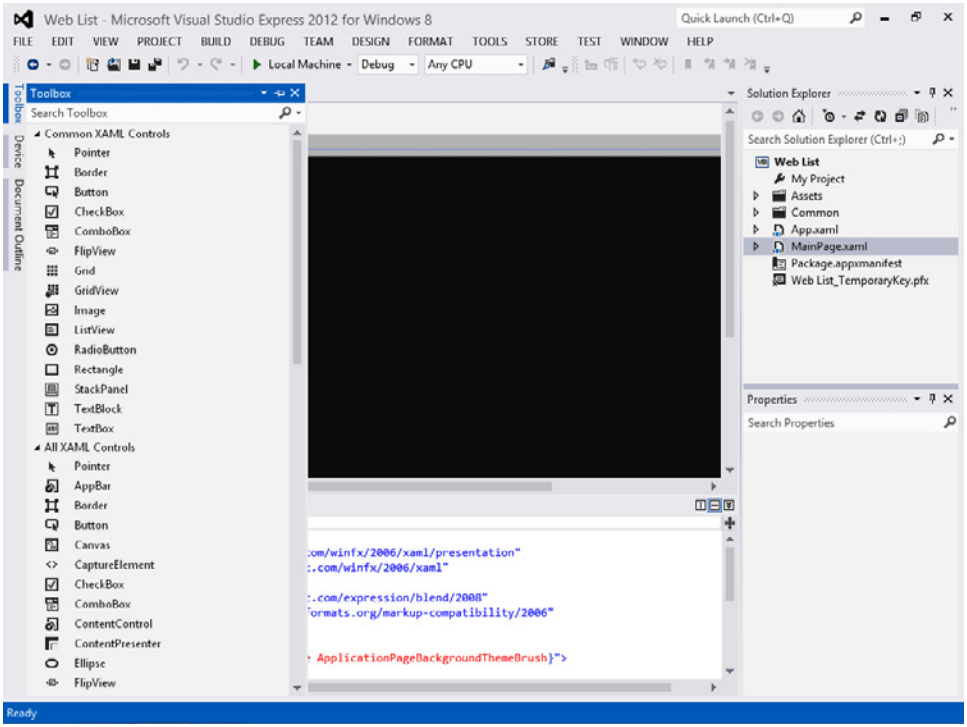
5. Click the Zoom drop-down button, and then click 100%.

   Now you'll open the Toolbox.

6. If the Toolbox is not currently visible, click the Toolbox tab or click the Toolbox command on the View menu.

   The Toolbox window contains a large collection of user interface controls that you can add to your application. Because you are building an application for Windows 8, the types of controls that are displayed in the Toolbox are so-called *XAML controls*—that is, structured elements that control the design of an application and can be successfully organized on a page by the XAML parser within Visual Studio. There are other types of Toolbox controls for other types of applications (Windows forms controls, HTML controls for web applications, and so on), but you don't have to worry about that now—Visual Studio automatically loads the proper controls into the Toolbox when you open a new solution.

   Your screen should look like this:



For convenience, the Toolbox controls have been organized into two groups: Common XAML Controls (those controls that appear in many applications), and All XAML Controls (a list of all

the XAML controls that are currently installed on your system). Keep in mind that the Toolbox window is like any other tool window in the Visual Studio IDE. You can move it, resize it, or pin it as needed. Most programmers have the Toolbox open while adding controls to a new page, and then they pin it to the side of the IDE, as you'll do in the following step.

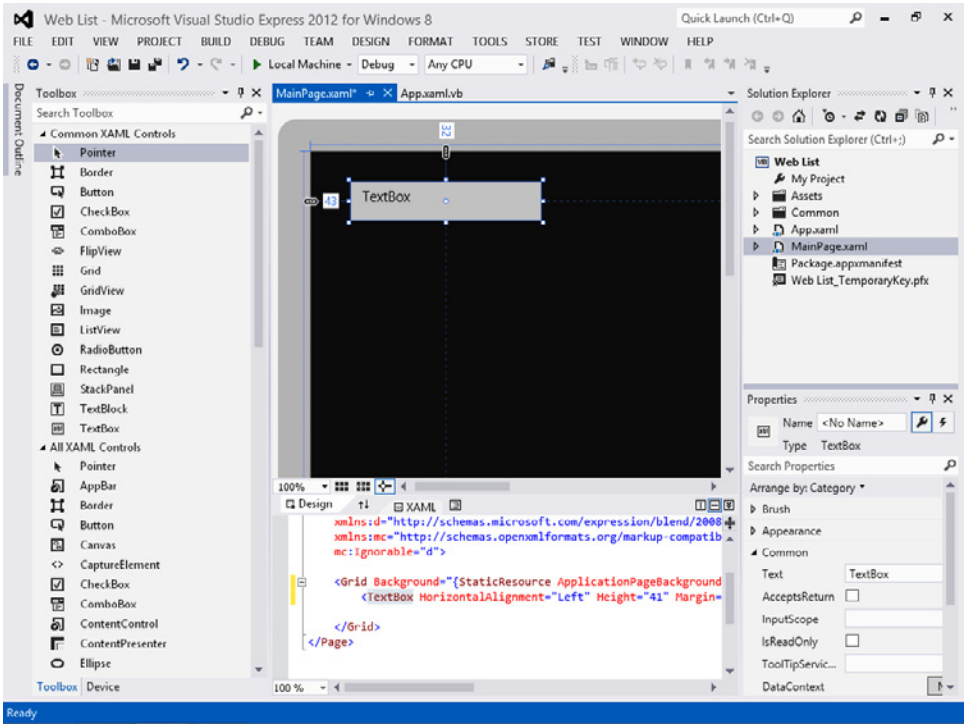
Next, you'll practice adding a text box object to the page.

7. Click the *TextBox* control in the Toolbox, and then click the Auto Hide button on the Toolbox title bar to pin the Toolbox to the IDE, if it is not already pinned.

   Remember that when you pin a window to the IDE, the Auto Hide pushpin points in the down direction. Putting the Toolbox in this position will stop it from obscuring any of the Designer window while you work.

8. Move the mouse pointer to the upper-left corner of the Designer window, and then drag right and down to create a rectangular-shaped text box object on the page.

   When you release the mouse button, Visual Studio creates a XAML text box object on the page, as shown here. (You may need to reposition the Designer window to see what is shown in the illustration.)



The text box is currently enclosed with selection handles, indicating that the object is selected in the IDE. The property settings of the selected object are loaded into the Properties window, and

below in the Code Editor, XAML markup for the text box object appears nested within a grid object. All Windows 8 app user interfaces created with XAML markup have a grid object as their base layout element, and the controls that you add to the page appear nested within this main grid.

Move and resize a text box

1. Point to the lower-right corner of the new text box object, click the sizing handle on the corner of the text box, and drag it to a new location.

   As you drag the corner, the text box object will be resized, although the upper-left corner of the text box will remain in place.

   Whenever an object is selected in the Designer window, you can resize it by using the sizing handles. As you make sizing adjustments, grid lines reveal the dimensions of the object in pixels, and you can use the grid lines to create objects of a uniform size, or to align an object with another object on the page. Your selected text box will also have little locking icons at the top and left edges of the text box. These locks indicate that the element is *locked*, or frozen, a set distance from the edge of the window, but you can adjust this by clicking the locks, which open and close like a toggle.

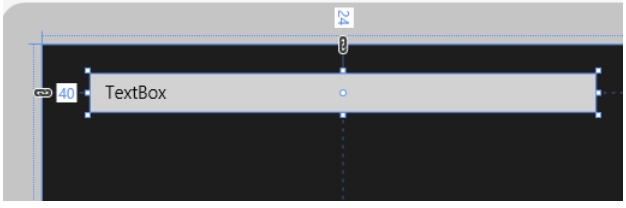
2. Click the middle of the text box object and slowly drag it around the application page.

   The text box floats around the surface of the page, and the grid lines and sizing information adjust as you move the object.

   It is very simple to move objects on a page in the Designer window. As you make these adjustments, note that your changes are recorded in XAML markup in the Code Editor, as well as in the Designer window.

3. Position the text box so that it is 40 pixels from the left edge of the page and 24 pixels from the top edge of the page, and has a height of 32 pixels and a width of 420 pixels.

   These dimensions will be fine for the single-line text box that the user will use to enter the web address (URL) that they want to browse to when the program runs. Your Designer window should look like this:



Now you'll add a button object to the page. The Toolbox should currently be visible, since you pinned it to the IDE.

Add a button object

1. Click the *Button* control in the Toolbox and then move the mouse pointer over the application page.

   The mouse pointer changes to crosshairs and a button icon. The crosshairs are designed to help you draw the rectangular shape of the button on the page. You can also create a button with the default size by double-clicking the *Button* control in the Toolbox.

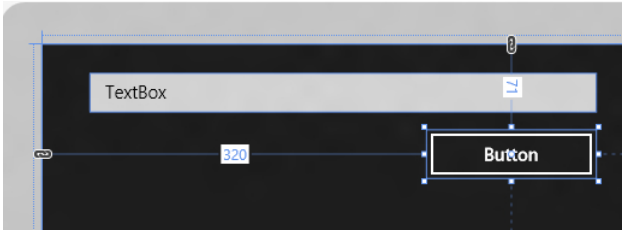
2. Drag the pointer down and to the right. Release the mouse button to complete the button, and watch it snap to the page.

3. Resize the button object so that it is 40 pixels high and 140 pixels wide.

Tip At any time, you can delete an object and start over again by selecting the object on the page and then pressing Delete. Feel free to create and delete objects to practice creating your user interface.

4. Move the button object so that it is below the text box object. Snap lines will appear as you move the object, and the right edge of the button will snap to the right edge of the text box when aligned.

   Your screen should look like this:



Now you'll add a second (larger) text box object to the page. This text box object will contain the list of web sites that you visited while the Web List program was running.

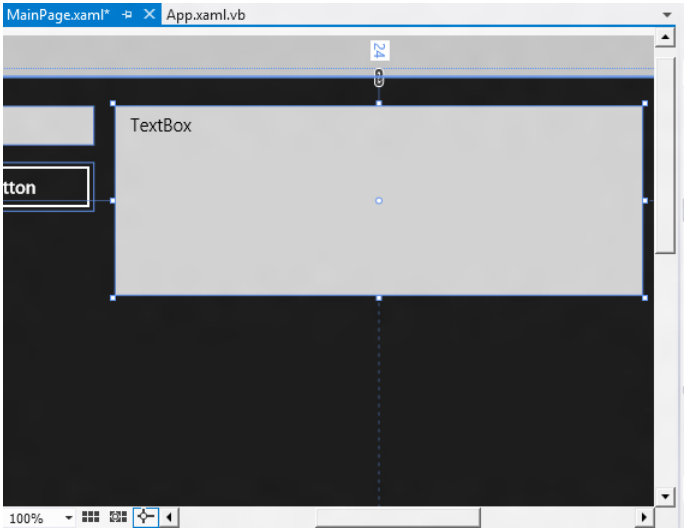
Add a second text box

1. Use the horizontal scroll bar in the Designer window to make the right side of the application page more visible.

   You're going to add the second text box object to the right side of the application page.

2. Click the *TextBox* control in the Toolbox, move the mouse to the Designer window, and then create a second text box object on the page. Make the text box about the same width but much taller than the first one.

   Visual Studio creates a second text box object on the page. Your screen will look like this:



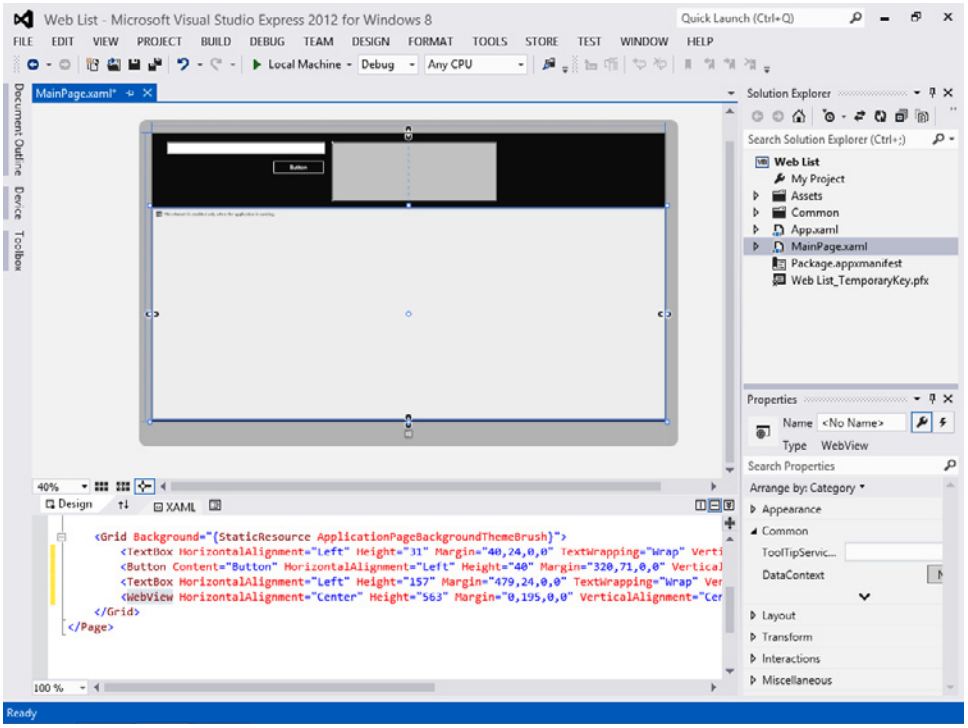
Now you'll add a large web browser window on the page to display information from the web sites that you visit. This object is created by using the *WebView* control in the Toolbox. *WebView* is not a full-featured web browser like Internet Explorer. However, it was added to the XAML Toolbox to give Visual Studio programmers a simple way to display live web information in a Windows 8 application.

Add a web view object

1. Click the *WebView* control in the Toolbox. (You'll find it in the All XAML Controls section.)
2. Using the drawing pointer for the control, create a very large rectangular box on the page below the button object and second text box object.

   The goal with this object is to display as much of the web browser as possible. However, for web content that extends beyond the viewing area, the web view object will allow users to scroll up and down to see more information.
3. After you create the object, you may wish to close the Toolbox window or adjust the amount of zoom magnification in the Designer window to make as much of the page visible in the IDE as possible.

Your final page should look like this in the Designer window:



In my Designer window, the zoom is set to Fit All (about 40 percent) so that I can see the entire application page. Also note how the Code Editor now shows XAML markup for four objects: a text box, a button, a second text box, and a web browser. Now you're ready to customize your interface by setting a few properties.

Setting the Properties

As you discovered in Chapter 1, you can change properties by selecting objects on a page and changing their settings in the Properties window. You'll start by changing the property settings for the first text box.

Set the web address text box

1. Click the first text box you created on the page. The text box object is selected and is surrounded by resize handles.
2. Open the Properties window.

Tip If the Properties window isn't visible, click the Properties Window command on the View menu, or press Alt+Enter.

3. At the top of the Properties window, click the Name property text box. The *Name* property will appear selected in the Properties window. Although not all XAML controls on a page need a name, you do need to specify one if you plan to use the control in Visual Basic program code. Setting the *Name* property for this text box will give you something that you can make reference to later.

4. Type **NewURL** and press Enter.

   As you name the objects on a page, it is useful to follow some basic naming conventions that make the objects easy to recognize in your code-behind file. In this case, I've specified *NewURL* because the object will hold the new web address or URL that the user of the program wants to browse to. Programmers sometimes name objects according to numeric patterns as well, such as *TextBox1*, *TextBox2*, and so on.

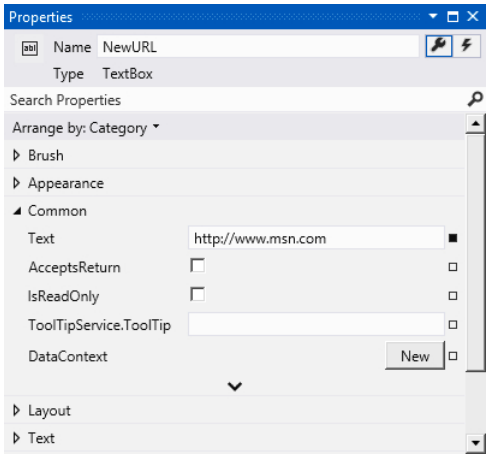
   Now you'll put a sample URL in the text box to show users the pattern that you want them to follow.

5. In the Common category, click the *Text* property, type **http://www.msn.com**, and press Enter.

   The *Text* property holds the text that is currently displayed in the text box object on the page, and if you look in the Designer window now, you'll see that the default text has changed from "TextBox" to "http://www.msn.com."

   A text box object can contain one or more lines of text. You can modify the text that appears in a text box by using the Properties window, by directly editing the XAML markup associated with the text box in the Code Editor, or by modifying the *Text* property of the text box in the Visual Basic code-behind file so that the text box changes while the program is running.

   The following screen shot shows what the Properties window looks like after you have set the *Name* and *Text* properties. It shows the window slightly expanded and floating over the IDE, which is probably a good way to use the tool when you first start using it. Once you get the hang of it, however, you'll find it easiest to use the Properties window in its docked position.



Now you'll change the *Name* property for the second text box object on the page.

6. Click the second text box object, and use the Properties window to change the *Name* property of the text box to **AllSites**.

   You'll give the larger text box this name because you'll use it to list all of the web sites that the user visits while the program runs. The text box should be big enough to list up to a dozen web sites.

7. Click the *Text* property for the *AllSites* text box, delete the text that is there, and then press Enter.

   The Properties window removes "TextBox" from the *AllSites* text box in the Designer. This is done to prevent text from being displayed in the text box when the program starts.

8. In the Common category, click the *IsReadOnly* check box.

   You'll see a check mark in the check box, indicating that the *IsReadOnly* property has been set to *True*. This setting will prevent the user from editing content in the *AllSites* text box while the program is running, although they can still copy information from the text box to the Clipboard by selecting text in the object with the mouse and pressing Ctrl+C.

Now you'll set a property for the button object in the program.

Set the Content property of the button object

1. Click the button object on the page.

   The button will be selected and surrounded by resize handles.

   The XAML *Button* control uses the *Content* property to store the text that is displayed on a button, so you'll edit that property now. The text that is currently displayed is "Button," but you'll change it to "Visit Web Site" to make the element more descriptive.

2. Use the Properties window to change the *Content* property of the button object to **Visit Web Site**.

   Once you make the change, the text is updated in the Properties window, in the button object on the page, and in the XAML markup in the Code Editor.

Now you'll set a few properties for the web view object in the program.

Set the properties of the web view object

1. Click the web view object on the page.

   The XAML *WebView* control is a no-frills web browser that allows you to quickly display web-page content in a Windows 8 application. It is useful because you can't easily start Internet Explorer or another web browser from within a Windows 8 program.

   Adding direct access to the web from a Windows application is an exciting feature. You'll update the *Name* property of the web view object now so that you can use this interesting tool in Visual Basic code.

2. Use the Properties window to change the *Name* property to **WebView1**.

Tip While you are working with the web view object, you might notice that in the Designer window, the object displays the following message: "This element is enabled only when the application is running." This means that the web browser will not work while you are creating your application, so you can't preview how web content will appear until you actually run the program. Not to worry—testing how your program works is part of the overall development process, which I'll discuss later in the book.

   Now you'll center the horizontal and vertical alignment of the web content so that the user can see it clearly within the application window that you have designed.

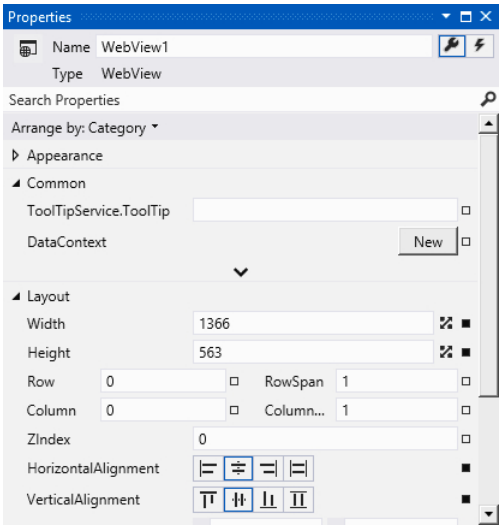
3. In the Properties window, expand the Layout category and scroll down a bit so that you can see the *HorizontalAlignment* and *VerticalAlignment* properties.

   These properties control how content is aligned within the web view object. The default is left for *HorizontalAlignment* and top for *VerticalAlignment*. However, you'll want to specify center alignment for both properties. You change these values in the Properties window by clicking one of the four alignment buttons, each of which contains a visual representation of the alignment.

4. Click the center-alignment button for the *HorizontalAlignment* property.

5. Click the center-alignment button for the *VerticalAlignment* property.

Your Properties window will look like this:



Congratulations! You are finished setting the properties for the Web List program. Now you'll write a few lines of Visual Basic program code to navigate to web sites as needed, and to keep track of the web sites that the user has visited.

Reading Properties in Tables

In this chapter, you've set the properties for the Web List program one step at a time. In future chapters, the instructions to set properties will be presented in table format unless a setting is especially tricky. Table 2-1 lists the properties you've set so far in the Web List program, as they will look later in the book. Settings you need to type in are shown in quotation marks. ("" means that you should delete the text currently in the property setting.) You shouldn't type the quotation marks.

TABLE 2-1 Web List Properties

Object	Property	Setting
Text box 1	*Name* *Text*	NewURL "http://www.msn.com"
Text box 2	*Name* *Text* *IsReadOnly*	AllSites "" True
Button 1	*Caption*	"Visit Web Site"
Web view	*Name* *HorizontalAlignment* *VerticalAlignment*	WebView1 Center Center

Writing the Visual Basic Code

Now you're ready to write the code for the My Web List program. Because most of the objects you've created already "know" how to work when the program runs, they're ready to receive input from the user and process it. The inherent functionality of objects is one of the great strengths of Visual Studio and Visual Basic—after XAML objects are placed on a page and their properties are set, they're ready to run. However, the core of the My Web List program—the code that starts the web browser and copies each web site that is visited to a text box—is still missing. This computing logic can only be built into the application by using program statements—code that clearly spells out what the program should do at each step of the way. Because the Visit Web Site button drives the program, you'll create an event handler that runs, or *fires*, when the user clicks this button. The event handler will be created using Visual Basic code in a file that is connected to the page you just built.

In the following steps, you'll create an event handler for a button click event using the Code Editor.

Use the Code Editor to create an event handler

1. In the Designer window, click the button object.
2. Open the Properties window, and next to the *Name* property text box, click the Event Handlers button (a square button displaying a lightning bolt icon).

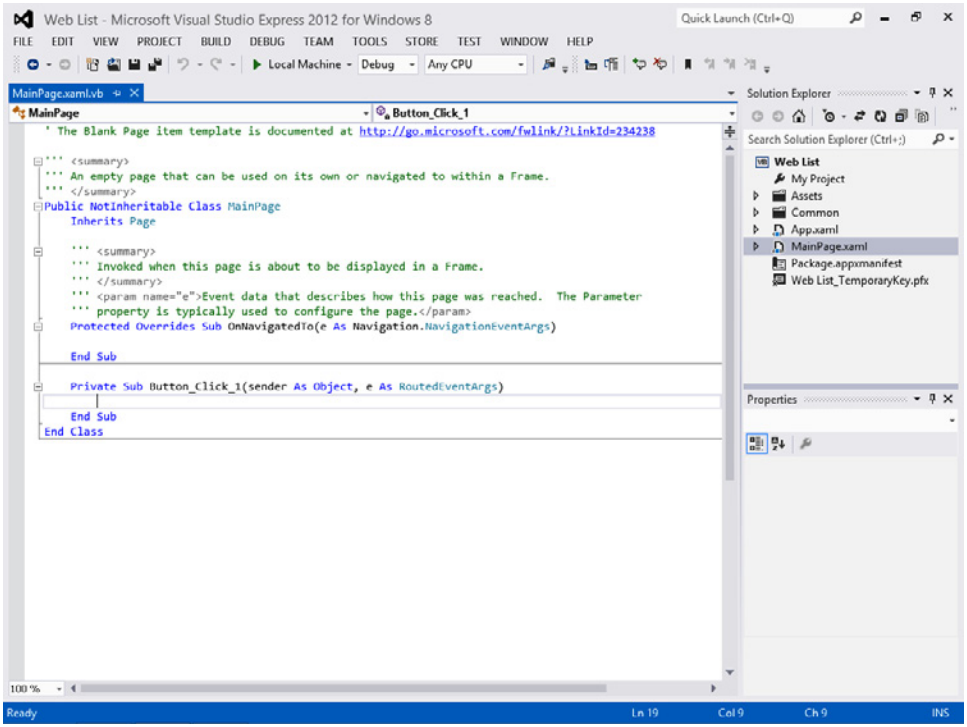
   A long list of events that the button object can detect fills the Properties window. Typical events that a button object might respond to include *Click* (a mouse click), *DoubleClick* (two mouse clicks in quick succession), *DragOver* (an object being dragged over a button), and *Drop* (an object being dragged over and dropped on a button). Since Visual Basic is at its core an event-driven programming language, much of what you do as a software developer is create user interfaces that respond to various types of input from the user, and then you write event handlers that manage the input.

   Most of the time, you will only need to write event handlers for a few events associated with the objects in your programs. (The list of events is quite comprehensive, however, to give you many options.)

   To create an event handler for a particular event, you double-click the text box next to the event in the Properties window. Since you want to load a webpage each time that the user clicks the button in your program, you'll write an event handler for the button's *Click* event.

3. Double-click the text box next to the *Click* event in the Properties window.

   Visual Studio inserts an event handler named *Button_Click_1* in the *Click* text box, and opens the *MainPage.xaml.vb* code-behind file in the Code Editor. Your screen should look like this:



Inside the Code Editor are program statements associated with the *MainPage* template that you opened when you started this project. This is Visual Basic program code, and you may notice right away that some of the code is organized into concise units, known as *procedures*. There is a procedure called *OnNavigatedTo*, and there is a new event handler procedure that you just created called *Button_Click_1*.

The *Sub* and *End Sub* keywords designate a procedure, and the keywords *Protected* and *Private* indicate how the procedure will be used. You'll learn more about these keywords later.

When you double-clicked the *Click* text box in the Properties window, Visual Studio automatically added the first and last lines of the *Button_Click_1* event procedure, as the following code shows:

```
Private Sub Button_Click_1(sender As Object, e As RoutedEventArgs)

End Sub
```

The body of a procedure fits between these lines and is executed whenever a user activates the interface element associated with the procedure. In this case, the event is a mouse click, but as you'll see later in the book, it could also be a different type of event. Programmers refer to this sequence as "triggering an event."

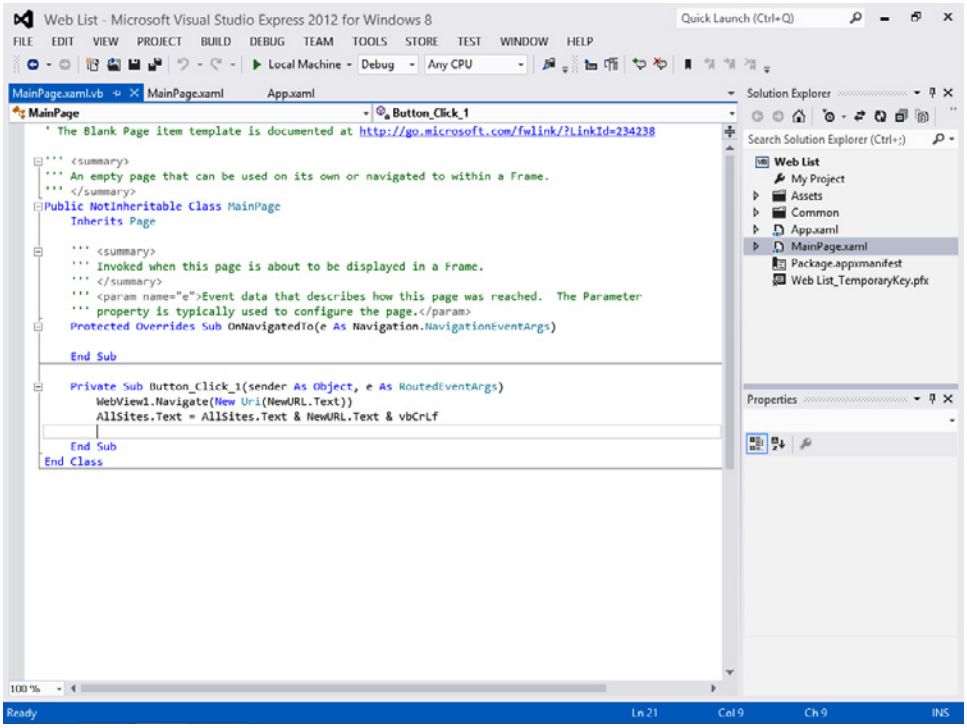
Tip You may also notice lines of text highlighted with green type in the Code Editor. In the default settings, green type indicates that the text is a *comment*, or an explanatory note written by the creator of the program, so that it might be better understood or used by others. The Visual Basic compiler does not execute, or *evaluate*, program comments.

4. Type the following program code and press the Enter key after the last line:

```
WebView1.Navigate(New Uri(NewURL.Text))
AllSites.Text = AllSites.Text & NewURL.Text & vbCrLf
```

As you enter the program code, Visual Studio formats the text and displays different parts of the code in color to help you identify the various elements. When you begin to type the name of an object property, Visual Basic also displays the available properties for the object you're using in a list box, so you can double-click the property or keep typing to enter it yourself.

Your screen should now look like this:



If Visual Basic displays an additional error message, you might have misspelled a program statement. Check the offending line against the text in this book, make the necessary correction, and continue typing. (You can also delete a line and type it again from scratch.)

Program statements are a little like complete sentences in a human language—statements can be of varying lengths but must follow the grammatical rules of the language. In Visual Studio, program statements can be composed of keywords, properties, object names, variables, numbers, special symbols, and other values. As you enter these items in the Code Editor, Visual Studio uses a feature known as IntelliSense to help you write the code. With IntelliSense, as Visual Studio recognizes language elements, it will automatically complete many expressions.

More Info You'll learn more about Visual Basic language fundamentals in Chapter 6, "Visual Basic Language Elements."

5. Click the Save All command on the File menu to save your additions to the program.

   The Save All command saves everything in your project—the project file, the pages, the code-behind files, the assets, the package manifest, and other related components in your application. Since this is the first time that you have saved your project, the Save Project dialog box opens, prompting you for the name and location of the project. (If your copy of Visual Studio is configured to prompt you for a location when you first create your project, you won't see the Save Project dialog box now—Visual Studio just saves your changes.)

6. Browse and select a location for your files.

   I recommend that you use the *My Documents\Start Here! Programming in Visual Basic\Chapter 02* folder (the location of the book's sample files), but the location is up to you. Since you used the "My" prefix when you originally opened your project, this version won't overwrite the practice file that I built for you on disk.

7. Clear the Create Directory For Solution check box.

   When this check box is selected, it creates a second folder for your program's solution files, which is not necessary for solutions that contain only one project (the situation for most programs in this book).

8. Click Save to save your files.

   Notice that the object names in the Code Editor (*WebView1*, *NewURL*, and *AllSites*) are now displayed in normal type.

Tip If you want to save just the item you are currently working on (the page, the code module, or something else), you can use the Save command on the File menu. If you want to save the current item with a different name, you can use the Save As command.

A Look at the Visual Basic Code-Behind File

The *Button_Click_1* event handler is executed when the user clicks the Visit Web Site button on the page. The procedure uses some interesting Visual Basic code, which is worth looking at before moving on.

The first statement uses the *Navigate* method of the *WebView1* object to load a webpage into the window you created earlier in this project:

```
WebView1.Navigate(New Uri(NewURL.Text))
```

Navigate is a *method*, or a statement that performs a specific action for an object in your program. The web view object has numerous methods, but the *Navigate* method is the one that prompts web view to load a webpage. Right now, you should notice that the *Navigate* method is connected to the *WebView1* object by a period (.), which is the same syntax that is used to reference a property in program code.

In parentheses following the *Navigate* method is a reference to the text that the user has entered into the first text box on the page (New*URL*). The text is stored in the *Text* property, and the *New Uri* keywords are used to put the user input into a standard format used for web addresses—the so-called *uniform resource indicator (URI)* format. Since this is a simple demonstration program, I am assuming that the user is entering the web address in the proper way. In fact, if an incorrect or badly formatted web address is entered, the program will not load the webpage and there will be little indication that something went wrong. This is not what you would do in a commercial application, of course, and I'll show you later in the book how to be much more deliberate about handling errors introduced by the user.

The second statement builds the list of web sites that the user visits while the program runs:

```
AllSites.Text = AllSites.Text & NewURL.Text & vbCrLf
```

Each web site is entered through the *Text* property of the *NewURL* object. This *Text* property is combined with the current contents of the *AllSites* text box through the string-concatenation operator (*&*), which appends each new web site that is entered to the bottom of the list. A line break is added to the end of each line by the *vbCrLf* constant. You'll learn more about the string-concatenation operator when you learn about how Visual Basic computes mathematical and textual operations in Chapter 6.

With just two lines of Visual Basic code, your program is complete. Now you are ready to run the application.

Running Visual Basic Applications

To run a Visual Basic program from the development environment, you can do any of the following:

- Click Start Debugging on the Debug menu.
- Click the Start Debugging (Local Machine) button on the Standard toolbar.
- Press F5.

Try running the My Web List program now. If Visual Studio has difficulty compiling your program or displays an error message, you might have made a typing mistake or two in your program code. Try to fix it by comparing the printed version in this book with the one you typed, or load Web List from this book's sample files and run it.

Tip When you run the My Web List program, you will use the WebView control to display live information from the Internet in your application. By default, if Visual Studio encounters any type of error when loading web pages into the WebView control, it will display an error message in a dialog box entitled "Visual Studio Just-In-Time Debugger." If you click "Yes" in this dialog box, you will enter debugging mode (or break mode) and be able to learn more about the error message. If you click "No", you can suppress the error message and keep running the My Web List program.

Internet script errors can occur for a variety of reasons on the web. Most of these errors are simple warning messages that are not too serious. To suppress Just-In-Time script debugging for now, while you are still getting your feet wet with Visual Studio, I recommend that you click Tools | Options | Debugging | Just-In-Time, remove the check mark from the Script check box, and then click OK. After you complete this chapter, you can restore this setting.

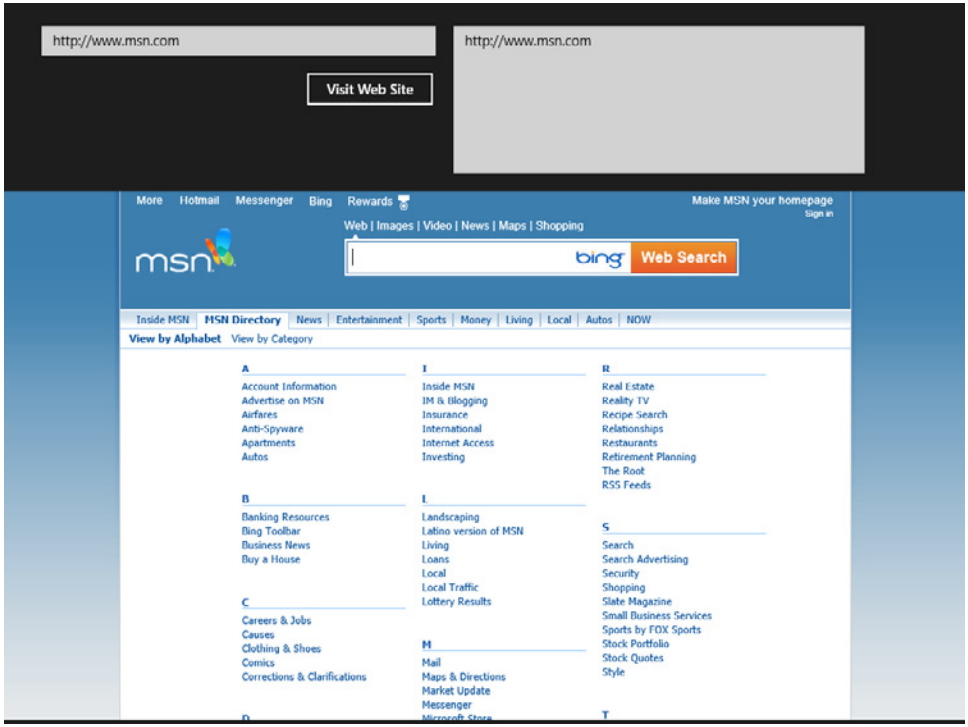
Run the My Web List program

1. Click the Start Debugging button on the Standard toolbar (the green arrow button with the words "Local Machine" next to it).

   The My Web List program will compile and run in the IDE. After a few seconds, the user interface appears, just as you designed it. The Microsoft MSN web site (*http://www.msn.com*) appears in the first text box as a sample web site you can browse to.

2. Click the Visit Web Site button.

   The program uses the *Navigate* method of the *WebView1* object to access the site. The web site appears in the web view object, and the web site URL appears in the second text box on the page. Your screen will look like this:



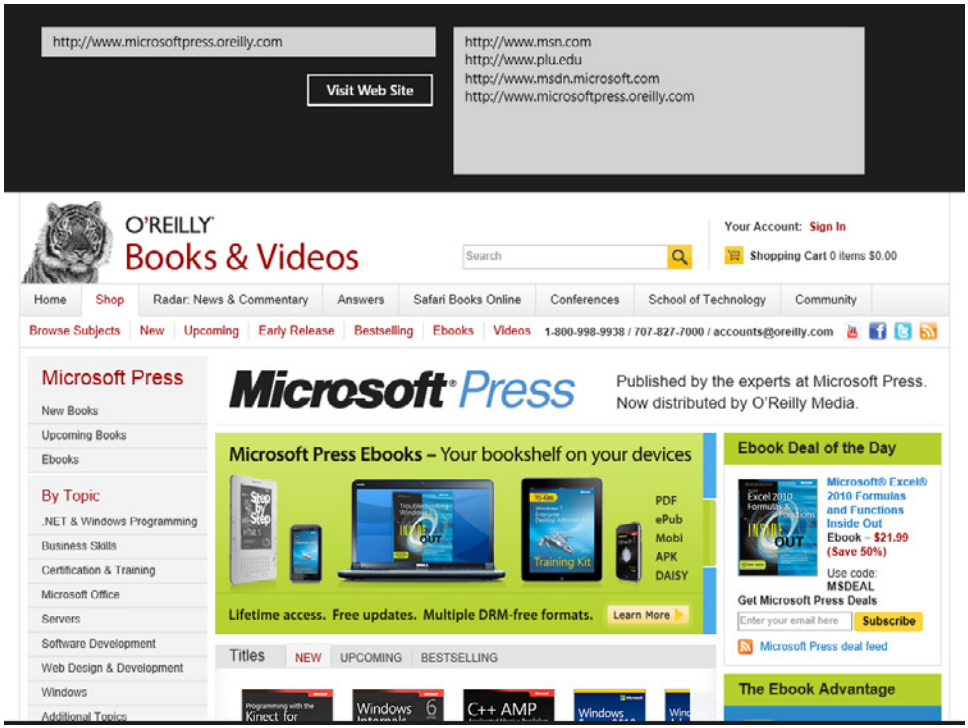
3. Enter a new URL in the first text box, such as **http://www.plu.edu**, and then click Visit Web Site. (This is the university where I teach, but you can substitute your own favorite web address.)

4. Visual Basic immediately adds the web site to the list of visited sites and loads the webpage into the web browser. Keep in mind that the content in the web browser is live—you can click around within the webpages and move from link to link as you would on a normal webpage. To see webpage content that is not currently visible, simply press the Down Arrow key or rotate the mouse wheel.

5. Enter a third URL in the text box, such as **http://msdn.microsoft.com**, and then click Visit Web Site.

6. Enter a fourth URL in the text box, such as **http://www.microsoftpress.oreilly.com**, and then click Visit Web Site.

Your screen will look like this:



Now that you've entered four or five web sites, you can begin to see the value of the web site address list that is slowly accumulating in the second text box. Visual Basic offers you the power to track all types of information, including a list of the web sites that you have visited. If you would like to save the list for future use, simply select the contents of the text box with your mouse, press Ctrl+C to copy the list to the Clipboard, open an application such as Notepad or Microsoft Word, and then press Ctrl+V to paste the list into the open document.

When you're finished experimenting with the Web List program, close the application. You've just tested your first Windows application!

Sample Projects on Disk

If you didn't build the My Web List project from scratch (or if you did build the project and want to compare what you created to what I built as I wrote the chapter), take a moment to open and run the completed Web List project now, which is located in the *My Documents\Start Here! Programming in Visual Basic\Chapter 02* folder on your hard disk (the default location for the practice files for this chapter). If you need a refresher course on opening projects, see the detailed instructions in Chapter 1. If you are asked if you want to save changes to the My Web List project, be sure to click Save.

Note The *Start Here!* programming series is designed to be a hands-on learning experience, so you will benefit most from building the projects on your own as you read this book. But after you have completed the projects, it is often a good idea to compare what you have with the practice file solution that I provide, especially if you run into trouble. To make this easy, I will give you the name of the solution files on disk before you run the completed program in most of the exercises that follow.

After you have compared the My Web List project to the Web List solution files on disk, reopen My Web List and prepare to compile it as an executable file. If you didn't create My Web List, use the book's solution file to complete the exercise.

Building an Executable File and Deploying

Your last task in this chapter is to complete the development process and create an application for Windows, or an executable file. Windows applications created with Visual Studio have the file name extension *.exe* and can be run on any system that contains Windows and the necessary support files. Visual Studio installs the support files that you need when you deploy a completed project—including the .NET Framework files—automatically.

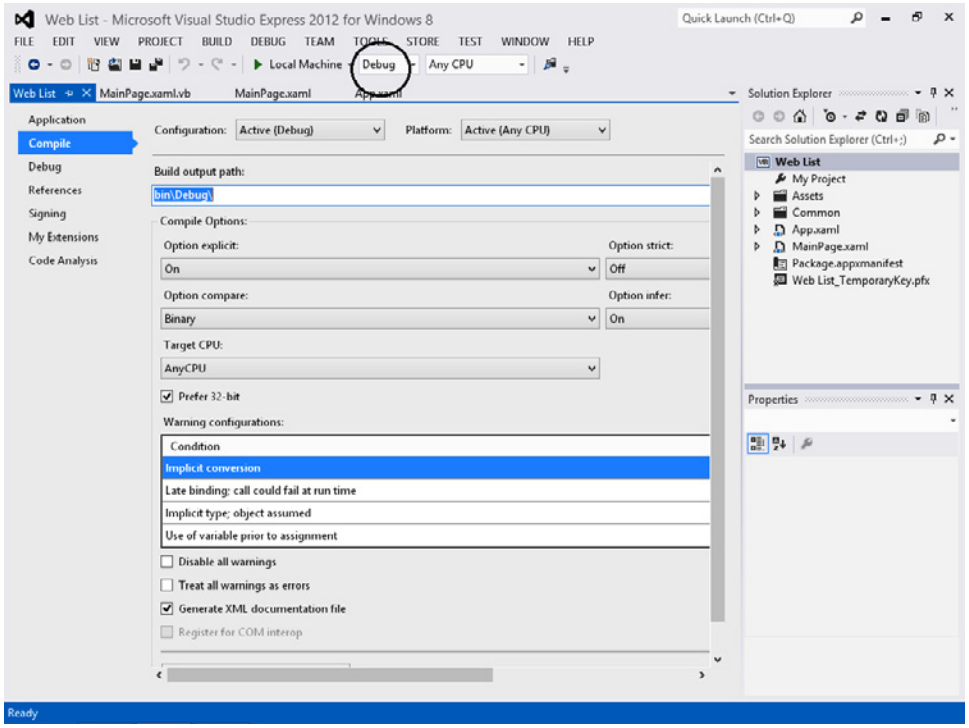
Since you are creating an application for the Windows 8 user interface, you will need to deploy this program on a computer running Windows 8, because your application is designed for that environment. Chapter 12, "Future Development Opportunities and the Windows Store" introduces you to the Windows Store, an online purchasing and distribution system that allows Visual Studio programmers to sell their Windows 8 applications to customers around the world.

Before you prepare your app for the Windows Store, however, you need to know a little more about programming, and also a little more about how applications are compiled and tested. When Visual Studio programmers complete the initial design and functionality of their application, they typically test their program systematically to verify that the code works as expected under a variety of operating conditions. Often, more than one developer, or *tester*, is involved in the process, and they typically use a variety of machines, operating systems, and computing scenarios to test the seaworthiness of the application. If you examine the Build, Debug, and Test menus in the Visual Studio IDE, you'll begin to see how elaborate this process can actually be.

To assist in the testing and compilation process, Visual Studio allows you to create two types of executable files for your Windows application project: a *debug build* and a *release build*.

Debug builds are created automatically by Visual Studio when you create and test your program. They are stored in a folder called *bin\Debug* within your project folder. The debug executable file contains debugging information that makes the program run slightly slower.

Release builds are optimized executable files stored in the *bin\Release* folder within your project. To customize the settings for your release build, you click the [*ProjectName*] Properties command on the Project menu, and then click the Compile tab, where you'll see a list of compilation options that looks like the following screen. The Solution Configurations drop-down list box on the Standard Visual Studio toolbar (circled in the following image) indicates whether the executable is a debug build or a release build. If you change the Solution Configurations setting, the path in the Build Output Path text box will also change.



The process of preparing an executable file for a specific computer is called *deploying* the application. As noted, when you deploy an application with Visual Studio, the IDE handles the process of copying all the executable and support files that you will need to register the program with the operating system and run it. Visual Studio allows you to deploy applications *locally* (on the computer you are using) or *remotely* (on a computer attached to the network or Internet).

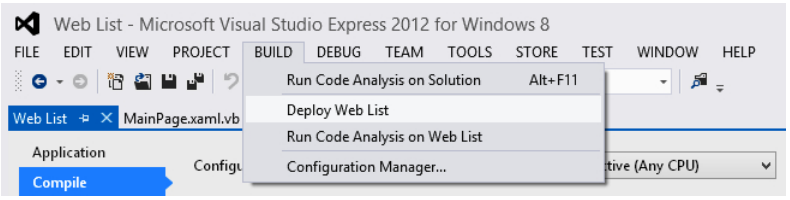
In the following exercise, you'll deploy a release build for the My Web List application locally and create an application icon for the program on the Windows Start page. In Chapter 12, you'll learn more about packaging applications that have been tested and prepared for the Windows Store.

Deploy a release build for a Windows 8 application

1. Click the Solution Configurations drop-down list box on the Standard toolbar, and then click the Release option.

   Visual Studio will prepare your project for a release build, with the debugging information removed. The build output path is set to *bin\Release*.

2. On the Build menu, click the Deploy My Web List command.



   The Build command creates a *bin\Release* folder in which to store your project (if the folder doesn't already exist) and compiles the source code in your project. The Output window appears to show you milestones in the assembly and deployment process. The result is an executable file named *My Web List.exe*, which Visual Studio registers with the operating system on your computer.

   Visual Studio deploys the application locally because Local Machine is currently selected on the toolbar next to the Start button. This is the desired behavior here, but you can also deploy applications on a remote machine (i.e., a computer attached to yours via a network or the Internet) by selecting the Remote Machine option. If you select this option, you'll be presented with a dialog box asking for more information about the remote connection. Keep in mind that remote deploying is mostly designed for testing purposes. The best way to install completed applications via the Internet is through the Windows Store.

   When you deploy an application built for the Windows 8 user interface, Windows automatically creates a new program icon for the application on the Start page. You can use this icon to launch the program whenever you want to run it. Try running My Web List now from the Windows Start page on your computer.

3. Open the Windows Start page, and browse to the list of applications that are currently installed. (The most recent applications are typically located on the right side of the Start page.)

My Windows Start page currently looks like this:

4. Click the My Web List application icon.

   The My Web List program will load and run in Windows.

5. Test the application again, browsing to several web sites. When you are finished, close the application.

6. Return to Visual Studio, and close the Output window and the My Web List properties page. Note that you can view and change compilation options whenever you want—the properties page is always there.

7. On the File menu, click Exit to close Visual Studio and the My Web List project.

8. Click Save if you are prompted to, and the Visual Studio development environment will close.

Congratulations on completing your first Windows 8 application!

Tip Would you like to add to a Visual Basic program after you've finished working with it? Simply restart Visual Studio and check the Recent Projects pane on the Start page. The last few projects that were open in Visual Studio will be listed there, and you can click the project and pick up where you left off. If the project is not there (if you worked on it some time ago), click the Open Project command on the File menu and browse your computer's hard disk to find it. Most programmers edit and revise their projects over a period of days, weeks, and months. It is very simple to add to an existing Visual Basic project or to move components from one project to the next.

Summary

This chapter described how to create a Windows 8 application by using Visual Studio and the Visual Basic programming language. The process is very straightforward conceptually. First, you create a user interface on an application page by using controls from the XAML Toolbox. The XAML Toolbox offers numerous controls for user interface features that have been optimized for use in Windows 8. The Designer window allows you to place and resize the controls on the application page so that they look good and take on the design of the Windows 8 user interface. Once a Toolbox control is placed on a page, it is referred to as an object.

The next step is adjusting the property settings for one or more objects by using the Properties window. Before you can adjust the properties for an object, you must select the object in the Designer window. Once an object is selected, its property settings fill the Properties window, and you can adjust them by clicking, typing, and selecting values from list boxes. In the Web List project you created in this chapter, you adjusted property settings for the text box, button, and web view objects.

The third step is creating event handlers for the objects in your program that are manipulated in some way by the user. Event handlers are written in the Code Editor with Visual Basic program code. This Visual Basic code, sometimes referred to as a Visual Basic code-behind file, follows the syntax rules of the Visual Basic programming language and connects the developer to the power and functionality of the Windows operating system. Event handlers and other Visual Basic routines are considered the core of a Visual Studio program; they process information, calculate values, set object properties, and use object methods. In the My Web List project, you created the *Button_Click_1* event handler, which opens a webpage and adds the current web site to a text box when the user clicks the Visit Web Site button.

After the user interface and code-behind file for a project are complete, the application is ready to be tested against a variety of operating conditions. When you are finished testing your project, you can deploy it locally or remotely, which involves compiling the project into an executable file and registering it with the operating system. Finally, you can package your finished application and upload it to the Windows Store for global sales and distribution, a process that will be discussed more fully in Chapter 12.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Z

About the Author

MICHAEL HALVORSON is the author or co-author of more than 35 books, including *Microsoft Visual Basic 2010 Step by Step*, *Learn Microsoft Visual Basic Now*, and *Microsoft Visual Basic 6.0 Professional Step by Step*. Halvorson has been the recipient of numerous nonfiction writing awards, including the Computer Press Best How-to Book Award (Software category) and the Society for Technical Communication Excellence Award (Writing category). Halvorson earned a bachelor's degree in Computer Science from Pacific Lutheran University and master's and doctoral degrees in History from the University of Washington. He was employed at Microsoft Corporation from 1985 to 1993, and has been an advocate for Visual Basic programming since the product's original debut at Windows World in 1991. Halvorson is currently an associate professor at Pacific Lutheran University. You can learn more about his books and ideas at *http://www.michaelhalvorsonbooks.com*.