PART I

Integrated Development Environment

- **CHAPTER 1:** A Quick Tour
- **CHAPTER 2:** The Solution Explorer, Toolbox, and Properties
- **CHAPTER 3:** Options and Customizations
- **CHAPTER 4:** The Visual Studio Workspace
- **CHAPTER 5:** Find and Replace and Help

1

A Quick Tour

WHAT'S IN THIS CHAPTER?

- Installing and getting started with Visual Studio 2017
- Creating and running your first application
- Debugging and deploying an application

Ever since software has been developed, there has been a need for tools to help write, compile, debug, and deploy applications. Microsoft Visual Studio 2017 is the next iteration in the continual evolution of a best-of-breed integrated development environment (IDE).

This chapter introduces the Visual Studio 2017 user experience and shows you how to work with the various menus, toolbars, and windows. It serves as a quick tour of the IDE, and as such it doesn't go into detail about what settings can be changed or how to go about customizing the layout because these topics are explored in the following chapters.

GETTING STARTED

Recent versions of Visual Studio have seen incremental improvements in the installation experience. However, Visual Studio 2017 has pretty much completely revamped the installation options and workflow. It has been designed to not only get you up and running quickly, but also to easily select only those options you need to have installed. This section walks you through the installation process and getting started with the IDE.

Installing Visual Studio 2017

The installer for Visual Studio 2017 is what Microsoft calls a "low-impact installer." The idea arose as Microsoft compared the footprint used by Visual Studio 2015 with the kinds of experiences that users were not only requesting, but also using. As surprising as it might seem, not

every developer needs to have Visual Studio support for Windows Forms, ASP.NET, WPF, Universal Apps, and C++ out of the box.

Visual Studio 2015 and earlier versions were optimized so that pressing F5 to run a program would work out of the box. It wasn't expected that you would need to install any other components in order to get the large majority of .NET applications running. While this was a definite plus regarding ease of use, it made for a large (some might say bloated) footprint for Visual Studio.

In Visual Studio 2017, the installation process takes a different point of view. Instead of automatically installing "everything," you get to pick and choose the different components that you want to install. Yes, you had a little bit of that in the past, but now the number of options that you have is greatly increased. However, more options doesn't necessarily mean a better installation experience. In fact, it's probably the opposite, as you try to figure out which of a hundred different options you need to install to work on your project. To address that challenge, the Visual Studio 2017 installer uses the concept of workloads.

When you launch the Visual Studio 2017 installation process (an application of only a couple of megabytes in size), you'll see the dialog in Figure 1-1 appear relatively quickly. Naturally, this is after you have read (in great detail, of course) and accepted the licensing information and privacy statements.

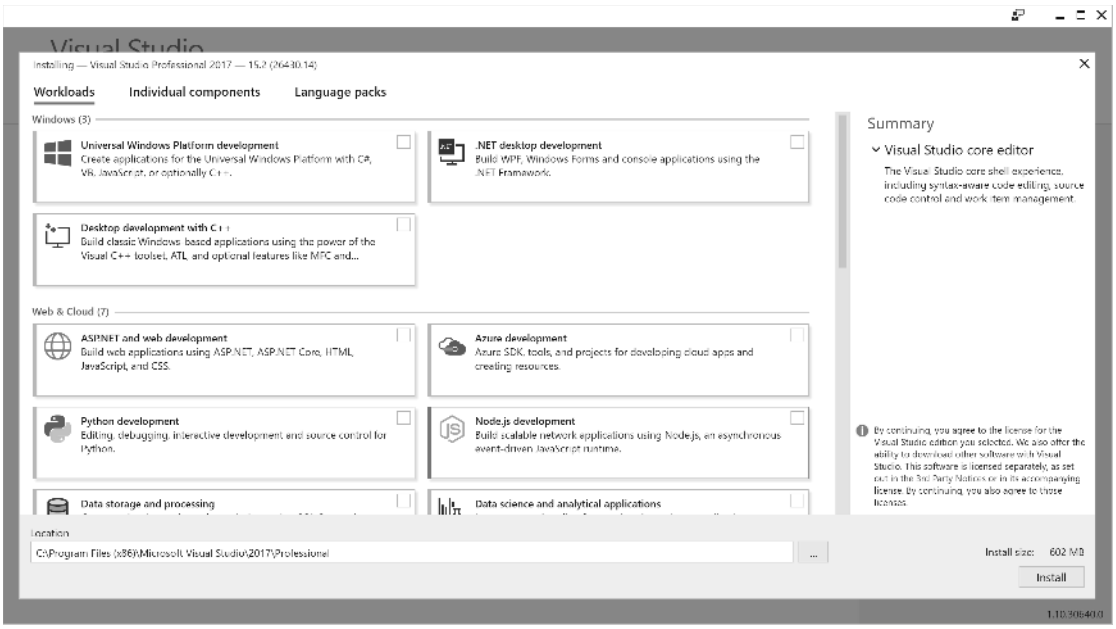

FIGURE 1-1

This is the main hub for the installer and the location where the desired components are specified. There are two modes for identifying the components. In Figure 1-1, you are looking at the workloads. The workloads have been divided into five different categories. To include a workload in the

installation, simply click on it, showing a blue checkbox in the top right corner. You can add any number of the workloads as part of the installation. The available workloads are:

- **Universal Windows Platform development:** Used if you are creating applications for the Universal Windows Platform, regardless of your language of choice.
- **.NET desktop development:** Allows you to create applications, using either WPF or Windows Forms. This is also where you find the Console application template.
- **Desktop development with C++:** Used to build classic Windows-based applications. This option is appropriate if you expect to be using Visual C++, the Active Template Library (ATL), or Microsoft Foundation Classes (MFC).
- **ASP.NET and web development:** Adds the components used to build web applications, including ASP.NET, ASP.NET Core, and plain old HTML/Javascript/CSS.
- **Azure development:** Includes the Azure SDK, tools, and project templates that allow you to create Azure-based cloud applications.
- **Python development:** Includes support for cookiecutter, Python 3, and tools that are used to interact with Azure. And, optionally, you can include other distributions of Python, such as Anaconda.
- **Node.js development:** One of the new tools supported by Visual Studio 2017, this workload includes the components that allow you to create network applications using the Node.js platform.
- **Data storage and processing:** Some recent additions to the Azure platform include Azure Data Lake, Hadoop, and Azure ML (Machine Learning). This workload includes the templates and tools to develop applications for this platform, along with the Azure SQL Server database.
- **Data science and analytical applications:** Brings together three languages that are also found in other workloads: R, Python, and F#. These tools can be used to build a wide variety of analytics-based applications.
- **Office/SharePoint development:** Used to create a wide variety of Office and SharePoint applications, including Office add-ins, SharePoint solutions, and Visual Studio Tools for Office (VSTO) add-ins.
- **Mobile development with .NET:** One of the three technologies that Visual Studio 2017 supports for mobile development, this workload allows you to create iOS, Android, or Windows applications using Xamarin.
- **Mobile development with JavaScript:** Similar concept to the previous entry, but instead of using Xamarin, your applications are developed using Tools for Apache Cordova and JavaScript.
- **Mobile development with C++:** And the last of the three mobile development environments allows you to create iOS, Android, and Windows applications using C++.
- **Game development with Unity:** Unity is a broadly used and very flexible cross-platform game development environment. This workload allows you to create 2D and 3D games using the Unity framework.

- **Game development with C++:** Supports the creation of games using C++ along with libraries like DirectX, Unreal, or Cocos2d.
- **Visual Studio extension development:** Lets you create add-ons and extensions for use in Visual Studio. Included in this are code analyzers and tool windows that take advantage of the Roslyn compiler functionality.
- **Linux development with C++:** Windows 10 includes an option to install an Ubuntu-based Linux Bash shell. This workload includes the set of tools and libraries that allow you to create applications that run in Linux using Visual Studio.
- **.NET Core cross-platform development:** .NET Core is a popular approach to cross-platform development. This workload allows you to create .NET Core applications, including web applications.

WORKLOADS USED IN THIS BOOK

In order to work through the examples in the book, there are a number of workloads that need to be installed. Specifically:

- Universal Windows Platform
- .NET desktop development
- ASP.NET and web development
- Azure development
- Node.js development
- Data storage and processing
- Data science and analytical applications
- Mobile development with .NET
- Mobile development with Javascript
- .NET code cross-platform development

The second mode for choosing components is more granular. If you select the Individual Components link at the top of the installation screen, the list of components shown in Figure 1-2 appears. From here you can select any or all of the individual components that you want to install on your machine.

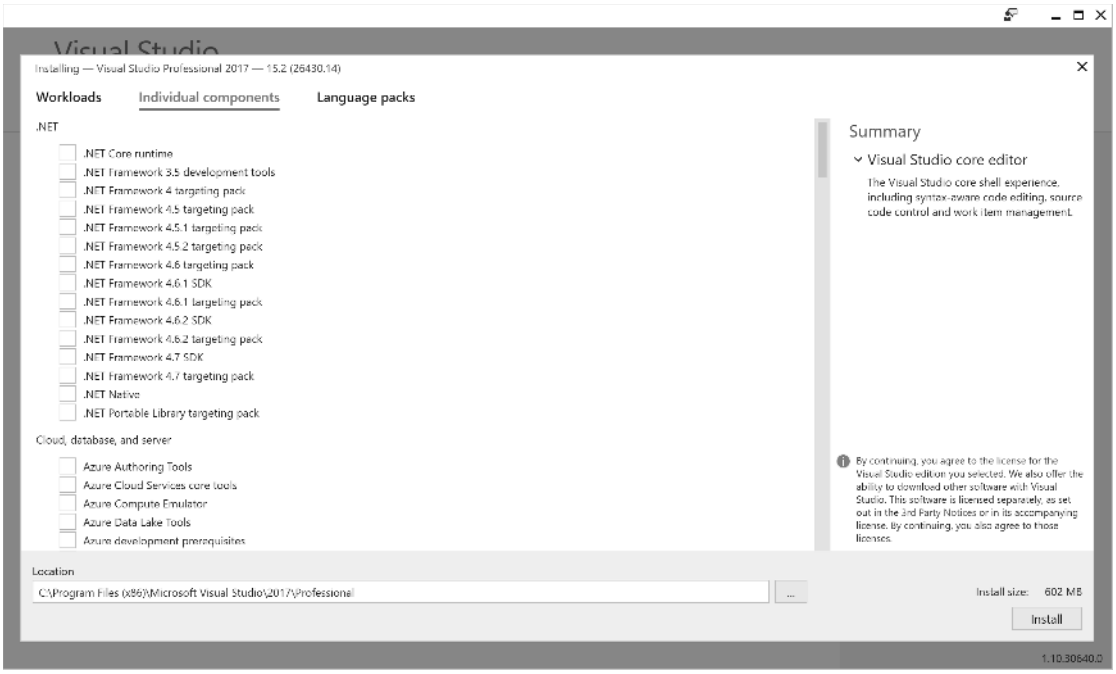

FIGURE 1-2

> **NOTE** *To see the relationship between the workloads and the more granular components that are included, simply select a workload. A list of the components that are included appears in the pane on the right side of the dialog.*

The third installation option you have for Visual Studio is to include one or more of the supported language packs. Clicking on the Language Packs link displays the list of language packs that are available (see Figure 1-3).

Once you have selected your components (either individually or as part of a workload), choose the installation location and click on Install. Now comes the longer part of the process. You'll see the progress dialog, an example of which is shown in Figure 1-4. Depending on which components you already have installed on your computer, you might be prompted to restart your computer midway through or at the end of the installation process. When all the components have been successfully installed, the original dialog changes slightly (as shown in Figure 1-5). This final dialog is also the starting point for adding features to Visual Studio in the future.

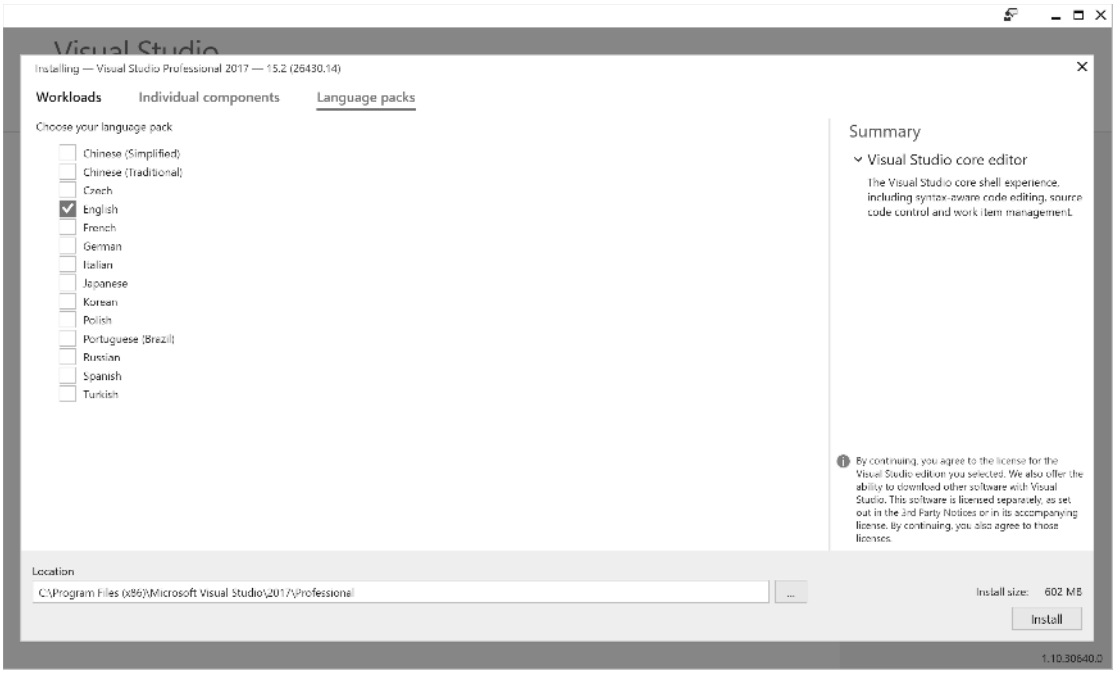

FIGURE 1-3

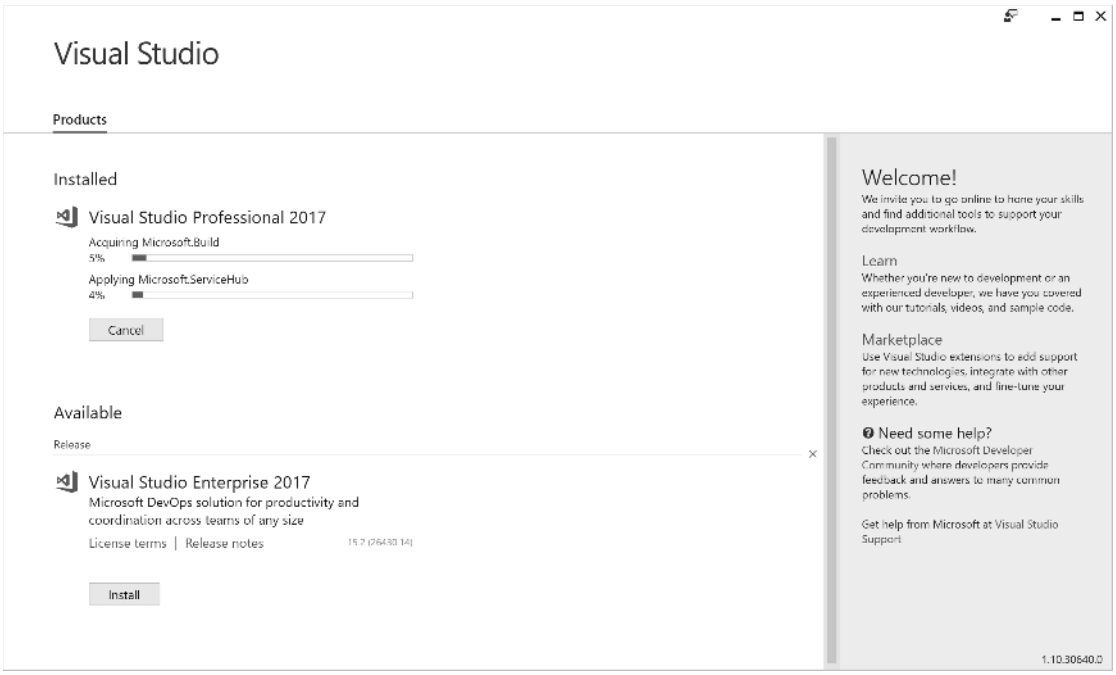

FIGURE 1-4

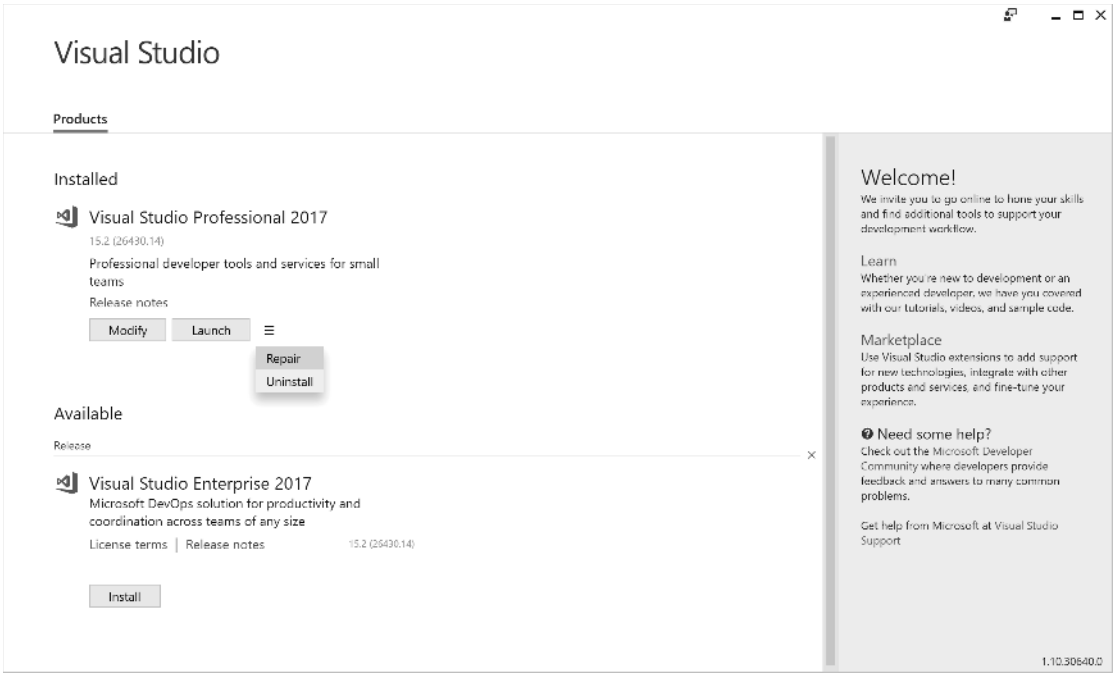

FIGURE 1-5

Running Visual Studio 2017

The first time you run Visual Studio 2017, you might be given the opportunity to sign in. If you had already signed in from within Visual Studio 2015, you won't be prompted to log in. Your credentials are remembered between the versions. However, if you're new to Visual Studio, then you'll be asked to provide a Microsoft Live account.

This behavior is part of an effort to cloud-enable Visual Studio—to connect Visual Studio settings and functionality to assets that are available across the Internet. There is no requirement for you to log in. The login page includes a Not Now, Maybe Later link. Clicking on that link skips a number of steps and lets you get to Visual Studio quickly. But there are some decent benefits that can be derived by signing in.

Is Visual Studio Really Cloud Enabled?

The quick answer is "Yes." A more accurate answer is "Yes, if you want it to be." Part of the research work behind creating this feature involved Microsoft gaining an understanding of how developers identified themselves to various online functions. In general, most developers have two or more Microsoft accounts that they use when they develop. There is a primary identity, which typically maps to the credentials used by the person while working. Then there are additional identities used to access external functions, such as Team Foundation Server, or to publish apps onto the various Microsoft stores.

To mimic how developers work with these multiple online identities, Microsoft introduces a hierarchical relationship between these identities within Visual Studio. When you sign in, the account you specify is the primary identity for the Visual Studio IDE. It should, in theory, represent you (that is you, the person). Every place you sign into Visual Studio with the same credentials, your preferred settings will follow you. This includes customizations like themes and keyboard bindings. And a change on one device will automatically flow to the other devices you are signed into.

To handle the secondary credentials, Visual Studio 2017 contains a secure credential store. This allows the connections that you have made to external services to be remembered and used without the need to provide authentication each time. Naturally, you can manually sign out from a particular connection and the credentials will be removed.

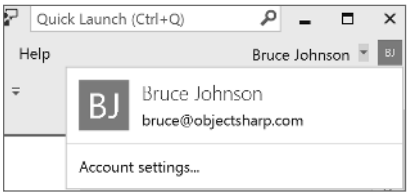

FIGURE 1-6

As part of the cloud enabling, you see your name (assuming that you logged in) in the top right of the IDE. If you click on the drop-down arrow (shown in Figure 1-6), you can see an Account Settings link. Clicking on that link takes you to a dialog (see Figure 1-7) in which you can manage the details of your account, including associating Visual Studio with different accounts.

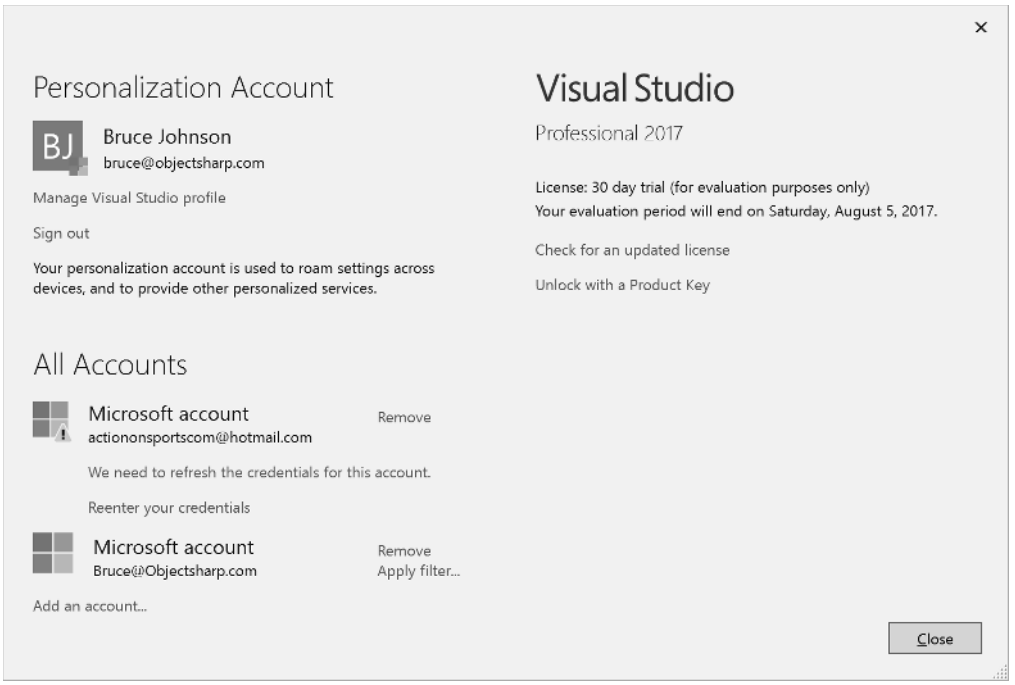

FIGURE 1-7

Along with providing a mechanism for editing the basic contact information for the profile, the dialog includes a list of the Microsoft Live accounts that have been "remembered" on your current machine.

THE VISUAL STUDIO IDE

The first time you launch Visual Studio 2017 , you will most likely see a dialog indicating that Visual Studio is configuring the development environment. When this process is complete, Visual Studio 2017 opens, ready for you to start working, as shown in Figure 1-8.

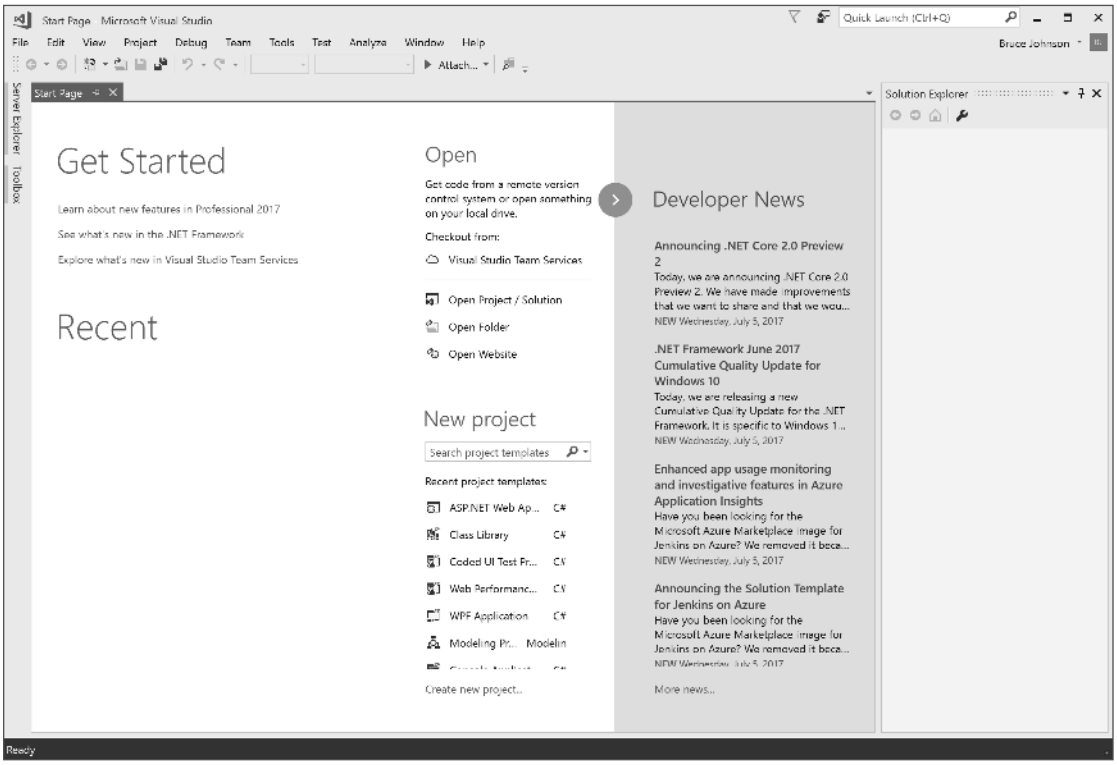

FIGURE 1-8

You'll see the Start page in the center of the screen. The bulk of the page contains links to the most common functions that you're likely to perform. For example, there is a list of Recent projects, along with links that allow you to open existing projects or create a new project. And in the latter case, the most commonly used templates are prominently on display. The previous version of the Start page included a news feed of interest to developers, and that feed is still present in Visual Studio 2017. And in the top left, there is also a Get Started section with links to information that is useful to new users of Visual Studio.

Before you launch into building your first application, you must take a step back to look at the components that make up the Visual Studio 2017 IDE. Menus and toolbars are positioned along the top of the environment, and a selection of subwindows, or panes, appears on the left, right, and bottom of the main window area. In the center is the main editor space. Whenever you open a code file, an XML document, a form, or some other file, it appears in this space for editing. With each file you open, a tab is created so that you can easily switch between opened files.

On either side of the editor space is a set of tool windows. These areas provide additional contextual information and functionality. For the general developer settings, the default layout includes the Solution Explorer and Properties on the right, and the Server Explorer and Toolbox on the left. The tool windows on the left are in their collapsed, or *unpinned*, state. If you click a tool window's title, it expands; it collapses again when it no longer has focus or you move the cursor to another area of the screen. When a tool window is expanded, you see a series of three icons at the top right of the window, similar to those shown in the top right corner of Figure 1-9.

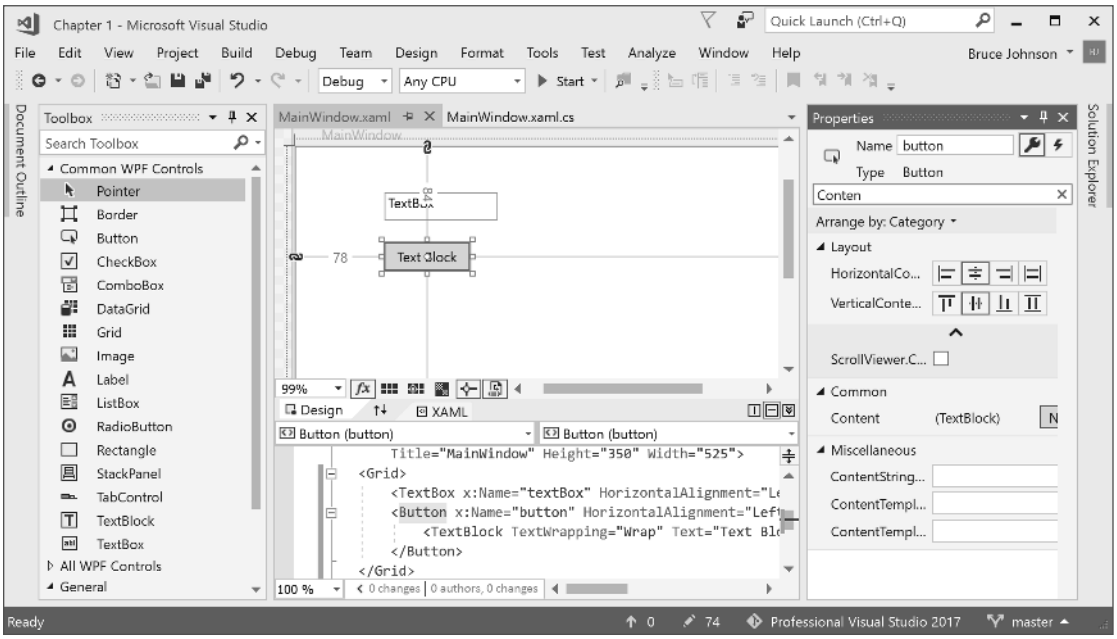

FIGURE 1-9

If you want the tool window to remain in its expanded, or *pinned*, state, you can click the middle icon, which looks like a pin. The pin rotates 90 degrees to indicate that the window is now pinned. Clicking the third icon, the X, closes the window. If later you want to reopen this or another tool window, you can select it from the View menu.

> **NOTE** *Some tool windows are not accessible via the View menu; for example, those having to do with debugging, such as threads and watch windows. In most cases these windows are available via an alternative menu item; for the debugging windows, it is the Debug menu.*

When the first icon, the down arrow, is clicked, a context menu opens. Each item in this list represents a different way to arrange the tool window. As you would imagine, the Float option enables the tool window to be placed anywhere on the screen, independent of the main IDE window. This is useful if you have multiple screens because you can move the various tool windows onto the additional screen, allowing the editor space to use the maximum screen real estate. Selecting the Dock as Tabbed Document option makes the tool window into an additional tab in the editor space. In Chapter 4, "The Visual Studio Workspace," you'll learn how to effectively manage the workspace by docking tool windows.

Developing, Building, Debugging, and Deploying Your First Application

Now that you have seen an overview of the Visual Studio 2017 IDE, this section walks you through creating a simple application that demonstrates working with some of these components. This is, of course, the mandatory "Hello World" sample that every developer needs to know, and it can be done in either Visual Basic .NET, or C#, depending on what you feel more comfortable with.

1. Start by selecting File ➪ New ➪ Project. This opens the New Project dialog, as shown in Figure 1-10. There is a tree on the left side of the dialog for grouping templates based on language and technology. And there is also a search box in the top-right corner. The right pane of this dialog displays additional information about the project template you have selected. Lastly, you can select the version of the .NET Framework that the application will target using the drop-down at the top of the dialog.

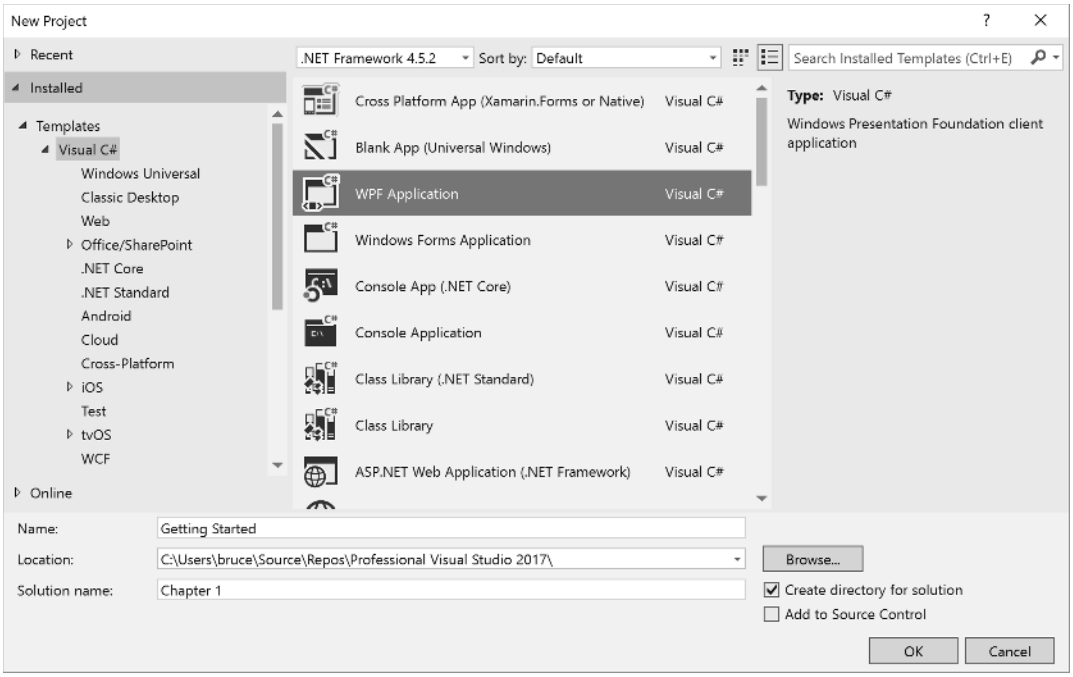

FIGURE 1-10

Select WPF Application from the Templates area (this item exists under the root Visual Basic and Visual C# nodes, or under the subnode Windows) and set the Name to **GettingStarted** before selecting OK. This creates a new WPF application project, which includes a single startup window and is contained within a Chapter 1 solution, as shown in the Solution Explorer window of Figure 1-11. This startup window has automatically opened in the visual designer, giving you an approximate graphical representation of what the window will look like when you run the application. The Properties tool window is collapsed and sits on the right side of the windows.

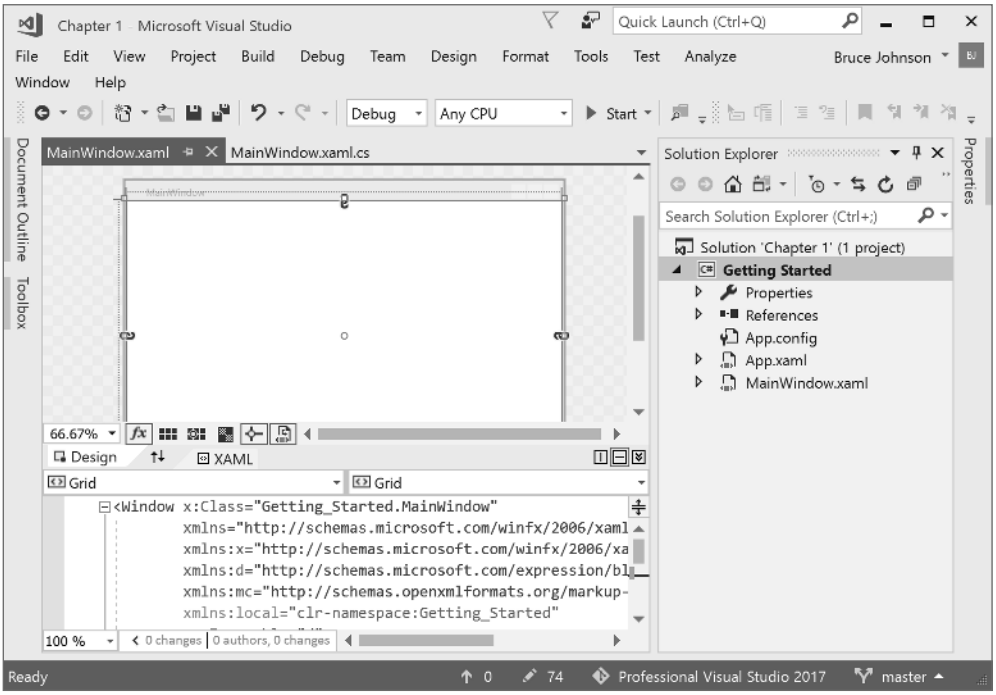

FIGURE 1-11

2. Click the collapsed Toolbox window, which appears on the left side of the screen. This causes the Toolbox to expand. Then click on the pin icon, which keeps the tool window open. To add controls to the window in the GettingStarted project, select the appropriate items from the Toolbox and drag them onto the form. Alternatively, you can double-click the item, and Visual Studio automatically adds them to the window.

3. Add a button and textbox to the form so that the layout looks similar to the one shown in Figure 1-12. Select the textbox, and select the Properties tool window. (You can press F4 to automatically open the Properties tool window.) Change the name of the control (found at the top of the Properties tool window) to **txtSayHello**. Repeat for the Button control, naming it **btnSayHello** and setting the `Content` property to **Say Hello!**

You can quickly locate a property by typing its name into the search field located beneath the Name field. In Figure 1-12 **Conten** has been entered to reduce the list of Properties so that it's easier to locate the Content property.

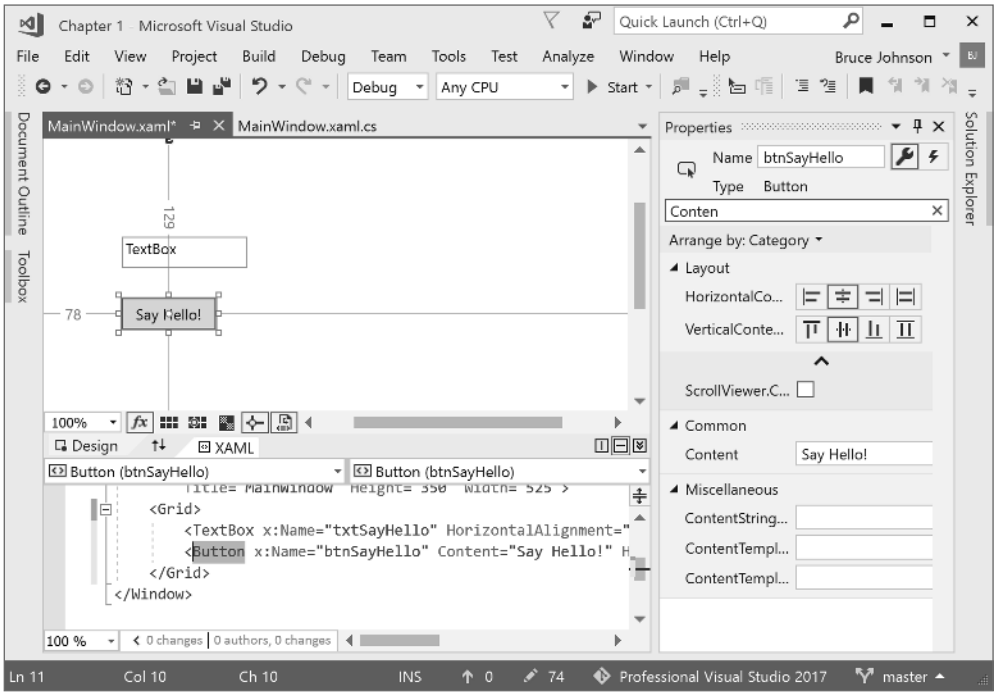

FIGURE 1-12

After you add controls to the window, the tab is updated with an asterisk (*) after the text to indicate that there are unsaved changes to that particular item. If you attempt to close this item while changes are pending, you are asked if you want to save the changes. When you build the application, any unsaved files are automatically saved as part of the build process.

> **NOTE** *One thing to be aware of is that some files, such as the solution file, are modified when you make changes within Visual Studio 2017 without your being given any indication that they have changed. If you try to exit the application or close the solution, you are still prompted to save these changes.*

4. Deselect all controls (you can click an empty spot on the screen to do this), and then double-click the button. This not only opens the code editor with the code-behind file for

this form, it also creates and wires up an event handler for the click event on the button. Figure 1-13 shows the code window after a single line has been added to echo the message to the user.

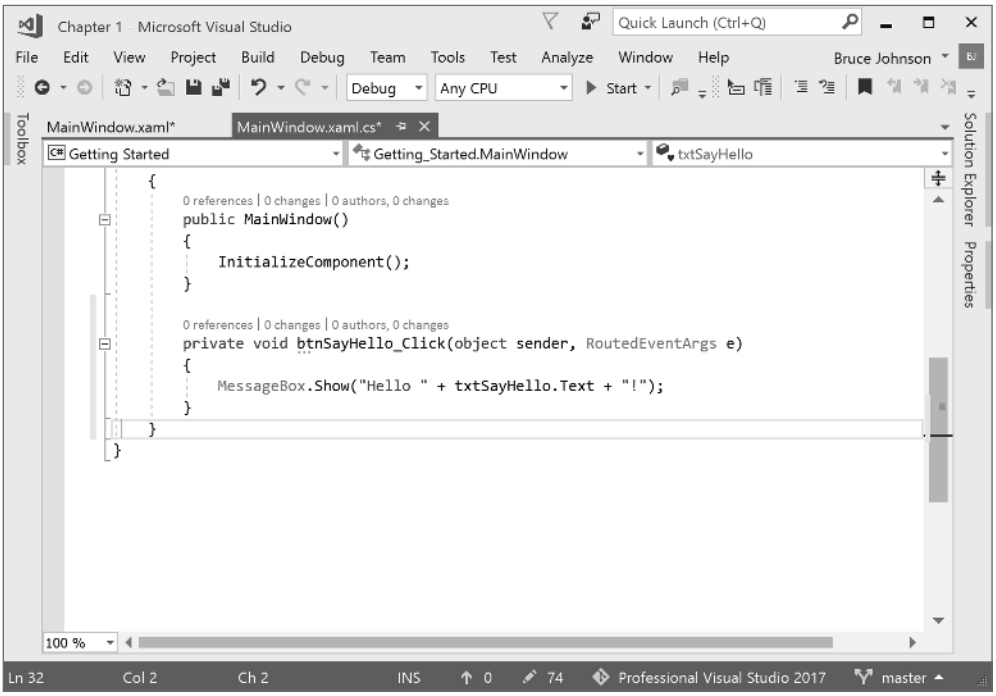

FIGURE 1-13

5. Before you build and execute your application, place the cursor somewhere on the line containing `MessageBox.Show` and press F9. This sets a breakpoint; when you run the application by pressing F5 and then click the "Say Hello!" button, the execution halts at this line. Figure 1-14 illustrates this breakpoint being reached. The data tip, which appears when the mouse hovers over the line, shows the contents of the `txtSayHello.Text` property.

   The layout of Visual Studio in Figure 1-14 is significantly different from the previous screenshots because a number of tool windows are visible in the lower half of the screen, and command bars are visible at the top. Also, the status bar at the bottom of the IDE is orange, as opposed to the blue that appears when in design mode. When you stop running or debugging your application, Visual Studio returns to the previous layout. Visual Studio 2017 maintains two separate layouts: design time and run time. Menus, toolbars, and various

windows have default layouts for when you edit a project, whereas a different setup is defined for when a project is executed and debugged. You can modify each of these layouts to suit your own style, and Visual Studio 2017 remembers them.

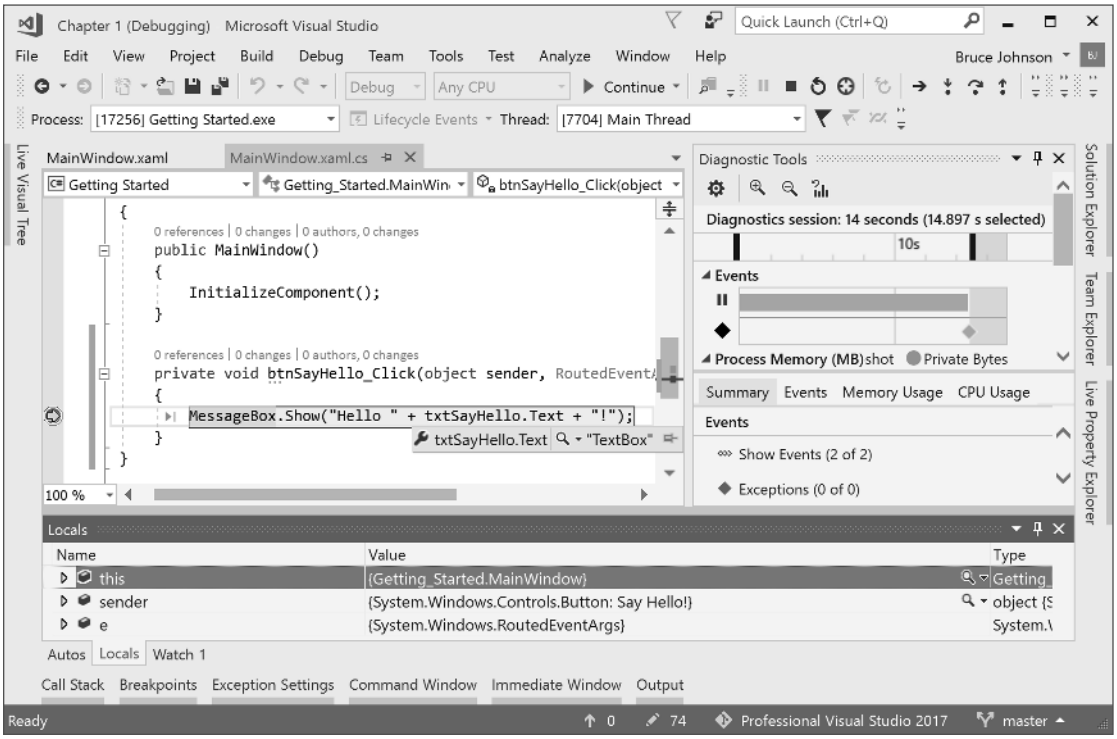

FIGURE 1-14

6. You need to deploy your application. Whether you build a rich client application using Windows Forms or WPF, or a web application using IIS, Azure, Node.js, or any of a number of other technologies, Visual Studio 2017 has the capability to publish your application. Double-click the Properties node in Solution Explorer, and select the Publish node to display the options for publishing your application, as shown in Figure 1-15.

   In Figure 1-15, the publishing folder has been set to a local path (by default, the path is relative to the directory in which the project is found), but you can specify a network folder, an Internet Information Services (IIS) folder, or an FTP site instead. After you specify where you want to publish to, clicking Publish Now publishes your application to that location.

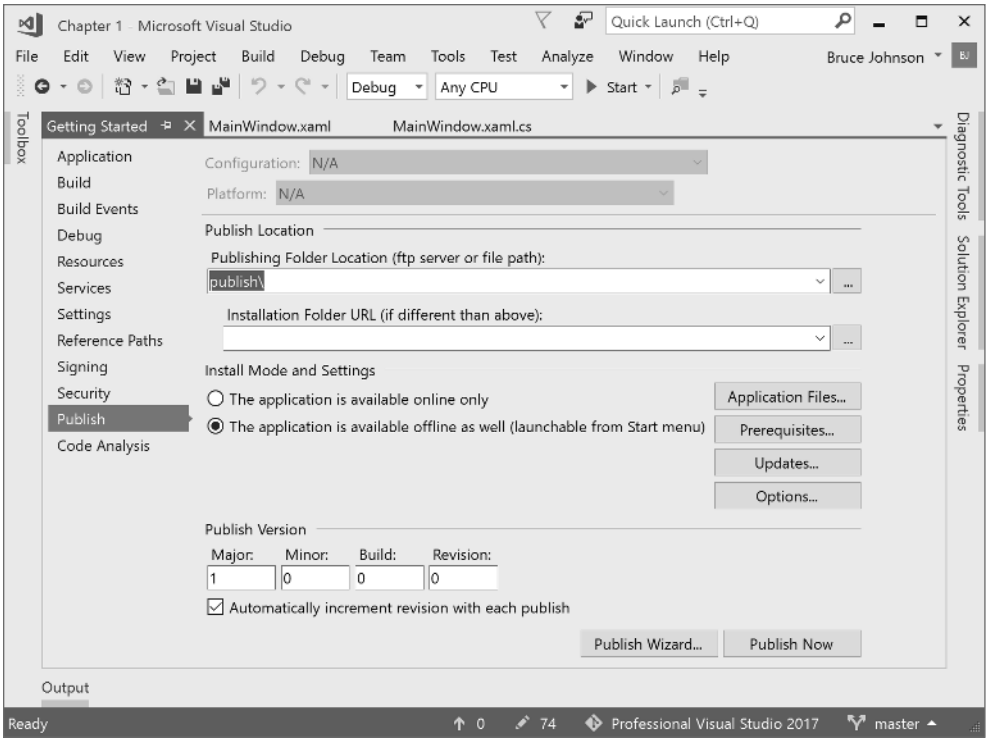

FIGURE 1-15

SUMMARY

You've seen how the various components of Visual Studio 2017 work together to build an application. The following list outlines the typical process of creating a solution:

1. Use the File menu to create a solution.
2. Use the Solution Explorer to locate the window that needs editing, and double-click the item to show it in the main workspace area.
3. Drag the necessary components onto the window from the Toolbox.
4. Select the window and each component in turn, and edit the properties in the Properties window.
5. Double-click the window or a control to access the code behind the component's graphical interface.
6. Use the main workspace area to write code and design the graphical interface, switching between the two via the tabs at the top of the area.

7. Use the toolbars to start the program.
8. If errors occur, review them in the Error List and Output windows.
9. Save the project using either toolbar or menu commands, and exit Visual Studio 2017.

In subsequent chapters, you'll learn how to customize the IDE to more closely fit your own working style. You'll also see how Visual Studio 2017 takes a lot of the guesswork out of the application development process and a number of best practices for working with Visual Studio 2017 that you can reuse as a developer.