Visual Studio 2017

Succinctly®

by Alessandro Del Sole

Syncfusion | Technology Resource Portal

Visual Studio 2017 Succinctly

By

Alessandro Del Sole

Foreword by Daniel Jebaraj

Important licensing information. Please read.

Technical Reviewer: James McCaffrey

Copy Editor: John Elderkin

Acquisitions Coordinator: Tres Watkins, content development manager, Syncfusion, Inc.

Proofreader: Graham High, senior content producer, Syncfusion, Inc.

Table of Contents

The Story behind the *Succinctly* Series of Books

Daniel Jebaraj, Vice President
Syncfusion, Inc.

Staying on the cutting edge

As many of you may know, Syncfusion is a provider of software components for the Microsoft platform. This puts us in the exciting but challenging position of always being on the cutting edge.

Whenever platforms or tools are shipping out of Microsoft, which seems to be about every other week these days, we have to educate ourselves, quickly.

Information is plentiful but harder to digest

In reality, this translates into a lot of book orders, blog searches, and Twitter scans.

While more information is becoming available on the Internet and more and more books are being published, even on topics that are relatively new, one aspect that continues to inhibit us is the inability to find concise technology overview books.

We are usually faced with two options: read several 500+ page books or scour the web for relevant blog posts and other articles. Just as everyone else who has a job to do and customers to serve, we find this quite frustrating.

The *Succinctly* series

This frustration translated into a deep desire to produce a series of concise technical books that would be targeted at developers working on the Microsoft platform.

We firmly believe, given the background knowledge such developers have, that most topics can be translated into books that are between 50 and 100 pages.

This is exactly what we resolved to accomplish with the *Succinctly* series. Isn't everything wonderful born out of a deep desire to change things for the better?

The best authors, the best content

Each author was carefully chosen from a pool of talented experts who shared our vision. The book you now hold in your hands, and the others available in this series, are a result of the authors' tireless work. You will find original content that is guaranteed to get you up and running in about the time it takes to drink a few cups of coffee.

Free forever

Syncfusion will be working to produce books on several topics. The books will always be free. Any updates we publish will also be free.

Free? What is the catch?

There is no catch here. Syncfusion has a vested interest in this effort.

As a component vendor, our unique claim has always been that we offer deeper and broader frameworks than anyone else on the market. Developer education greatly helps us market and sell against competing vendors who promise to “enable AJAX support with one click,” or “turn the moon to cheese!”

Let us know what you think

If you have any topics of interest, thoughts, or feedback, please feel free to send them to us at succinctly-series@syncfusion.com.

We sincerely hope you enjoy reading this book and that it helps you better understand the topic of study. Thank you for reading.

Please follow us on Twitter and “Like” us on Facebook to help us spread the word about the *Succinctly* series!

About the Author

Alessandro Del Sole has been a Microsoft Most Valuable Professional (MVP) since 2008. Awarded MVP of the Year in 2009, 2010, 2011, 2012, and 2014, he is internationally recognized as a Visual Studio expert and a Visual Basic and .NET authority. Alessandro has authored many books and e-books on programming with Visual Studio, including *Visual Studio Code Succinctly*, *Visual Basic 2015 Unleashed*, *Roslyn Succinctly*, and *Visual Basic 2012 Unleashed*. He has written numerous technical articles about .NET, Visual Studio, and other Microsoft technologies in Italian and English for many developer portals, including MSDN Magazine and the Visual Basic Developer Center from Microsoft. He is a frequent speaker at Italian conferences, and he has released a number of Windows Store apps. He has also produced a number of instructional videos in both English and Italian. Alessandro works as a senior .NET developer, trainer, and consultant. You can follow him on Twitter at @progalex.

Introduction

“Any developer, any platform, any device.” If you attended (in person or online) a recent Microsoft conference, such as //Build 2016, Ignite, or Connect(); 2016, these words will not sound new. They represent the vision and ambition behind the release of Microsoft Visual Studio 2017 as the ultimate development tool supporting the most recent strategies at Microsoft. Until a few years ago, Microsoft Visual Studio was the development environment of choice if you wanted to build Windows and web applications based on the .NET Framework with C#, F#, Visual Basic, and C++. If you wanted (or needed) to build applications for different operating systems, platforms, or devices other than a PC, you had to use proprietary development tools and native frameworks on specific platforms.

In recent years, Microsoft has significantly changed its strategy, opening up to other platforms, embracing open source, and focusing even more on cloud services. In fact, the company has been making significant investments for bringing technologies, platforms, developer tools, frameworks, and services to other operating systems such as Linux and Mac OS, and to typically non-Microsoft (and sometimes hostile) audiences by focusing on services much more than in the past. In this strategy, .NET Core, the modular open source, cross-platform runtime, enables C# developers to write applications that run on Windows, Linux, and Mac. With Xamarin, you can write mobile applications that run on Android, iOS, and Windows with a single, shared C# codebase. SQL Server 2016 now has a preview that runs on Linux—a revolutionary milestone for Microsoft. A preview of Visual Studio for Mac is currently available, and it fully enables C# developers to write cross-platform apps with .NET Core and Xamarin on Mac OS. In this cross-platform and cross-device vision, the cloud is even more important. In fact, Azure hosts all the new and existing services Microsoft is offering, and it grows according to what the market demands—for example, hosting Docker containers on Linux.

Being the premier development environment from Microsoft, Visual Studio 2017 fits perfectly into this mobile-first, cloud-first world; developers can use Visual Studio 2017 to build apps that run on any platform and any device with the language and framework of their choice. For example, Visual Studio 2017 allows you to write Node.js applications, native iOS and Android applications, and web apps that run on Linux and Mac OS. The good news is that you, as the developer, can still use the same powerful tools you already know, such as the debugger, IntelliSense, and profilers against all the supported development platforms. In this e-book, you will find a comprehensive description of new features in the Visual Studio 2017 IDE that will not only help you to write better code but will also help you understand how you can use it for building apps for any platform and any device.

As with its predecessors, Visual Studio 2017 is available in different editions, such as Community, Professional, and Enterprise. If a feature requires a specific edition, it will be highlighted when appropriate. You can download the Community edition for free.

A final disclaimer—this e-book has been written based on the Release Candidate (RC) of Visual Studio 2017, which means some tools might be subject to slight changes in the final version.

This book is dedicated to my girlfriend Angelica. I’m so grateful for all you do for us.

Alessandro

Chapter 1 A New Installation Experience

Microsoft Visual Studio has always been an extremely powerful development environment. One of the reasons for its power is that it can target multiple development platforms and, with versions 2013 and 2015, it even added the option to target non-Microsoft technologies and operating systems. For example, think of Android and iOS development with both Xamarin and Apache Cordova. But great power also means a complex infrastructure, and in the past Visual Studio required many hours for installation and a huge amount of space on disk. In Visual Studio 2017, Microsoft brings a new installation experience that simplifies the process and saves both time and disk space. Installing Visual Studio 2017 will be your first experience with the new version, and it deserves a thorough discussion.

Solving the complexity of the Visual Studio installation

Keeping in mind the importance of the overall performance and efficiency of its premiere development tool, one of Microsoft's goals for Visual Studio 2017 was to simplify the installation process to save developers time and disk space. With Visual Studio 2015, a full installation required many gigabytes on disk and several hours to complete. Solving this problem meant rethinking the entire IDE infrastructure and changing both the way Visual Studio consumes the components it relies on and the way it allows targeting different development platforms. Based on these changes, Visual Studio 2017 is now made of a core shell that includes the code editor and essential tools, referred to as **Visual Studio core editor**. This provides capabilities for writing code in a number of languages, including (but not limited to) Visual Basic, C#, C++, and Python, along with syntax colorization, IntelliSense, and debugging support, all in the familiar Visual Studio environment (see Chapter 5, "Working with solutions, folders, and languages").

Additionally, Visual Studio 2017 supports adding sets of components, each targeting a specific development scenario. Each set of components is referred to as a **workload**. Workloads make installation and maintenance easier and allow developers to install what they actually need without unnecessary components, SDKs, and tools. You could even decide to install only the Visual Studio core editor without any additional workloads in order to get the basic coding environment, which would reduce the space required by the Visual Studio installation to about 750 megabytes. You will probably need more than just syntax highlighting, which means you will want to select the proper workloads—this just gives you an idea of how Visual Studio's infrastructure has been revisited. The next section will describe how to install Visual Studio 2017 and explain more about workloads and what you can do with each.

Installing Visual Studio 2017

Tip: Visual Studio 2017 can be installed side-by-side on a machine that has earlier versions of the IDE installed.

When you start the setup program, you are prompted with a completely new user interface that presents a list of workloads. This is represented in Figure 1.

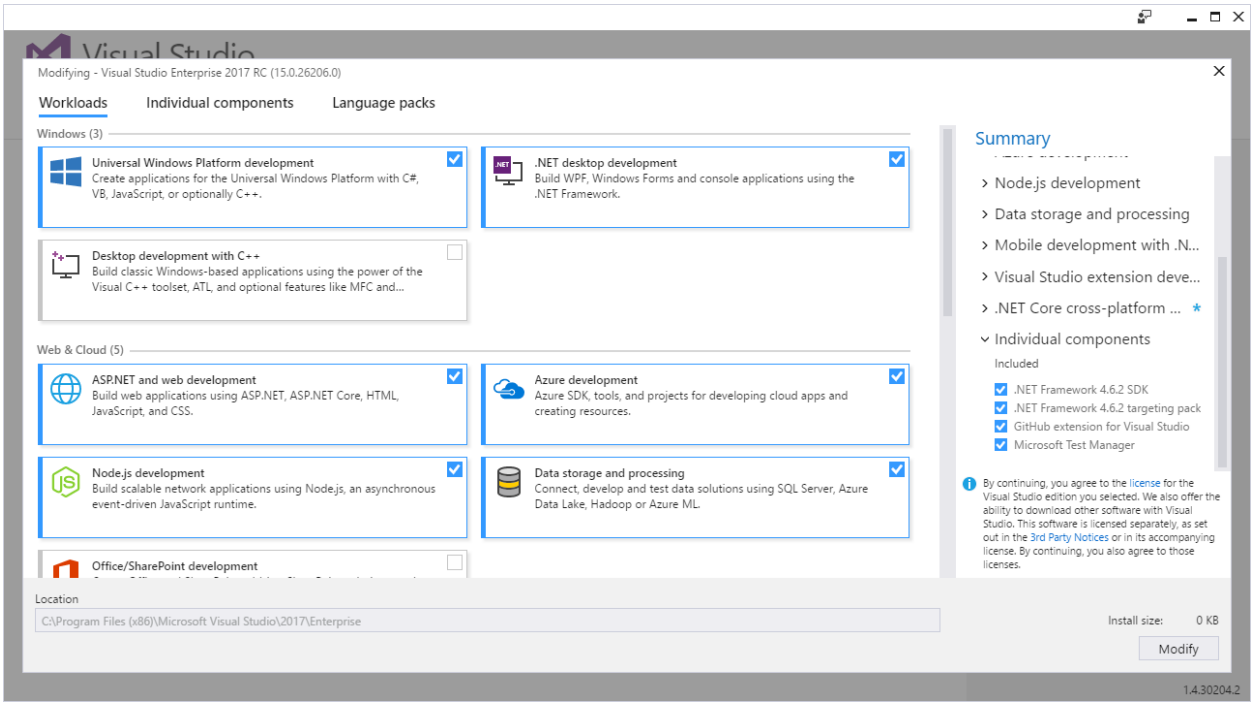

Figure 1: Starting the Installation of Visual Studio 2017

As you can see, workloads are grouped by the following categories:

- Windows
- Web & Cloud
- Mobile & Gaming
- Other Toolsets

All the available workloads can be discovered by scrolling through the list (see Figure 2).

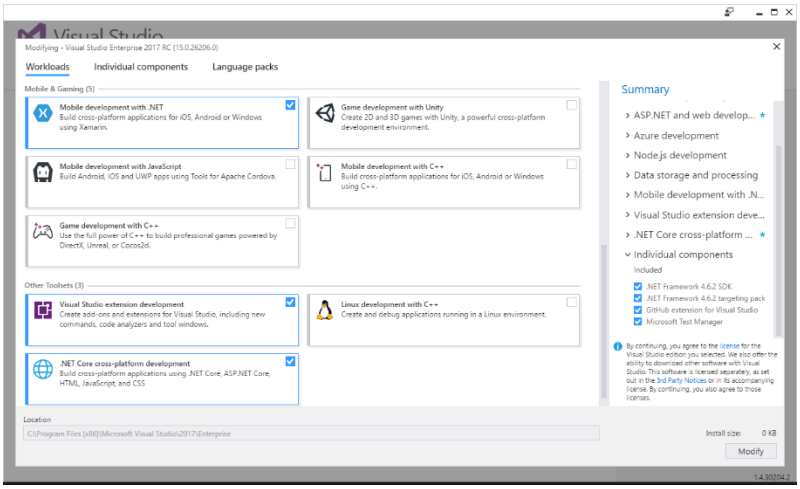

Figure 2: Additional Workloads

You can select all the workloads you'll need for your daily work with Visual Studio 2017. If you select no workloads, only the Visual Studio core editor will be installed. On the right side of the dialog, you can expand the workload name and see which components it includes and a list of additional, individual components. Table 1 depicts available workloads in more detail.

Table 1: Available Workloads for Visual Studio 2017

Name	**Description**
Universal Windows Platform development	Select this workload if you want to write universal applications for Windows 10, including PC, tablets, smartphones, HoloLens, Xbox, and IoT devices.
.NET desktop development	Select this workload if you want to build classic Windows desktop applications with WPF, Windows Forms, and console apps using the .NET Framework. This provides shorter solution load time and improved IntelliSense, code navigation, and refactorings. It includes new features such as XAML Edit and Continue and Run to Click debugging, both of which are discussed in this e-book.
Desktop development with C++	Select this workload if you wish to create, build, and debug native, classic desktop applications that run on versions ranging from Windows XP to the latest Windows 10 release, using the C++ language and environment.
ASP.NET and web development	Select this workload to develop web applications using ASP.NET and standards-based technologies such as HTML, JavaScript, CSS, and JSON. As with .NET desktop, this workload includes shorter solution load time, improved IntelliSense, code navigation, and more refactoring, and it enables you to quickly deploy your app to a web server or to Azure.
Azure development	This workload installs the latest Azure SDK for .NET and tools for Visual Studio 2017. It allows you to view resources in Cloud Explorer, create resources using Azure Resource Manager tools, and build applications and services ready to be hosted in Azure.
Node.js development	This workload adds everything you need to build apps for Node.js, including IntelliSense, local and remote debugging, profiling, npm integration, an interactive window, test runners, and Azure integration.
Data storage and processing	This workload provides tools for accessing on-premises SQL Server databases as well as SQL databases on Azure and Azure Data Lakes resources. It also provides support for U-SQL, Hive, and Big Data on Azure.
Office/SharePoint development	This workload provides the Office developer tools, which allow for creating Office and SharePoint add-ins and solutions.

Name	Description
.NET Core cross-platform development	This workload installs all the tools you need to write cross-platform web applications with .NET Core, with support for deployment to Docker containers.
Mobile development with .NET	This workload installs Xamarin, the technology that allows you to create native iOS, Android, and Universal Windows Platform apps using a shared C# codebase.
Game development with Unity	Select this workload if you want to develop cross-platform 2D and 3D games using the Unity framework and integrated tools for Visual Studio 2017.
Mobile development with JavaScript	This workload installs Apache Cordova for creating cross-platform mobile apps using HTML and JavaScript within Visual Studio.
Mobile development with C++	Select this workload if you want to create cross-platform mobile apps using C++.
Game development with C++	Select this workload if you want to create games using C++.
Visual Studio extension development	This workload installs the Visual Studio SDK and allows you to write extensions such as new commands, tool windows, and templates.
Linux development with C++	This workload enables you to author C++ code for Linux servers, desktops, and devices from within Visual Studio 2017.

For the instructional purposes of this e-book, I have installed all the available workloads. You are not required to do the same—feel free to select only those you need. You can later install additional workloads as required.

Customizing the installation with individual components

Although workloads help keep the installation simple, you might still need to install individual components. In Figure 2, you will see an item called **Individual components**. Click it to access the full list of individual components that can be installed regardless of selected workloads (see Figure 3).

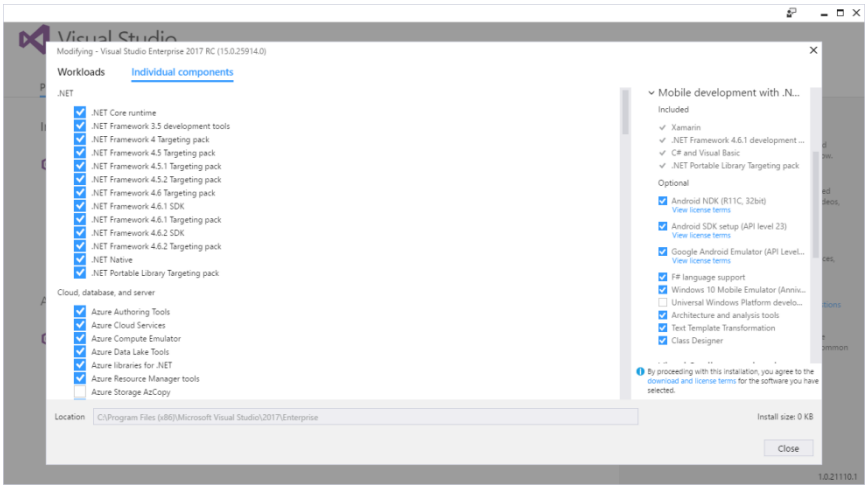

Figure 3: Selecting Individual Components

As an example, the GitHub extension for Visual Studio 2017 is not selected by default, which means you might want to select this component if you plan to work with Git repositories on that popular service. Once you have made your selection, click **Close** and select the edition of Visual Studio 2017 you want to install. A dialog will show the progress of the operation (see Figure 4).

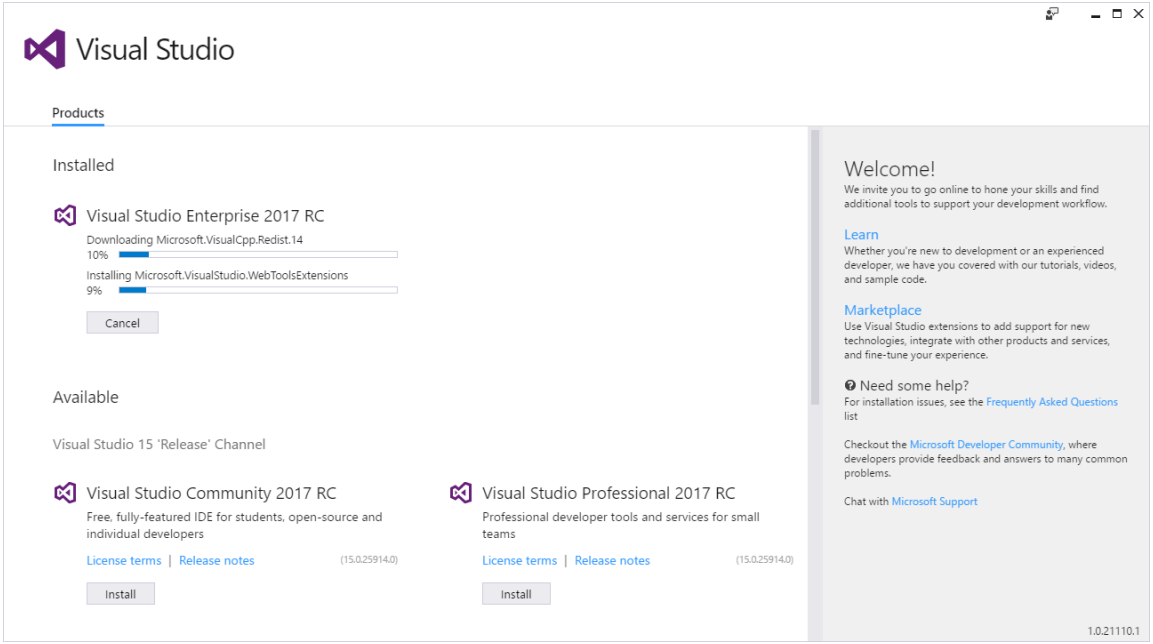

Figure 4: Installation Progress

The time needed for complete installation depends on the number of workloads and on your Internet connection. But even with many workloads selected, you will notice it goes much faster than previous Visual Studio installations.

Installing multiple editions

With Visual Studio 2017, you can finally install multiple editions on the same machine. This great feature means you can install the Community, Professional, and Enterprise editions on the same machine (or any two of them). This is now possible because each edition is installed into a specific subfolder on the system. Apart from running separate installers, you could launch a new installation directly from the current setup program. Figure 4 shows an Install button for both the Community and Professional editions while Enterprise is being installed.

Note: Visual Studio core assemblies are no longer installed to the Global Assembly Cache (GAC). In order to support the installation of multiple editions, most assemblies required by Visual Studio 2017 now reside in C:\Program Files (x86)\Microsoft Visual Studio\2017\%editionName%\Common7\IDE\PublicAssemblies, where %editionName% is the installed edition (such as Community, Professional, or Enterprise).

Modifying the Visual Studio 2017 installation

Visual Studio 2017 offers a new shortcut for modifying an existing installation. You simply go to the **Windows** > **All Programs** menu and select the **Visual Studio Installer** shortcut. This will start the setup program, and you will have an option to add or remove workloads or individual components.

Launching Visual Studio 2017

As with its predecessor, Visual Studio 2017 launches using the same-named shortcut in the All Programs menu. When it starts for the first time, Visual Studio will ask for your Microsoft account credentials to log in (optional). As you might know, entering a Microsoft account will allow for synchronizing settings across machines. This will also automatically restore customized settings you might have on an existing VS 2017 installation. When launching Visual Studio, you will immediately recognize better performance and faster startup than Visual Studio 2015. Other new features at startup are described in the next chapter.

Chapter summary

Visual Studio 2017 introduces a completely new installation experience based on the Visual Studio core editor and a number of workloads, each targeting specific development scenarios. This not only simplifies the setup process, but installing Visual Studio is now much faster and more efficient. You still maintain full control of the installed components, and you can even install multiple editions of Visual Studio 2017 on the same machine. Now that you have set up the environment, you will find some new features when launching Visual Studio 2017 for the first time.

Chapter 2 The Start Page Revisited

Visual Studio 2017 has improved performance and efficiency in many ways. You will immediately notice this when you launch the IDE, which will load faster than its recent predecessors. Not limited to performance, the new version provides an enhanced start experience through a revised Start Page.

Optimized start experience

The **Start Page** is the first contact you will have with Visual Studio 2017, which means it plays an important role. In the new version, the Start Page has been reorganized and optimized to offer more space for common shortcuts and tools. Figure 5 shows the new Start Page.

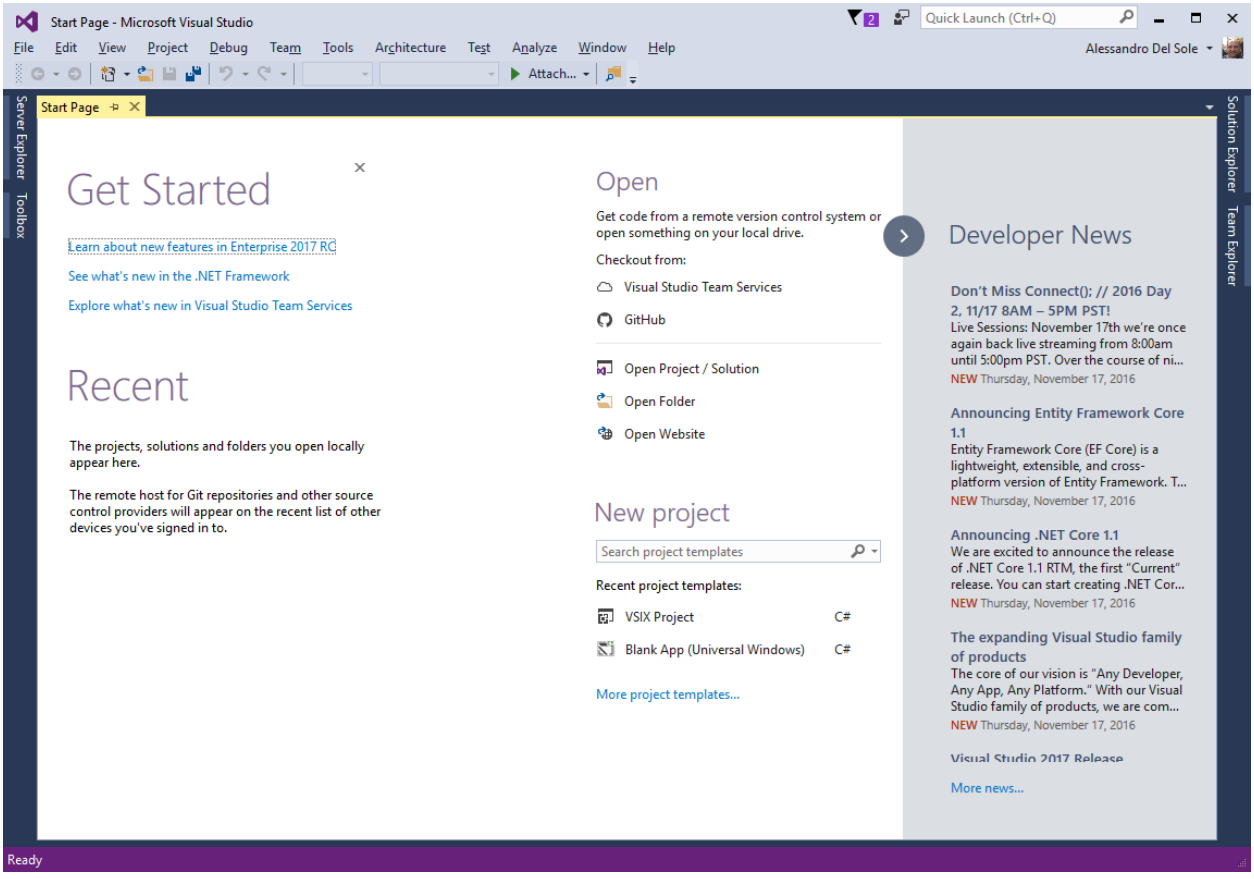

Figure 5: The New Start Page

The first new feature in the Start Page allows the **Get Started** area to be removed by clicking the **X** icon at the upper-right corner. This area contains shortcuts to learning resources and documentation, but you can also hide it to save space for other contents. These are described in the next sections.

Tip: In Visual Studio 2017, the menu command to open the Start Page has been moved from the** View **menu to the** File **menu.

Staying up to date: Announcements and news

As with its predecessors, Visual Studio 2017 offers a list of announcements and news from official Microsoft channels (see Figure 5). However, in the new version, this list has been moved to a collapsible panel on the right side of the page called **Developer News**. Figure 6 shows how the Start Page appears with the Developer News panel hidden.

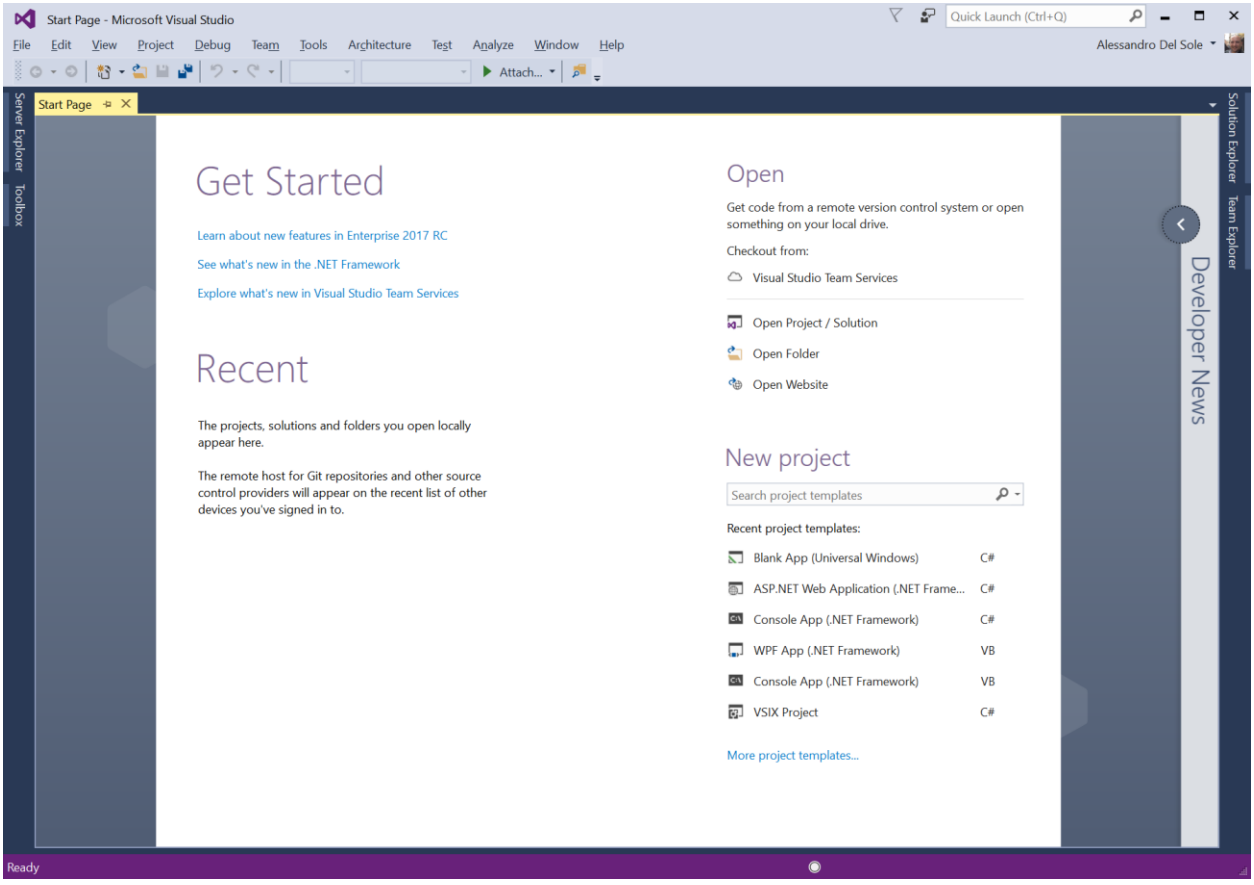

Figure 6: Collapsing the Developer News Panel

By collapsing the Developer News panel, you can have more space for recently used items and tools. Just click the arrow to restore the panel. However, note that when it is collapsed, you will be notified of updated news via an orange glyph that will overlay the arrow.

Shortcuts for solutions, projects, and folders

Visual Studio 2017 offers shortcuts to open and create projects quickly. This is not in fact new, but there are several changes and improvements in the new version.

Working with most recently used projects

The list of most recently used projects (MRUs) is on the left side of the Start Page. Along with recently used projects, Visual Studio 2017 will also show the list of repositories you have recently cloned from Visual Studio Team Services and GitHub, which are represented with a folder icon. Even more interesting, this list is synced across machines if you log into Visual Studio with a Microsoft Account, which means that you will see this list on any of your installations of Visual Studio 2017. This will make it easier to clone the same repositories on all your machines. Figure 7 shows the list of MRUs, including cloned repositories.

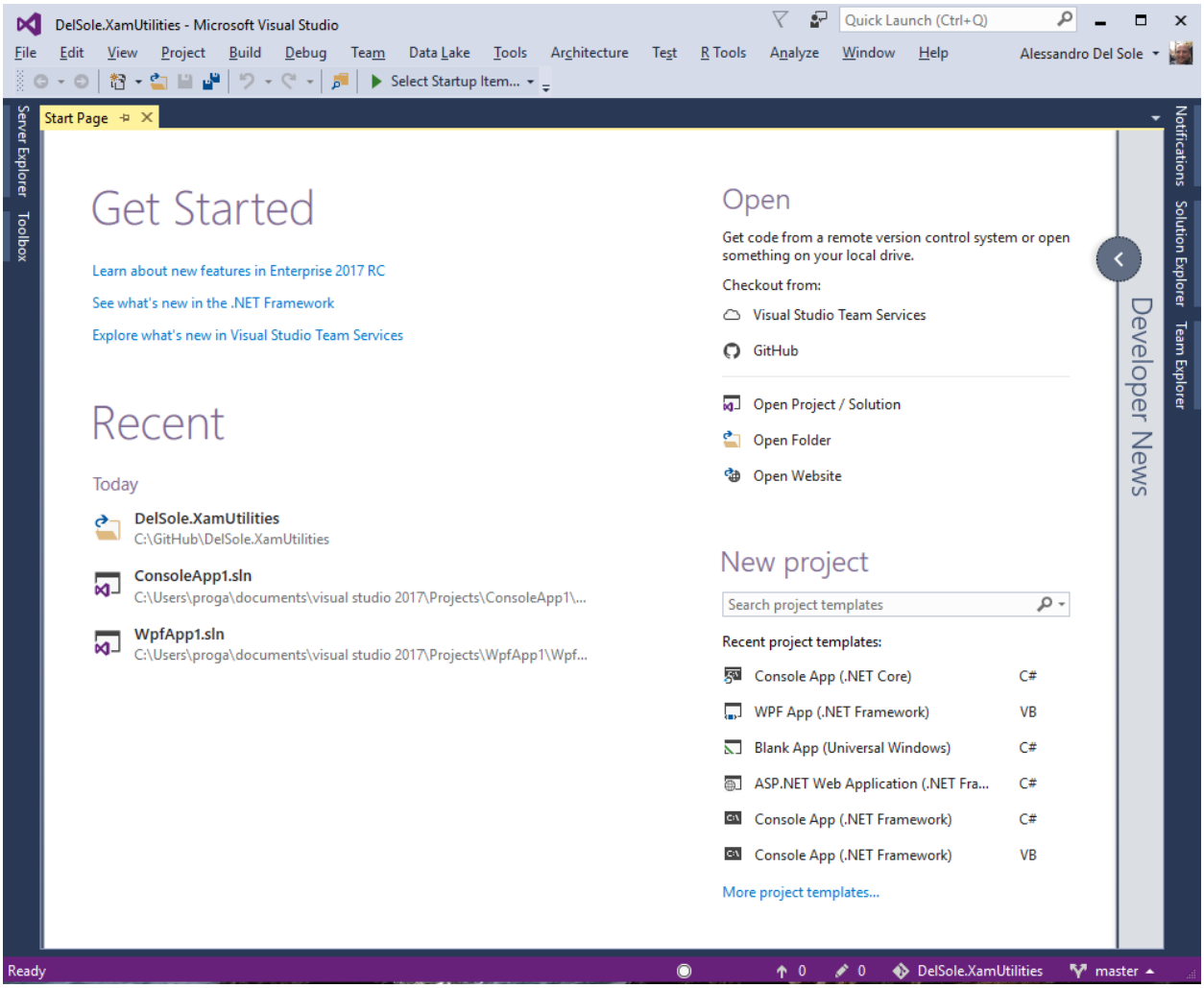

Figure 7: List of MRUs and Cloned Repositories

Accessing project templates

The new Start Page makes it easier to create new projects by selecting different templates. In the **New project** area, which you can see in all the previous figures, you can find a list of recently used project templates that you can click to create a new project based on that template. This list is synchronized across machines, and it also shows the programming language used with the project template. In fact, when you click a recent template, Visual Studio 2017 shows the **New Project** dialog with the specified template already selected. Also, by clicking **More project templates**, the New Project dialog will appear and provide an option for deciding which project template to use. Finally, you can search for project templates directly within the Start Page by typing in the **Search project templates** text box.

Opening projects, folders, and repositories from source control

The **Open** area in the Start Page provides shortcuts to open projects and websites. There are several very interesting new features in Visual Studio 2017. The first feature comes with a shortcut called **Open Folder**. This allows you to open folders containing loose assortments of code files that are not based on proprietary project systems. This will be examined thoroughly in Chapter 5, "Working with solutions, folders, and languages."

The second new feature makes it easier to open projects from source control engines such as Team Foundation Server, Visual Studio Team Services, and Git. As you can see in Figure 7, the group called **Checkout from** offers two shortcuts:

- Visual Studio Team Services
- GitHub

The first shortcut will allow you to open team projects or Git repositories from both Visual Studio Team Services and Team Foundation Server. The second shortcut allows you to open or clone Git repositories from GitHub. Of course, you can still manage team project connections with the Team Explorer tool window.

Tip: The GitHub shortcut is available only if you install the GitHub extension for Visual Studio 2017. This can be easily selected from the list of individual components within the Visual Studio Installer.

Chapter summary

Visual Studio 2017 improves developer productivity from the moment it starts up. With the new Start Page, the available space has been reorganized in order to offer more shortcuts to commonly used tools. Also, new features have been introduced to support team projects and repositories, including the list of recently cloned repositories (which is synced across machines), and the option to open projects from source control engines such as Git and Visual Studio Team Services.

Chapter 3 Code Editor Improvements

The code editor is the place where you spend most of your developer life, and Microsoft introduces productivity improvements at every major release of the IDE. Visual Studio 2017 is no exception—it introduces a number of excellent features to improve your coding experience.

Note: All the topics described in this chapter apply to both C# and Visual Basic, except where expressly specified. Also, remember that C#, Visual Basic, C++, and F# also provide a number of new language features that are not covered in this chapter, which instead focuses on the code editing experience. Visit the Visual Studio documentation for further information on what's new with programming languages in Visual Studio 2017.

IntelliSense improvements

The IntelliSense tool has long been a best friend of every developer using Microsoft Visual Studio. In Visual Studio 2017, IntelliSense gets major improvements that will help you save time while coding. First, IntelliSense is now smarter with filtering. Instead of suggesting the top item in the list, it automatically shows the best matching result based on what you typed. And, not limited to this, it starts highlighting the words in bold as you type, as shown in Figure 8.

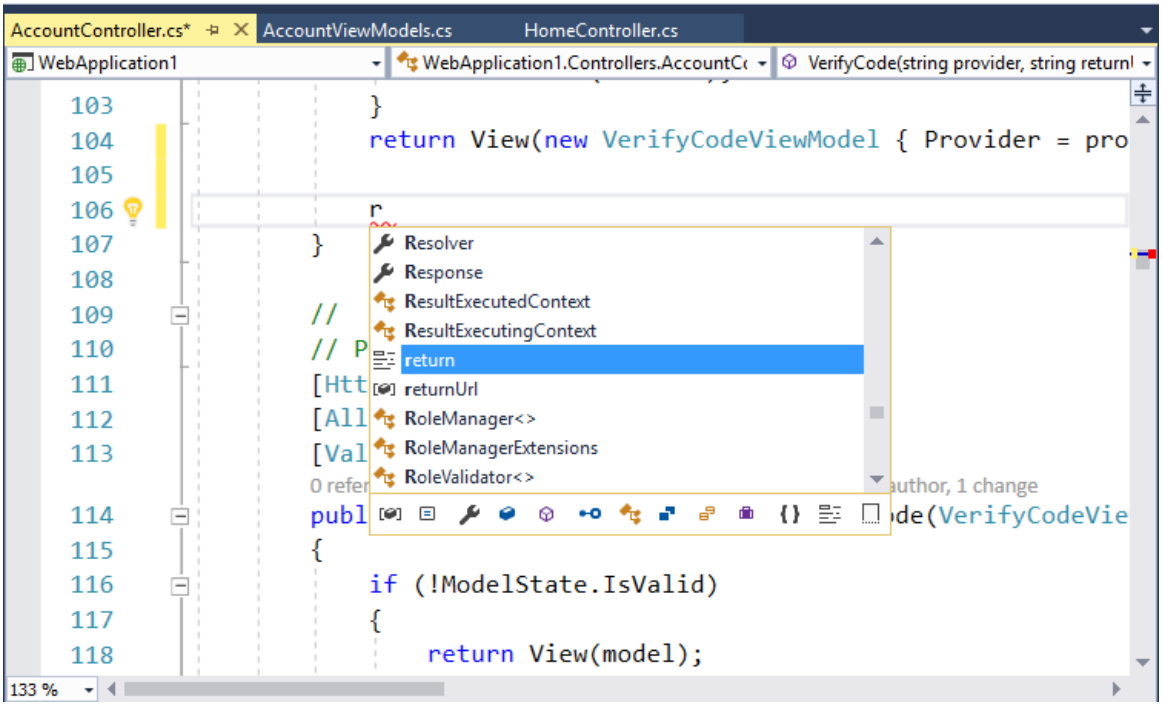

Figure 8: IntelliSense Suggestions and Filters

Filtering based on specific types or members is another important improvement in Visual Studio. As you can see in Figure 8, there is a new bar with many icons at the bottom of IntelliSense, each icon representing a particular kind of object or member such as local variables, constants, properties, fields, methods, interfaces, classes, value types, enumerations, delegates, namespaces, reserved words, and even code snippets. For instance, Figure 9 shows how to restrict IntelliSense's members searches to methods, classes, and code snippets.

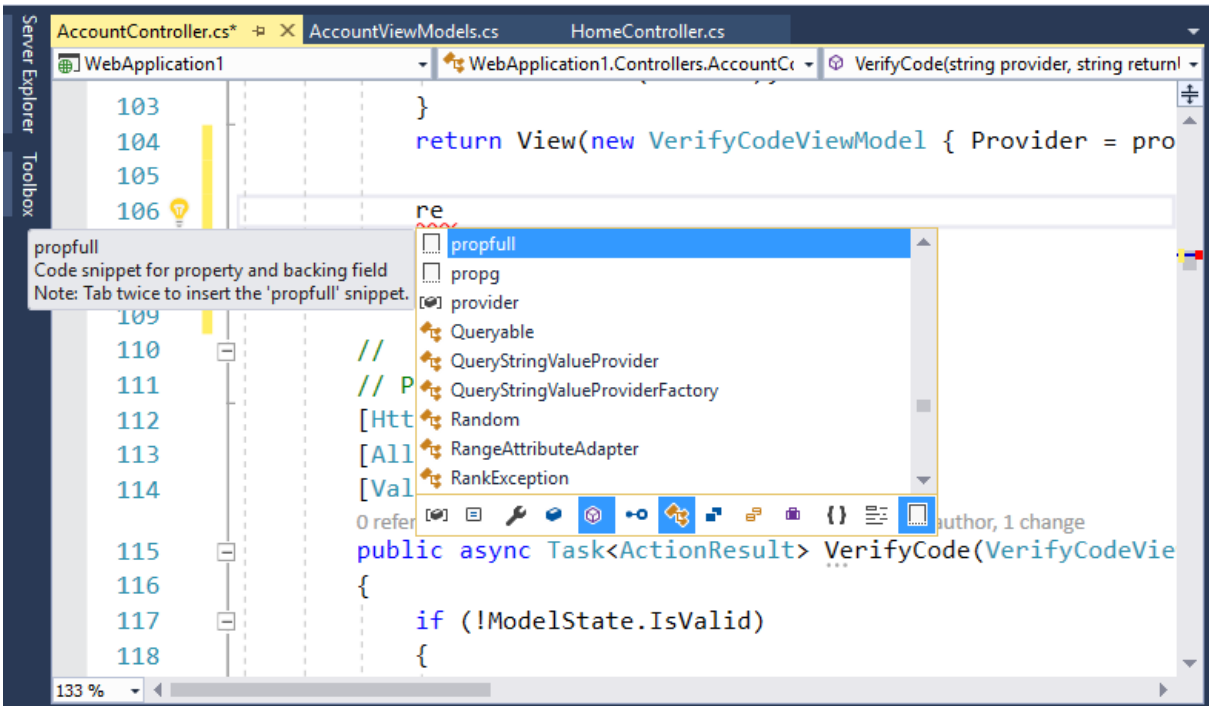

Figure 9: Filtering IntelliSense Search

Click an icon to add or remove the specified filter. This new option dramatically simplifies the way you can leverage word completion, especially when you know in advance what kind of objects you need to work with.

Code navigation made easier

Moving quickly between type definitions and member invocations or assignments is crucial for productivity, especially with dozens of code files. Visual Studio 2017 introduces a number of interesting improvements to code navigation, thereby enhancing your productivity.

Find All References

Find All References is a popular tool window you can use to see where and how an object or member has been used across your solution. It is still more useful for developers who cannot take advantage of the CodeLens tooling, which is only available in the Enterprise edition. With previous versions of Visual Studio, Find All References showed a list of lines of code where an object or member was used, including its definition, and provided an option to double-click a line in the list and be immediately redirected to the line of code in the editor. In Visual Studio 2017, Find All References groups object and member references by project and then by type, with syntax colorization and a more intuitive user interface. Figure 10 shows an example based on references to a class called **`Person`**, which is used in two different projects.

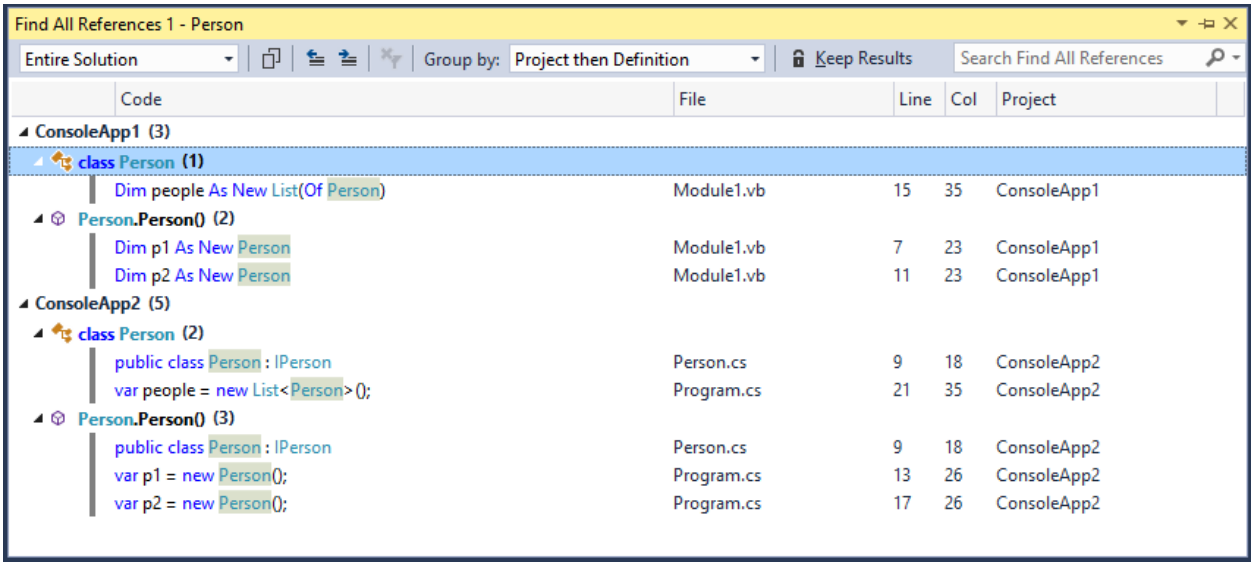

Figure 10: Find All References Groups References with Syntax Colorization

As you can see, the new view makes it easier to see where and how an object has been used. The tool window shows the number of references near the project name, its types, and their members. You can still double-click an item to open the code editor on the selected line. If you hover over a reference, a colorized tooltip shows a preview of the code block that references the object or member (see Figure 11).

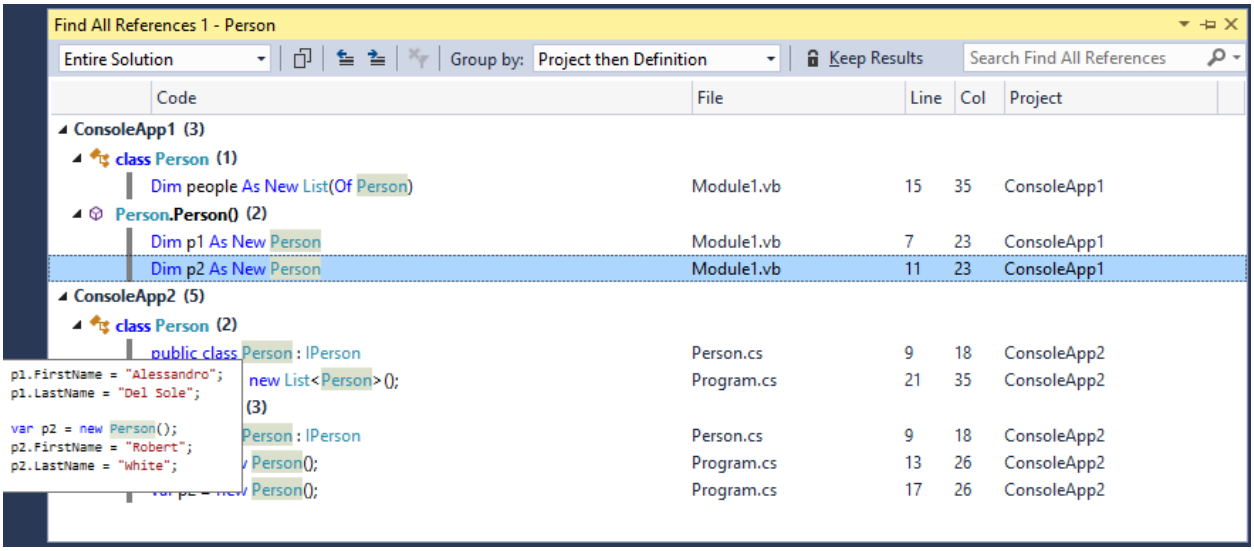

Figure 11: Colorized Tooltip Shows a Preview of the Code Block Referencing an Object

By default, Find All References shows references across the entire solution. You can filter the list through open documents, the current project, the current document, and documents with pending changes. Also, by default, Find All References groups items by project, then by object definition. Table 2 shows a list of available grouping objects you can find in the **Group by** combo box.

Table 2: Grouping Options for Find All References

Option	**Description**
Project then Definition	Groups by project, then by object definition (default).
Definition Only	Groups by object or member definition, without the project hierarchy.
Definition then Project	Groups by object definition, then by project.
Definition then Path	Groups by object definition, then by the path of the code file that contains the definition.
Definition, Project then Path	Groups by object definition, then by project, and then by the path of the code file.

You can also search among results with the **Search Find All References** text box and lock the current results by clicking **Keep Results**.

Navigating code with Go To

Visual Studio 2017 introduces a new navigation feature called **Go To**, which replaces and improves another popular tool known as Navigate To. To enable Go To, just press Ctrl+T. A navigation pop-up appears and automatically lists all the occurrences of the identifier where the cursor was when you pressed Ctrl+T. Figure 12 shows an example.

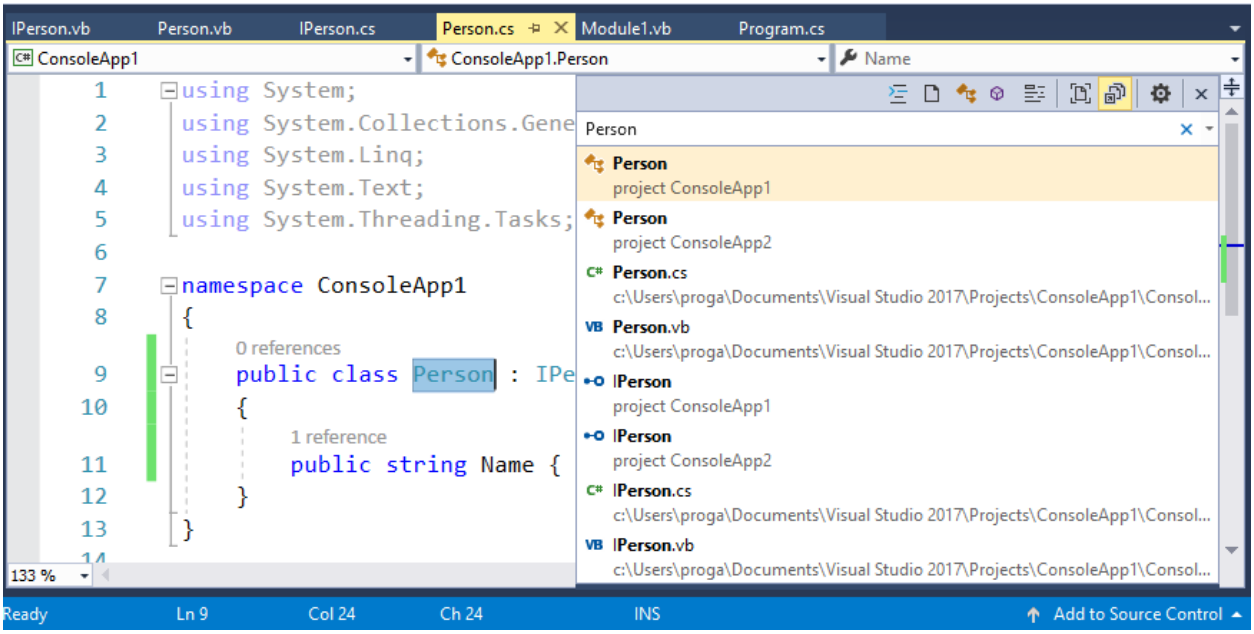

Figure 12: Go To Listing Occurrences of the Specified Identifier

You can refine your search by typing in the search bar. The list will be automatically updated with any words matching what you typed. Like IntelliSense, Go To has a new toolbar at the top that allows for easy filtering based on the object type. You can filter by code file, type, method, object members, and line of code. With Go To, you can only apply one filter at a time. Notice that when you click the filter buttons, the search box shows special characters before a word or identifier. For instance, the **`t`** character preceding any words in the search box will filter the list by type, while the **`#`** character will filter by method. Other supported characters are **`f`** (files), **`m`** (members), and **`:`** (line numbers). These are very useful. If you know in advance what kind of object or member you are searching for, they can speed up your search.

Go To has settings that can be customized. If you click the **Show Settings** button at the right corner of the tool bar, you will be able to decide whether to show a preview tab or enable a detailed view. Figure 13 shows a customized view that filters by type and shows details such as the project, code file, and line number containing the selected type.

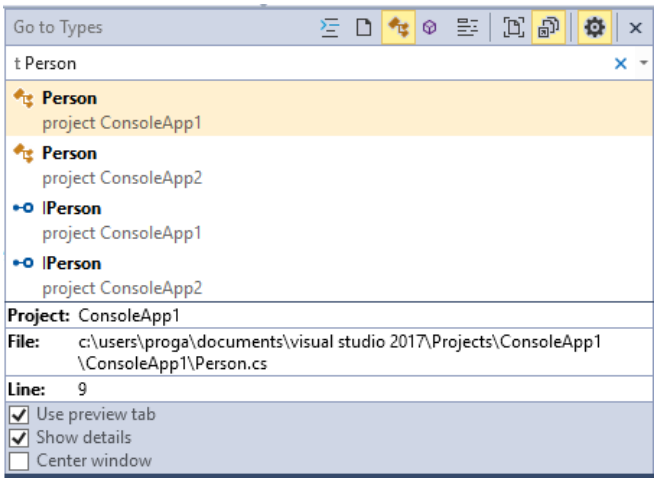

Figure 13: Customizing the Go To View

Actually, Ctrl+T is a shortcut for the **Go To All** command in the **Edit** menu. The Edit menu has additional Go To commands (e.g., Go To Type and Go To Member), each mapped to a filter in the Go To tool window. If you select one of these commands, Go To appears with the appropriate filter enabled. This can speed up your code navigation, especially if you use the provided keyboard shortcuts.

Structure guide lines

Visual Studio 2017's code editor introduces a feature known as **Structure Visualizer** that was previously available in the Productivity Power Tools extension for Visual Studio 2015. With this feature enabled, the editor draws structure guide lines—small gray vertical lines near each code block that make it easier to view the structure of your code. If you hover over structured guide lines, the editor shows a tooltip with a preview of the parent code for the current block. This is demonstrated in Figure 14.

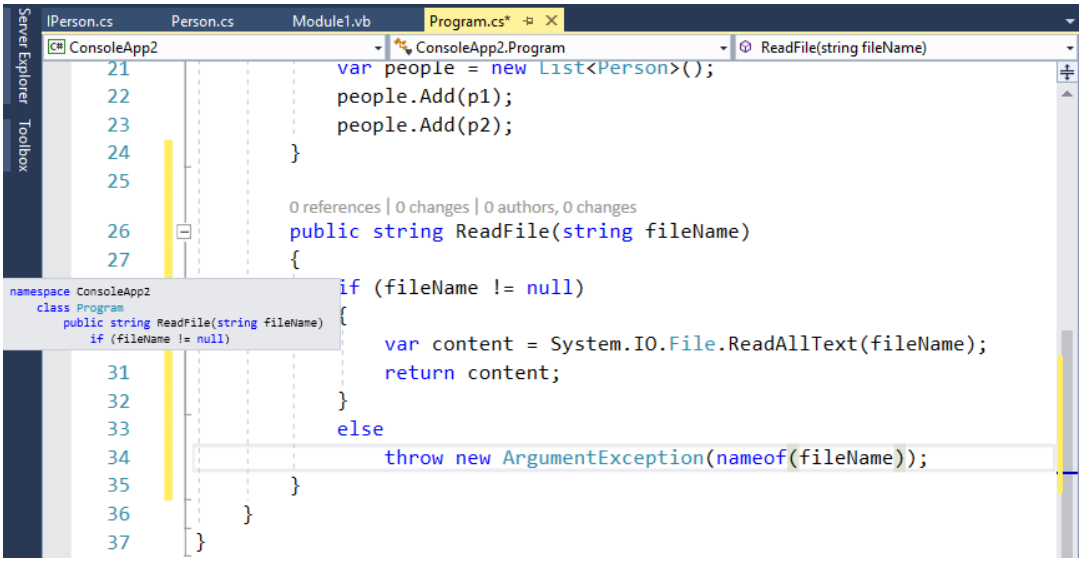

Figure 14: Visualizing Code Structure with Guide Lines

This feature is very useful with long code blocks, and it is enabled by default. If you want to disable it, go to **Tools** > **Options** > **Text Editor**, and clear the **Show structure guide lines** option.

Roslyn code analysis

Along with Visual Studio 2017, Microsoft is releasing C# 7.0 and Visual Basic 15 with an updated version of the .NET Compiler Platform, also known as Project Roslyn. The .NET Compiler Platform provides open source C# and Visual Basic compilers with rich code analysis APIs. With this platform, compilers are offered as a service and developers can take advantage of their APIs to perform a number of operations against source code. If you want to know more about Roslyn, you can read the e-book *Roslyn Succinctly* (also written by me). Starting with Visual Studio 2015, a huge number of tools, including the code editor, are no longer powered by the IDE itself; instead, they are powered by Roslyn. These tools include the live code analysis that detects issues while typing, and code refactoring. As described in *Roslyn Succinctly* and *Visual Studio 2015 Succinctly*, when the compiler detects code issues as you type, the editor offers the light bulb and quick actions to fix an issue or to refactor a code block. Visual Studio 2017 takes some steps forward, providing new refactoring tools and an enhanced coding experience based on Roslyn.

More Roslyn refactorings

The new release of Roslyn offers additional interesting code refactorings. The following is a detailed list of what's new.

Simplify object initialization

The first new refactoring simplifies object initialization. It replaces an object initialization based on property assignments with another one based on object initializers. Figure 15 shows this refactoring in action. Notice that when the code editor detects a possible refactoring for an object initialization, it underlines the constructor invocation with three gray dots.

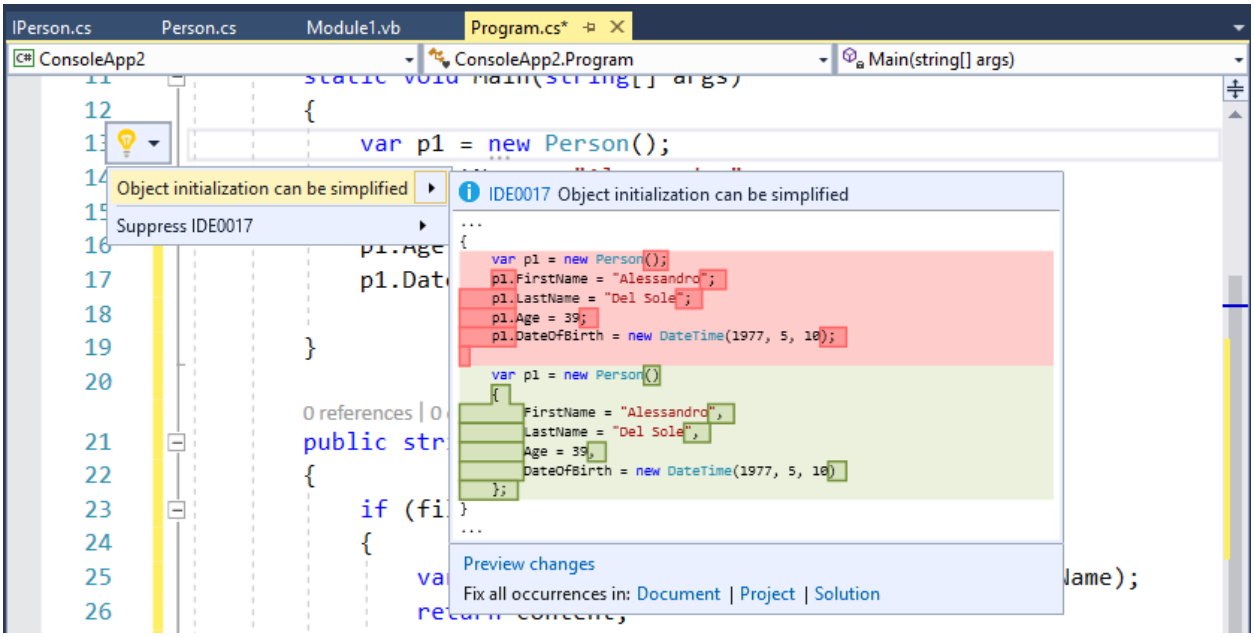

Figure 15: The Object Initialization Refactoring

This code refactoring also provides the proper indentation when applied.

Convert to interpolated string

Another very useful refactoring converts an invocation to **`string.Format`** to an interpolated string. This is shown in Figure 16.

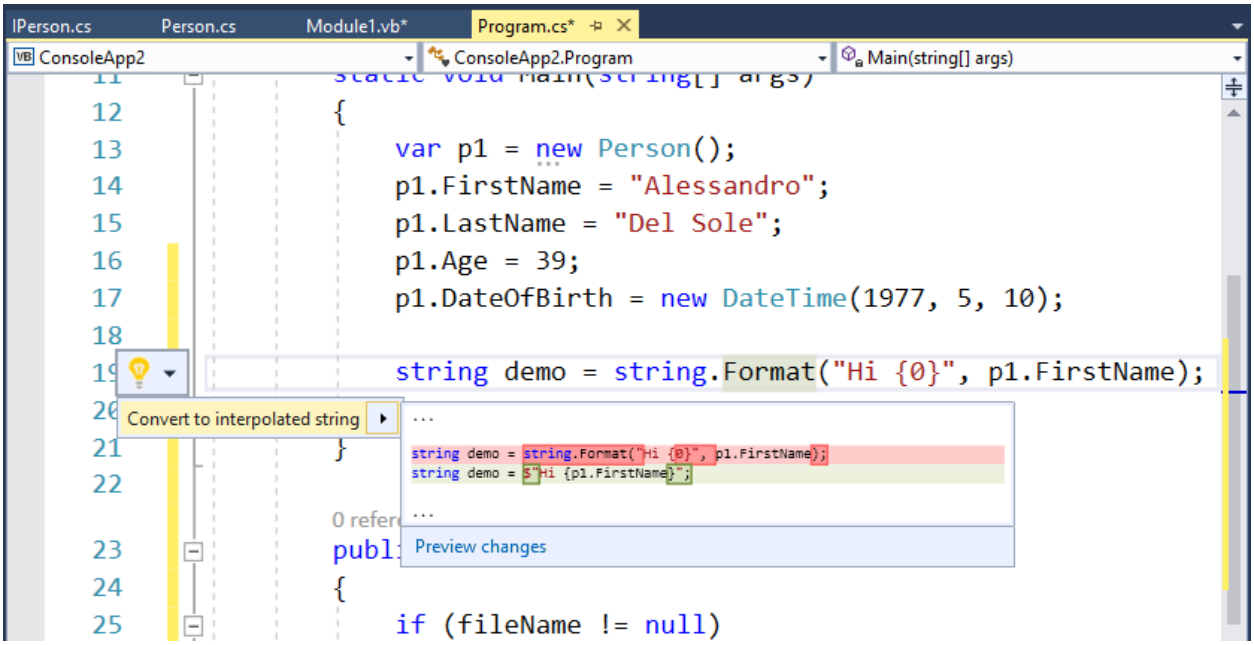

Figure 16: Converting string.Format to String Interpolation

You can enable this refactoring by right-clicking the **`string.Format`** invocation, not the assignment. If you want to know more about string interpolation, visit the documentation page on MSDN.

Move type to matching file

Suppose you have a class that is not defined within an individual file but rather is defined inside a code file that contains other type definitions or, more generally, other code. A new Roslyn refactoring allows you to quickly move the specified type into a new file that will have the same name as the type. For instance, Figure 17 shows a class called **`Person`** that is defined inside Program.cs. If you right-click **`Person`**, you will see a refactoring that offers to move the type into a new file called Person.cs.

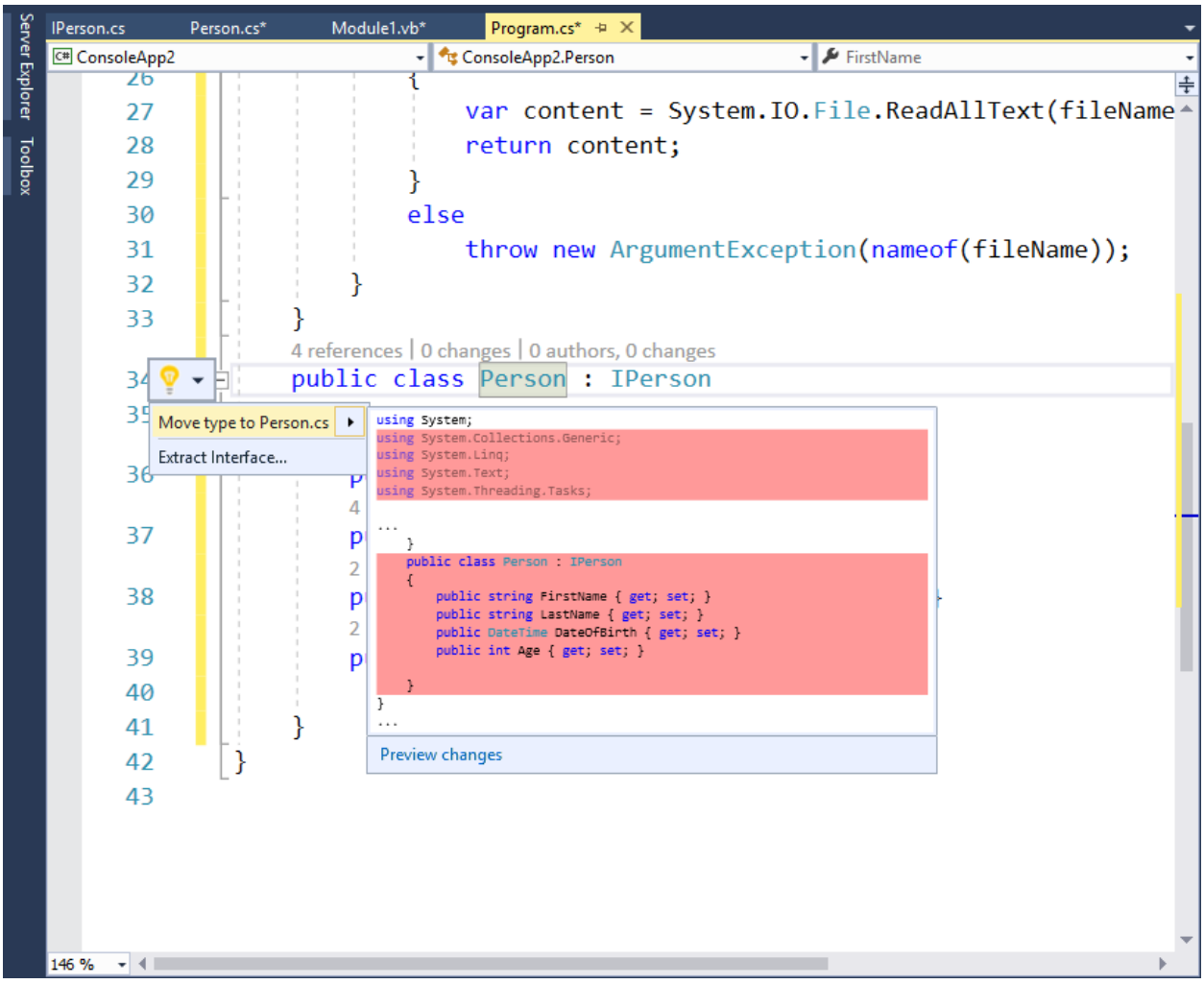

Figure 17: Moving a Type Definition into a Matching File

This feature is extremely useful if you use it to define multiple types in one code file. The only caveat is that, if the target file already exists, Visual Studio will generate a new file and will not use the existing one. In the current example, if Person.cs already exists, then Visual Studio will generate Person1.cs and place the type into the newly created file.

Synchronizing type name and file name

If you have a type definition inside a file whose name doesn't match the type name, a new refactoring will simplify the work of keeping the type name and file name in sync. For instance, if you have a **`Person`** class defined inside a file called Human.cs, you can either rename the file or rename the type to make them match. Figure 18 shows this refactoring in action, which you enable by right-clicking a type name.

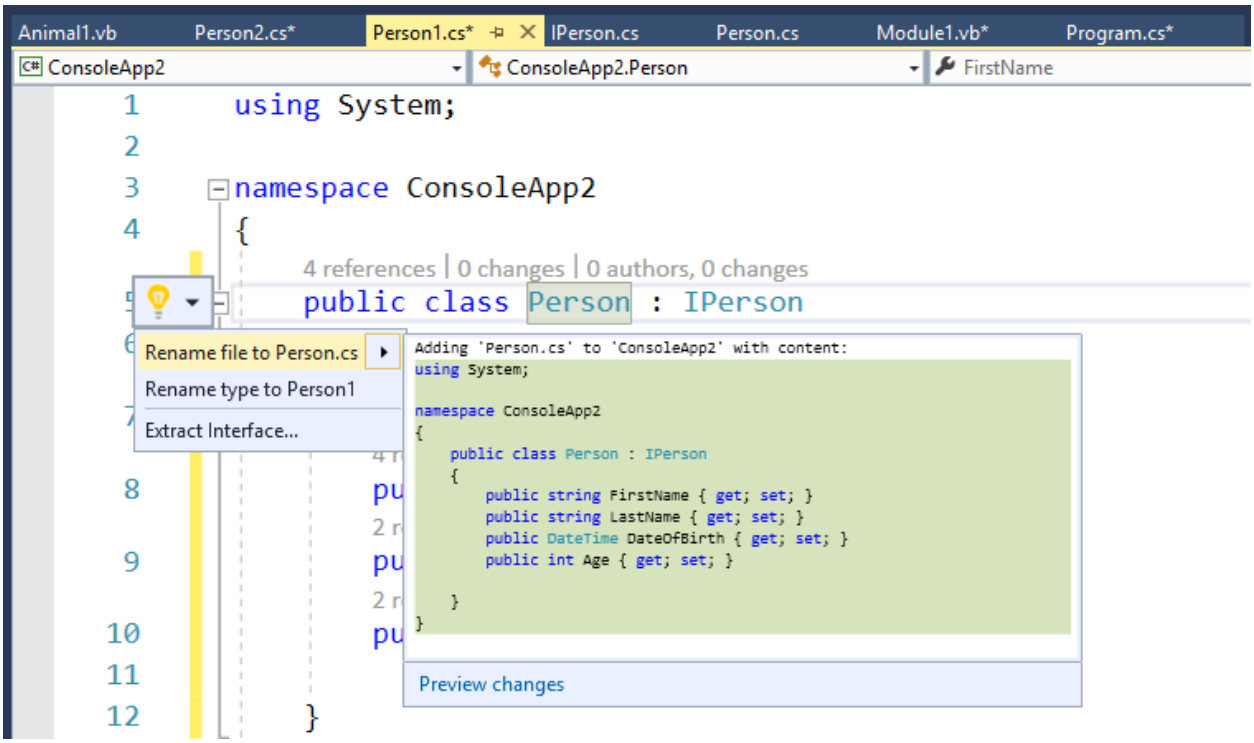

Figure 18: Keeping Type Name and File Name in Sync

The rename file refactoring offers to rename the file to match the type name. Alternatively, rename type offers to rename the type to match the file name.

Inline out variable declarations

Note: This refactoring is not available in Visual Basic 15.

C# 7.0 introduces a new feature known as out variables, which provide the ability to declare a variable right at the point where it is used as an out argument. Roslyn now offers a new refactoring to support this feature, as shown in Figure 19.

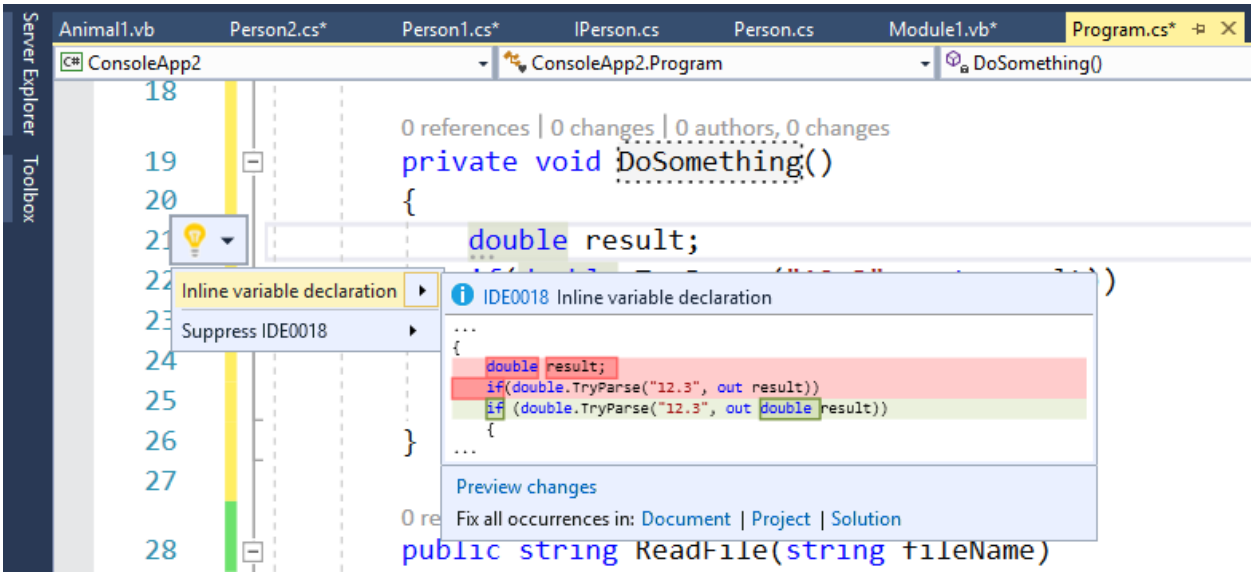

Figure 19: Introducing an Inline Out Variable

With the inline variable declaration refactoring, you can simplify code that uses out variables introducing an inline declaration.

Controlling live analysis with code style

With Roslyn, compilers can report code issues as you type, no matter what kind of Roslyn-based IDE is being used. With Visual Studio and before Roslyn, the code editor could only analyze code for rules designed at Microsoft. One of the biggest advantages of Roslyn is that it allows you to introduce your own analysis rules (with fixes and refactorings), and the compiler will report issues for code that does not adhere to those rules. While we certainly have full control over rules we write, before Visual Studio 2017 we had no control over rules coded at Microsoft, except for disabling warnings. For instance, in Visual Studio 2015 the background compiler always reports as redundant the usage of the **`this`** and **`Me`** keywords in C# and VB unless the keyword is used to reference a member that has the same name as the variable we are assigning (e.g., **`this.name = name`**).

As another example, in Visual Studio 2015 the compiler always suggests that we replace the Framework type names with the corresponding keywords; for instance, it suggests we use **`int`** (C#) or **`Integer`** (VB) instead of **`System.Int32`**. However, there are plenty of reasons why you might want to use those coding styles. In order to give developers the opportunity to decide which coding styles they want to use, with Visual Studio 2017, Microsoft has introduced a new tool called code style. This tool enables you to decide how the compiler should treat some of our coding preferences, including naming. You reach the code style settings by selecting **Tools** > **Options** > **Text Editor**, then **C#** or **Visual Basic**, and finally the **Code Style** node. I will provide a description based on C#, but the same applies to Visual Basic (I will later summarize the available preferences for both). In C#, code style has general, formatting (C# only), and naming options.

Tip: Behind the scenes, this feature is based on the EditorConfig file format. If you wish to further configure code styles, you can read this blog post from the .NET Team at Microsoft.

General code style

The general settings are probably the most interesting options for altering code style. They allow you to change preferences over some built-in coding rules. Each group of preferences allows you to specify the preference and the severity level. If your code does not match the preference, the compiler will report a suggestion, warning, or error, depending on the severity level you choose. The default severity is None, which means that the compiler will simply ignore the preference and report nothing.

Tip: Remember that the Error severity level will prevent you from building your project until the code issue is solved.

For instance, suppose you want to use the **`this`** keyword when referencing an object's property and that the compiler should report a suggestion if your code does not use the keyword. Figure 20 shows how to accomplish this.

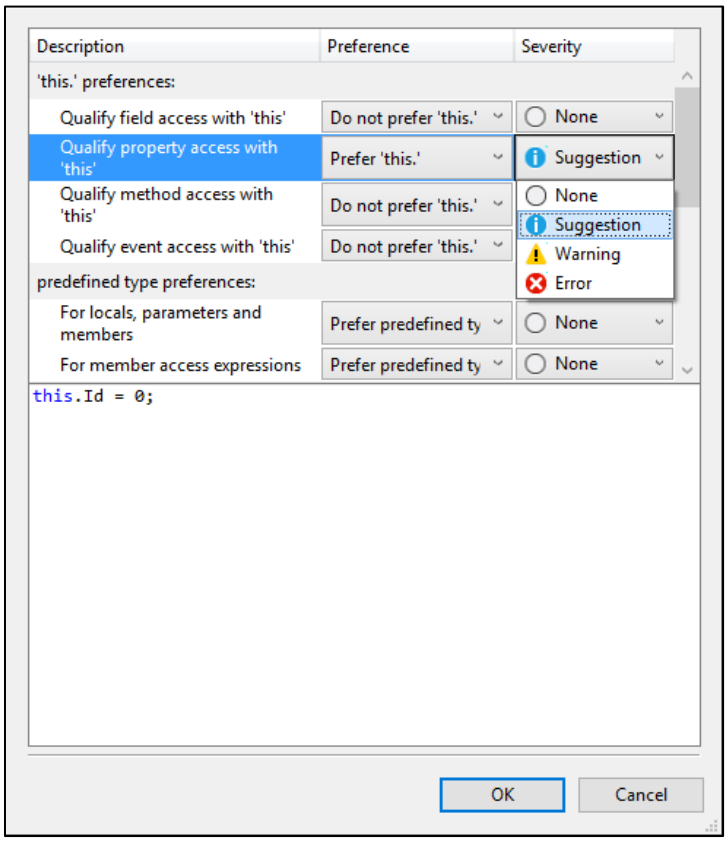

Figure 20: Changing Code Style Preferences

Notice how a preview of your code is shown at the bottom of the dialog in Figure 20, whereas Figure 21 shows the result of setting this preference—the compiler reports a suggestion over the property assignment without **`this`**, and the light bulb offers to fix the issue.

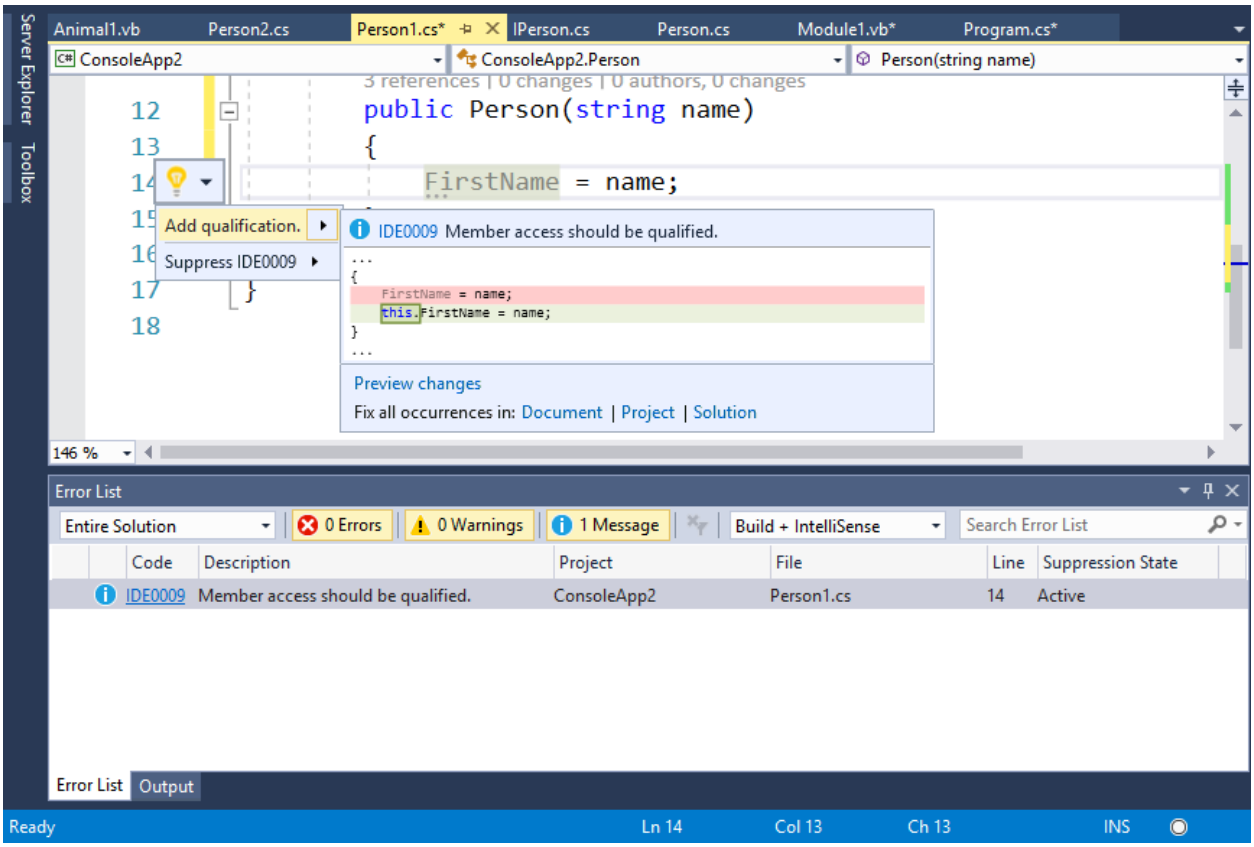

*Figure 21: Changing the Code Style for the **`this`** Keyword*

As another example, suppose you want to control the code style for the **`var`** keyword. More specifically, suppose you want to avoid using **`var`** with built-in types and you want to avoid **`var`** when the type is apparent from assignment expressions. Figure 22 shows how to accomplish this.

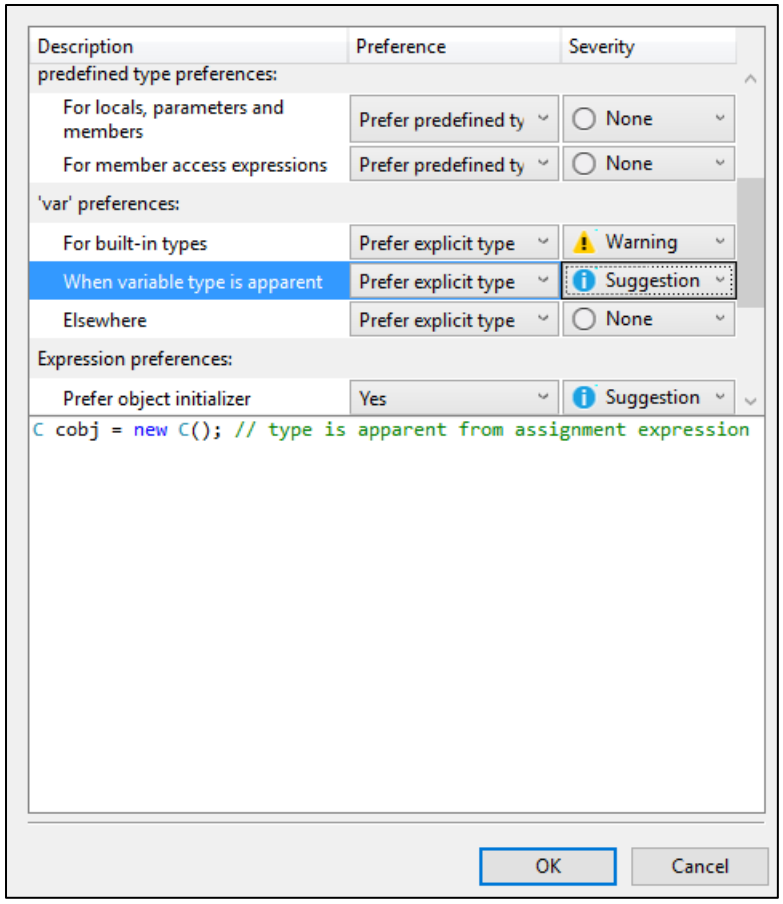

*Figure 22: Changing the Code Style for the **var** Keyword*

In the first case, the compiler will report a warning each time you use **var** instead of a built-in type, and it will offer the proper code fix, as shown in Figure 23.

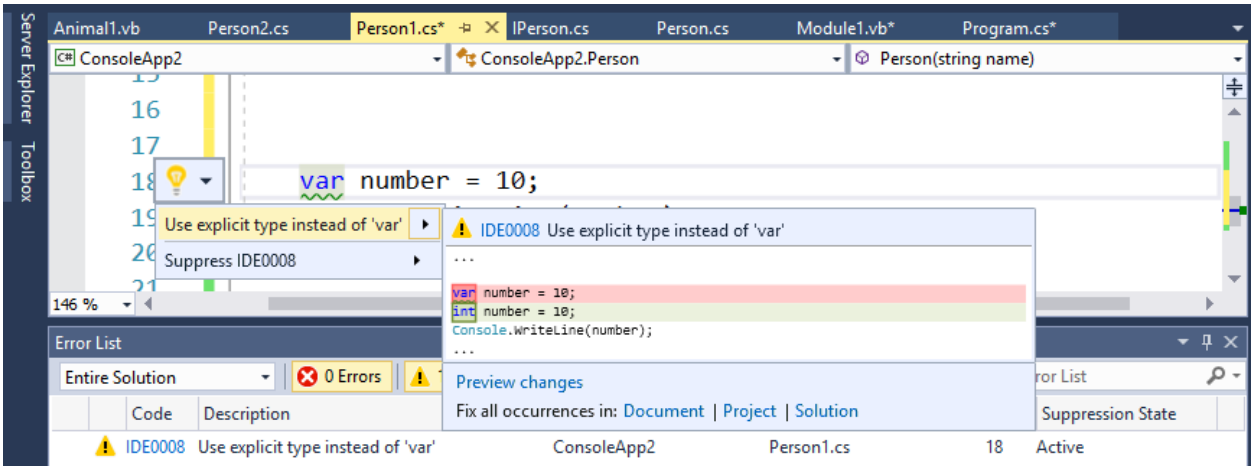

*Figure 23: Compiler Detects and Fixes Improper **var** Usage*

In the case of apparent assignments, Figure 24 shows how the compiler detects that assigning an instance of the **`Person`** type to a variable is an apparent assignment. The compiler therefore reports the suggestion to replace **`var`** with the explicit **`Person`** type, offering the proper fix through the light bulb.

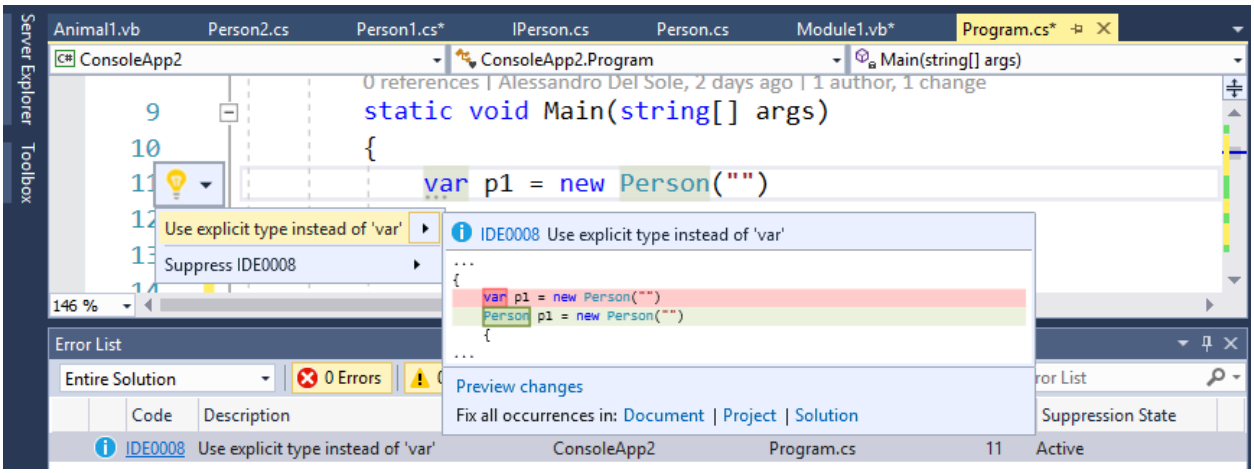

*Figure 24: Compiler Detects an Improper **`var`** Usage with an Apparent Assignment*

Tables 3 and 4 summarize some of the available preferences for C# and Visual Basic, respectively. Remember that a code preview for each style is shown in the **Options** dialog.

Table 3: Code Style Preferences for C#

Preference Group	**Preference**	**Description**
'this.' preferences	Qualify field access with 'this'	Sets whether you should prefer the **`this`** keyword to qualify access to fields.
'this.' preferences	Qualify property access with 'this'	Sets whether you should prefer the **`this`** keyword to qualify access to properties.
'this.' preferences	Qualify method access with 'this'	Sets whether you should prefer the **`this`** keyword to qualify access to methods.
'this.' preferences	Qualify event access with 'this'	Sets whether you should prefer the **`this`** keyword to qualify access to events.
predefined type preferences	For locals, parameters, and members	Sets whether you should prefer the predefined type (e.g., **`int`**, **`double`**) over the Framework type (e.g., **`System.Int32`**, **`System.Double`**) with local variables, method parameters, and object members.

Preference Group	Preference	Description
predefined type preferences	For member access expressions	Sets whether you should prefer the predefined type over the Framework type when accessing type members.
'var' preferences	For built-in types	Sets whether you should prefer an explicit type instead of the **`var`** keyword with built-in types such as **`int`**, **`string`**, **`double`**, etc.
'var' preferences	When variable type is apparent	Sets whether you should prefer an explicit type instead of the **`var`** keyword when the type can be easily discovered by the assignment of an expression.
'var' preferences	Elsewhere	Sets whether you should prefer an explicit type instead of the **`var`** keyword in all other cases.
Code block preferences	Prefer braces, For methods, For constructors, For operators, For properties, For indexers, For accessors	Sets whether code blocks should be implemented using a block body or an expression body (that is, with lambda expressions).
Expression preferences	Prefer object initializer	Sets whether you should prefer an object initializer when instantiating an object and populating its members. When reported, this invokes the simplify object initialization refactoring described in the previous section.
Expression preferences	Prefer collection initializer	Sets whether you should prefer a collection initializer when instantiating a collection of objects. When reported, the editor offers the proper fix.
Expression preferences	Prefer pattern matching over 'is' with 'cast' check	Sets whether code should enforce cast check with pattern matching.
Expression preferences	Prefer pattern matching over 'as' with 'null' check	Sets whether code should use pattern matching instead of using the **`as`** conversion operator for null checks.
Expression preferences	Prefer explicit tuple name	Tuples in C# 7.0 allow for implicit names. This option sets whether code should use explicit or implicit tuple names.

Preference Group	Preference	Description
Variable preferences	Prefer inline variable declaration	Sets whether you should prefer using inline variable declaration. When reported, this invokes the inline out variable declaration refactoring described in the previous section.
'null' checking'	Prefer throw expression	With null checks, sets whether you should prefer using the new throw expression feature in C# 7.0 with the null-coalescing operator (**??**) instead of comparing variables to **null**.
'null checking'	Prefer conditional delegate call	With null checks, sets whether you should prefer the null-conditional operator (**?.**) and delegates' **Invoke** method over evaluating an object with the **!= null** expression.
'null checking'	Prefer coalesce expression	With null checks, sets whether you should prefer the coalescing operator (**??**) instead of expressions based on the conditional operator (**?.**)
'null checking'	Prefer null propagation	With null checks, sets whether you should prefer the null propagation (**?.**) instead of conditional comparisons based on the conditional operator **?:**.

Table 4: Code Style Preferences for Visual Basic

Preference Group	Preference	Description
'Me.' preferences	Qualify field access with 'Me'	Sets whether you should prefer the **Me** keyword to qualify access to fields.
'Me.' preferences	Qualify property access with 'Me'	Sets whether you should prefer the **Me** keyword to qualify access to properties.
'Me.' preferences	Qualify method access with 'Me'	Sets whether you should prefer the **Me** keyword to qualify access to methods.
'Me.' preferences	Qualify event access with 'Me'	Sets whether you should prefer the **Me** keyword to qualify access to events.

Preference Group	Preference	Description
Predefined type preferences	For locals, parameters, and members	Sets whether you should prefer the predefined type (e.g., **`Integer`**, **`Date`**) over the Framework type (e.g., **`System.Int32`**, **`System.DateTime`**) with local variables, method parameters, and object members.
Predefined type preferences	For member access expressions	Sets whether you should prefer the predefined type over the Framework type when accessing type members.
Expression preferences	Prefer object initializer	Sets whether you should prefer an object initializer when instantiating an object and populating its members. When reported, this invokes the simplify object initialization refactoring described in the previous section.
Expression preferences	Prefer collection initializer	Sets whether you should prefer a collection initializer when instantiating a collection of objects. When reported, the editor offers the proper fix.
Expression preferences	Prefer explicit tuple name	Tuples in Visual Basic 15 allow for implicit names. This option sets whether code should use explicit or implicit tuple names.
'nothing' checking	Prefer coalesce expression	Sets whether you should prefer calling the **`If`** operator with short-circuit evaluation for null checks.
'nothing' checking	Prefer null propagation	Sets whether you should prefer the null conditional operator **`?.`** for null checks.

This feature is extremely useful, especially if you find some of the default suggestions from the compiler to be annoying.

Formatting code style

Formatting code style is available only to C#. This makes sense because the primary goal of formatting is to control automatic formatting and indentation of curly braces and code that is or should be inside braces as you type. Formatting also allows you to control how some code blocks or statements should be added to specific constructs. All the available options have self-explanatory names and the **Options** dialog shows a preview of how the code will look with a particular option selected. As an example, Figure 25 shows how a property definition looks with the **Leave block on single line** option selected (under the **Wrapping** node), whereas Figure 26 shows how the same property definition will look with that option not selected.

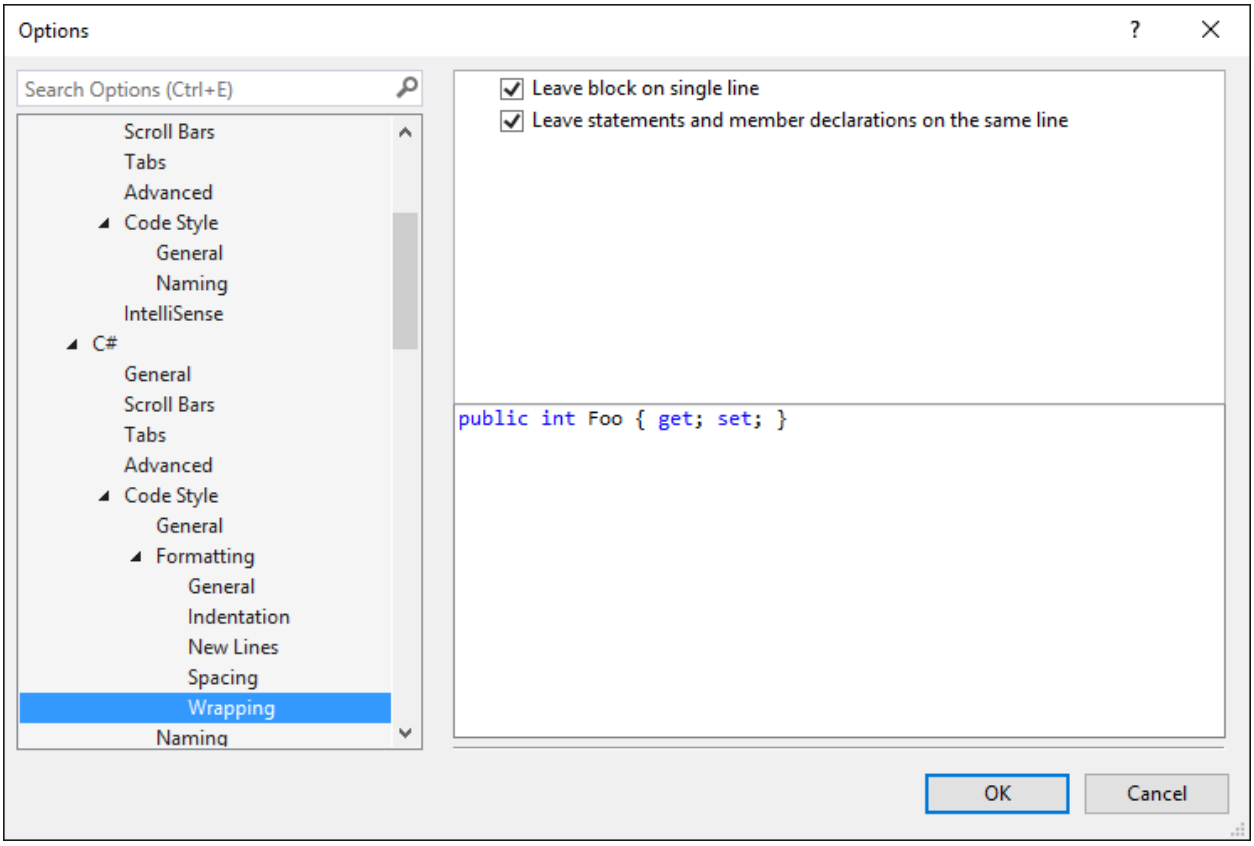

Figure 25: Code Blocks Should Be on a Single Line

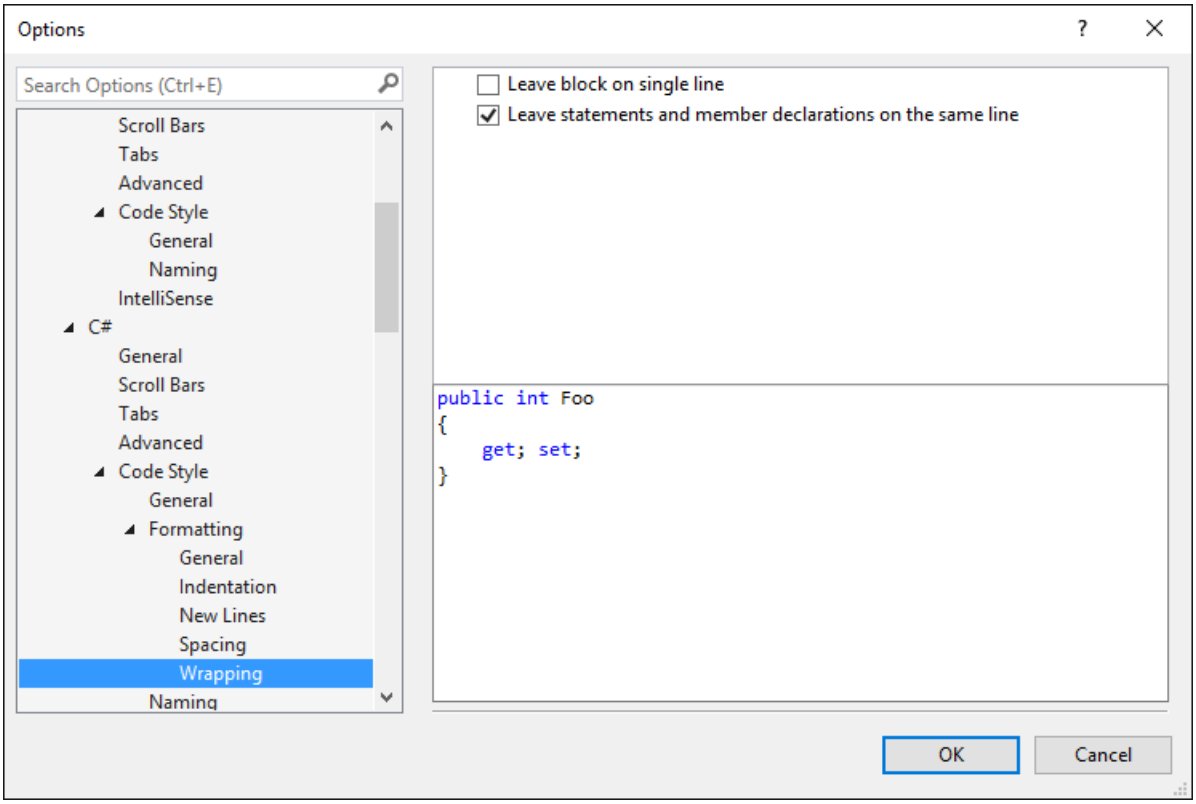

Figure 26: Code Blocks Should Be on Multiple Lines

Naming code style

The naming code style preferences address .NET naming conventions for types and members. With the naming specifications, you can control how the compiler should report issues about code that does not adhere to the specified naming conventions. For instance, if the name of an interface does not begin with **`I`**, the compiler will report a suggestion to fix its name. Each specification can have one style and severity level. The severity level can be None, Suggestion, Warning, or Error. There are three built-in styles and four built-in specifications—these are summarized in Tables 5 and 6, respectively. By default, all specifications have the Suggestion severity level.

Table 5: Built-In Naming Styles

Style name	**Description**
Begins with I	The name of the type or member selected in the specification to which it is assigned should begin with **`I`**.
Pascal Case	The name of the type or member selected in the specification to which it is assigned should follow the Pascal case naming convention.

Table 6: Built-In Naming Specifications

Specification name	Description
Interface	Determines that an interface's name should follow the naming rule assigned in the Style column. The default style is Begins with I.
Types	Determines that a type name should follow the naming rule assigned in the Style column. The default style is Pascal Case.
Non-Field Members	Determines that object members different than fields should follow the naming rule assigned in the Style column. The default style is Pascal Case.

Note: I strongly recommend that you not change these four specifications. They reflect some of the most important general naming conventions in .NET and should be left as they are.

Figure 27 shows the default settings for the Naming code style options.

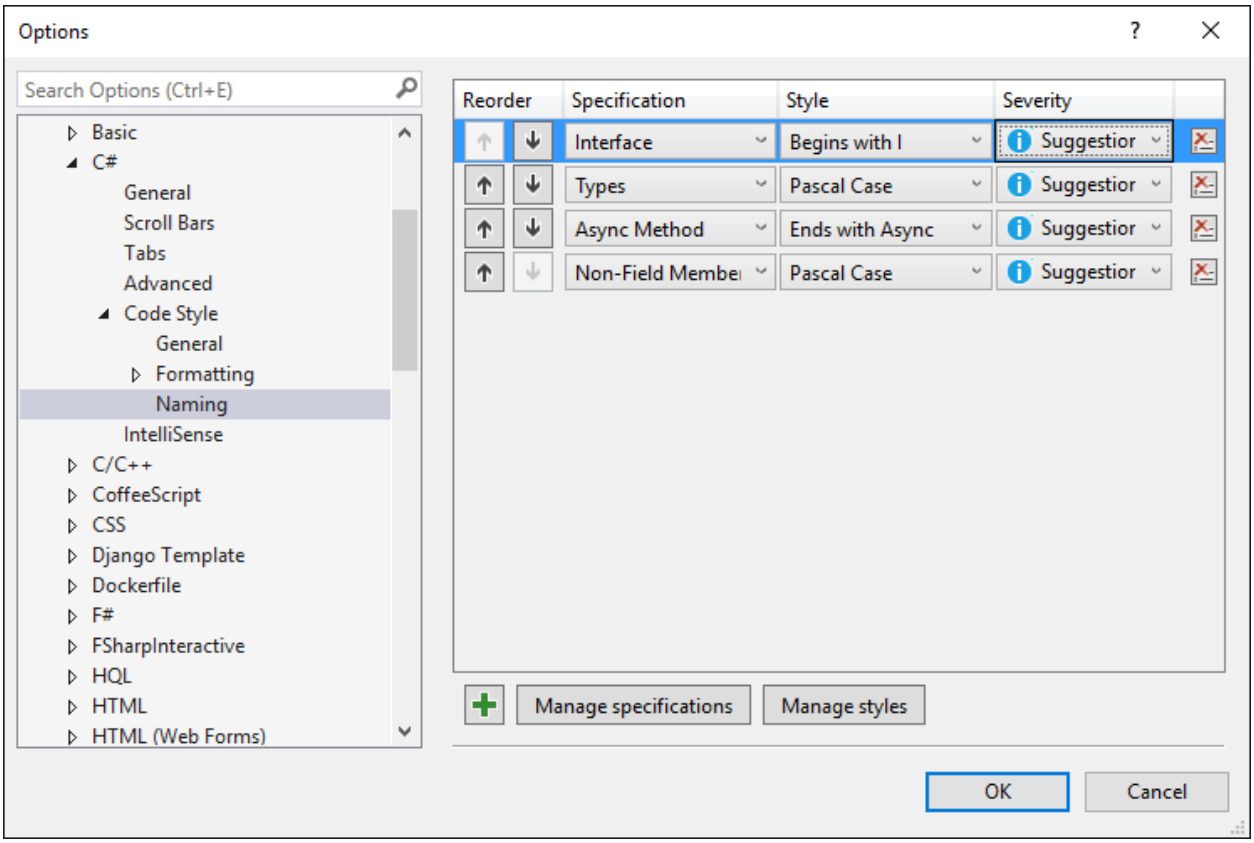

Figure 27: Default Settings for the Naming Code Style

You have significant control over naming settings. In fact, you can add your own specifications and styles, and you can combine multiple styles for the same specification. You can even edit existing specifications and styles by clicking **Manage specifications** and **Manage styles**. For instance, suppose you want to add a new naming convention for private enumerations and you want the style to be camel case. To accomplish this, click **Manage specifications**. This will show the full list of active specifications (see Figure 28).

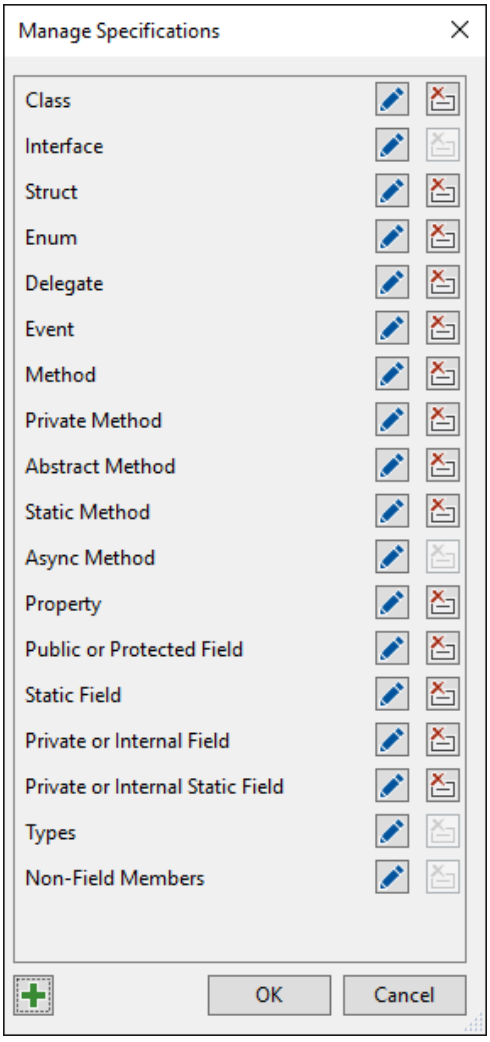

Figure 28: The List of Available Specifications

For each specification, you can click **Edit** (with the pencil icon) to change its properties or **Delete** to remove it (not recommended).

Tip: Editing an existing specification will open the same Symbol Specifications dialog described shortly (and visible in Figure 29).

As you can see, a specification for **`Enum`** types already exists, but let's say you want to add one specifically for private enumerations. Click **Add** (the green **+** icon). In the **Symbol Specification** dialog that appears, you will see a list of available types, modifiers, and accessors. Make sure that only **`enum`** and **`private`** are selected, then assign the **Private Enums** title in the **Symbol Specification Title** text box (see Figure 29) and, finally, click **OK**.

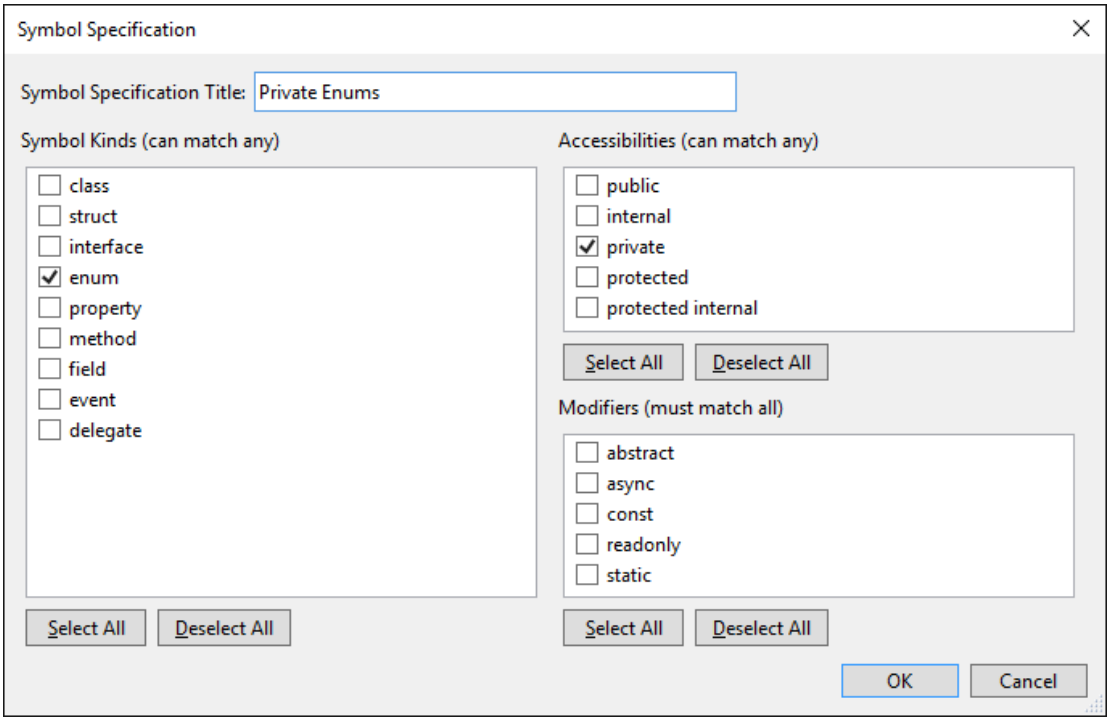

Figure 29: Creating a Custom Specification

Click **OK** again to close the Manage Specifications dialog. At this point, the new specification will be listed in the Naming code style list. Now you need to create a new style, so click **Manage styles**. The **Manage Naming Styles** dialog will show the list of current styles, which contains the three styles described in Table 5. Click **Add** to add a new one. In the **Naming Style** dialog (see Figure 30), provide a title for the new style, such as Camel Case, then select the **camel Case Name** option from the **Capitalization** combo box. As you can see, you also have the option to specify a prefix, a suffix, and a word separator (not required in this case).

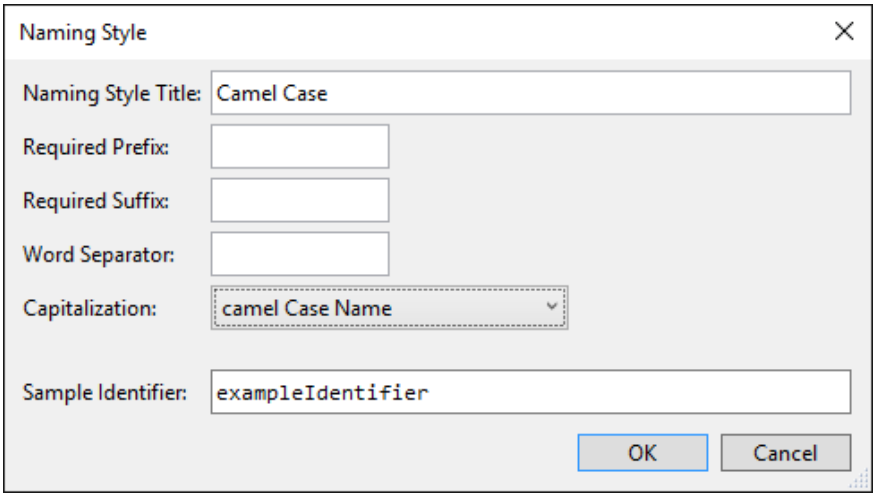

Figure 30: Creating a Custom Style

Click **OK** to close the Naming Style dialog, then click **OK** to close the Manage Naming Styles dialog. At this point, you will be able to assign the new style for the previously created specification, as shown in Figure 31 (notice that a severity level of Warning has been also assigned).

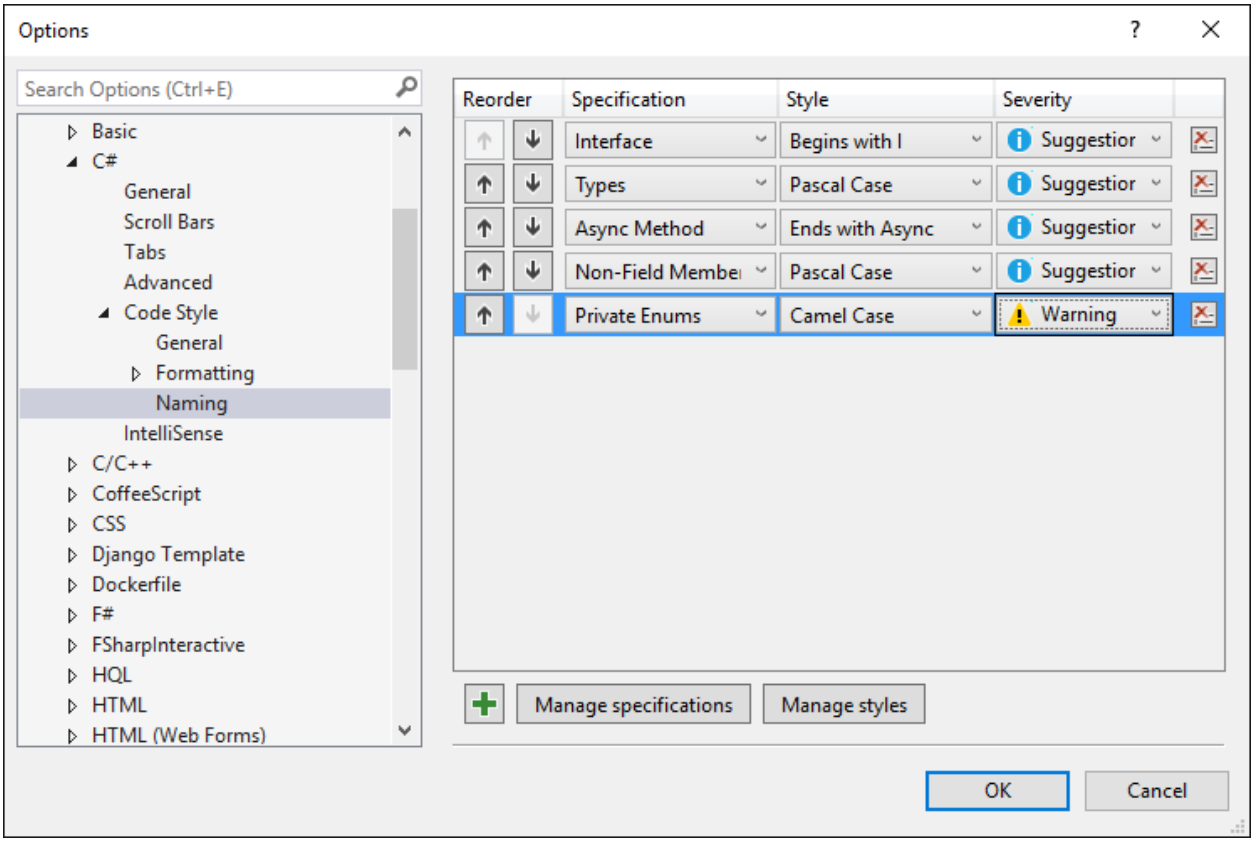

Figure 31: Assigning a Custom Style to a Custom Specification

If you want to see the result of your work in action, you must take an additional step. Because the Types specification includes all types with any visibility accessor, you should edit this specification and remove the private accessor from the list. You can use Figures 28 and 29 as a reference. Of course, we do this only for demonstration purposes, but the default specifications should not be changed. However, with these custom specifications and styles, the compiler now detects a violation of the naming rule for private enumerations, shows a warning message, and offers the appropriate fix, as shown in Figure 32.

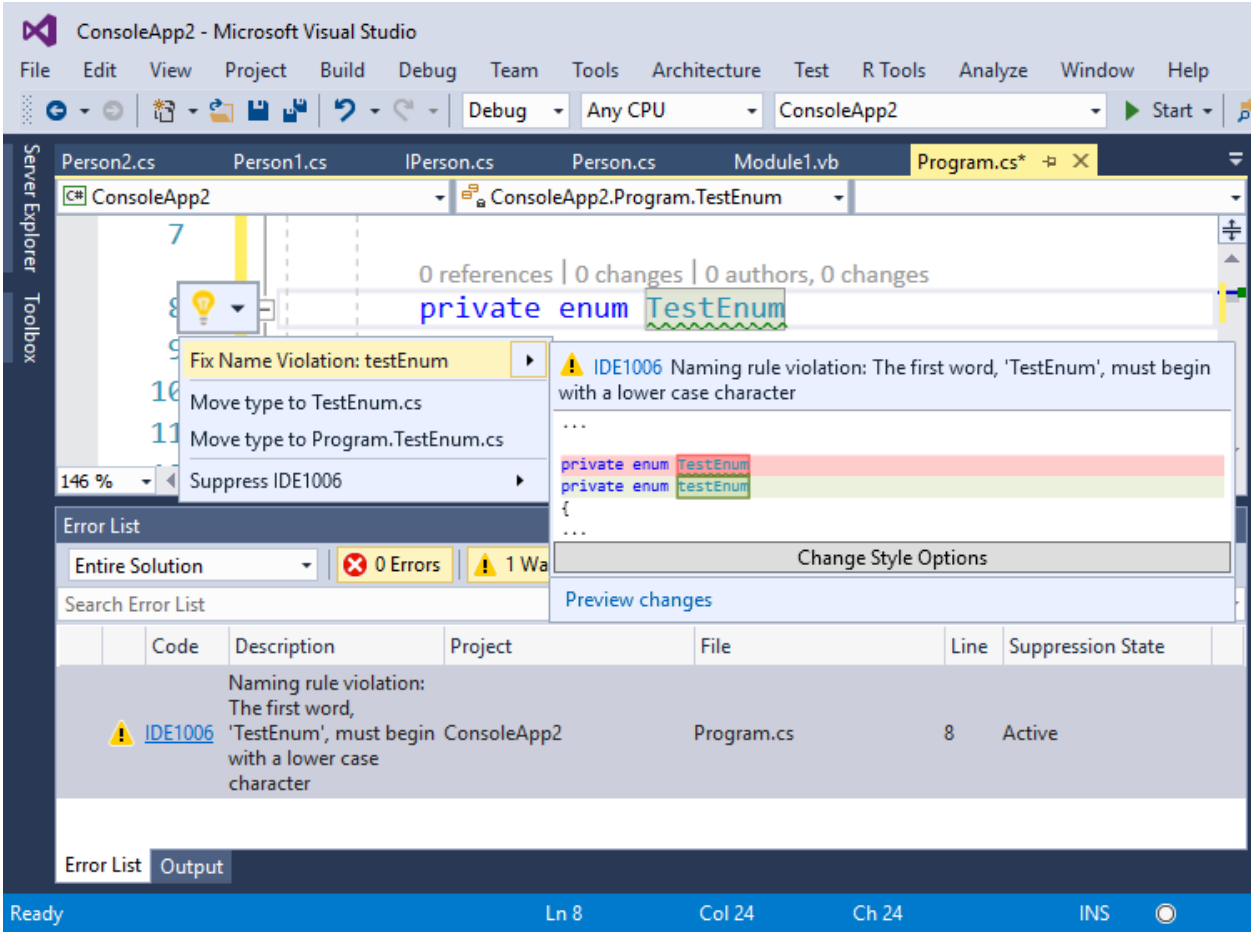

Figure 32: Detection of Custom Naming Rules Violations and a Code Fix

So, you are not limited to creating custom naming rules that will be checked by the live analysis engine, but you will also get code fixes for violations, which is really amazing.

Editing improvements for C++ and F#

Visual Studio 2017 introduces improvements to the code editing experience for C++ and F#, too. For C++, features such as IntelliSense, Go To Definition, and Find All References now rely on a SQLite database that stores the information the IDE needs in order to quickly move among lines of code, and that dramatically improves performance. Also, the code editor for C++ now provides two new refactorings: change signature and extract function. The first refactoring allows you to more quickly change the signature of a method, while the second one allows you to encapsulate a code block into a function. With F#, the code editor now offers improved IntelliSense with better completions and filters, along with type colorization. Also, the F# code editor is now built on Roslyn Workspaces, and many features are now powered by Roslyn, including Go To Definition, Peek Definition, brace matching, indentation, breakpoint resolution, and debugging data tips.

Chapter summary

Microsoft has increased the code editor's productivity for C# and Visual Basic in Visual Studio 2017 by investing in IntelliSense, code navigation, and integrated Roslyn code analysis. IntelliSense has better performance and allows for type filtering, which makes it easier to use the objects and members you need. For code navigation, Find All References now presents grouped lists of objects with syntax colorization. Go To is an improved replacement of Navigate To, with type filtering and quick search among large codebases. Finally, structure guide lines help us understand to which code block a snippet belongs. With live code analysis powered by Roslyn, new refactorings are available to both C# and Visual Basic, and you can now control styles and naming rules coded at Microsoft with the new code style features. Code editor improvements are not limited to C# and Visual Basic—the XAML code editor has exciting new and updated features as well, which is discussed in the next chapter.

Chapter 4 XAML Improvements

The *eXtensible Application Markup Language* (XAML) is the language for designing the user interface of Windows Presentation Foundation (WPF) and Universal Windows Platform (UWP) applications. Therefore, it's crucial for Microsoft. In Visual Studio 2017, the tooling for XAML has been dramatically enhanced. On the performance side, switching between tabs in the XAML code editor is more than 90% faster in some cases, and many investments have been made to avoid delays when typing XAML markup in the editor. Additionally, many other improvements have been made to the coding experience and the diagnostic tools. This chapter will describe such improvements, showing how much more productive you can be in Visual Studio 2017. If you want to try it yourself, simply create a blank UWP or WPF project and open a .xaml file.

Note: This chapter describes features that are common to both WPF and UWP. Chapter 8, "Visual Studio 2017 for mobile development," will address additional improvements that target UWP. Topics discussed in this chapter are not available to XAML for Xamarin.Forms.

XAML Edit and Continue

One of the most important additions to the XAML tooling in Visual Studio 2017 is XAML Edit and Continue. This feature allows us to edit XAML code while the application is running in Debug configuration. The application will immediately reflect those edits without the need to break the application execution and restart after making some edits. Figure 33 shows how you can edit property values in debugging mode (demonstrated by the orange color of the status bar at the bottom of the main Visual Studio shell).

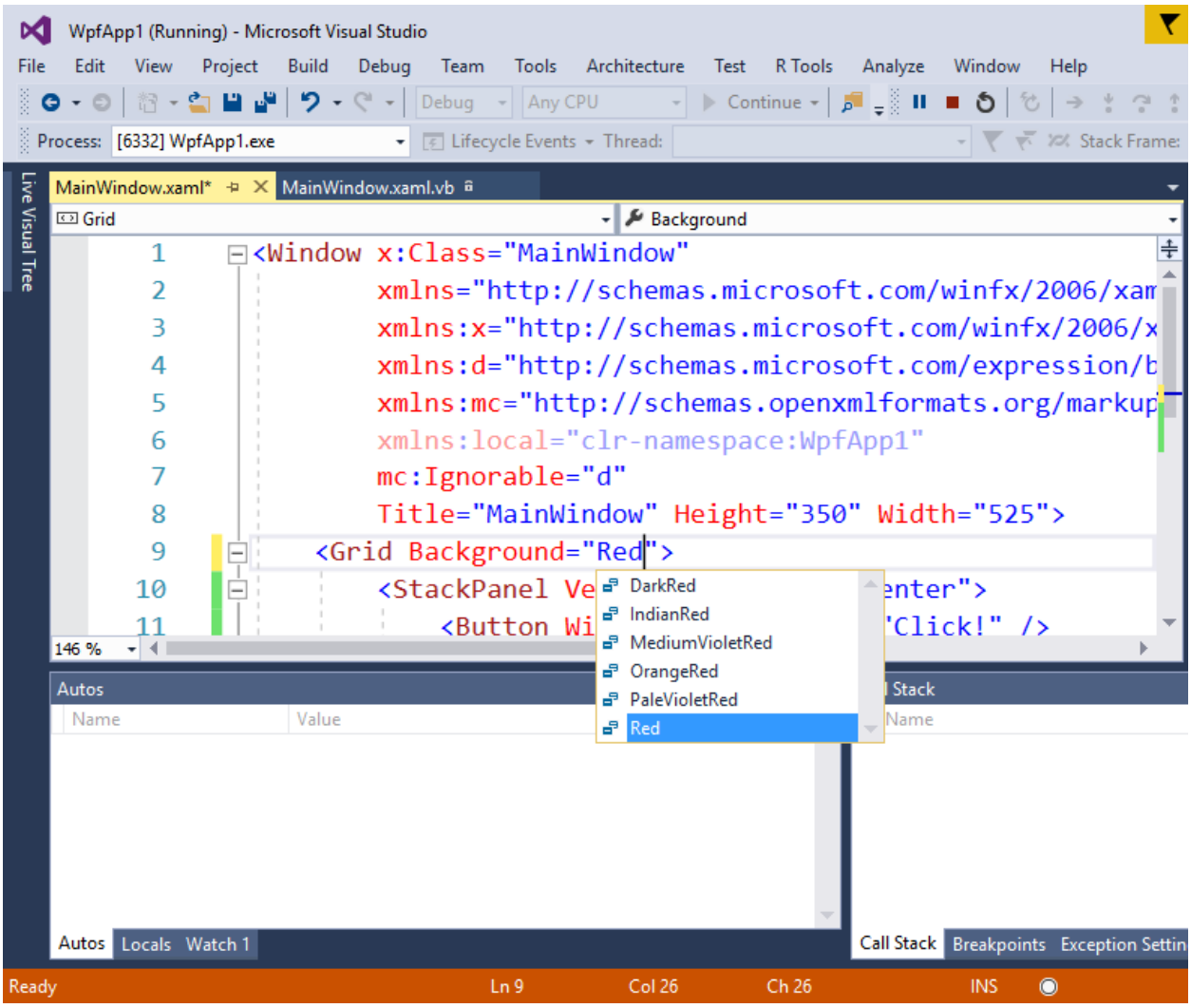

Figure 33: Modifying XAML at Runtime with Edit and Continue

You are not limited to editing controls' or panels' property values—you can also add new controls, new panels, and everything you might need to improve your user interface. You can combine using this feature with the Live Visual Tree and Live Property Explorer windows to get an enhanced experience. Edit and Continue has been a heavily requested feature and is finally available to all editions of Visual Studio 2017.

XAML code editor improvements

More often than not, you will write XAML code manually in order to design or at least to fine-tune the user interface of a WPF or UWP app. For this reason, Microsoft has introduced additional improvements to the XAML code editor that will greatly improve your coding experience. Among others, the XAML editor now has the **structure guide lines** feature we examined in Chapter 3. Additional features address IntelliSense and code refactoring, and they are described in the following paragraphs.

Note: Figures in this chapter have been captured from a UWP project, but the same applies to WPF.

Navigating code with Go To

In Chapter 3, we looked at the Go To feature, which makes navigating code easier and faster. This is also available for the XAML code editor and can be enabled by pressing Ctrl+G.

IntelliSense filtering

IntelliSense for XAML has been updated to show only those members that best match what you type. Additionally, if you type only the capitalized letters of a control name, IntelliSense will show the list of full control names that match those capitalized letters. For instance, if you type **SV** in UWP, IntelliSense will filter the completion list showing only the **ScrollViewer** and **SplitView** element names (see Figure 34).

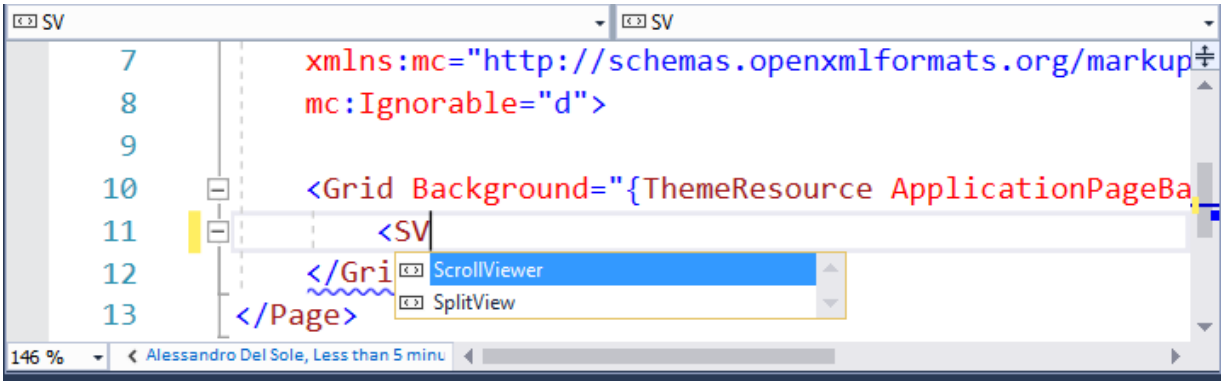

Figure 34: IntelliSense Filtering

Refactoring namespaces

Numerous improvements have been introduced for better management of XAML namespaces. These include features that were previously available to C# and Visual Basic, such as inline renaming, resolving missing namespaces, and removing redundant namespaces.

Removing and sorting namespaces

The XAML code editor detects redundant namespaces by presenting them with a lighter syntax colorization, exactly as with C# and Visual Basic since Visual Studio 2015. You can easily remove redundant namespaces with the help of the light bulb as shown in Figure 35.

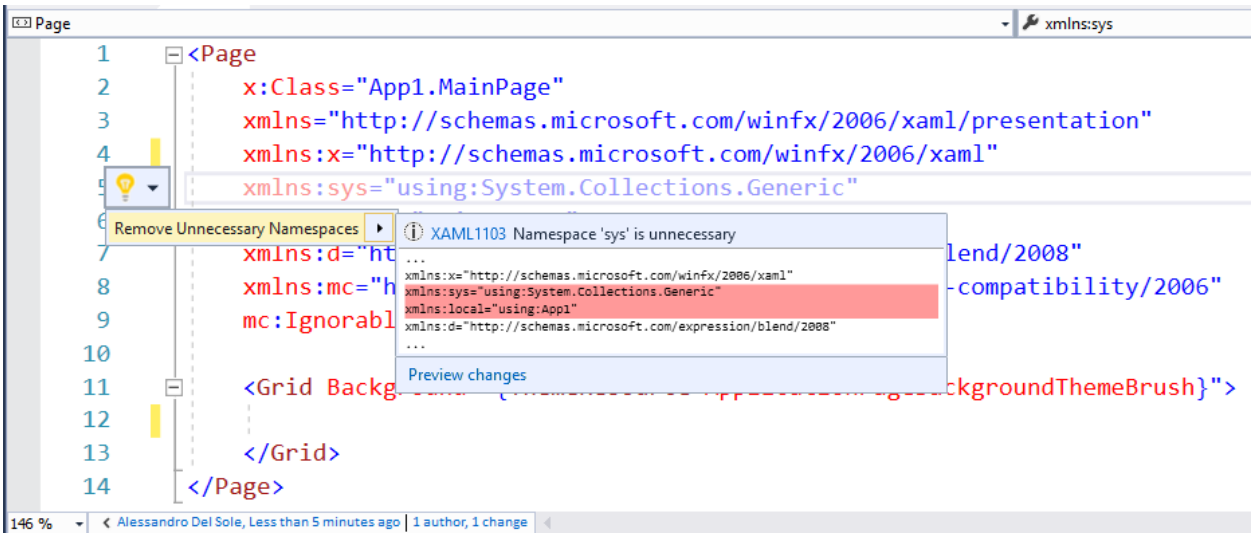

Figure 35: Removing Redundant Namespaces

You can also quickly remove redundant namespaces and sort the remaining namespaces by right-clicking the XAML code then selecting **Remove and Sort Namespaces**, or by pressing Ctrl+R, then Ctrl+G.

Inline namespace rename

The inline rename feature was first introduced in Visual Studio 2015 for the C# and Visual Basic languages, and it allows renaming a symbol directly in the editor, without any modal dialogs. In Visual Studio 2017, this feature now comes to the XAML code editor, providing an option to rename namespaces more efficiently. You can right-click the name of a namespace, then select **Rename**. This will also highlight all the references to the selected namespaces, as shown in Figure 36.

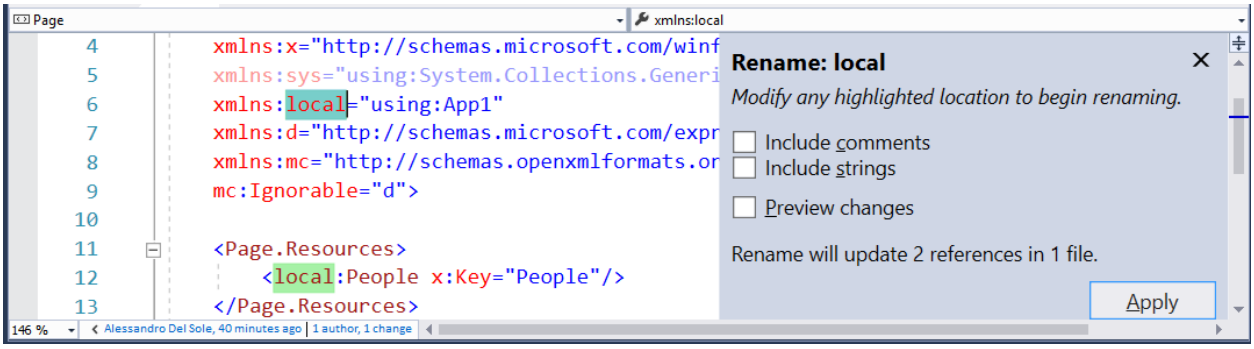

Figure 36: Enabling Inline Rename on Namespaces

Simply type the new name on any highlighted location and it will be applied to all references in the entire solution (see Figure 37).

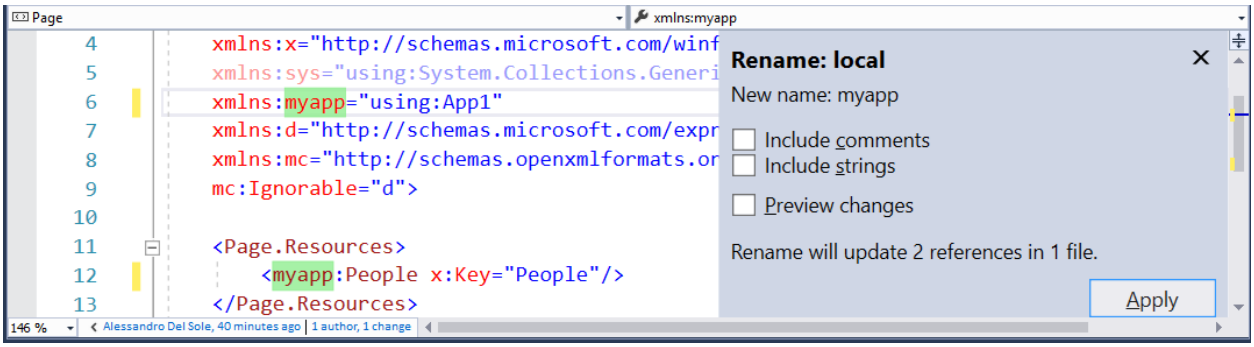

Figure 37: Renaming Namespaces as You Type

As with C# and Visual Basic, you can also include namespace references within comments and strings for a consistent rename.

Resolving missing namespaces

Note: In order to demonstrate this feature, I have added the Microsoft.Toolkit.Uwp.Controls NuGet package to a blank UWP project. However, it applies to any control in any library and namespace, both to UWP and WPF.

We often need to add a reference to a library that provides some controls, which can be a Microsoft or third-party library. However, you might not know which XAML namespace exposes the control, which can result in time wasted searching for the proper namespace and adding its declaration. In Visual Studio 2017, the code editor can easily resolve missing namespaces, adding the proper declaration on your behalf. Figure 38 shows how you can type the name of a control with the light bulb and then suggest a quick action to add the proper, missing namespace declaration.

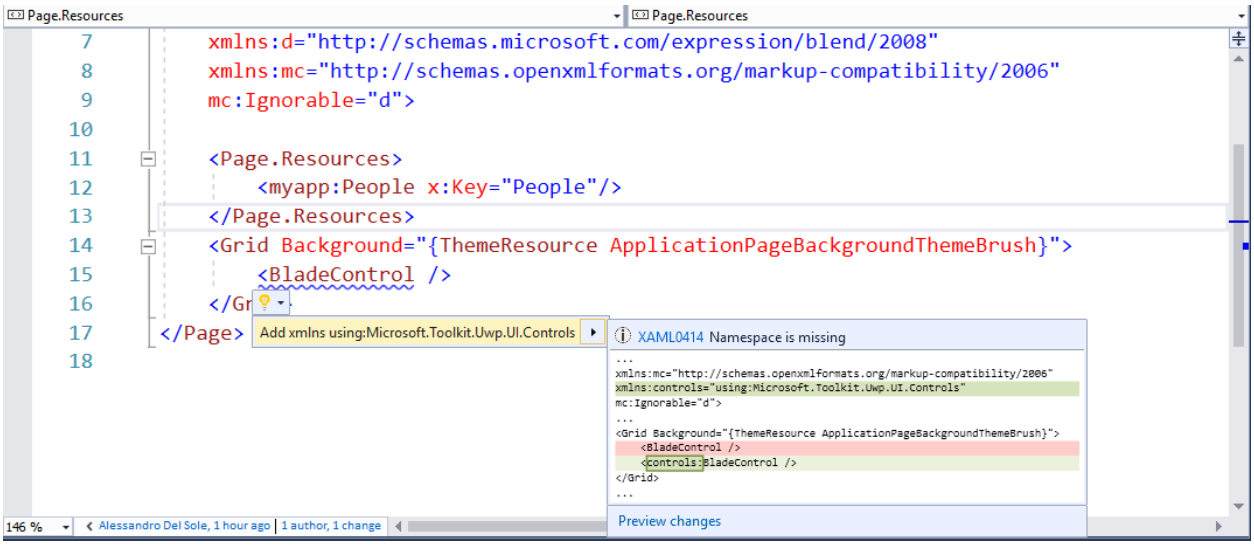

Figure 38: The Light Bulb Suggests Quick Actions for Missing Namespaces

As a result of the quick action, Visual Studio adds the proper XAML namespace. In this particular case, it adds a **`xmlns:controls`** directive that points to a namespace called Microsoft.Toolkit.Uwp.UI.Controls, as shown in Figure 39.

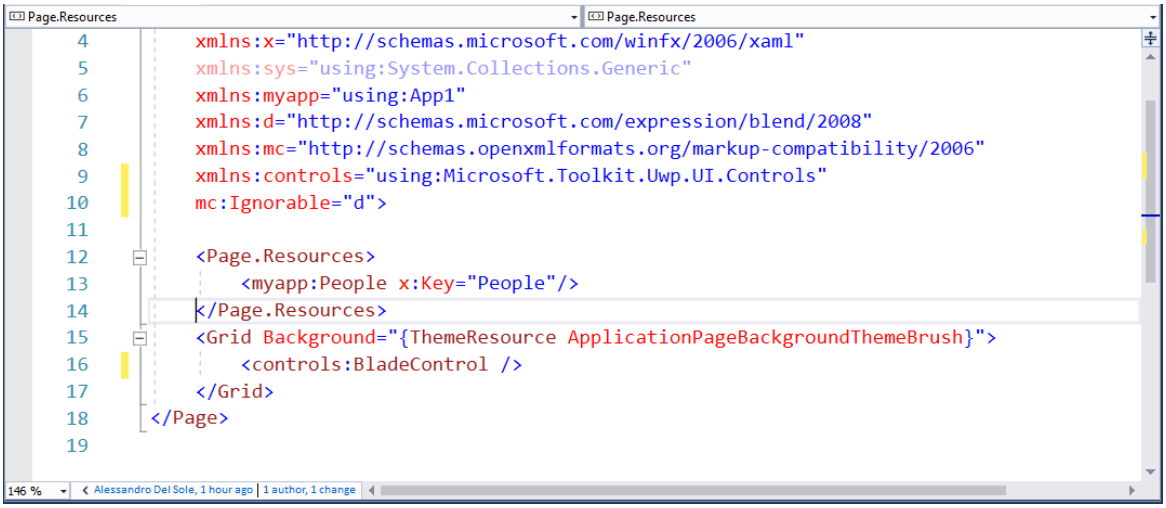

Figure 39: Namespace Added and Issue Fixed

Notice that the identifier for the directive (**`control`** in this case) is generated based on the .NET namespace name. You can then use inline renaming to provide a different identifier and automatically update all references.

XAML Diagnostics

Visual Studio 2017 includes a feature called XAML Diagnostics. This feature was first introduced with Visual Studio 2015 Update 2 under the name XAML In-App Menu, but it deserves some explanation, especially if you are new to the Visual Studio development environment. XAML Diagnostics consists of a black, collapsible toolbar that appears when you debug a WPF or UWP application. Figure 40 shows this toolbar over a very simple app.

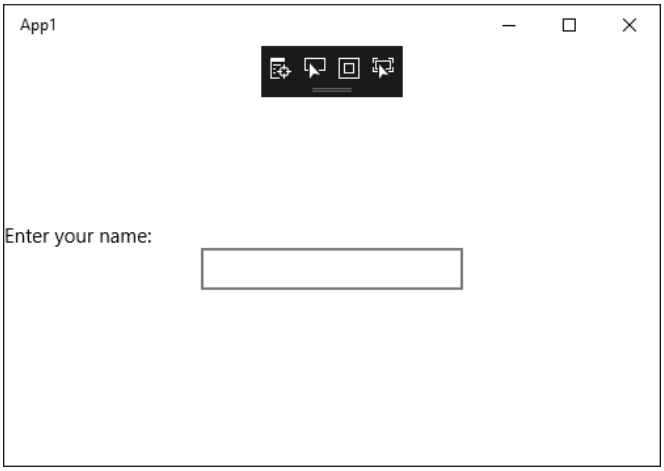

Figure 40: The XAML Diagnostics Toolbar

Tip: The In-App Menu is enabled by default. If you want to disable it, go to** Tools **>** Options **>** Debugging**, and clear the** Show runtime tools in application **option.

The In-App Menu can be minimized, which is useful for avoiding overlaying parts of the UI—but for now, leave it open. The menu has four buttons, each described in the next paragraphs (starting from left to right).

Go to Live Visual Tree

As the name implies, this button simply opens the Live Visual Tree tool window. I suggest that you dock the Live Visual Tree window so that you will immediately see the result of the other buttons. More information about the Live Visual Tree can be found in the MSDN documentation.

Enable Selection

This button allows you to select controls on the user interface. When you select a control, it is surrounded with a red border, and the Live Visual Tree window automatically shows the selected control within the visual tree. Figure 41 shows an example.

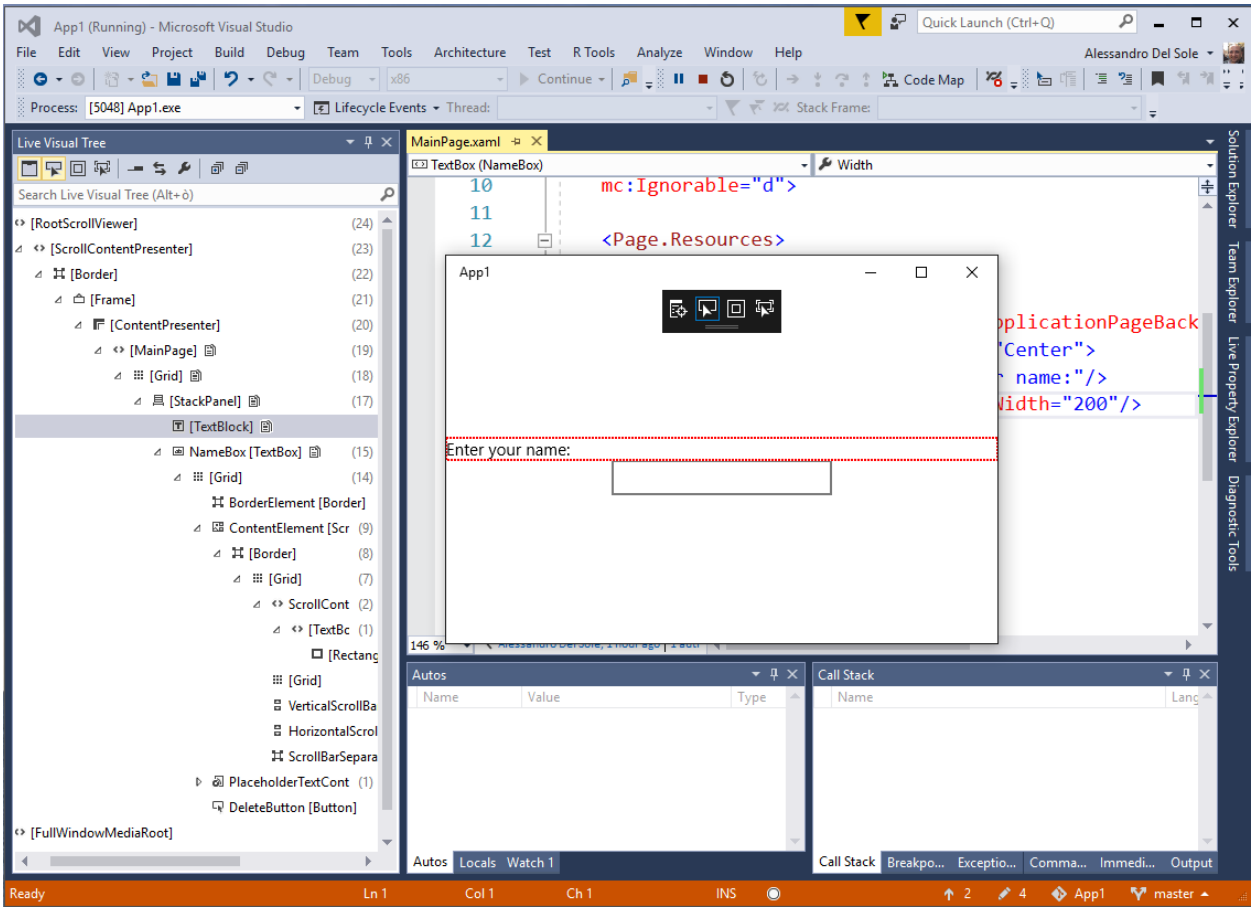

Figure 41: Enabling Element Selection in XAML Diagnostics

This is useful for focusing more precisely on a specific control or UI element, including primitive elements in the visual tree.

Display Layout Adornments

This button allows you to highlight the surface of a control or UI element. If combined with Enable Selection, an element is highlighted and selected. Selection is also reflected in the Live Visual Tree window. This is useful for understanding the delimiters of a control. Figure 42 shows an example based on the combination of both buttons.

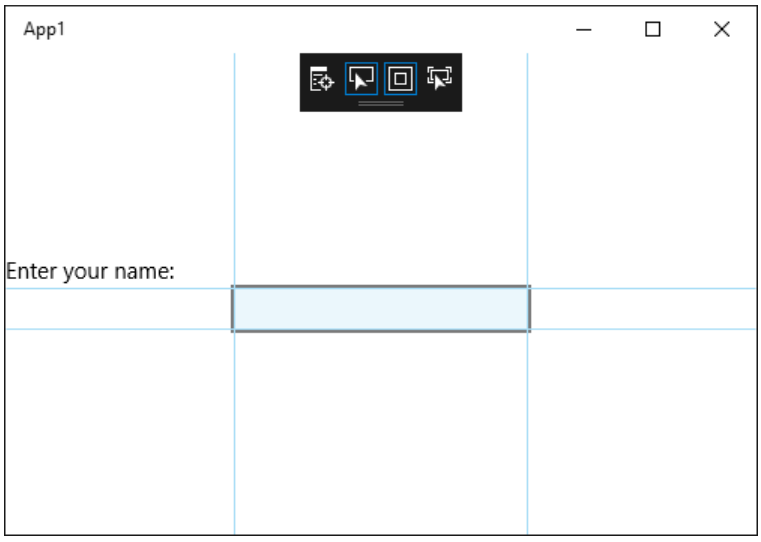

Figure 42: Displaying Layout Adornments

Track Focused Element

The Track Focused Element is similar to Enable Selection in that it allows selecting a control and reflecting the selection in the Live Visual Tree, but you should note that it only allows selecting controls that can receive focus (for instance, **`TextBox`** controls).

Chapter summary

Due to its importance in the Universal Windows Platform and Windows Presentation Foundation, the XAML code editor has received a number of improvements in Visual Studio 2017. With Edit and Continue, you can now modify the visual tree and see changes while the application is in debugging mode. IntelliSense has improved filtering, Go To allows you to quickly navigate your markup, and new refactorings have been introduced to support namespace management. But Visual Studio 2017 is not simply the environment for managed languages and XAML. It is the development environment for any development on any platform. In the next chapter, you will discover amazing new features for developers who do not work with MSBuild solutions and projects.

Chapter 5 Working with Solutions, Folders, and Languages

Visual Studio 2017 introduces updates to the solution model and to the way the IDE manages projects. This brings improved performance and reliability, especially with large solutions. Visual Studio 2017 also supports a larger number of languages and code files, regardless of the project system to which they belong. All these goodies are described thoroughly in this chapter.

Lightweight Solution Load

Often, enterprise applications consist of very large solutions, sometimes with dozens of projects. However, loading a very large solution can result in a negative impact on the overall performance of Visual Studio. For this reason, Visual Studio 2017 has introduced a new feature called Lightweight Solution Load. This feature optimizes the process of loading very large solutions, providing better responsiveness and performance. This feature is not enabled by default, so you can enable it manually by navigating to **Tools** > **Options** > **Projects and Solutions**, and then selecting **Lightweight Solution load** (see Figure 43).

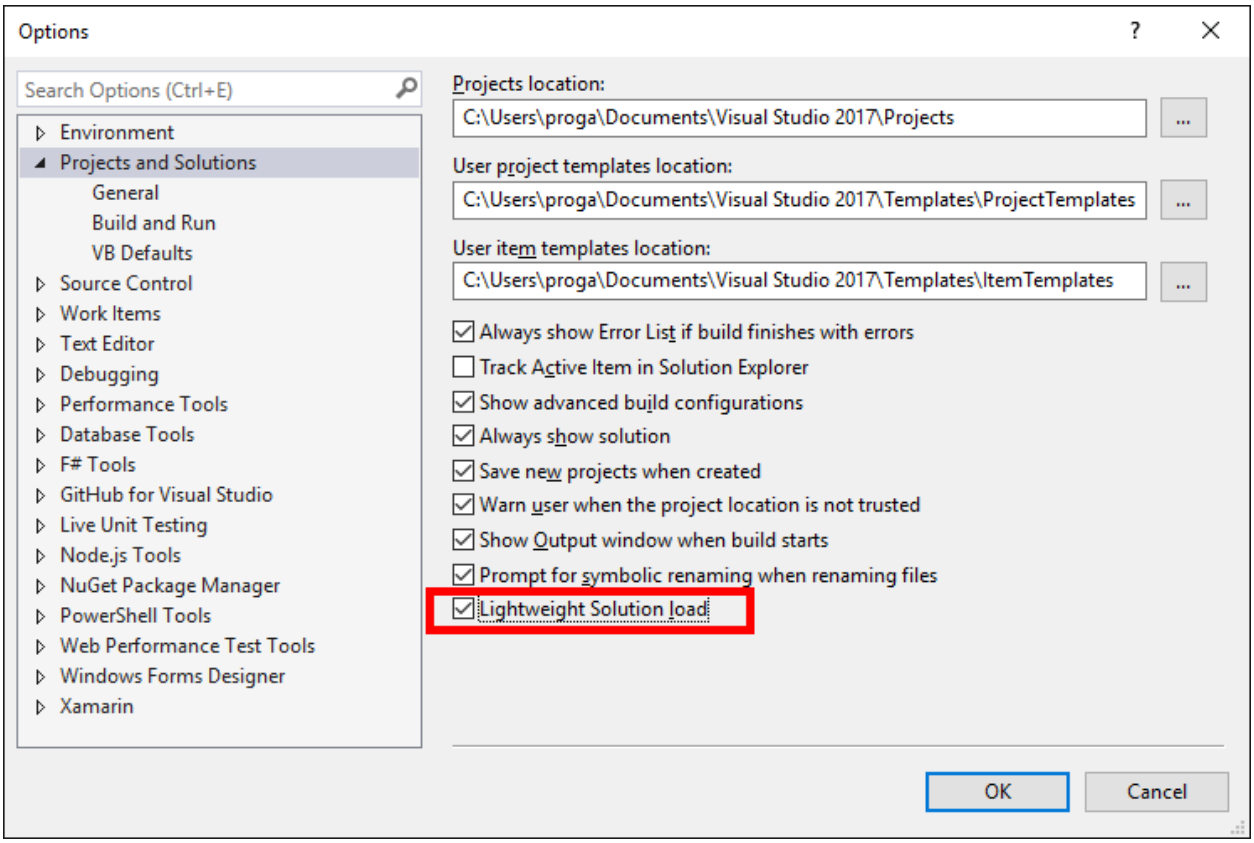

Figure 43: Enabling Lightweight Solution Load

You must understand the reasons why this feature is not enabled by default before you use it:

- You benefit from using this feature only with large solutions.
- It is optimized to work with solutions made of C#/VB projects or a mix of C# and C++ projects. Not all project types benefit from this feature.
- Visual Studio 2017 can automatically detect if loading a given solution might be optimized with this feature. In this case, it will prompt you with a message and ask your permission to enable the feature for you.

Behind the scenes, Lightweight Solution Load delays loading some projects until you actually need them. This implies that some features, such as code refactoring, code navigation, or inline renaming, might require some extra milliseconds the first time you invoke them. In conclusion, Lightweight Solution Load is a very useful addition, but only with large solutions made of C#/VB projects or with mixed C#/C++ solutions.

Extended language support

Out of the box, Visual Studio 2017 provides built-in support for a broader set of languages than past versions. This support consists of syntax colorization, code snippets, navigation, and code completion through the Visual Studio core editor, which means it is available even if no workloads are installed. If you install workloads that target specific languages (e.g., .NET desktop development that targets C# and VB), those languages will receive more extensive support, such as, but not limited to, IntelliSense, light bulbs and quick actions, debugging, and so on. The value of this feature will be demonstrated in the next section, “Open Folder: Working with any codebase.” In the meantime, take a look at Table 7, which summarizes the list of supported languages and features per language.

Table 7: Supported Languages and Features in Visual Studio 2017

Features	**Languages**
Only syntax colorization and code completion	Batch, Clojure, CoffeeScript, CSS, Docker, F#, INI, Jade, JSON, LESS, Make, Markdown, Objective-C, PowerShell, Python, Rust, ShaderLab, SQL, YAML
Code snippets, syntax colorization, and code completion	CMake, Groovy, HTML, Javadoc, Lua, Perl, R, Ruby, ShellScript, Swift, XML
Go To, code snippets, syntax colorization, and code completion	C++, C#, Go, Java, JavaScript, PHP, TypeScript, Visual Basic

By supporting these languages, you can work with source code for different platforms and take advantage of evolved editing features—even with no installed workloads. If you are impatient, you can try to open individual code files in any of the supported languages, but the biggest benefit is working with folders. The next section provides some practical examples.

Open Folder: Working with any codebase

Many development environments have their own project system, and Visual Studio is no exception. A project contains everything needed to build an application, including source code files, references to external libraries, assets, and metadata. Microsoft Visual Studio has always had its own proprietary project system based on solutions (.sln files) and projects with the .xxproj extension, where xx represents the targeted development platform, the utilized programming language, or both. A solution can be considered as a collection of projects, and in the .NET terminology it is often referred to as an MSBuild solution, because MSBuild.exe is the tool Visual Studio uses to produce an application that compiles all the projects in a solution in the proper order. Visual Studio 2017's ambition is to be the development environment for any developer on any platform, so it takes an important step forward by supporting a large number of non-.NET languages (see Table 7). Visual Studio 2017 also supports working with folders; this means that you can open folders on disk containing multiple code and asset files and Visual Studio will organize them in the best way possible in that environment. This makes Visual Studio independent from any proprietary project system (except for solutions, of course). It loads all the code files in a folder as a loose assortment, providing a structured view by organizing files and subfolders into a root folder for easy navigation through **Solution Explorer**. The root folder is actually the name of the folder you opened. Let's look at a couple of examples.

Setting up the demo

For the examples, I'll be using a nice open source project called Haikunator, which provides functions that generate random subdomain names for the Heroku platform. This project is available in various programming languages and with no proprietary project system, so it is perfect for our needs. I'll be using two versions: the version written in Go and the version written in Python. The first example will be used to showcase the capabilities of the Visual Studio core editor, and the second example will be used to demonstrate extensive language support with the debugger and IntelliSense. That said, follow these steps:

1. Go to github.com/Atrox/haikunatorgo for the Go version.
2. Click **Clone or download**, then **Download ZIP**.
3. Extract the content of the downloaded .zip archive into a folder of your choice.
4. Go to github.com/Atrox/haikunatorpy for the Python version, then repeat steps two and three.

Take note of the folders where you extracted both .zip archives because they will be used shortly.

Basic language support in Visual Studio 2017

Our first example is based on the version of the Haikunator project written with the Go programming language. In Visual Studio 2017, select **File** > **Open** > **Folder**. Browse the disk until you find the folder where you extracted the code, then click **Select Folder**. After a few seconds, **Solution Explorer** will show the root folder for navigation and files inside that folder (see Figure 44). If you double-click any of the files with the .go extension, you will see how Visual Studio 2017 offers syntax colorization for this supported language even if no workloads have been installed and even if the current language does not belong to the .NET family (see Figure 44).

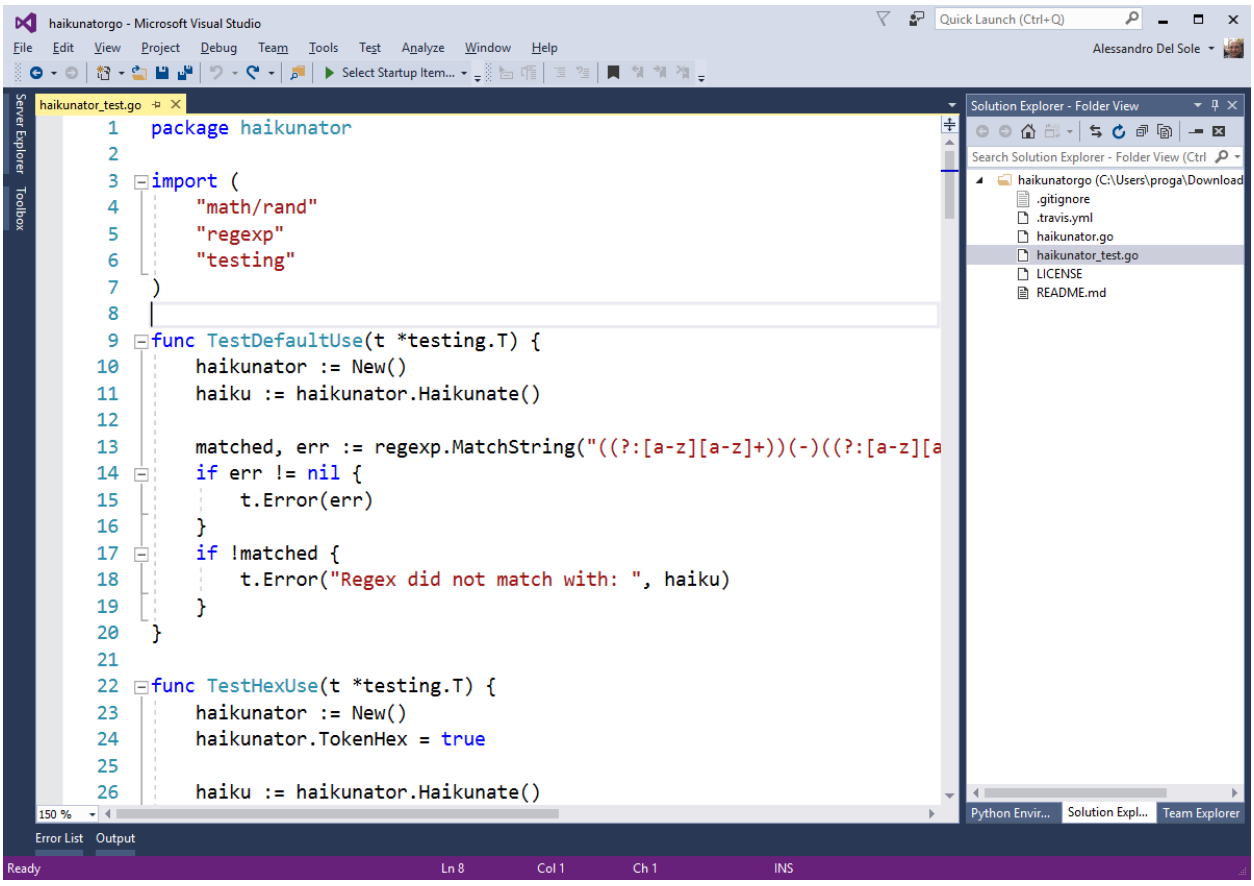

Figure 44: Opening a Folder and Editing Any Supported Code Files

Additionally, you get code navigation features such as Go To and structure guide lines as demonstrated in Figure 45, where you can see how the list of items can be filtered as you type.

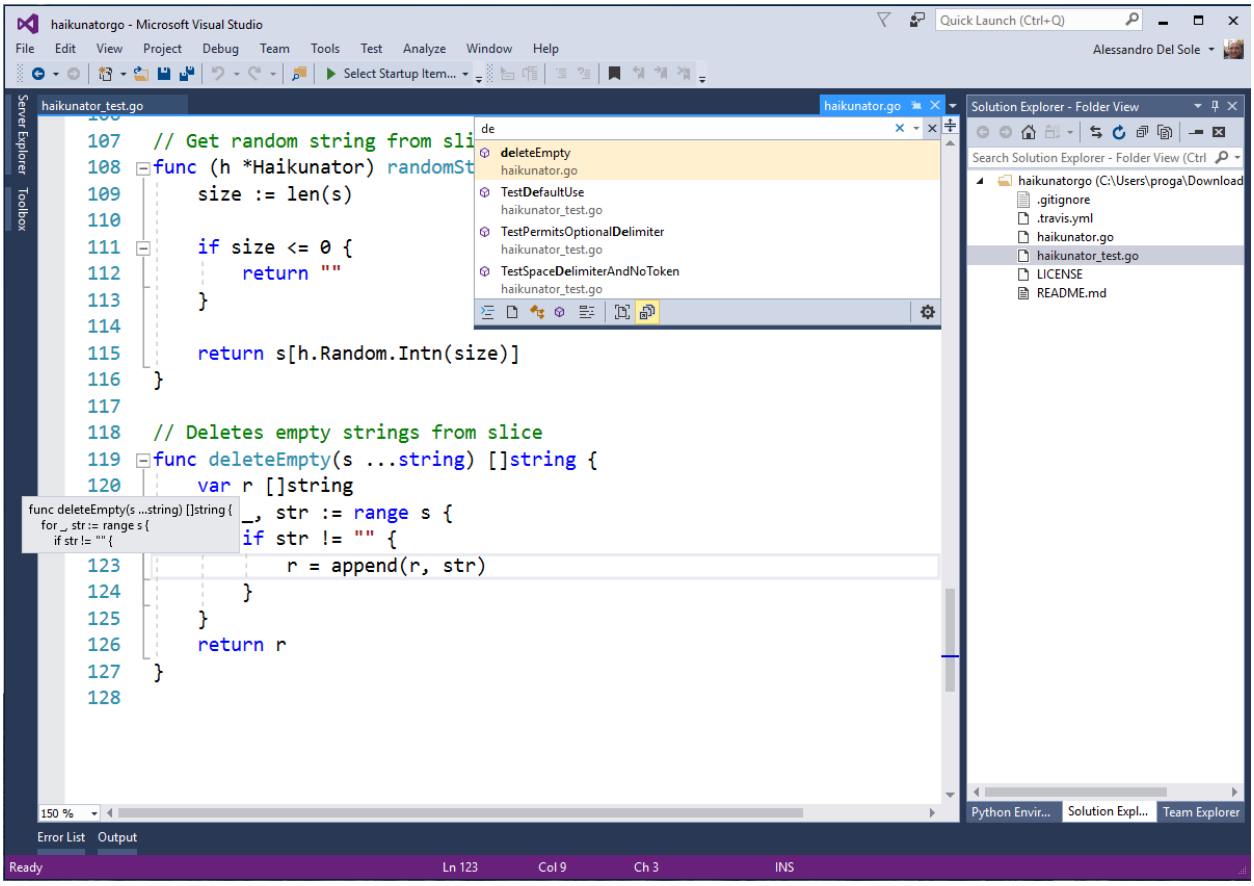

Figure 45: Code Navigation Features in Visual Studio's Core Editor

For every supported language, you also get code completion, a feature that makes writing and editing code faster and easier. Figure 46 shows an example.

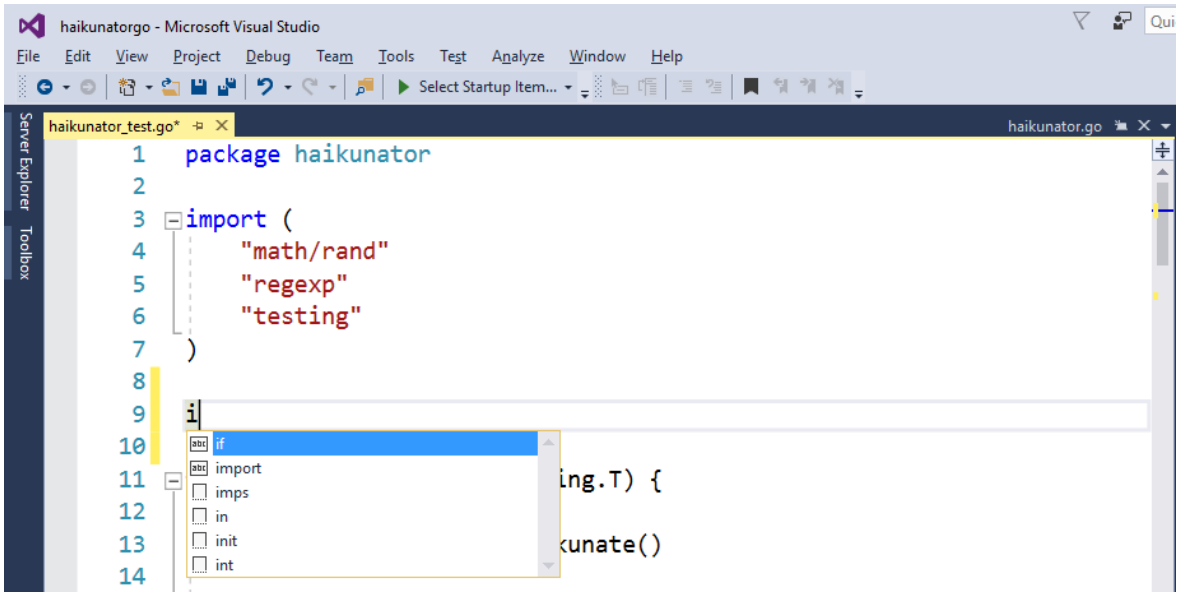

Figure 46: Code Completion Simplifies Writing Code

Some languages, including Go, also support TextMate code snippets, which are available through the code completion pop-up. Code snippets can be recognized through the icon of a blank sheet (see Figure 46). For example, if you select the **init** snippet and then press **Tab**, the code editor will insert a function stub called **`init`**, as shown in Figure 47.

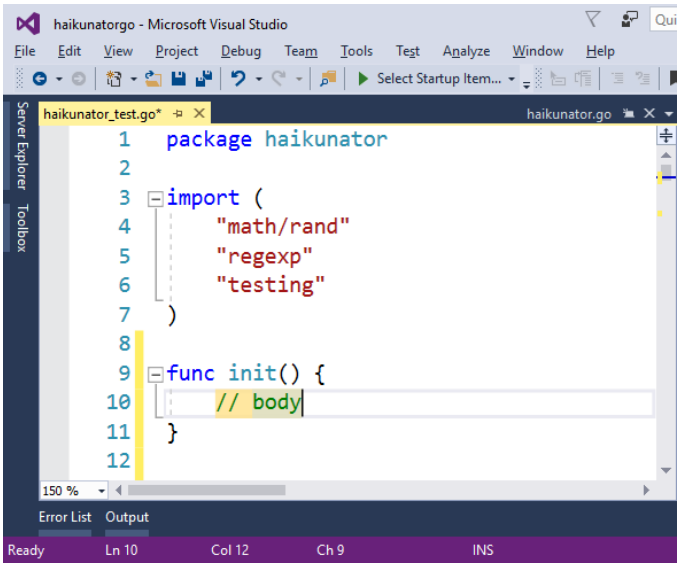

Figure 47: Inserting Code Snippets

Notice how the code editor highlights in yellow the code that should be replaced with your code. Remember that code snippets are not available to all languages, but you can refer to Table 7 as a quick reference for this feature's availability. The important thing to emphasize here is that you have all you would expect from an evolved code editor regardless of the language, project systems, and installed workloads.

Note: Code completion is not IntelliSense. This is an important clarification. IntelliSense is an evolved tool, typically powered by a background compiler or by a completion database that allows an IDE to display and group items based on types, members, identifiers, and so on. Not surprisingly, IntelliSense in Visual Studio is available only to a restricted number of languages that have extensive support provided by specific workloads and tools. Code completion is more limited, often based on a small built-in list of keywords and on literals that the editor detects in the code, but it is in fact available to any language.

Extensive language support through workloads and tools

For some languages, Visual Studio 2017 allows for adding extensive support through specific installation workloads or third-party tools. Think of Python, which is a very popular programming language. The Visual Studio core editor provides basic support for it, including syntax colorization and code completion. However, Visual Studio 2017 allows executing external programs against your source code; this means that if you install the Python interpreter, you can execute your Python code from the development environment.

Tip: The last sentence applies to all the supported languages, not only Python.

In pre-releases of Visual Studio 2017, Microsoft also included a specific installation workload for Python development which included debugger integration and other enhanced tools to work with Python from the IDE. For RTM, the Python workload has been removed from the Visual Studio Installer. The Visual Studio Tools for Python for Visual Studio 2017 will be offered separately and is planned to be released a few days after Visual Studio 2017 reaches the RTM milestone. So, the next example based on the Python version of the Haikunator project is actually based on the pre-release of Visual Studio Tools for Python previously included with the Visual Studio Installer and will still be valid once the updated tools ship.

Note: The reason why I'm showing an example based on Python though it will only be available after RTM ships is that it's the perfect demonstration of extensive language support with debugging features and code refactorings.

When you are done, open the folder containing the Python project with the **Open Folder** command. Then, open any .py code file to see how the code editor provides the expected syntax colorization. Figure 48 shows an example based on the tests.py file.

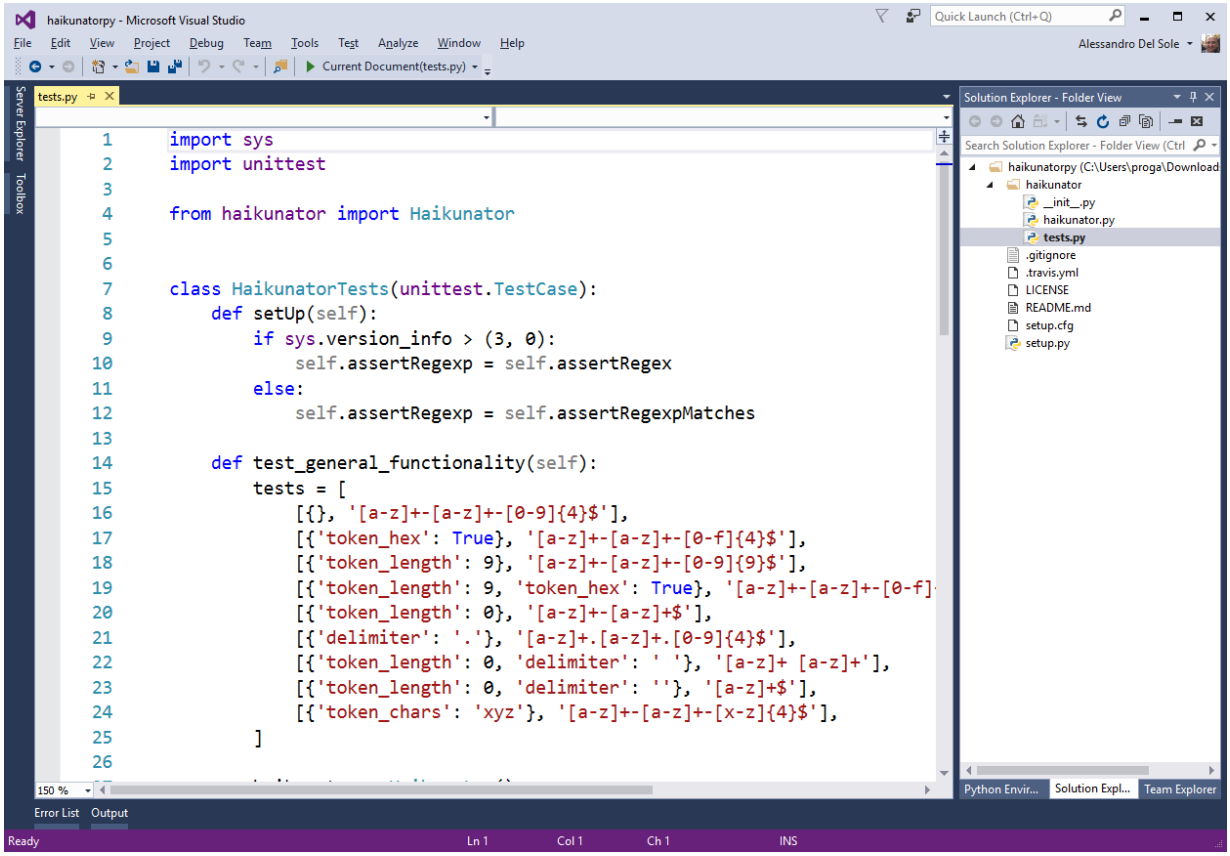

Figure 48: Syntax Colorization for Python

When you start typing, you will see IntelliSense in action, not just code completion. Figure 49 demonstrates how IntelliSense for Python shows items with icons that explain an item's scope, such as keywords, functions, and types. Remember that IntelliSense is available for Python because of the specific installation workload, otherwise only code completion would be available.

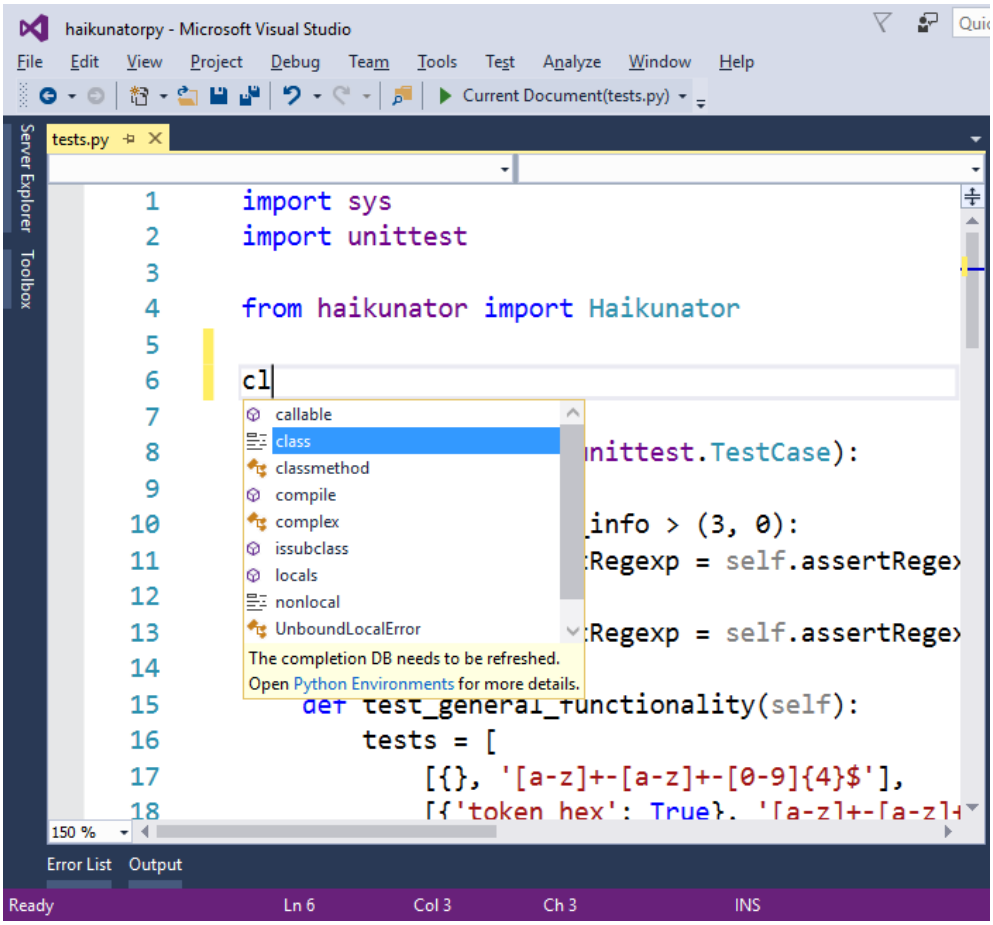

Figure 49: IntelliSense for Python

You might see a warning message that the completion DB needs to be refreshed. This happens because IntelliSense for Python is populated with an external database, which is different from what happens with C# and VB, in which the Roslyn APIs expose available members to Visual Studio. If you see the warning message, click the **Python Environments** hyperlink in the IntelliSense pop-up (see Figure 49). This action will open the Python Environments tool window, where you can see a list of available environments and interpreters (see Figure 50). The Python development workload installs the latest version of the interpreter from the official Python Software Foundation website—in this case version 3.5. Depending on your machine, the workload installs the 32-bit or 64-bit version. You will see a **Refresh** button near the Python environment—click it to start refreshing the completion database.

Figure 50 shows this operation in progress. Notice that not all of the available environments are also installed locally. For instance, in Figure 50 you can see how the Anaconda environment appears disabled, which means available but not installed. This is available as an individual component in the Visual Studio Installer or can be installed separately.

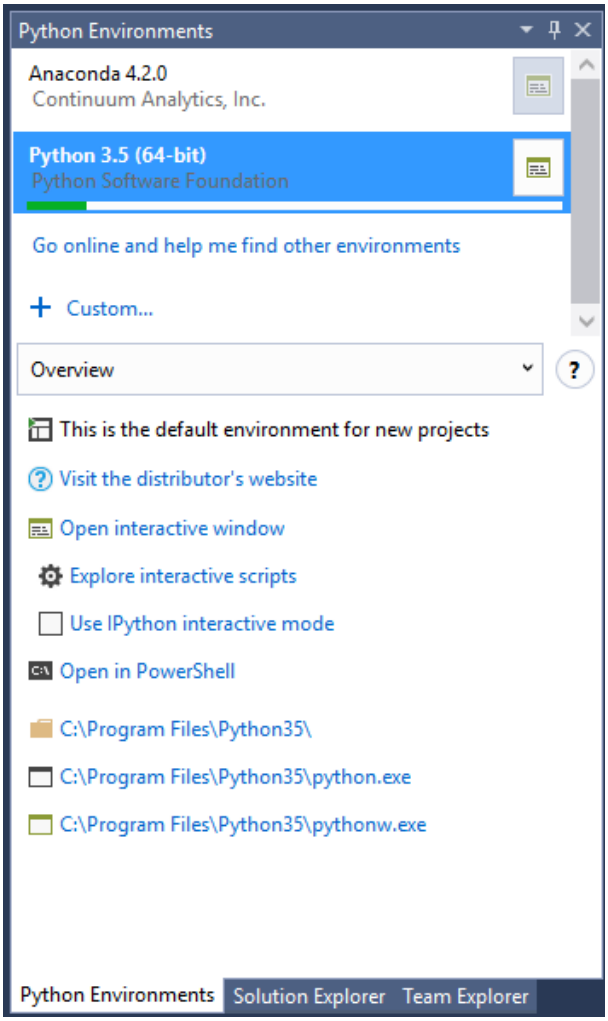

Figure 50: The Python Environments Tool Window

There are other shortcuts in this window, but they are strictly related to the Python tools for Visual Studio—addressing them is beyond the scope of this e-book. Now, it is time to discover some benefits of the Visual Studio editor by writing a bit of Python code.

Setting up a startup item

As we know, a great benefit that comes from installing tools that extend a language's support is being able to run a program directly from within Visual Studio. This is true for the Python language, too. In order to run code, Visual Studio 2017 needs us to set up a startup item, which is the code file it will use to start a program. If your folder already contains a startup item, you are set. In this current example, the Haikunator project is a library that does not contain a startup item. To add one, right-click the root folder in **Solution Explorer**, then select **Add** > **New File**. You will see a new file in the folder. You can type any name you like, but for consistency in this chapter, enter **Startup.py** and press Enter. At this point, type the code shown in Code Listing 1. You will get IntelliSense as you type, and this will make your coding experience truly awesome.

Code Listing 1

```
from haikunator import Haikunator
haikunator = Haikunator()
name = haikunator.haikunate()
print(name)
```

The code is very simple—it imports the **`Haikunator`** type that is defined in the haikunator package and generates a random name invoking the **`haikunate`** method. Then it simply prints the name on screen. This is enough to demonstrate a number of features of the code editor. When you open the code editor on a specific file, either new or existing, this will automatically be set as the startup item. In this case, Startup.py is automatically set as the startup item for the program. If you wish to use a different startup file, simply open it or right-click its name in **Solution Explorer**, then select **Set as Startup Item**.

Running a program

As you would do with any other Visual Studio project, you can run a program by pressing F5 (debug mode), Ctrl+F5 (without debugging), or by clicking **Start** on the standard toolbar. When working with folders, the Start button shows the name of the startup item (see Figure 51).

Figure 51: Start Button Displays the Name of the Startup Item

Whatever start option you choose, your program will run as expected. In this particular case, you do not need to manually invoke the Python interpreter. Starting the sample program will produce the result shown in Figure 52 (the generated random name will vary on your machine).

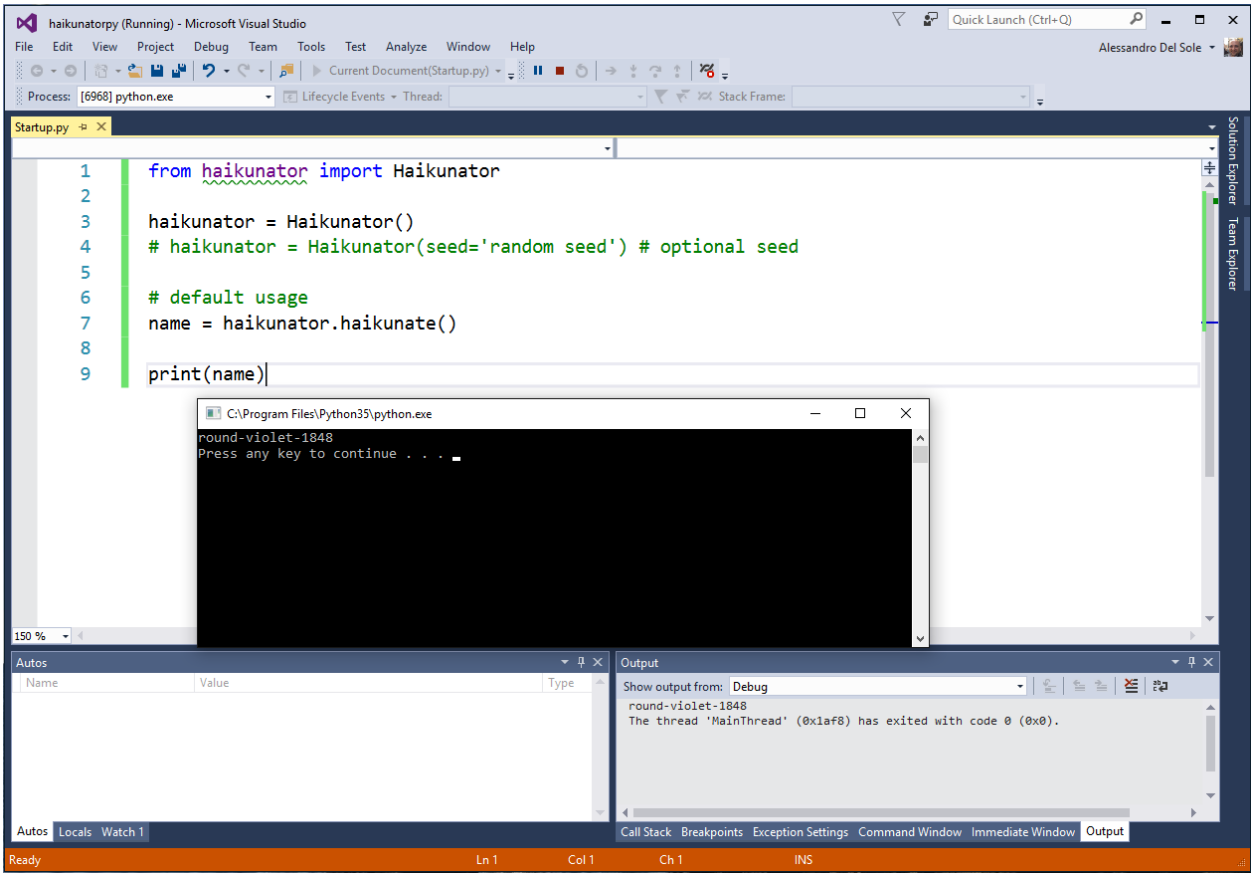

Figure 52: Running a Python Program from Visual Studio 2017

Notice that with specific regard to Python, the installation workload automatically configures Visual Studio 2017 to run the interpreter from within the IDE against the selected startup item. For other languages and with other tools, Visual Studio 2017 must be manually configured to run compilers or interpreters. We will look at those situations later in this chapter. For now, take a closer look at Figure 52—notice how the status bar of Visual Studio is orange, which means it is in debugging mode. This also implies that the new IDE can use external debugging tools.

Debugging

Visual Studio 2017 can extend the debugger to support additional languages. Specific workloads or third-party tools can install debugging components that the IDE can integrate into its powerful development experience. For a better understanding of this, open the **haikunator.py** file in the code editor. Move to the last line of code and place a breakpoint exactly as you would in C# or Visual Basic. Make sure you open the **Startup.py** file so that this is set again as the startup item, then run the program. After a few seconds, the debugger will break the application execution because of the breakpoint. At this point, you will be able to use powerful debugging features you already know, such as data tips, local variables, and other debugging windows. Figure 53 shows an example.

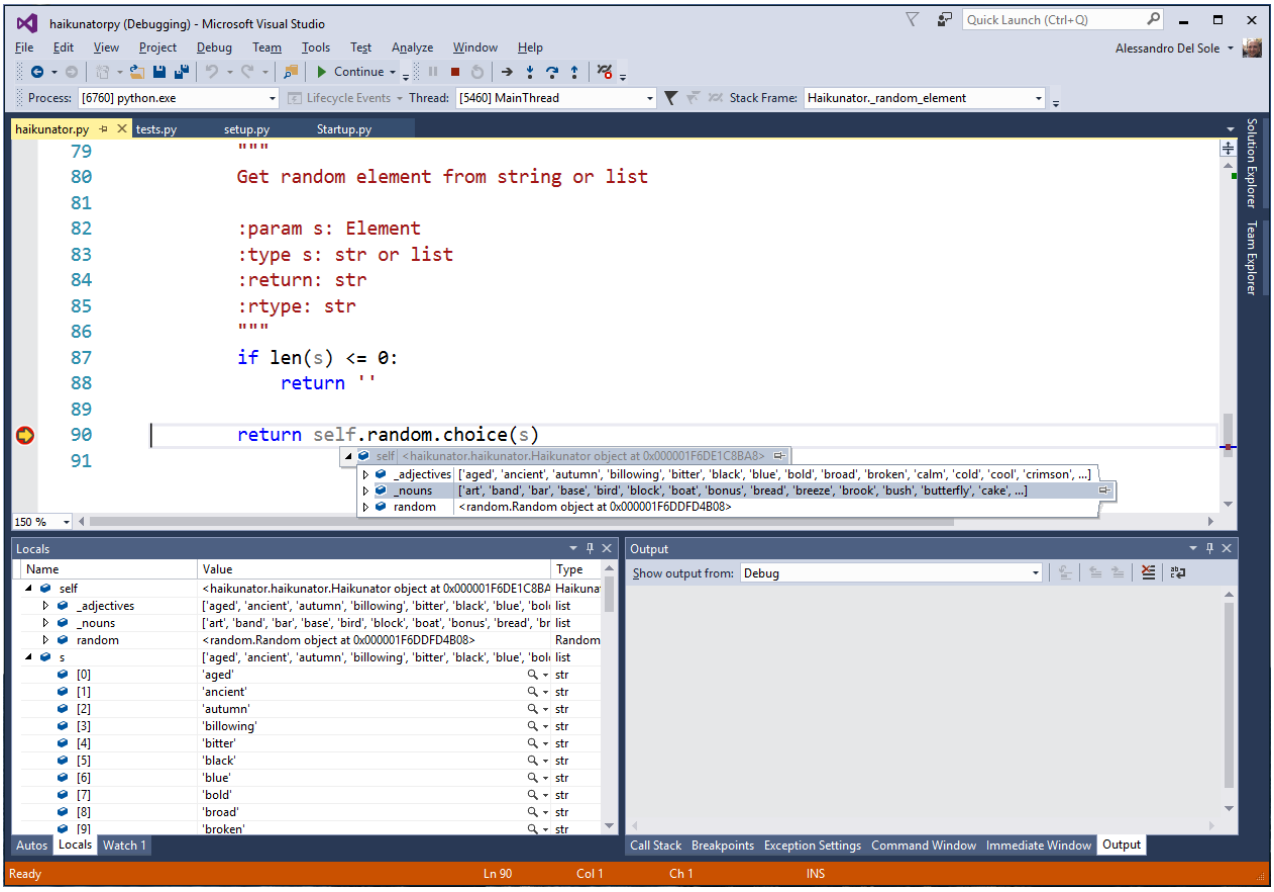

Figure 53: Debugger Support and Integration for Python

We need to remember that features such as debugging and IntelliSense are not available to all the supported languages, but rather they are available through specific workloads or third-party tools. However, Visual Studio 2017 offers its robust environment to non-.NET languages, and this is a tremendous added value for any developer.

Code refactoring

For specific languages, the Visual Studio 2017 code editor also offers light bulb and quick actions to provide an evolved code editing experience. For example, in Python you can select a code block and extract a method as you would with C# and Visual Basic refactorings. Figure 54 demonstrates this.

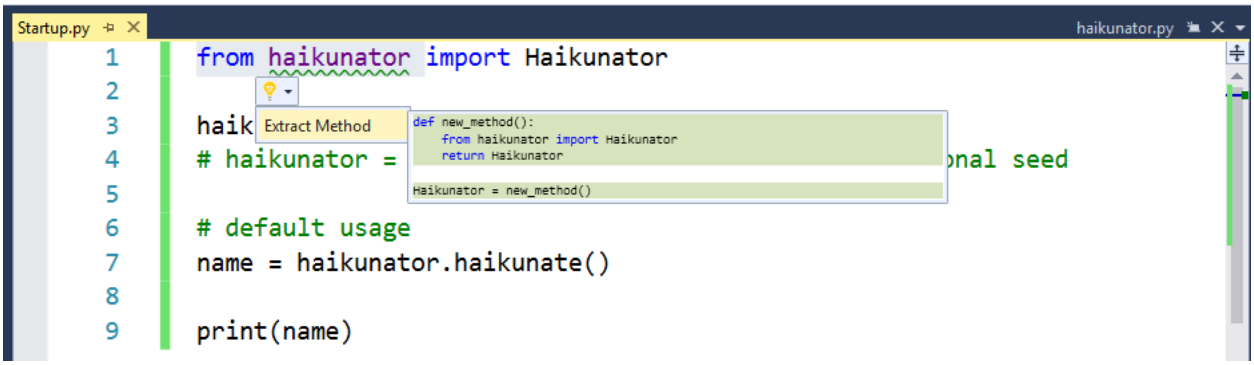

Figure 54: Code Refactoring for Python

With support for integrating features such as IntelliSense, refactorings, and debuggers, Visual Studio 2017 certainly lives up to its claim as the development tool for any developer on any platform.

Customizing tasks

Previously, we saw how easy it is to run a Python program by simply pressing F5. This is true because the Python tools for Visual Studio automatically configure the environment to run the interpreter against the selected source code file. Specific integrated tools do not exist for all languages, but for many we can still automate the execution of external programs, such as interpreters, compilers, or Windows commands by creating and customizing **tasks**.

Tip: If you are familiar with tasks in Visual Studio Code, you will notice a very close similarity with tasks in Visual Studio 2017. They have different locations and different syntax, but their purpose is exactly the same.

A task is an operation with a number of settings and is represented with JSON markup. Settings include the command to execute and its command-line arguments. An explanation will be provided shortly, but for now suppose you want to be able to compile or run a Go program. You first need a Go compiler and the proper settings. The Go Programming Language website has everything you need, including binaries and documentation. First, download and install the binaries for Windows. Second, create a folder in which you will write a simple Go program and configure the **GOPATH** environment variable as explained in the documentation, which also explains everything you need to know about writing, compiling, and running code with Go. The installer automatically configures the **GOROOT** environment variable so that you can run the Go.exe compiler from any location. In the current example, I'll use a folder called C:\GoWork. Now, everything we need to compile and run a Go program is ready.

Note: Obviously, different programming languages work with different binaries and system configurations, which means that configuring tasks requires the same steps with any language but with different settings. In this e-book, I'm providing an example based on Go because installing its binaries is very easy, though configuring the system requires some manual steps.

In Visual Studio 2017, open the folder you created (such as my C:\GoWork). When ready, add a new file called **HelloWorld.go**. Next, enter the code shown in Code Listing 2.

Code Listing 2

```
package main

import "fmt"

func main() {
    fmt.Printf("hello world from Visual Studio 2017!\n")
}
```

I'm using a new "Hello World!" program instead of the Haikunator project because Go requires that we install and configure packages before we can use them. We should compile and install the Haikunator library, then provide the proper configuration (which requires us to be somewhat familiar with Go rules).

While providing these details is outside of the scope of this e-book, the Go documentation offers good explanations. Code Listing 2 simply defines a package called **`main`** and imports a base library called **`fmt`**, then prints a simple message on screen. Now, in **Solution Explorer**, right-click the **Helloworld.go** file and select **Customize Task Settings**. This action will create a new JSON file called tasks.vs.json, which is located in a hidden folder called **.vs** and that resides in your workspace. This JSON file contains the directives Visual Studio needs to know in order to execute a number of tasks, including launching the startup item. The following is a list of JSON properties common to every task:

- **`taskName`**: A name for the task that will be shown in the **Solution Explorer** context menu.
- **`appliesTo`**: Specifies the target of the current task, typically a specific file.
- **`type`**: Represents the type of task, typically command.

You must edit tasks.vs.json and specify that it must execute the Go.exe compiler against the current source code file. Code Listing 3 shows how to accomplish this by adding the **`command`** and **`args`** properties.

Code Listing 3

```
{
  "version": "0.2.1",
  "tasks": [
    {
      "taskName": "Run Hello World",
      "appliesTo": "Helloworld.go",
      "type": "command",
      "command": "Go.exe",
      "args": ["run", "Helloworld.go"]
    }
  ]
}
```

Notice how **command** specifies the name of the external program that must be executed. In this case, you specify the compiler file name with no path because an environment variable has been set by the installer. Instead, **args** specifies any command-line arguments for the command in the form of a JSON array. In this case, the first argument is an option called **run** that instructs the compiler to build and run a program, and the second argument is the name of the file to be compiled.

Note: Tasks.vs.json can contain many tasks, such as for building a whole folder or cleaning the build output. For additional configuration options, you can read this blog post from the Visual C++ Team in which they provide further explanations about multiple task configurations, supported constants, literals, and macros.

At this point, right-click the **Helloworld.go** file name in **Solution Explorer**, then select **Run Hello World** from the context menu (this is the **taskName** you supplied before). This action will launch the Go compiler and will make the program output visible in the **Output** window, as shown in Figure 55.

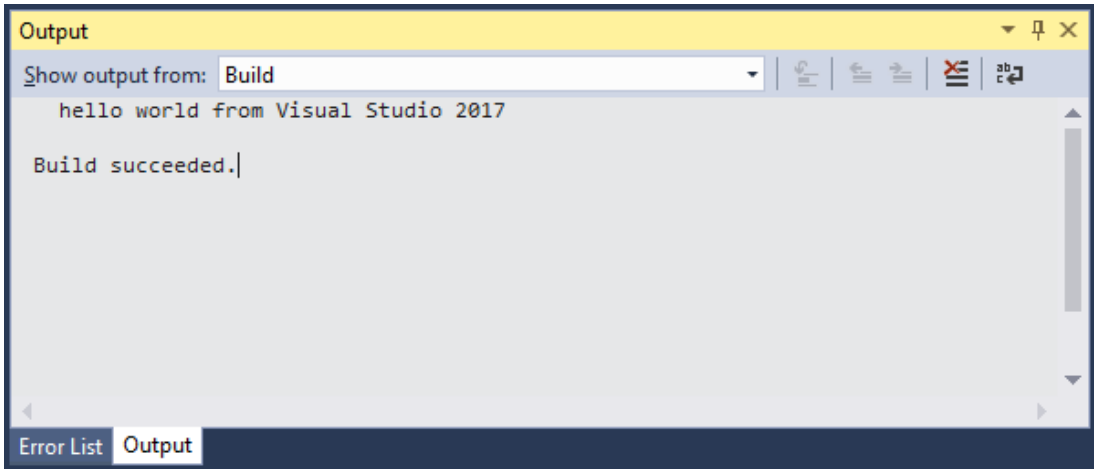

Figure 55: Running a Go Program with a Task

Though you don't have debugging support or a console window for the execution, you have been able to run a Go program from within Visual Studio 2017. Many other possibilities for task automation are available to you that provide greater control over all supported languages.

What's new for source control and team projects

Visual Studio 2017 also has something new for team collaboration. In fact, you can now use a new dialog to connect to Visual Studio Team Services and Team Foundation Server. For instance, in Team Explorer, click **Manage Connections**, then **Connect to Team Project**. At this point, you will see a new dialog called **Connect to a Project**, which shows all the Visual Studio Team Service accounts and TFS servers you are subscribed to (see Figure 56). By default, Visual Studio looks for repositories associated with the Microsoft account you used to log into the IDE, but you can add multiple accounts through the account drop-down. When you click **Connect**, Visual Studio will open Team Explorer showing a button labeled **Map & Get**, which provides a shortcut to map the remote repository to a local folder and get the latest version of the source code from the server.

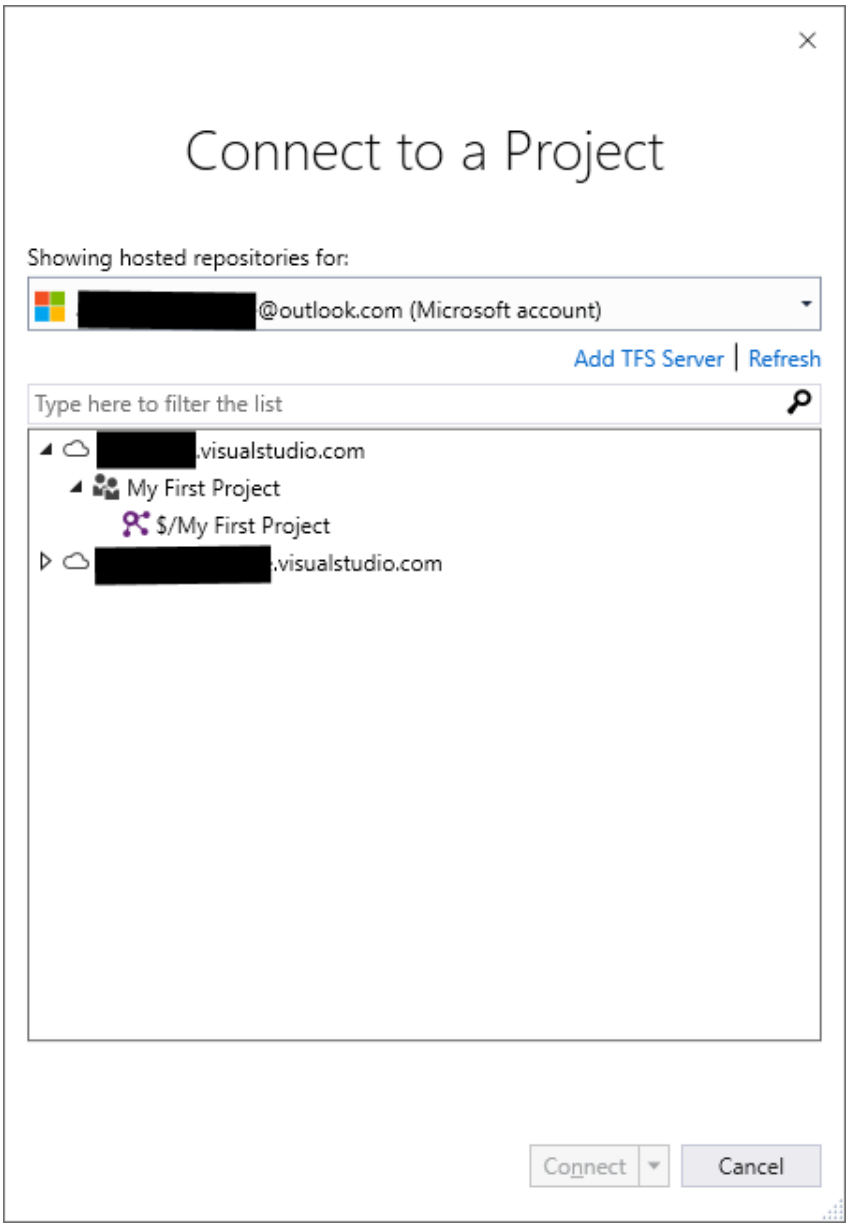

Figure 56: Connecting to Visual Studio Team Services Repositories

If you click **Add TFS Server**, you will be able to add and connect to on-premises TFS instances. Figure 57 shows an example based on fictitious information.

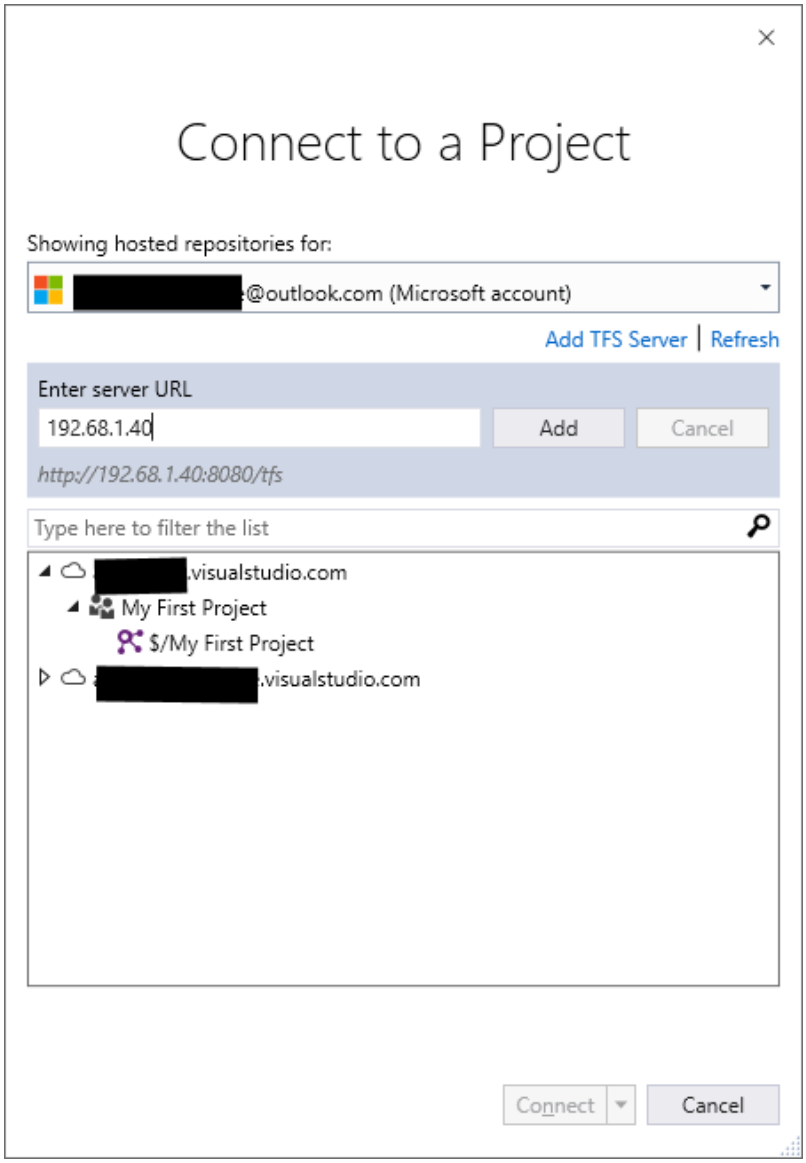

Figure 57: Adding an On-Premises TFS Instance

As with the **Team Services** tab, here you are still able to map the repository to a local folder and get the latest version of the source code. All the other features related to team collaboration, including Team Explorer, remain unchanged.

Tip: You can also enable the connection dialog through the** Manage Connections **command in the** Team **menu.

Note: Visual Studio 2015 Update 2 introduced more Git commands in Team Explorer and shortcuts for Git repositories at the bottom-right corner of the status bar. Visual Studio 2017 inherits these features, but it does not introduce anything new. If you want to learn more about these features, read the Visual Studio IDE and Team Explorer sections of the Visual Studio 2015 Update 2 release notes.

Chapter summary

Visual Studio 2017 introduces very important improvements and features for solutions, projects, and code files. With Lightweight Solution Load, large solutions can be loaded more efficiently and faster. The new IDE also supports a broader set of programming languages, even with no workloads installed, and the core editor can work with folders containing code files, providing not only a structured, organized representation, but also offering basic features such as syntax colorization and code completion, as well as evolved features like IntelliSense, debugging, and code snippets. In order to support these and other new or updated features, the architecture of Visual Studio 2017 is very different from previous editions, and this affects extensibility, as we'll see in the next chapter.

Chapter 6 Extensions and Extensibility

Changes in the architecture of Visual Studio 2017 have an impact on the extensibility model. This chapter will describe the improvements in extensions and extension management, but it will also provide information for extension authors looking to upgrade their Visual Studio 2015 extensions to the Visual Studio 2017 extensibility model.

What's new with extensions in Visual Studio 2017

In Chapter 1, I described the new installation model based on workloads and how we can now install multiple editions of Visual Studio 2017 on the same machine. For example, a developer might want to install an extension for the Enterprise edition but not for the Community edition, while another developer might have only the core editor installed and no workloads so that some extensions would not work. Consequently, extensions must be versatile enough to fit into these situations. In order to support these scenarios, Microsoft changed the extensibility model by introducing version 3 of the .vsix file format, which represents Visual Studio extension installers. This presents a number of breaking changes that will be described later in this chapter. For now, you must know that you will only be able to install extensions specifically written and compiled for Visual Studio 2017. An extension for Visual Studio 2015 cannot also target Visual Studio 2017, while an extension for Visual Studio 2017 can target past versions of Visual Studio, down to and including Visual Studio 2012. Let's now focus on new features that save time when using third-party extensions.

Roaming Extension Manager

The first important new feature regarding extensions is the **Roaming Extension Manager**. This tool allows for synchronizing installed extensions on every Visual Studio installation you have on different machines. The Roaming Extension Manager is included in the **Extensions and Updates** dialog that you enable through **Tools** > **Extensions and Updates**. Figure 58 shows what the tool looks like.

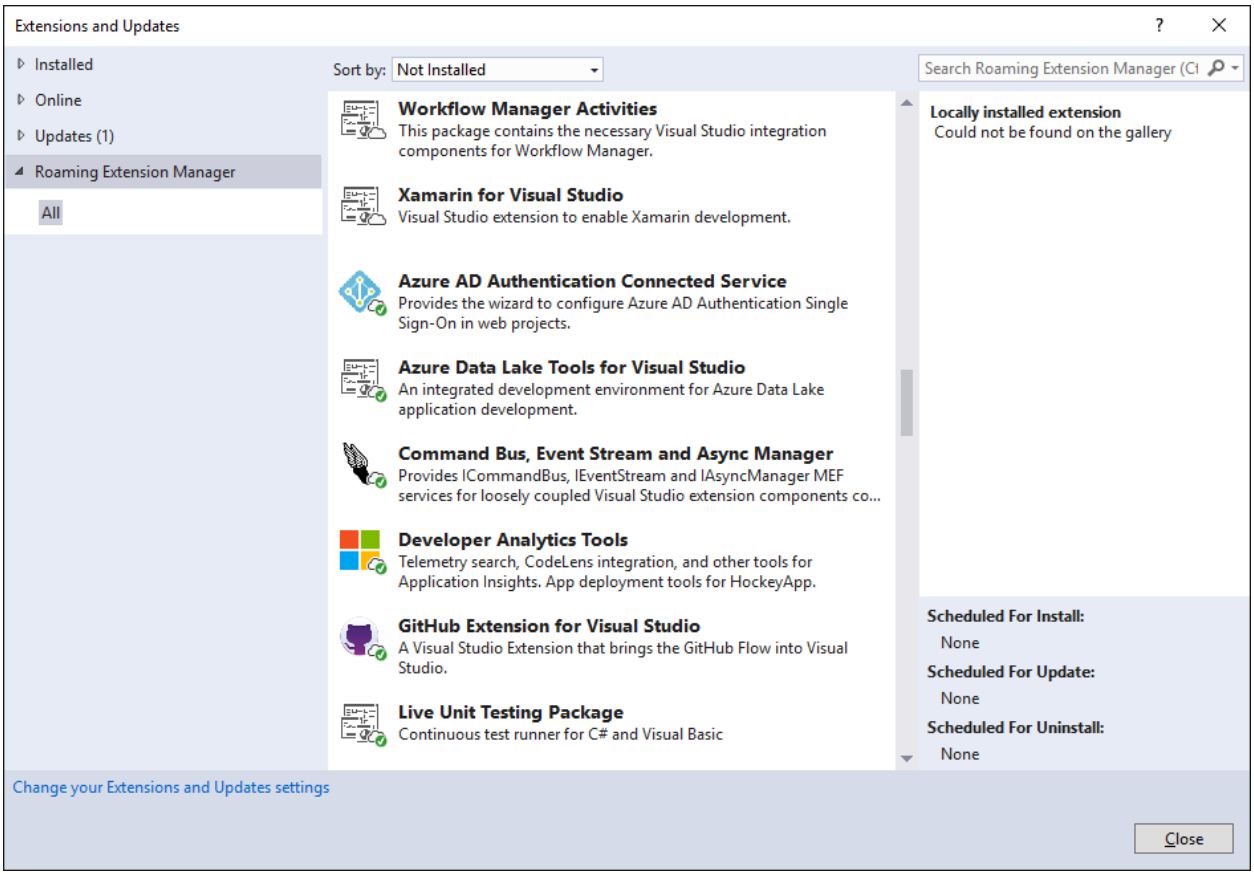

Figure 58: Activating the Roaming Extension Manager

The state of an extension is represented by one of the following three icons:

- A white cloud means the extension is subject to roaming but is not installed locally.
- A white cloud with a green check symbol means the extension is installed locally and roams.
- A green check symbol means the extension is installed locally but does not roam.

You can control extension roaming by selecting an extension and then clicking **Stop Roaming** (if it roams) or **Start Roaming** (if it's only installed locally). In Visual Studio 2017, this is one of the synchronized settings you enable once you log in with your Microsoft account.

Scheduling operations over extensions

Visual Studio 2017 also introduces a new way of installing, updating, and uninstalling extensions. In fact, the IDE now allows you to schedule multiple extensions for installation, update, or removal. These operations will be executed in bulk when you close Visual Studio. In order to understand how this works, consider Figure 59, in which you can see the **Extensions and Updates** dialog. In the figure, you can see two extensions marked with a clock icon. This icon appears after you download an extension, and it means it will be installed after Visual Studio shuts down.

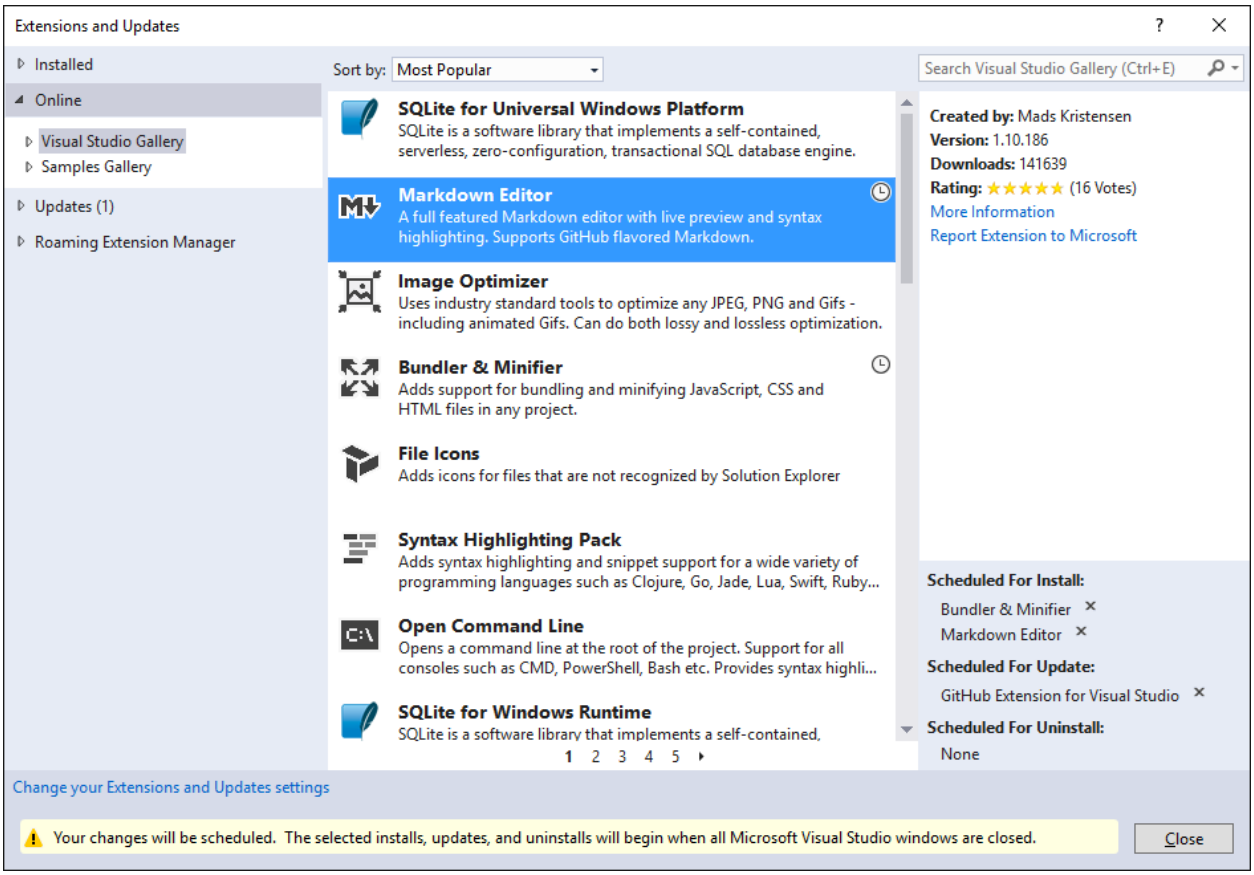

Figure 59: Scheduling Operations for Extensions

In the bottom-right corner, you can see the list of scheduled operations grouped by Scheduled For Install, Scheduled For Update, and Scheduled For Uninstall. For example, Figure 59 shows the number 1 next to the **Updates** node, which means there is one update available for an extension. When you click the **Updates** node and then click the **Update** button for each update available, the extension will be listed in the Scheduled For Update group. Figure 59 shows how, on my machine, an update is available for the GitHub Extension that will be installed when Visual Studio shuts down, along with the other extensions listed in the Scheduled For Install group.

This also applies to extensions you want to remove. You can exclude an extension from the schedule simply by clicking the **X** symbol next to its name. If you are familiar with Visual Studio 2015, you might remember how the Extensions and Updates dialog offers buttons to restart the IDE in order to complete the installation or update of one or more extensions. In Visual Studio 2017, you no longer have such buttons. Instead, you have only an option to close the dialog. All the operations will be executed in bulk when the IDE shuts down. This is a more convenient way of managing extensions that avoids moving your focus away from the development environment.

What's new with extensibility

Note: This section is dedicated to extension authoring. If you are interested in this topic, make sure you have installed the Visual Studio extension development workload. I will assume that you are familiar with extension authoring in previous versions of Visual Studio, so some steps and concepts will not be addressed in detail.

Extension authoring for Visual Studio 2017 requires facing some changes that break with previous editions because of the new extensibility model and version 3.0 of the .vsix file format. Breaking changes are due mainly to the following scenarios that Visual Studio 2017 must support:

- Developers might install multiple editions on the same machine, so extensions can no longer be installed into a centralized location, but must be installed with each edition.

- Developers might install only the Visual Studio core editor with no workloads, or only a restricted number of workloads. That means extensions must be able to detect if the workloads and components they need to work are installed.

There is an important implication here: Visual Studio 2017 can only accept extensions specifically written for this version of the IDE and installed through version 3.0 of a .vsix package. An extension written for Visual Studio 2015 cannot simply be updated or recompiled to target Visual Studio 2017. However, an extension written for Visual Studio 2017 can target through Visual Studio 2012. As an extension author, you must be aware of these considerations if you have existing extensions that you want to port to Visual Studio 2017.

Creating a blank extension for demo purposes

In order to describe the new features in extension authoring, an extensibility project is required. You can now simply create a blank project, as the new features reside in the VSIX manifest. Follow these steps:

1. Select **File** > **New** > **Project**.
2. In the **New Project** dialog, expand the language of your choice (C# or VB), then select the **Extensibility** node.
3. Select the **VSIX Project** template, supply a different project name if you want, and then click **OK**.
4. Right-click the project name in **Solution Explorer**, then select **Add New Item**.
5. In the **Add New Item** dialog, click the **Extensibility** node.
6. Select one of the available item templates, then click **OK**. If you want to be consistent with this e-book, choose the **Custom Tool Window** template.

When everything is ready, double-click the **source.extension.vsixmanifest** file. This is the extension manifest that contains all the information and metadata required to build the .vsix package.

Specifying extension prerequisites

In the manifest designer, you will see a new item called **Prerequisites** (see Figure 60).

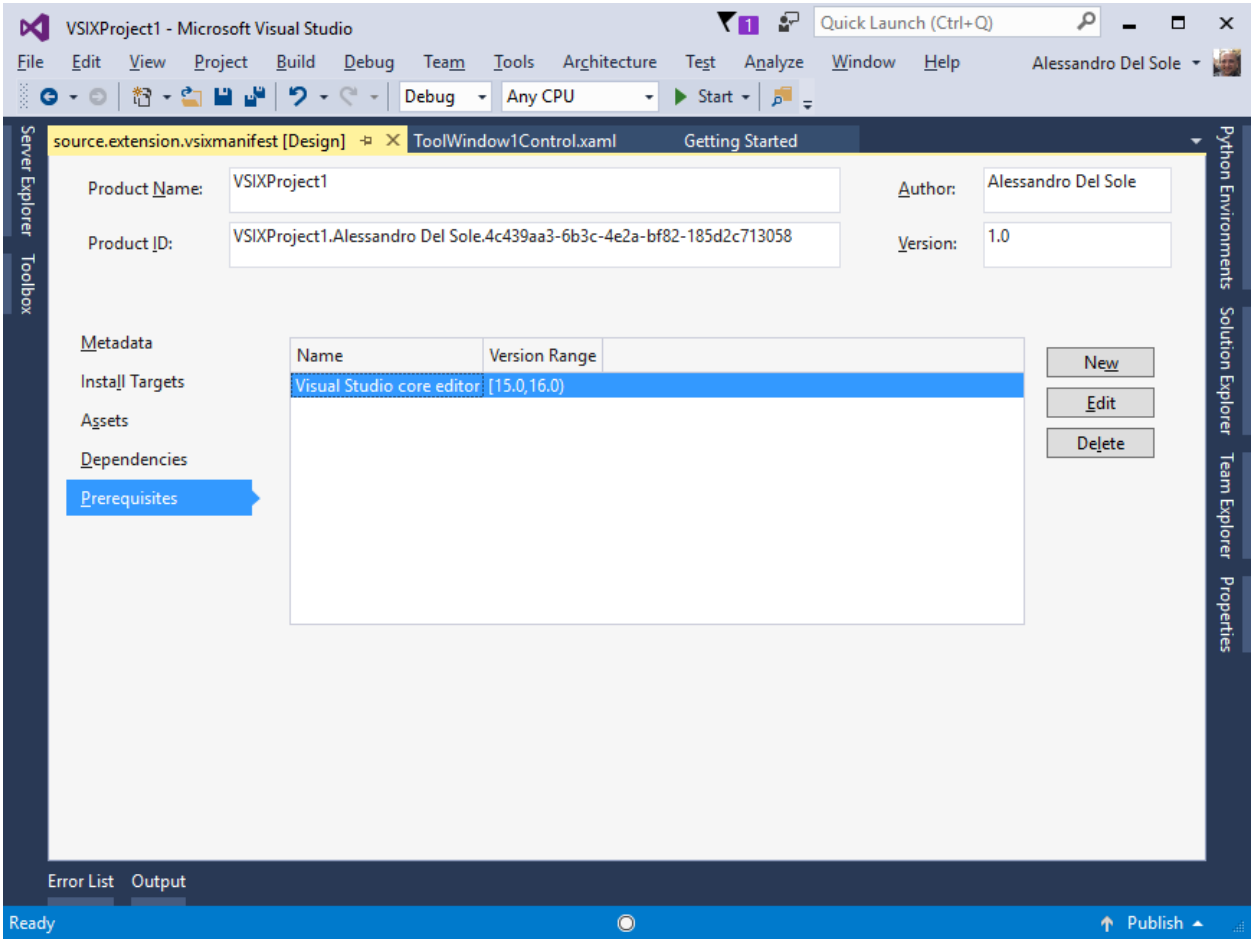

Figure 60: The New Prerequisites Item in the Manifest Designer

This new element is very important because it allows for specifying the workloads or individual components your extension requires in order to work inside Visual Studio 2017. As you can see, a prerequisite called Visual Studio core editor is available by default and targets the current and future versions of the IDE. Click **New** to add a new prerequisite. The **Add New Prerequisite** dialog will appear, and you can select a workload or component from the combo box, as shown in Figure 61.

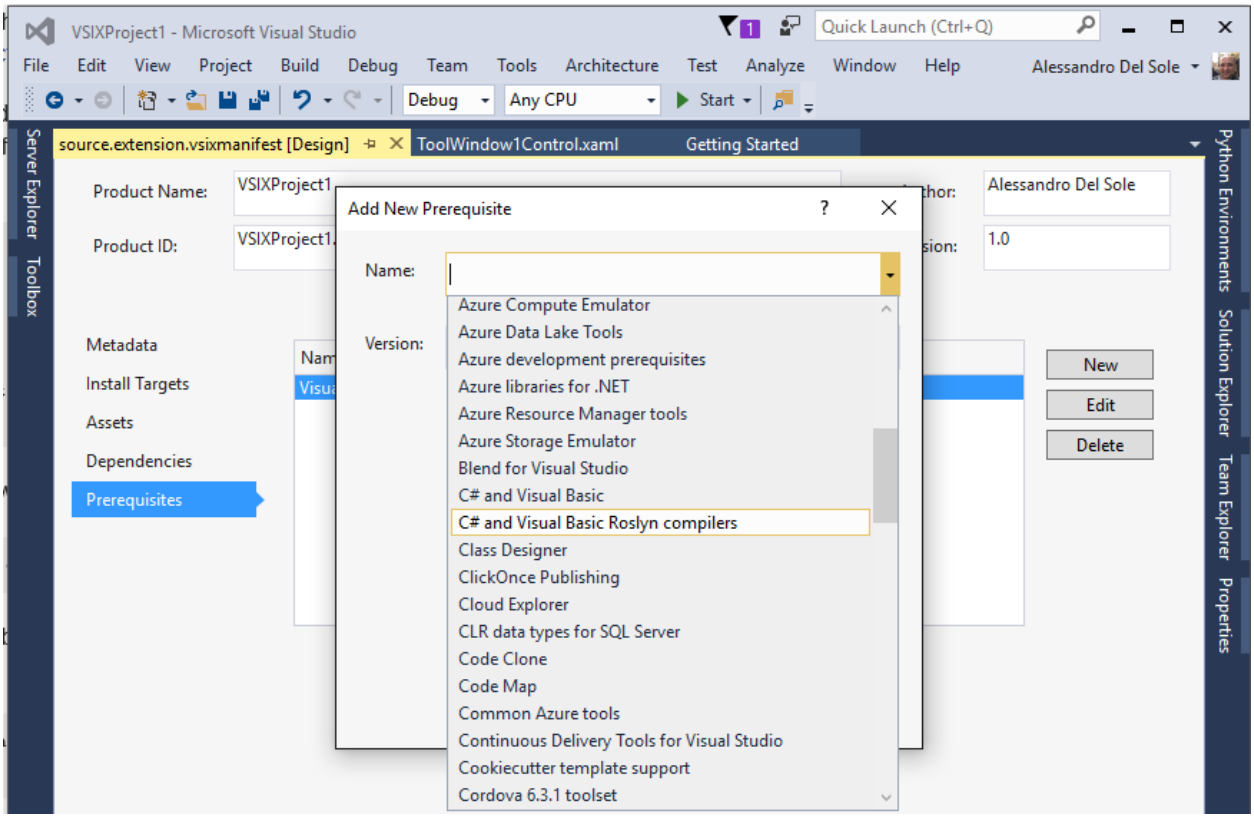

Figure 61: Selecting a Prerequisite

Once selected, you will be able to specify a version number for the prerequisite, as shown in Figure 62.

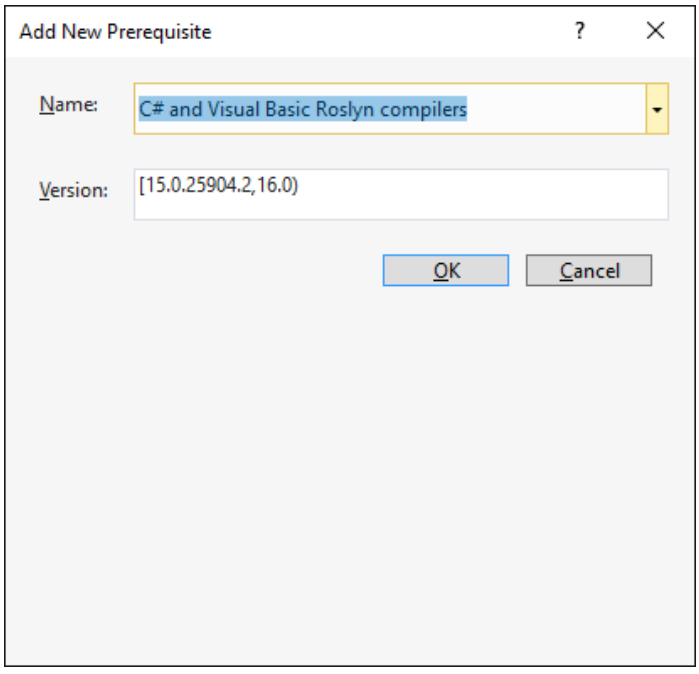

Figure 62: Specifying the Version Number for a Prerequisite

Specifying a version is optional because Visual Studio 2017 automatically selects the current IDE version detected on the development machine.

Note: This section has been written in conjunction with the Release Candidate of Visual Studio 2017, which means the default version number might be slightly different when Visual Studio reaches the RTM milestone.

After you specify the required prerequisites, you can build your .vsix package. When launched on the target machine, the VSIX installer will show the list of required prerequisites.

Ngen support and custom file installation

The new .vsix format supports the Native Image Generator tool Ngen, which allows creating native images of the extension assembly and the referenced assemblies. This can be set using the **Properties** window, as shown in Figure 63.

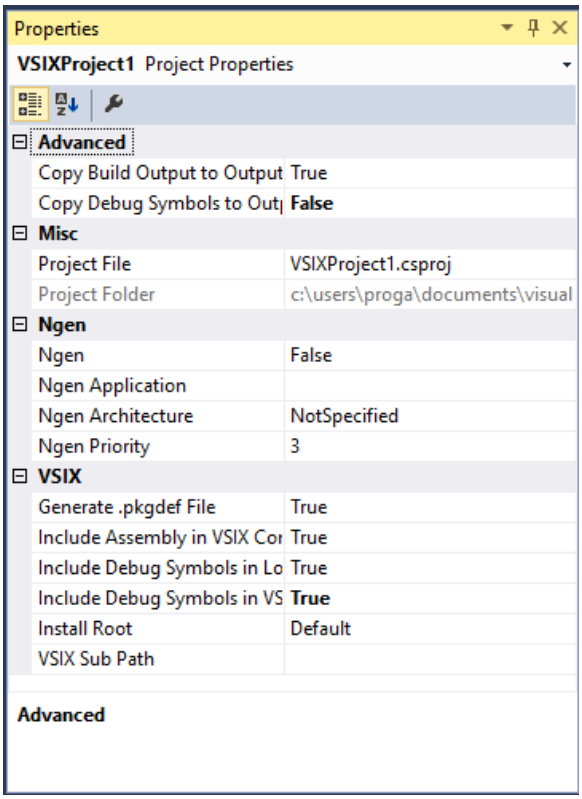

Figure 63: Ngen and File Destination Properties

The following Ngen options are available:

- Ngen: Sets whether to use Ngen. Options are True or False (default).
- Ngen Application: The application to pass into Ngen via the /ExeConfig switch.
- Ngen Architecture: Specifies the architecture among x86, x64, and All.
- Ngen Priority: Specifies the Ngen priority level.

In Figure 63, you can also see a group called VSIX, which allows you to control the behavior of some important assets, such as the .pkgdef file. Properties are self-explanatory, but it is worth emphasizing how you can specify a different installation target through the Install Root property. Here you can pick a destination folder from a predefined list, but unless you have very specific requirements, Default is a good option.

Chapter summary

Visual Studio 2017 introduces two new tools for consuming third-party extensions—the Roaming Extension Manager, which makes it easy to make the same extension available across machines, and support for scheduling operations over extension installation, update, and removal. This avoids the need to close and restart the IDE each time. For extension authors, Visual Studio 2017 introduces version 3.0 of the .vsix file format, which requires specifying extension prerequisites (workloads or components), and allows controlling the extension behavior by creating native images of the assemblies and by controlling the destination folder.

Chapter 7 Debugging and Testing Improvements

Debugging and testing code are two fundamental tasks in the application development lifecycle, and it should be no surprise that Visual Studio 2017 introduces important improvements for both. Such improvements make debugging and testing faster but still efficient while you keep your focus on the active editor. This chapter will describe new features, but it will also walk through updated tools you already know and that, in Visual Studio 2017, come to a new life.

Introducing Run to Click

When we are debugging, stepping through lines of code is one of our most common operations. We use breakpoints, and then use features such as Step Into, Step Over, or Step Out in order to understand the behavior of code blocks, local variables, and more generally, the application execution flow. Before Visual Studio 2017, we had to introduce temporary breakpoints to continue the execution from a breakpoint to a certain point in our code. Visual Studio 2017 makes a step forward and introduces a new feature called **Run to Click**. To understand how this feature works, consider Figure 64, in which you can see that a breakpoint has been hit and the application is in break mode.

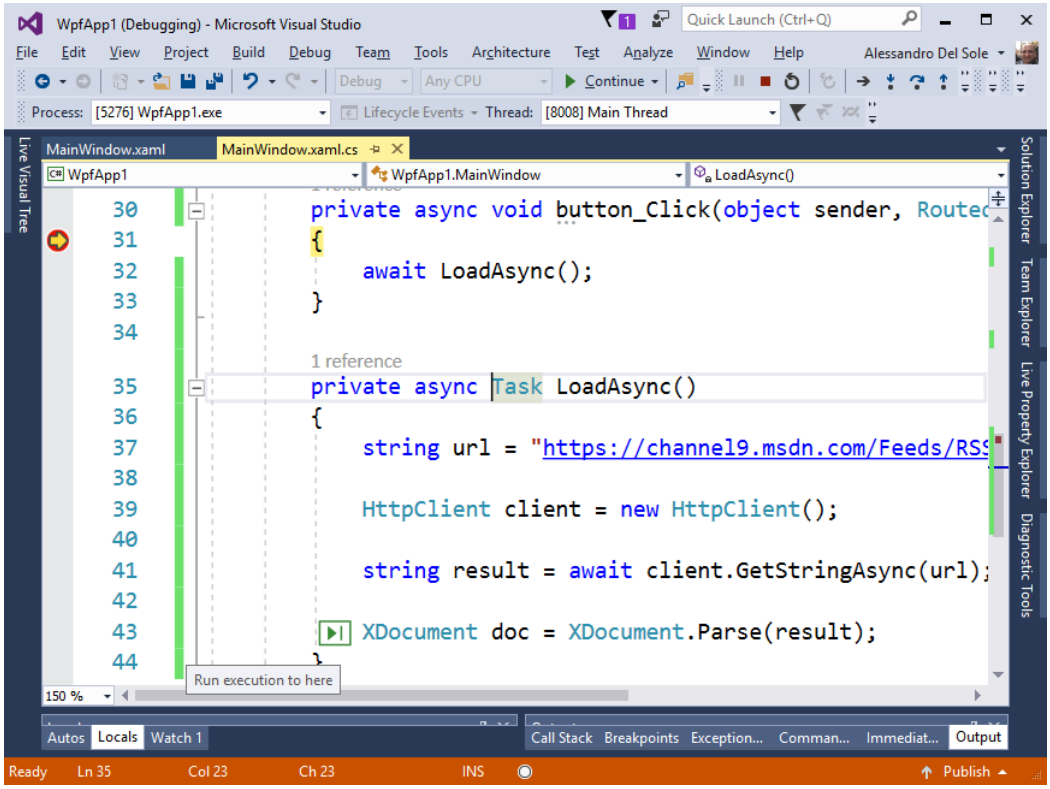

Figure 64: The Run to Click Icon

When you hover over a line of code, a green glyph appears near the line, as shown on line 43 of the code file in Figure 64. This glyph represents the Run to Click button. If you click it, your code will be executed to that line, without the need for temporary breakpoints. Actually, the line where Run to Click is activated is highlighted but not executed, exactly as it would happen if a breakpoint was set on that line (see Figure 65).

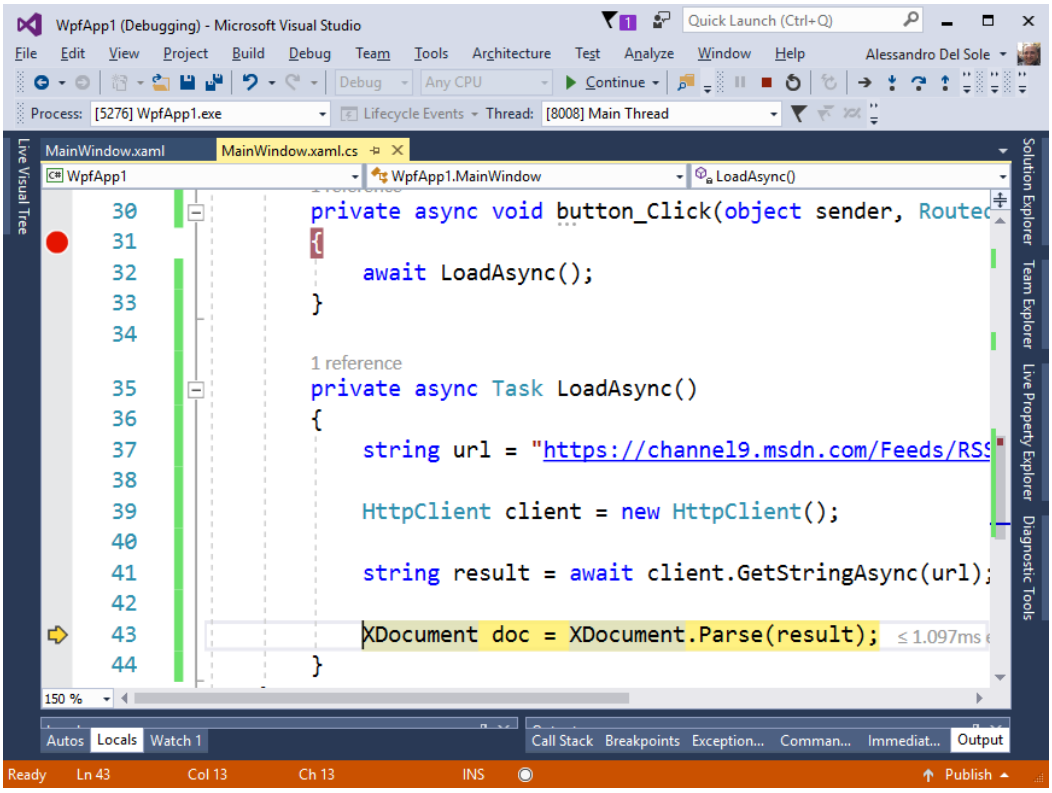

Figure 65: Run to Click Stops at a Line of Code without a Breakpoint

You can then continue to use the **Run to Click** button and execute code to a specific point, thereby avoiding temporary breakpoints.

Tip: Keep in mind that Run to Click works only in break mode, which means it is not available when the application is running.

Updated diagnostic windows

Visual Studio 2015 introduced the **Diagnostic Tools** window, which shows application events, memory and CPU usage, and other diagnostic information at debugging time. This tool window appears by default when you start debugging your application. In Visual Studio 2017, the Diagnostic Tools window has been updated with a view called **Summary** (see Figure 66).

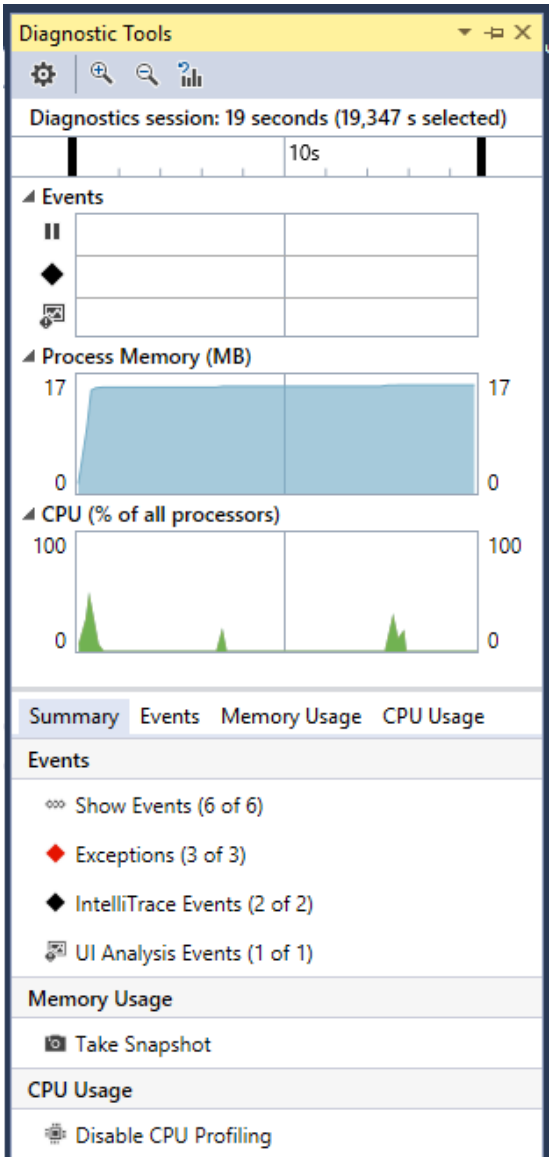

Figure 66: The New Summary View in Diagnostic Tools

This new view contains a summary of the number of application events (Show Events), exceptions, IntelliTrace events, Application Insights (if applicable), and UI Analysis events for UWP apps. It provides shortcuts to take snapshots of the managed heap and to enable or disable CPU profiling.

Tip: You can enable or disable specific IntelliTrace events by going to** Tools **>** Options **>** IntelliTrace **>** IntelliTrace Events**.

Note: UI Analysis events are a new feature in Universal Windows Platform that detects accessibility issues. They will be addressed in Chapter 8, "Visual Studio 2017 for mobile development."

This new view provides quick insight into what's happening at debugging time, then you can use the other tabs for further details.

Analyzing exceptions with the Exception Helper

The Exception Helper is a pop-up that appears when an unhandled or thrown exception occurs at debugging time. Through the Exception Helper, Visual Studio shows details about the exception. Previous versions of Visual Studio offered the Exception Assistant and Exception Dialog tools, which took different approaches to productivity. In Visual Studio 2017, the Exception Helper has a completely new look and improved behavior. Before we examine the new benefits, let's look at Figure 67, which shows the updated Exception Helper in action.

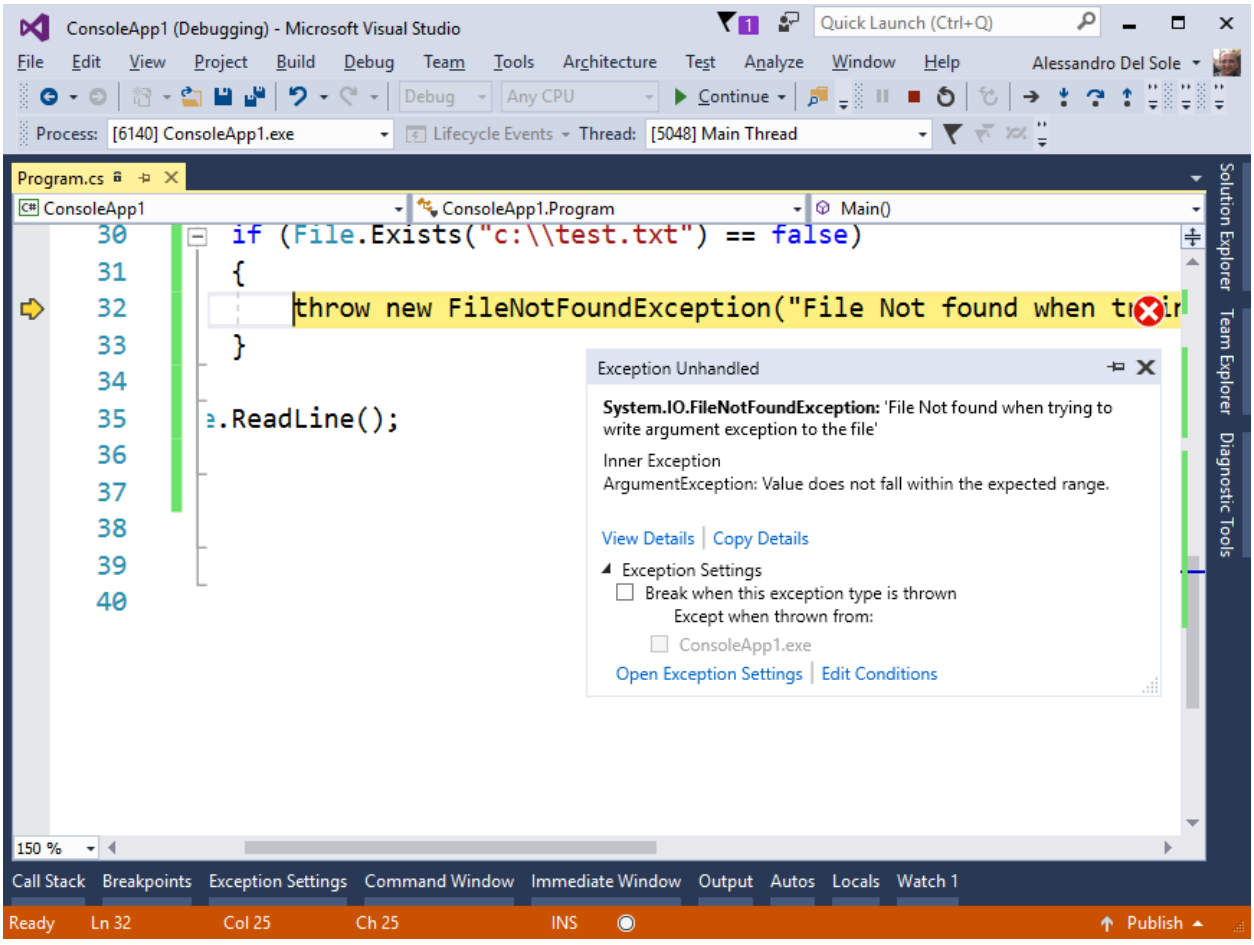

Figure 67: The Updated Exception Helper

Let's summarize what's new:

- The Exception Helper will appear whether you are debugging managed or unmanaged code.
- When an unhandled exception occurs, the entire line of code is highlighted. This improves code readability. An exception error icon will help you understand why the execution was interrupted.
- The Exception Helper pop-up is smaller, nonmodal, and less distractive.
- At a glance, the pop-up shows only the exception type, the error message, and whether the exception was thrown or unhandled.
- The pop-up immediately shows any inner exceptions—without the need of additional dialogs.
- In the **Exception Settings** group, you can specify if the debugger must break the execution when the exception is thrown, and you can exclude specific modules from breaking the execution.
- If you click the **Open Exception Settings** hyperlink, the exception information will be shown inside the **QuickWatch** dialog (see Figure 68). This makes it easy to investigate the exception details and to reevaluate an expression.
- You can click the **Edit Conditions** hyperlink to specify when the execution should be broken by including or excluding specific modules (see Figure 69). This is the only case in which you interact with a modal dialog from the Exception Helper.

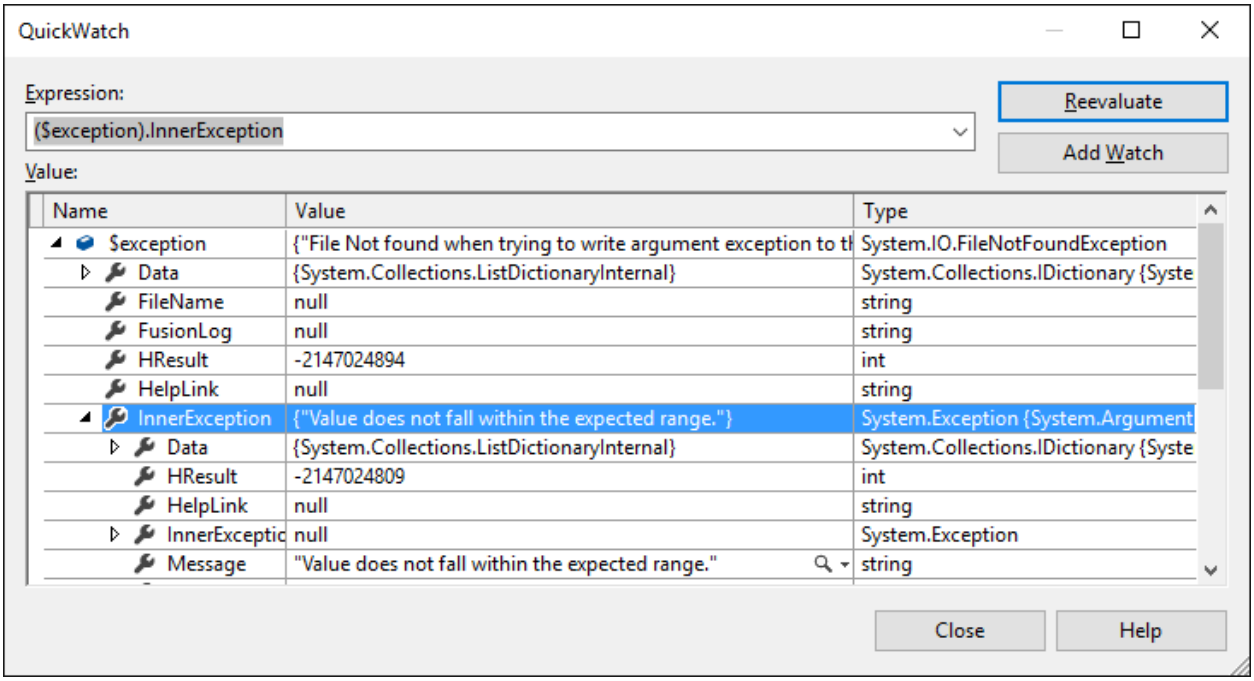

Figure 68: The Exception Settings within the QuickWatch Dialog

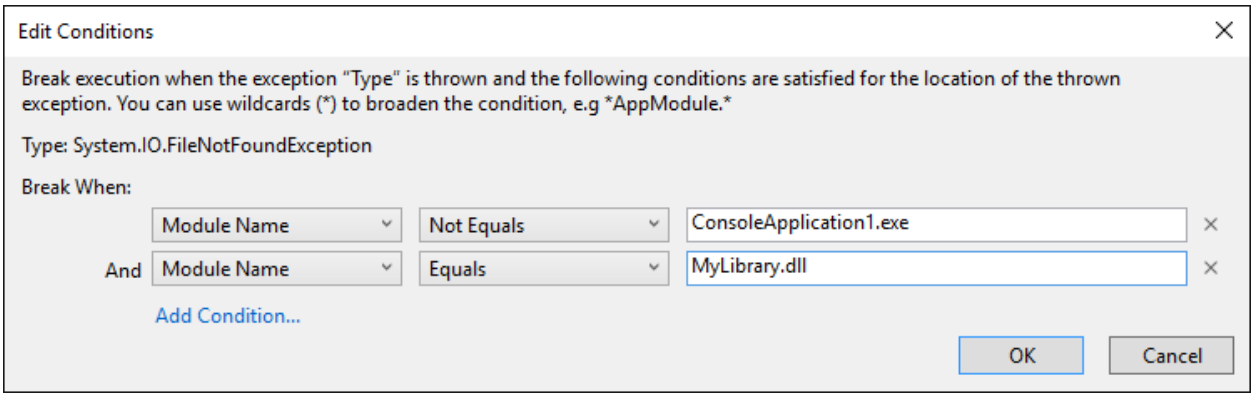

Figure 69: Editing Exception Conditions

With its new look and behavior, the Exception Helper improves productivity by providing all the information you need while letting you focus on the code.

Introducing Live Unit Testing

Note: This feature is available only in the Enterprise edition and only for C# and Visual Basic, and it assumes you already have basic knowledge of unit testing. If you need some guidance, the MSDN documentation has everything you need to get started.

Live Unit Testing is a new, exciting feature in Visual Studio 2017. It allows for executing unit tests in the background and showing their results and coverage in the code editor as you type. The best way to understand how this feature works is with an example. Create a console application and consider Code Listing 4.

Code Listing 4

```
namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            var rh = new RectangleHelpers();
            double result = rh.CalculateArea(10, 10);
        }
    }
```

```
    public class RectangleHelpers
    {
        // Calculate the area of a rectangle.
        public double CalculateArea(double width, double height)
        {
            return width * height;
        }
    }
}
```

The **RectangleHelpers** class provides a very simple **CalculateArea** method that returns the area of a rectangle when given width and height. The class is instantiated and the method is invoked in the **Main** method of the **Program** class. Now you need to create some unit tests. Right-click the **CalculateArea** method, then select **Create Unit Tests**. The **Create Unit Test** dialog (see Figure 70) appears and asks you to specify basic information for a test project. By default, the test framework is MSTestv2 and the offer is to create a new test project. Leave all the default settings and click **OK**. Of course, you can change the default settings if you have previous experience with unit testing in Visual Studio. At the time of writing, the supported test frameworks are MSTest, xUnit, and NUnit.

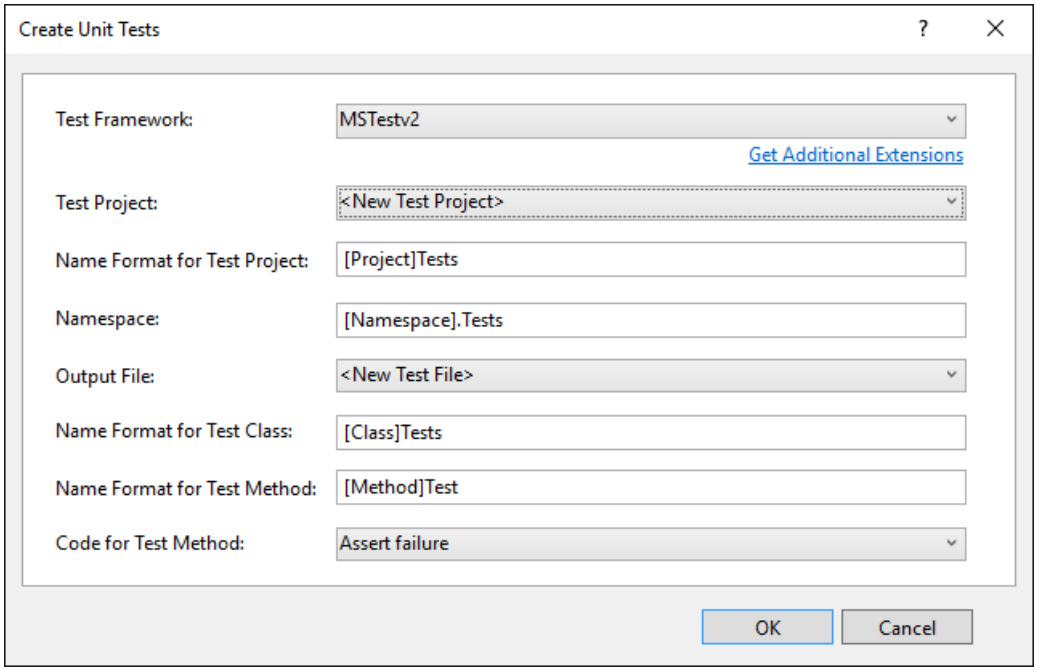

Figure 70: Creating a Test Project

Now you have a test class called **RectangleHelperTests** and a method called **CalculateAreaTest**, both available in the RectangleHelpersTests.cs file of the test project. They are decorated with the **TestClass** and **TestMethod** attributes, respectively, to instruct Visual Studio that they are within a test project and that they will be used by the proper testing framework and tools. At this point, you need to implement your unit test. Code Listing 5 shows a very simple implementation that checks for equality between the expected and actual results.

Code Listing 5

```
namespace ConsoleApp1.Tests
{
    [TestClass()]
    public class RectangleHelpersTests
    {
        [TestMethod()]
        public void CalculateAreaTest()
        {
            var rectHelpers = new RectangleHelpers();
            double width = 3;
            double height = 2;

            Assert.AreEqual(6, rectHelpers.CalculateArea(width, height));
        }
    }
}
```

Now go back to the Program.cs file. Select **Test** > **Live Unit Testing** > **Start**. After a few seconds, you will see how Visual Studio runs the unit test in the background, showing the results live in the editor (see Figure 71).

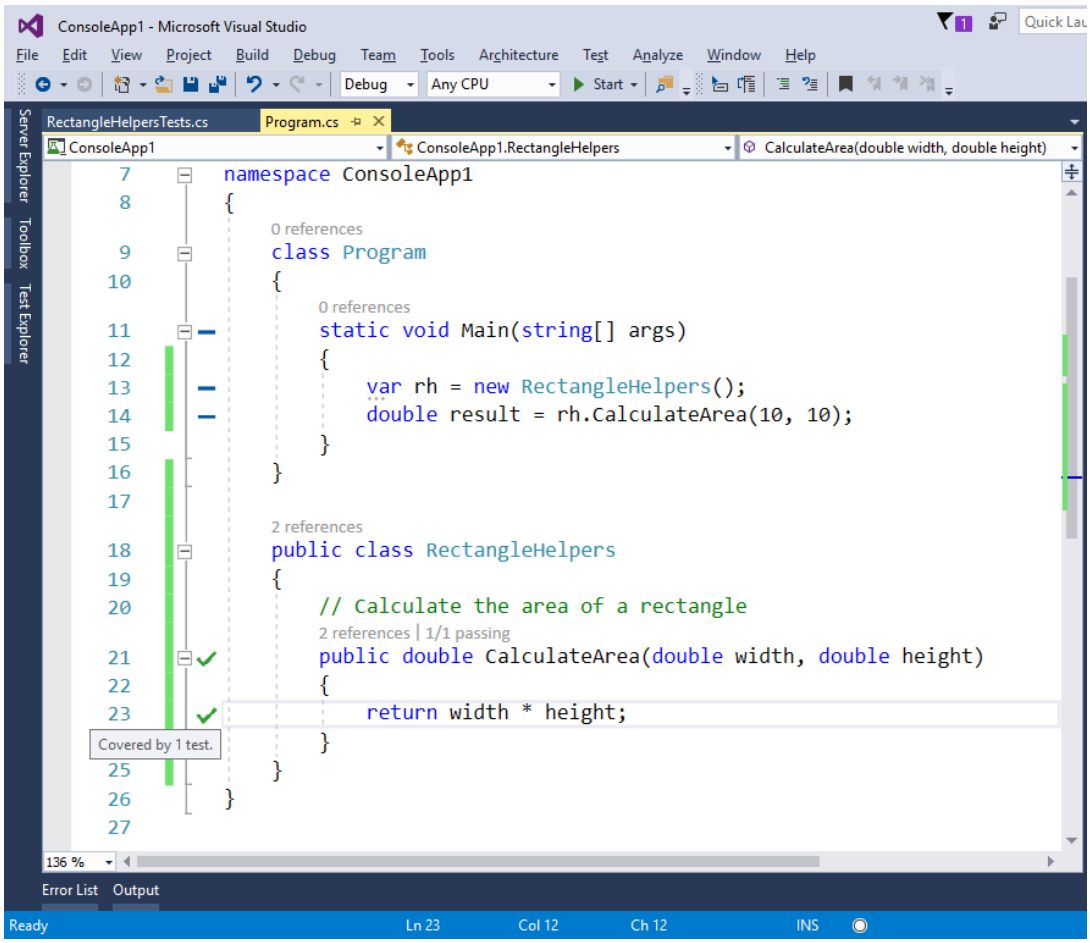

Figure 71: Live Unit Test Results in the Code Editor

Instead of running unit tests with Test Explorer, Visual Studio 2017 is able to run tests in the background and, most importantly, associate coverage and results based on the actual code, not on unit tests.

Note: Visual Studio 2017 can associate Live Unit Testing to your code only if unit tests have a reference to an object. In our example, the unit test defines an instance of the `RectangleHelpers` class, and so it creates a reference. Then Visual Studio 2017 associates the `CalculateAreaTest` method to the `CalculateArea` method based on this reference.

Icons you see in the code editor will help you understand test results and code coverage. More specifically:

- A green icon with a check symbol means that a piece of code has been covered by a passing unit test.
- A blue icon of a horizontal line represents code that is not covered by any unit tests.
- A red **X** icon represents code that has been covered by a unit test that did not pass.
- Any of these icons with an overlaid clock icon indicates code that is currently being edited.

Now edit the **`Assert.AreEqual`** statement in the **`CalculateTestArea`** method as follows to make the test fail:

```
Assert.AreEqual(5, rectHelpers.CalculateArea(width, height));
```

While editing, you will notice that the icons show an overlaid clock. When you are done, you will see that the code editor is updated to show the unit test result (failed, in this case), as shown in Figure 72.

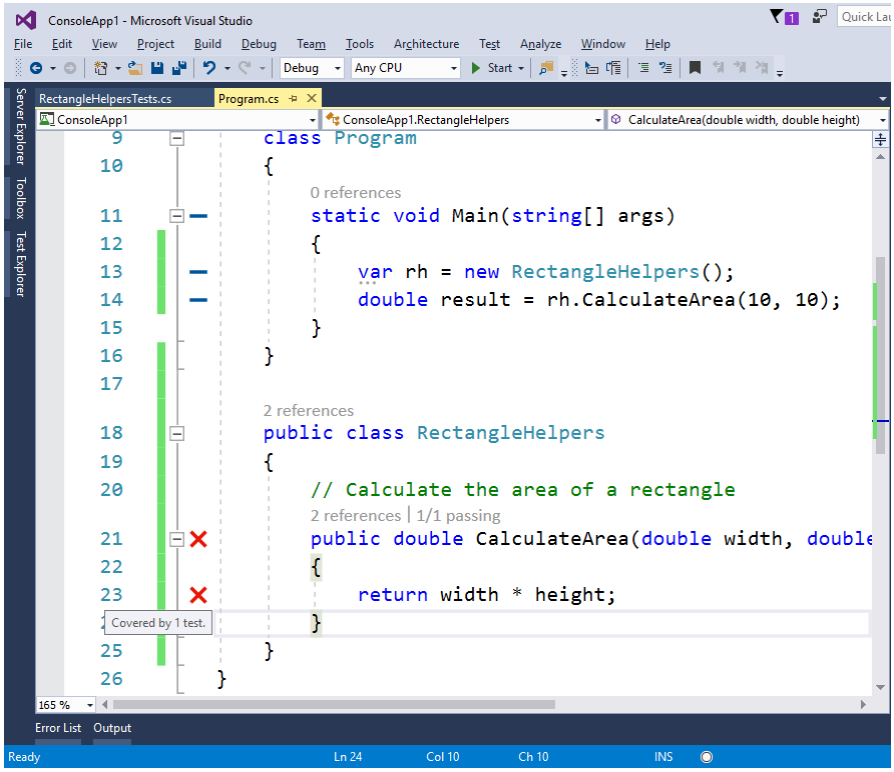

Figure 72: The Code Editor Updated with Unit Test Results

Tip: If you hover over an icon, you will get details about the test method that was invoked in the background (and its results).

Whatever changes you make to your code, Live Unit Testing will run in background, showing the result in the code editor. You can control Live Unit Testing with commands inside **Test** > **Live Unit Testing**—for example, Pause, Stop, and Restart. These are only visible when the feature is active.

Miscellaneous improvements

In addition to the major updates described in this chapter, Visual Studio 2017 introduces miscellaneous improvements that make the debugging experience even better. These are briefly summarized in this section.

Accessibility improvements

Most debugging windows (Locals, Watch, QuickWatch, Autos, Call Stack) have been improved for readability for screen readers and other accessibility features.

IntelliTrace events for .NET Core

IntelliTrace, the historical debugger in Visual Studio, now supports tracing events in .NET Core applications. More specifically, you can enable IntelliTrace to track ADO.NET, MVC, and HttpClient events. Go to **Tools** > **Options** > **IntelliTrace** > **IntelliTrace events** to set events you want to record with .NET Core.

Profiling tools updates

Profiling tools such as the Performance Profiler, CPU Usage, GPU Usage, and Performance Wizard can now attach to a running process. An option for this becomes available when you start the desired profiling tool. The CPU Usage tool has also been improved when working with external code to provide more detailed information.

Support for Chrome with JavaScript

With ASP.NET, if you debug an application using Chrome as the browser, the debugger will run against JavaScript code running in Chrome.

Chapter summary

Debugging and testing are very important tasks in the application development lifecycle, and Visual Studio 2017 introduces new goodies for both. With Run to Click, you no longer need temporary breakpoints to run code to a specific point while in break mode. The Diagnostic Tools window now provides a Summary tab with shortcuts that allow you to keep your focus on the IDE—it provides this along with an updated version of the Exception Helper that shows exception details in a simplified and focused manner. And with Live Unit Testing, you can write code and run unit tests in the background, getting test results and code coverage directly in the code editor, all live as you type.

Chapter 8 Visual Studio 2017 for Mobile Development

Enabling developers to build apps for any platform and any device is a major goal of Visual Studio 2017. This certainly includes mobile app development. With the 2016 Microsoft acquisition of Xamarin, and with improvements to the Universal Windows Platform and to the tools for Apache Cordova, Microsoft makes Visual Studio 2017 the ultimate development environment for building apps that run on Android, iOS, and Windows with shared codebases. This chapter is not intended to be a guide to mobile development with Visual Studio 2017, rather it will highlight what's new in the most popular tools for cross-platform development with Microsoft's IDE. Links to the documentation for each tool are provided where appropriate.

Visual Studio 2017 and the Universal Windows Platform

Tip: Documentation to get started with Universal Windows Platform is available at developer.microsoft.com/en-us/windows/apps/getstarted.

Note: Building Universal Windows apps requires selecting the Universal Windows Platform development workload in the Visual Studio Installer.

The Universal Windows Platform (UWP) offers an incredibly rich set of APIs for building apps that run on Windows 10 and on equipment such as PCs, tablets, smartphones, Xbox, HoloLens, and Internet of Things (IoT) devices. In most cases, you code once and your UWP app will run on all the aforementioned devices; in some cases, you will need to make adjustments for specific SDKs.

In March 2016, Microsoft announced the Anniversary Update for Windows 10, a major update delivered in July 2016 that added new features and improvements to the operating system and included some goodies for developers via the Windows 10 Anniversary Update SDK. The latter is already included in Visual Studio 2017 through the UWP development workload. The SDK contains integrated tools for development with Visual Studio 2017 and emulators that support the Anniversary Update. You can still choose the target version and minimum version for your UWP apps when creating a new project. For example, select **File** > **New** > **Project**, then select the **Blank App (Universal Windows)** project template in the **Windows Universal** folder of the C# and VB languages. Before generating the solution, Visual Studio will ask you to specify the target and minimum versions with the dialog shown in Figure 73.

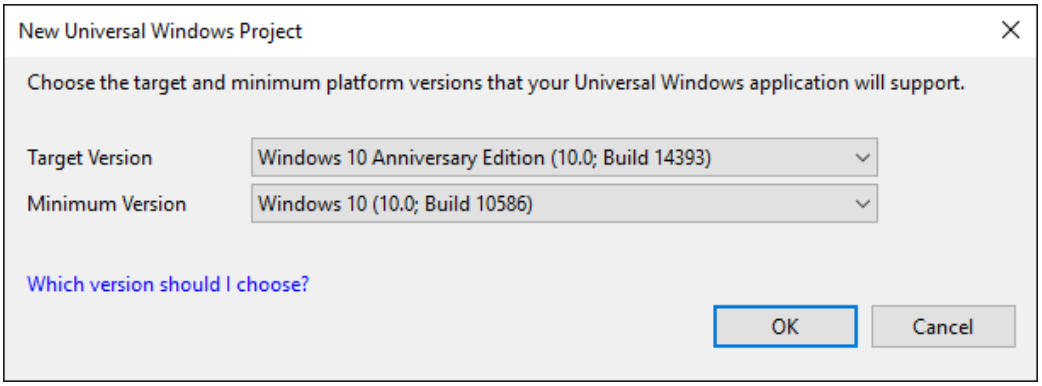

Figure 73: Selecting Target and Minimum Versions for UWP Apps

In most cases, leaving the default selection is the best choice. When you click **OK**, Visual Studio generates a solution that supports the specified Windows 10 versions. Do not delete the new project—keeping it enables the new project to serve as a test environment for future topics. Along with support for the Anniversary Update, Visual Studio 2017 includes the following improvements for UWP.

Note: The Universal Windows Platform also receives all the improvements described in Chapter 4, "XAML improvements."

Updates to .NET Native

As you might know, compiling a Universal Windows app in Release mode involves the .NET Native tool chain. Think of .NET Native as a compiler that directly produces a native image of the app instead of generating intermediate language (IL) that would need a just-in-time compilation. As a consequence, starting up a Universal Windows app is much faster than a .NET application or a Windows Phone app. With Visual Studio 2017, the .NET Native tool chain is updated with more than 600 bug fixes, runtime performance optimization, and overall improvements to the entire tool.

Note: If you have ever written a UWP app, you know that compiling in Release mode requires much more time than in Debug mode because for Release, Visual Studio 2017 invokes .NET Native. Behind the scenes, .NET Native is generating IL from your C# or VB code, then converting the IL into C++—finally producing native binaries. This is why it takes so long, but this is also why universal apps are so efficient.

Updated NuGet packages

Universal Windows apps rely on the Microsoft.NETCore.UniversalWindowsPlatform NuGet package. For full alignment with the Anniversary Update, .NET Core, and Visual Studio 2017, the aforementioned NuGet package is updated to version 5.2.2. This version mostly addresses issues reported by developers, and existing apps can be easily updated with the NuGet Package Manager tool in Visual Studio 2017.

XAML improvements for UWP

Visual Studio 2017 introduces some improvements to XAML that target only UWP. The first improvement addresses UI elements you create by using drag-and-drop from the Toolbox. Visual Studio 2017 now reduces the number of auto-generated XAML tags, which means the resulting markup is clearer and more readable. The second improvement addresses the XAML Designer—Visual Studio 2017 introduces a new button called **Device Preview Settings**, which can be found in the upper bar of the Designer. Double-click the **MainPage.xaml** file to open the XAML Designer if it is not already available. The new button is represented by a gear icon and allows for quick changes to contrast and theme settings. When you click it, the **Device Preview Settings** dialog appears (see Figure 74).

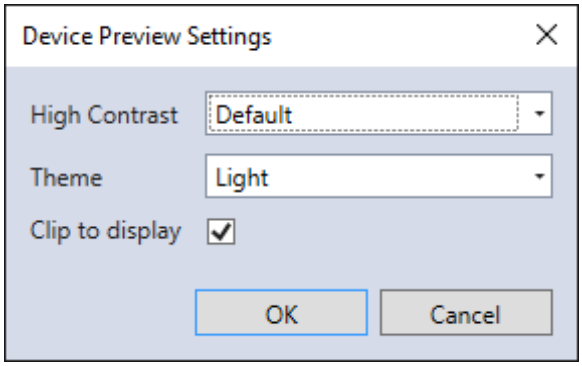

Figure 74: Changing Contrast and Theme Settings at Design Time

You can choose among a number of possible contrast settings and between the Light and Dark themes. This makes it easy to see how the UI of your app behaves with different settings at design time. The third improvement addresses the **Properties** window. You can now assign properties with basic mathematical equations that will be evaluated immediately, and the resulting value will be assigned to the selected control in XAML. Figure 75 shows an example in which the **`Width`** property of a **`Button`** control is assigned the sum of 60 + 40.

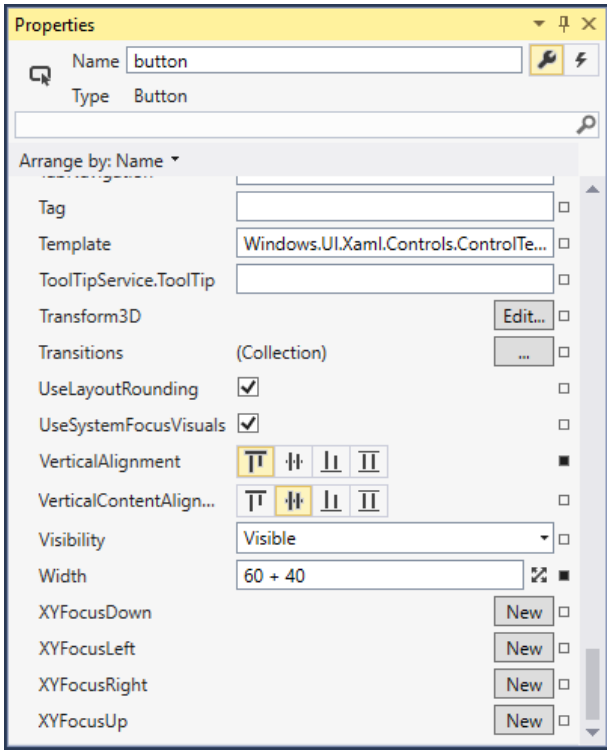

Figure 75: Basic Equations—Evaluated and Assigned to UI Elements' Properties

In this case, the evaluation result (100) will be assigned to the **`Button.Width`** property and will be visible in the XAML code. It supports sums, subtractions, multiplications, and divisions. Why is this feature useful? Think of the **`Button.Width`** with a starting value of 60, then suppose you wish to see how the button appears by increasing the width by 40. Instead of calculating the new value in your head, you can simply write the expression 60 + 40.

Updated Manifest Designer

Probably the most important update for UWP, from the IDE perspective, addresses the Manifest Designer, which you enable by double-clicking the **Package.appxmanifest** file in **Solution Explorer**. More specifically, the **Visual Assets** tab now has a new feature called **Asset Generator**, which can automatically generate all the required assets starting from a single source image.

Note: The Visual Assets Generator in Visual Studio 2017 is only available for Visual Basic and C#. Support for C++ and JavaScript is planned for a future release.

You simply need to supply an image file then click **Generate** and the Asset Generator will create tiles, logos, icons, and splash screens at any or all scales to fit every type of device your app targets. Figure 76 shows how the **Visual Assets** tab appears in Visual Studio 2017, with a sample image supplied as the source.

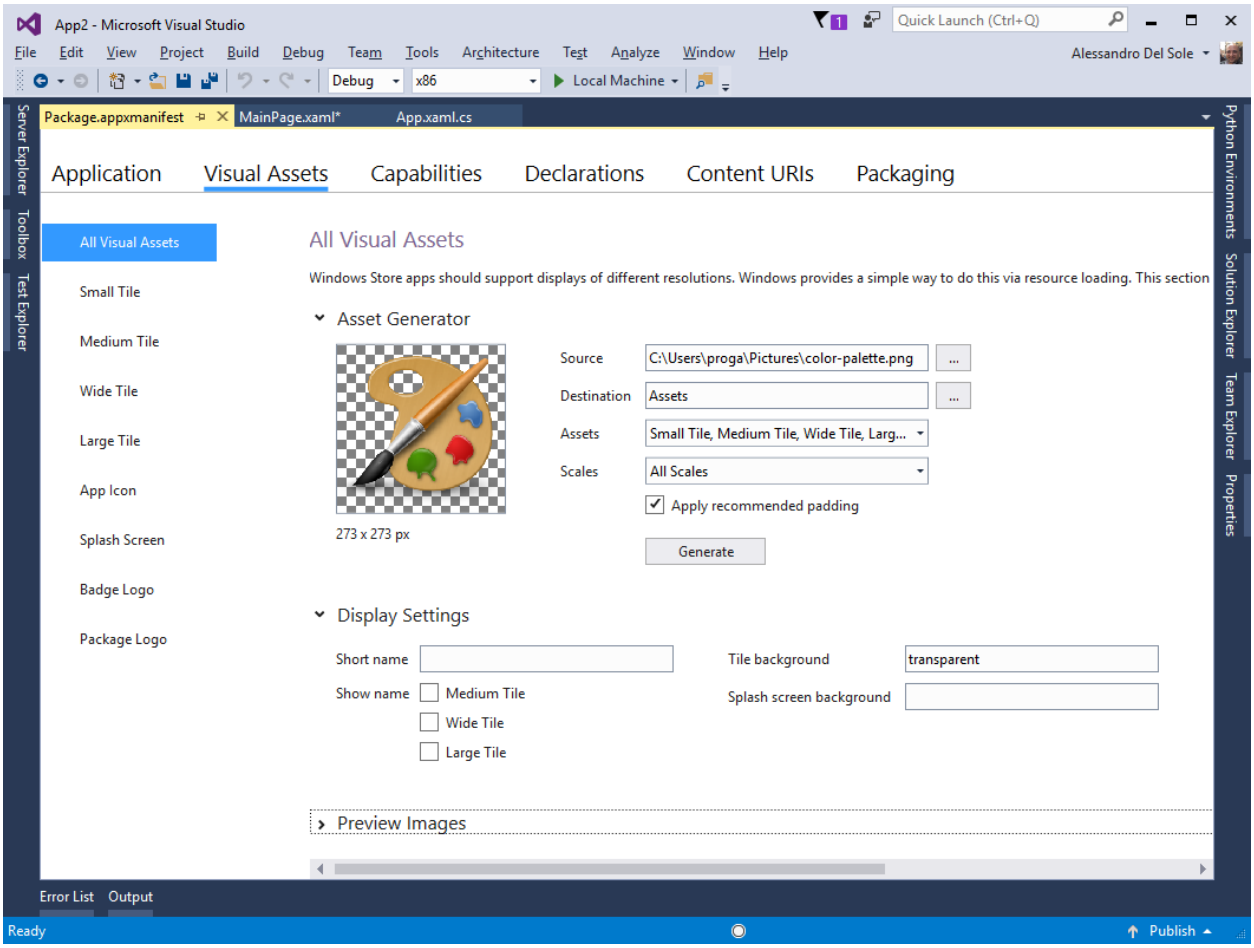

Figure 76: The New Asset Generator in Visual Studio 2017

When you click **Generate**, Visual Studio generates all the required assets, as you can see by scrolling down the window (see Figure 77). All the generated image files will be listed in **Solution Explorer**. By default, the **Visual Assets** tab shows all the available visual assets, but you can also select a specific asset category from the list on the left.

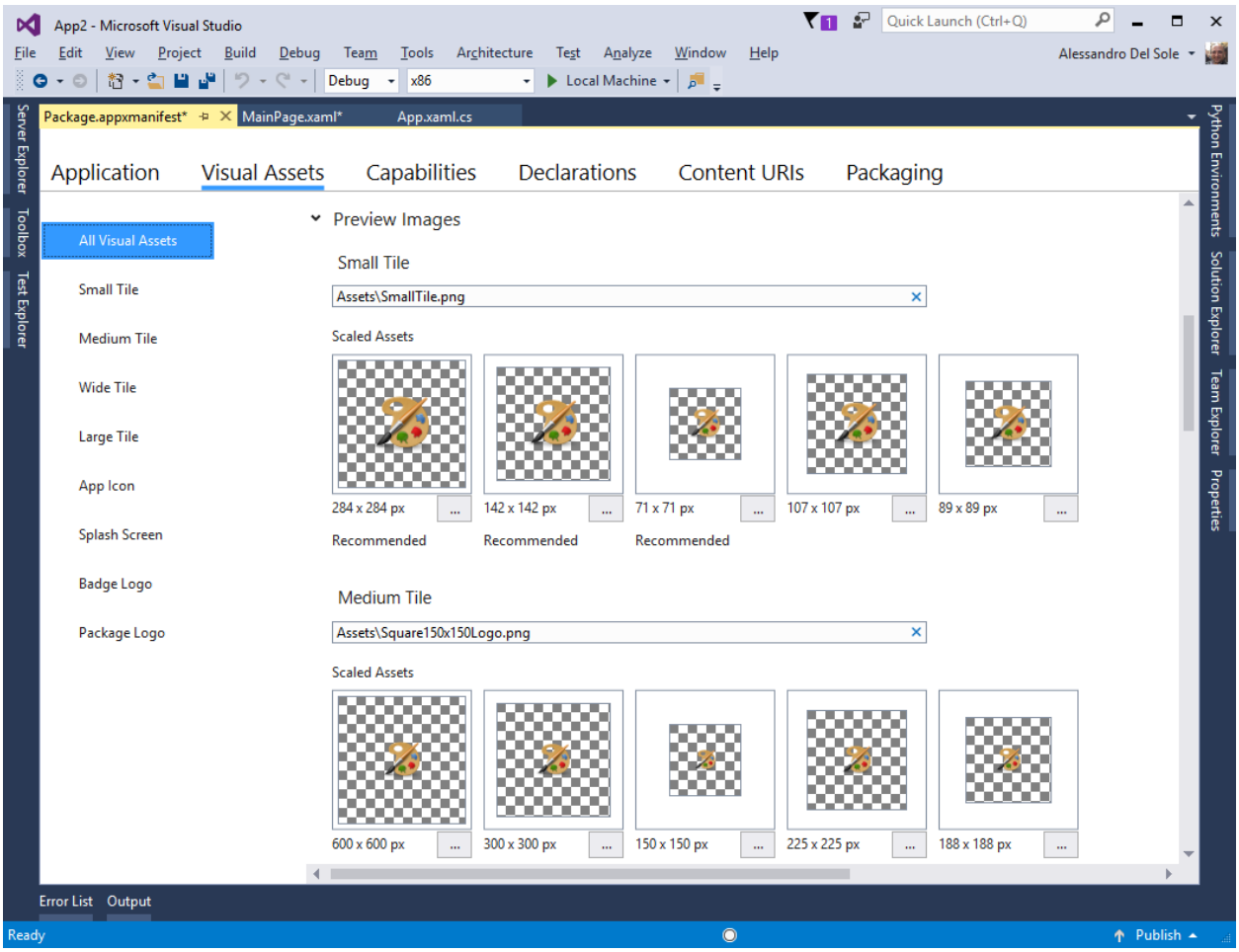

Figure 77: The Auto-Generated Assets

Visual Studio 2017 generates assets, making sure they adhere to all the design guidelines suggested for Windows 10 apps, such as padding and background colors. This feature is extremely useful and will help you save a huge amount of time because you will no longer need to supply individual assets separately.

Tip: The Badge Logo (for lock screen notifications) and Package Logo assets must be generated individually by selecting the corresponding items in the assets list. You will still supply a single source image, and you will click the** Generate **button. Visual Studio will then generate the required assets according to the Windows 10 guidelines. For instance, for the Badge Logo, the generator will automatically reduce colors as appropriate.

UI Analysis tool

The **Diagnostic Tools** window in Visual Studio 2017 introduces a special analysis tool for UWP apps called **UI Analysis**. This tool analyzes an app for accessibility and performance issues while debugging. UI Analysis is not enabled by default, which means that the first time you start debugging an app in the Diagnostic Tools window, you must click **Select Tools**, then select the **UI Analysis** item (see Figure 78).

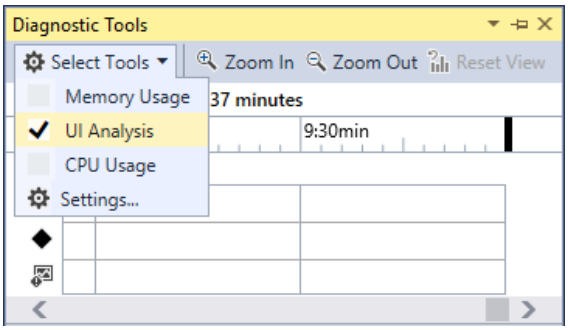

Figure 78: Enabling UI Analysis

Once enabled, UI Analysis will be available only the next time you start debugging the app, which means you must first stop the execution and then restart it. In order to understand how this feature works, let's introduce a couple intentional issues. In the XAML markup of the MainPage.xaml file, insert the code shown in Code Listing 6 inside the default **`Grid`**.

Code Listing 6

```
<ListView>
    <ListView.ItemsPanel>
        <ItemsPanelTemplate>
            <StackPanel/>
        </ItemsPanelTemplate>
    </ListView.ItemsPanel>
</ListView>
```

This code has two issues: first, the **`ListView`** is not virtualized because it is using a **`StackPanel`** container instead of an **`ItemsStackPanel`**, so the **`ListView`** can have performance problems when bound to a collection of objects. Second, its **`Name`** property has not been assigned, which means there is no chance to interact with it in code. At this point, start debugging by pressing **F5**, then keep an eye on the **Diagnostic Tools** window. In the **Summary** tab you will see two UI Analysis Events, and if you click the **Events** tab, you will see the full list with details for each event (see Figure 79).

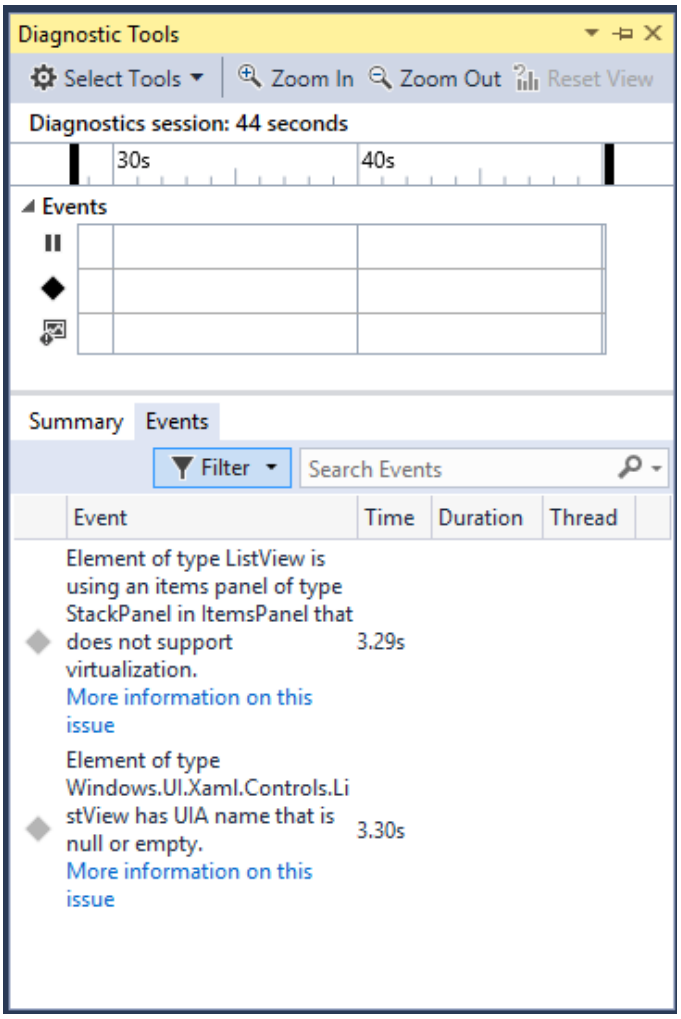

Figure 79: The List of UI Analysis Events with Details

As you can see, the list of UI Analysis events shows detailed information for each event. If you double-click an event, you will be redirected to the line of XAML code that caused the issue. You can also click the **More information on this issue** hyperlink to open the documentation for an issue, as shown in Figure 80.

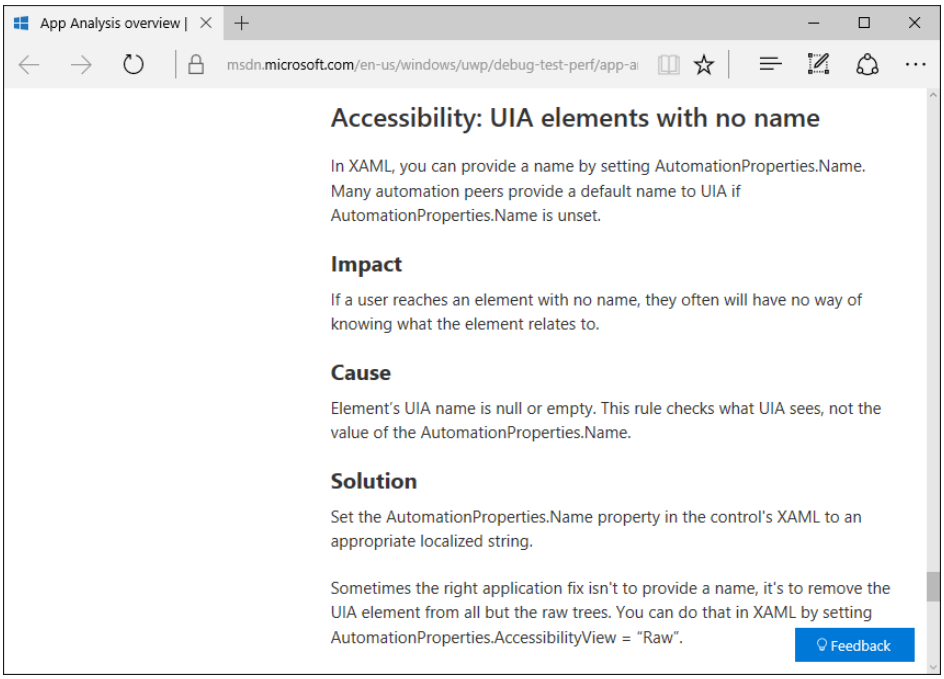

Figure 80: The Documentation of a UI Analysis Issue

The documentation also offers a page called App Analysis overview that contains the full list of issues the UI Analysis tool can detect. The UI Analysis tool is an extremely useful addition because it allows you to discover a large number of potential problems before you submit the app to the Windows Store for approval and publication.

Cross-platform development with Apache Cordova

Note: Building apps with Apache Cordova requires selecting the Mobile development with JavaScript workload in the Visual Studio Installer. Also, the Chrome browser is required for new features such as Cordova Simulate.

Apache Cordova is a development platform that allows us to create cross-platform apps for Android, iOS, and Windows using HTML and JavaScript. Visual Studio 2017, as well as its predecessor, includes the **Tools for Apache Cordova** (TACO) so that you can use your favorite powerful IDE to write, debug, and publish apps. Also, VS 2017 supports Ionic, a popular front-end JavaScript framework that works with Cordova. There are a number of changes and improvements in TACO for Visual Studio 2017 that enhance performance and productivity. These new additions will be demonstrated using the WeatherApp sample application included in the official Microsoft examples for Cordova. You can download these from GitHub as a .zip archive. The **weather-app** folder contains the solution you can open in Visual Studio 2017. Of course, you can also use any other Cordova project.

Tip: When opening any Cordova solutions built with previous versions of Visual Studio, VS 2017 needs to perform a one-time upgrade. It will ask you to reload the solution after the upgrade process.

Supported versions and platforms

In Visual Studio 2017, Tools for Apache Cordova no longer support Windows 8.1. Existing projects should be updated to target Windows 10. This release adds support for iOS 10 and Xcode 8 on the iOS development side. Finally, the minimum supported version for Cordova is 6.0.0. At the time of writing, Visual Studio 2017 ships with Cordova 6.1.3. Existing projects built with previous versions will need a one-time upgrade when the solution is opened.

In-browser simulation with Cordova Simulate

If you've already used Cordova, you might know it offers a number of plugins that can be thought of as libraries that provide shared APIs to access device capabilities such as GPS, sensors, battery status, and so on. Visual Studio 2017 introduces a new feature called **Cordova Simulate** that allows you to simulate and control the behavior of plugins on Android apps by simulating the Android environment inside the Chrome browser without installing emulators or using physical devices. As a consequence, Visual Studio 2017 can now debug JavaScript code running in Chrome. When you press F5, Visual Studio 2017 launches an instance of Chrome, attaches the debugger, and shows a new tool window called **Cordova Simulate**. Figure 81 shows an example based on an app that uses the Geolocation plugin.

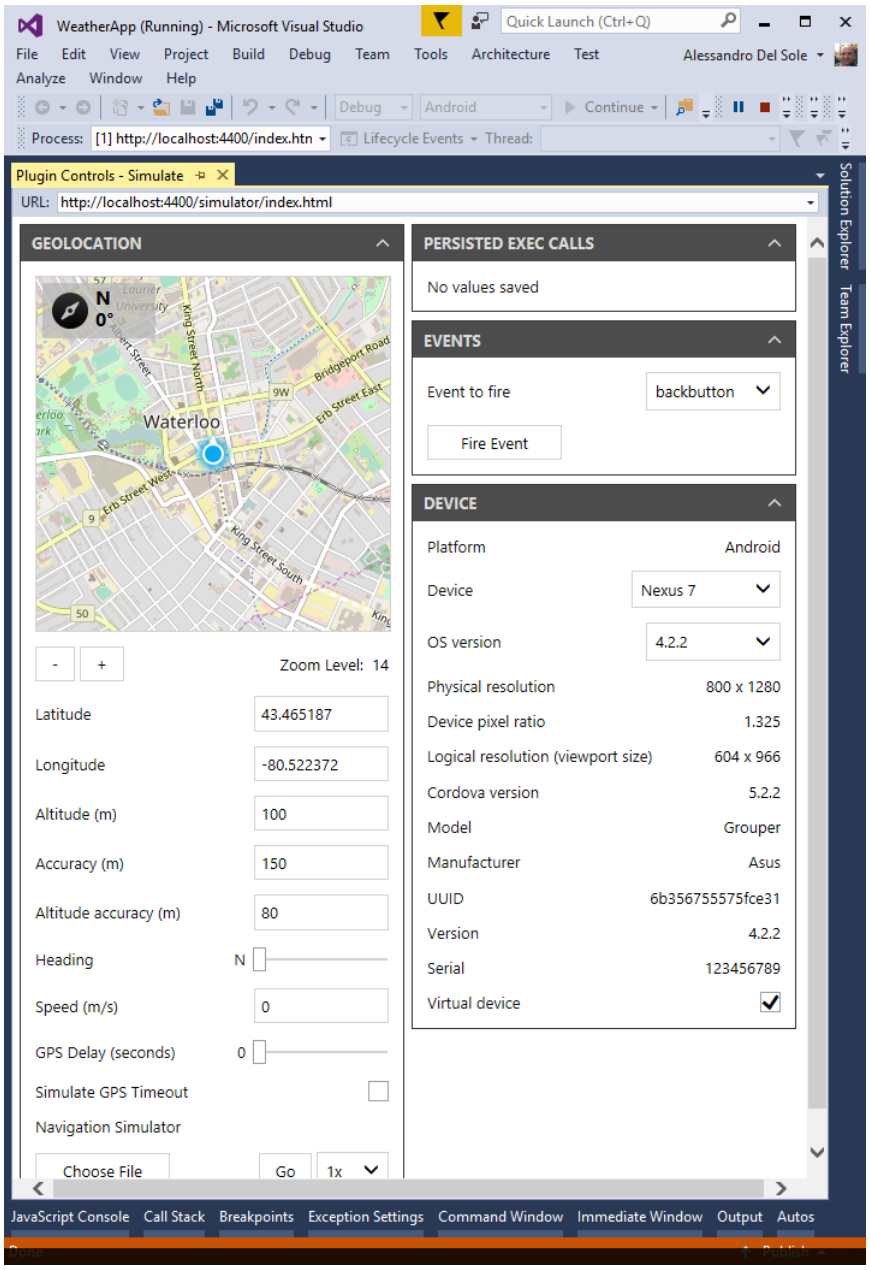

Figure 81: Controlling Plugins with Cordova Simulate

The Cordova Simulate window not only allows you to control the in-browser Android simulation (see the **Device** group), but it also allows you to manage any plugins your app is using with a convenient user interface. Additionally, you can simulate specific actions by using the **Fire Event** button in the **Events** group. For instance, you can simulate pressing the hardware Back button. In Figure 81, you can see how Cordova Simulate allows for controlling the Geolocation plugin by changing the position on the map, simulating navigation. This tool window works with many other plugins—for instance, you can control the battery status or simulate sensors such as the compass. For each plugin, Cordova Simulate will show a specific group of properties that you will be able to change to simulate different situations.

Message colorization

Cordova tools produce very verbose output messages, and it is often difficult to distinguish between important messages, errors, and compilation messages. Because the latter can often end up ignored, Visual Studio 2017 introduces message colorization for relevant messages that are now shown in blue. Figure 82 shows an example based on the build output of the WeatherApp sample.

Figure 82: Message Colorization in Tools for Apache Cordova

In-product acquisition of development tools

In Visual Studio 2017, selecting the Mobile development with JavaScript workload does not automatically include build tools for Android and Windows, which are now optional. This makes sense for two reasons: the product installation is faster, and now you have the choice of installing the tools you actually need according to how development continues. Though you can still select the build tools as individual components in the Visual Studio Installer, you can also take advantage of a new feature called **In-product acquisition**. In the drop-down near the **Start** button on the toolbar, you will find a new option labeled **Install build tools** referring to the current platform. Figure 83 shows an example based on Android.

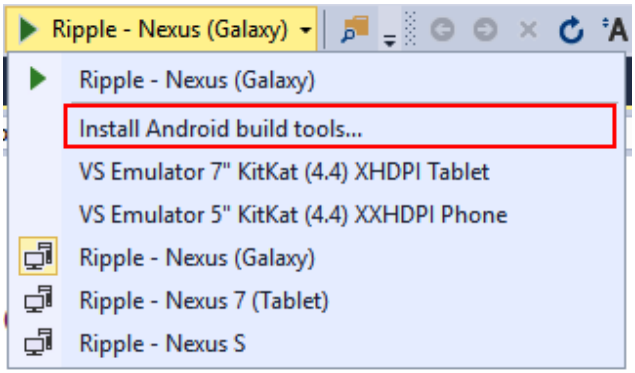

Figure 83: Installing the Required Tools at Build Time

This option allows you to install specific build tools only when and if you need them.

Tip: In-product acquisition is not specific to Cordova. In fact, the IDE can generally suggest SDKs and tools based on languages and platforms you are working with.

Cross-platform development with Xamarin

Note: Building apps with Xamarin requires selecting the Mobile development with C# and .NET workload in the Visual Studio Installer. Note that Xamarin development is not available for Visual Basic.

Xamarin is the name of both a company and a very popular development platform that allows building native iOS, Android, and Windows apps via writing and sharing C# code. Microsoft recently acquired Xamarin and is making significant investments in order to increase productivity with the Xamarin tools for Visual Studio. As with other platforms, my goal here is not to describe what you can do with Xamarin. Instead, you will find a thorough explanation of what's new with Xamarin tools in Visual Studio 2017. If you need a place to get started, you can visit the official developer portal, which includes all the resources you need for both Visual Studio and Xamarin Studio. The first important update is that Visual Studio 2017 supports Xamarin 4.3, which significantly improves the XAML editing experience in Xamarin.Forms and introduces the features detailed in the sections that follow.

Tip: Features described in this section are offered by version 4.3 of the Xamarin tools for Visual Studio, and they are not exclusive to Visual Studio 2017. This means they will also be available for previous versions (such as Visual Studio 2015 Update 3 and Visual Studio 2013 Update 2) if you upgrade the Xamarin tools. At the time of writing, Xamarin 4.3 is available as a preview release through the Alpha Updater Channel.

Automatic fix for missing Android dependencies

When you create a Xamarin project, some components required by the Android platform might not be detected. If this happens, the **Error List** window shows an error message describing which components are missing (typically JavaScript dependencies). In Visual Studio 2015, you were required to download these components and extract them manually into the proper locations. In Visual Studio 2017, the **Error List** window offers to download and install the missing components by double-clicking the error message. At this point, the IDE will download and install the required components in the appropriate way. Note that this might take a few minutes.

Updates to project templates

Xamarin 4.3 brings updated project templates to Visual Studio 2017. You can easily see this new feature when you select **File** > **New Project**. In the **New Project** dialog, select **Visual C#** > **Cross-Platform**. As you can see in Figure 84, the **New Project** dialog now shows only three templates.

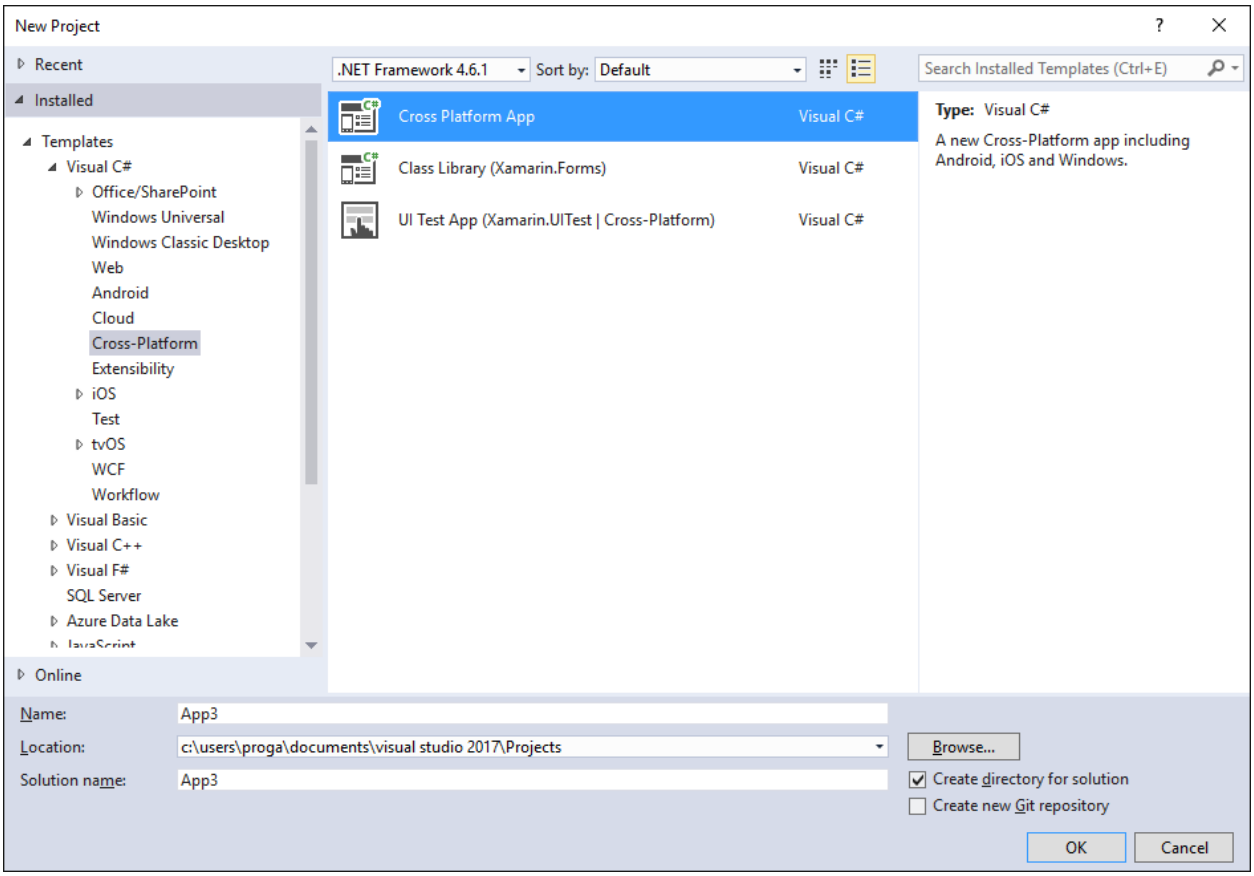

Figure 84: Updated Project Templates for Xamarin

The Class Library and UI Test App templates are taken from past versions, while the Cross Platform App template is a new addition. If you double-click this template, you will access the **New Cross Platform App** dialog (see Figure 85).

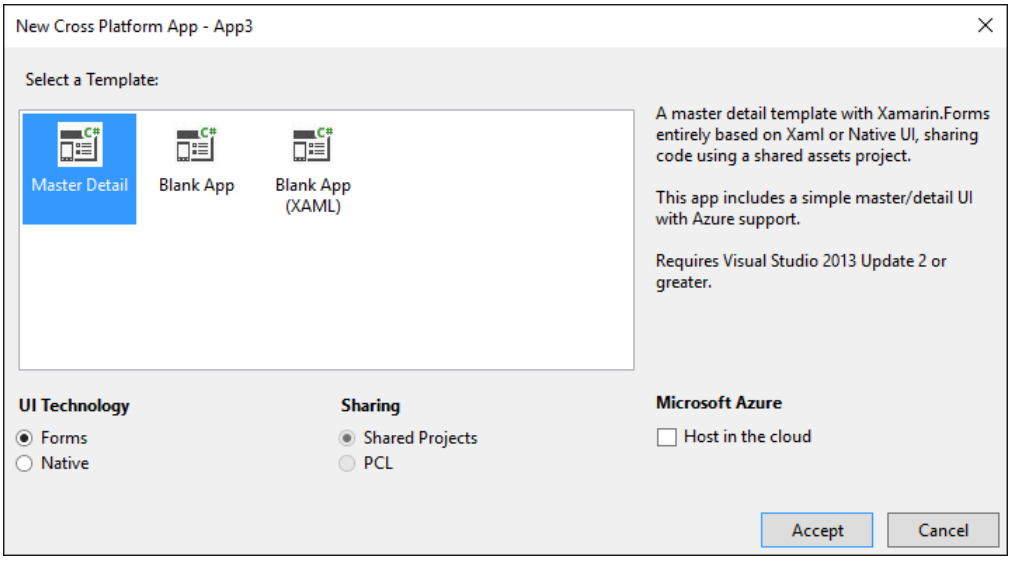

Figure 85: The New Dialog for Creating Xamarin Projects

Here you have three templates:

- Master Detail, which generates a master–detail user interface with either XAML (Xamarin.Forms) or native UI. It's based on shared projects and is ready to be hosted on Microsoft Azure.
- Blank App, which allows for generating a blank project based on either Xamarin.Forms or native APIs. You can choose between a Portable Class Library (PCL) and Shared Projects to share your C# code across platforms.
- Blank App (XAML). This is the project template you want to use for Xamarin.Forms development using XAML for the user interface (see Figure 86). You can choose between PCL and Shared Projects, but in most cases you will use the PCL option.

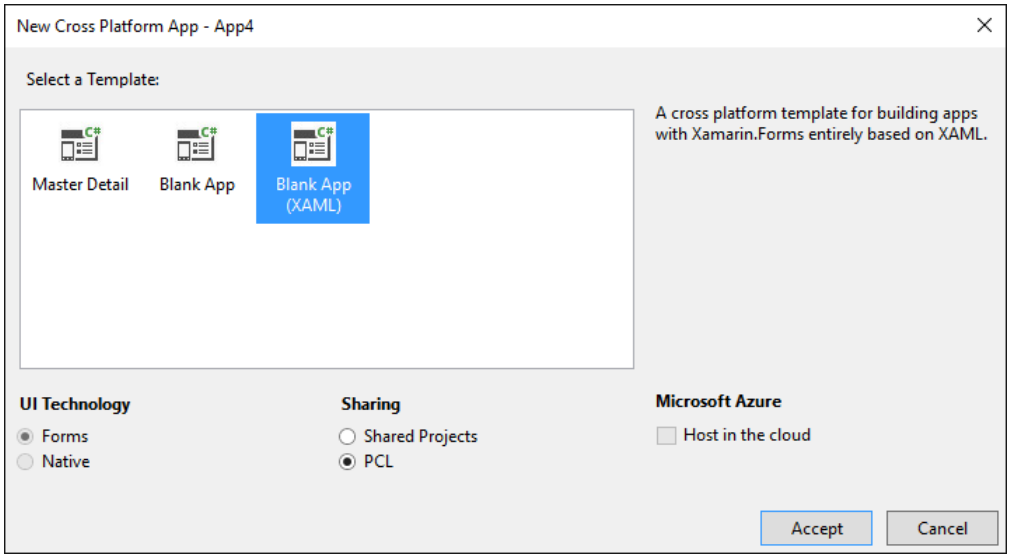

Figure 86: Creating a Project Based on Xamarin.Forms and XAML

Note: Microsoft has invested heavily in .NET Standard libraries, which share the same APIs across a number of platforms. It is no secret that Xamarin should provide fully integrated support for .NET Standard in the future, so keep in mind that the PCL support might be replaced by .NET Standard in the future. This blog post from Xamarin provides more information on this topic.

When you have made your choice, click **Accept** and wait for Visual Studio 2017 to generate your solution. In addition to cross-platform templates, Xamarin 4.3 introduces new project templates for native iOS development that you can find in the **iOS** and **tvOS** nodes of the **New Project** dialog, under **Visual C#**. There are new templates for the Apple Watch, Apple TV, and iOS extensions.

Unified .plist editor for iOS

Note: This section assumes you are somewhat familiar with iOS development with Xamarin.

In iOS development, you supply app metadata, assets, capabilities, and requirements with .plist files such as info.plist and entitlements.plist. In Visual Studio, most of these settings can be provided through the project **Properties** window, then the IDE will bundle the specified information into the proper .plist files. In addition to the **Properties** window, the Xamarin tools for Visual Studio have always offered built-in, specific editors for .plist files that allow fine tuning settings with a convenient user interface. Xamarin 4.3 integrates a new hierarchical, unified editor that provides better organization of all the information and allows for working with all .plist files in one place. Assuming you have either a Xamarin.Forms or a Xamarin.iOS project opened in Visual Studio 2017, double-click the **info.plist** file in **Solution Explorer** to call up the new editor. Figure 87 shows how it will appear.

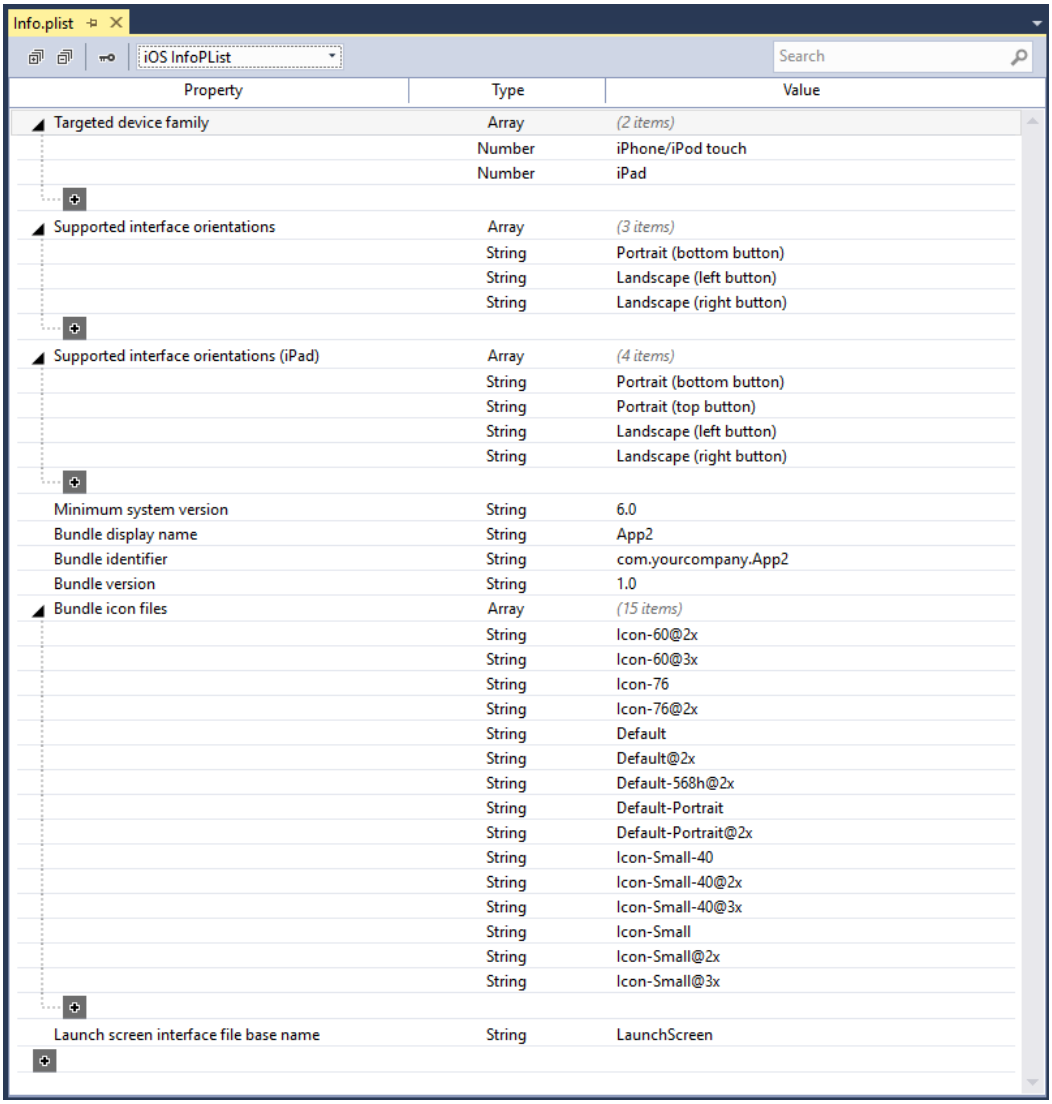

Figure 87: The New .plist Editor for iOS in Xamarin 4.3

As you can see, the new editor provides a unified view of your app settings and allows for making edits quickly. You can easily add custom settings by clicking the + icon at the bottom of each group. You can also filter the list using the drop-down at the top—the filter options are **iOS InfoPList** (full view), **iOS Entitlements** (only entitlements information), and **iOS Settings Bundle** (only information related to bundle signing). Notice that this editor works with any .plist file, so if you double-click **entitlements.plist** in **Solution Explorer**, you will get the same editor but the view will be restricted to entitlements. If you want to change the view, just change the selected item in the drop-down.

Chapter summary

Mobile development is crucial for Microsoft's strategies, and Visual Studio 2017 introduces important improvements to the development experience for any supported platform. For the Universal Windows Platform, you have a faster .NET Native compiler, updated NuGet packages, better design-time and diagnostic tools, and an updated Manifest Designer that automatically generates assets from a single image file. Regarding Tools for Apache Cordova (TACO), Visual Studio 2017 allows for simulating the Android environment inside the Chrome browser, which is useful for debugging apps that use plugins. The IDE also provides message colorization and in-product acquisition so that you can install the required tools only when and if you need them. For Xamarin, you get updated project templates and an enhanced XAML editing experience, a new editor for .plist files in iOS, plus many fixes in all platform-specific projects.

Chapter 9 Visual Studio 2017 for Cloud and Web Development

Cross-platform development is not only mobile development. Microsoft has been heavily investing in building tools that developers can leverage to write applications that run on Windows, Mac OS, Linux, and its most popular distributions. The result of such investments is .NET Core, the cross-platform framework that allows you to build applications that run on multiple systems using C#. Web apps you write on .NET Core need a robust infrastructure, and Microsoft Azure is the perfect companion on the cloud. When it comes to deploying .NET Core apps to Azure, many developers decide to adopt Docker containers. Visual Studio 2017 has integrated support for all of these technologies and, once again, proves to be the perfect environment to write applications that run on any platform and any device. This chapter provides a high-level overview of what's new in the tooling for cloud and web development in Visual Studio 2017, and you'll get links to the official documentation for further study.

Note: This chapter assumes you are familiar with basic Azure concepts and terminology. In fact, you will find mentions of resource groups, app service plans, Azure container registries, and other terms. If you are not familiar with Azure, or if something is not clear, the official documentation will help.

Building cross-platform apps with .NET Core 1.1

Note: This section requires installing the .NET Core cross-platform development workload.

The .NET Core is an open source, cross-platform, modular runtime that runs on Linux, Mac OS, and Windows. With .NET Core, you can write applications that run on multiple operating systems and platforms by using your existing C# skills. .NET Core ships with a command-line interface (CLI) and exposes a rich set of APIs that are shared across operating systems and, put succinctly, it allows you to deploy an application while including only the libraries the application effectively needs plus a component called Core CLR. The latter can be considered as a portable Common Language Runtime, and it allows an application to run. Discussing .NET Core in detail would require an entire book, so this chapter will focus on new tooling in Visual Studio 2017 that supports version 1.1 of the framework. Visual Studio 2017 provides a new node called **.NET Core** in the **New Project** dialog, under **Visual C#**, as shown in Figure 88.

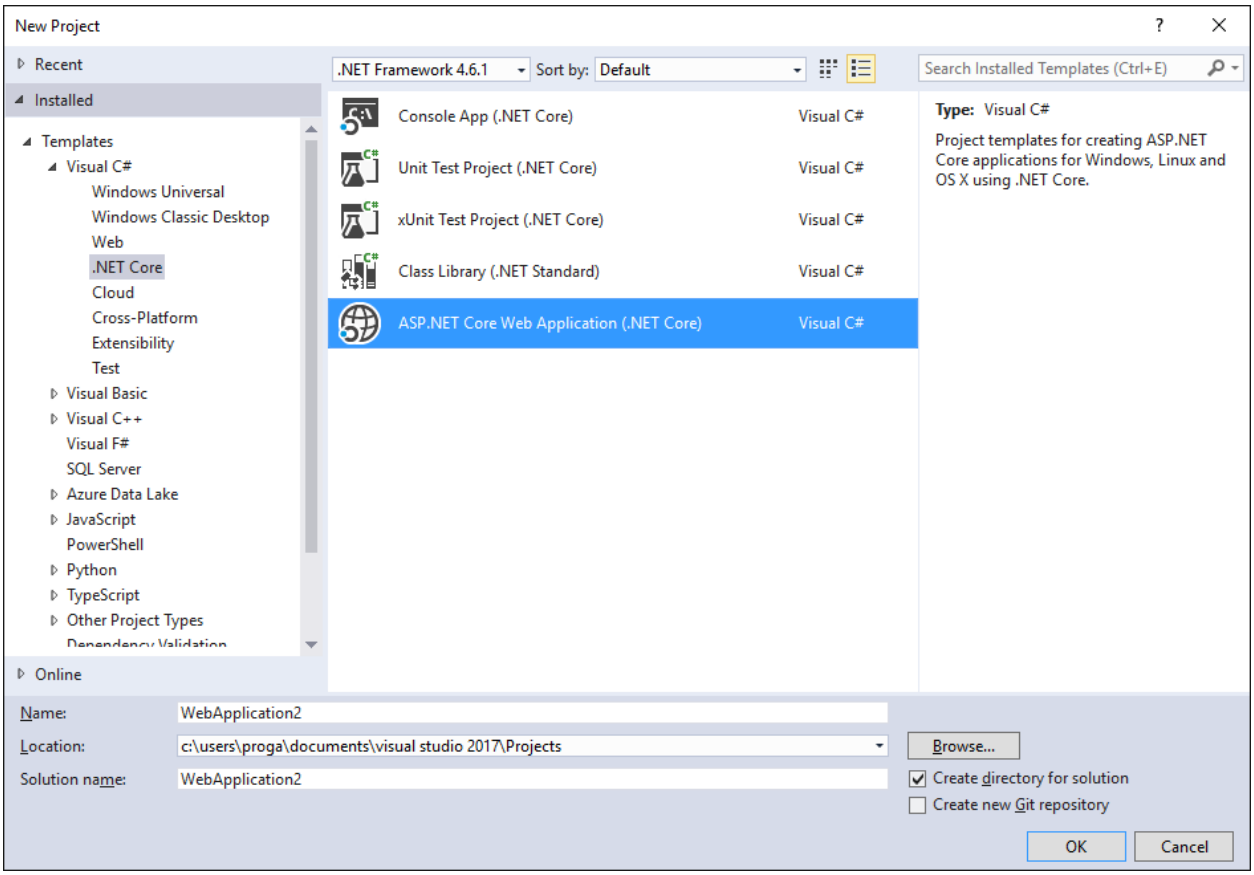

Figure 88: Available Project Templates for .NET Core

Two project templates (Unit Test Project and xUnit Test Project) are related to unit testing. With the Console App (.NET Core) template, you can write a console application that runs from the command line on Linux, Mac OS X, and Windows. With the ASP.NET Core Web Application (.NET Core) template, you can create a C# cross-platform web application based on the MVC pattern. The Class Library (.NET Standard) template allows you to create a library that is usable across all .NET runtimes, including .NET Core, Mono, and .NET Framework.

Tip: The .NET Standard Library specification, currently in version 1.6, is growing quickly, and it will be much more important in the next releases. If you author libraries, I strongly recommend that you consider .NET Standard libraries instead of Portable Class Libraries. Detailed information and explanations about .NET Standard can be found in the official documentation and in this blog post from the .NET Team at Microsoft.

For now, select the **ASP.NET Core Web Application** template, give a name to the project, then click **OK**. At this point, you will be able to specify which kind of application you want to create in the new **ASP.NET Core Web Application (.NET Core)** dialog, as represented in Figure 89.

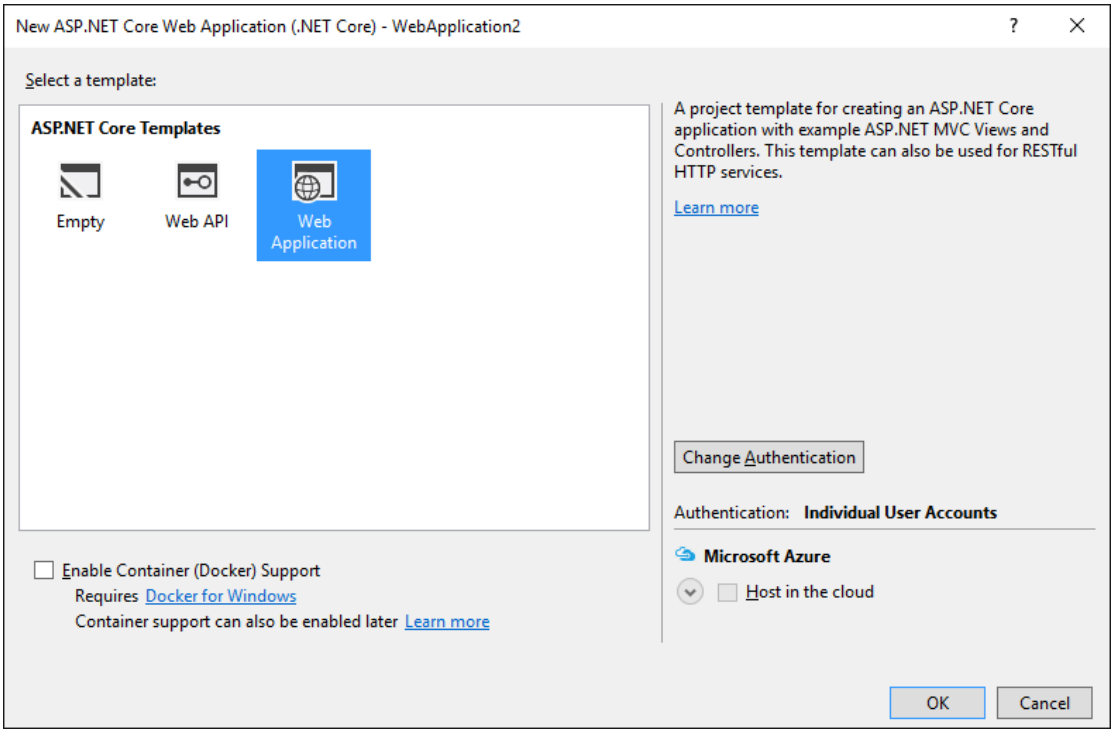

Figure 89: Creating a New ASP.NET Core Web Application

As you can see, you can decide to create an empty application, a cross-platform Web API service, or a cross-platform MVC web application. Select the **Web Application** for consistency with the next examples.

With both Web API and MVC web application, you also get a chance to supply which kind of authentication the application must offer. The default is no authentication, so click **Change Authentication**, then select **Individual User Accounts**. For now, disable the **Enable Container (Docker) Support** in the dialog, and, finally, click **OK**. Docker support will be discussed later in this chapter. As you would expect from an MVC application, in **Solution Explorer** you can see folders containing C# controllers, models, startup files, and services needed to manage registration and credentials (see Figure 90).

.NET Core applications can access SQL databases using a cross-platform version of the Entity Framework called Entity Framework Core. This is demonstrated by the availability of the Data\Migrations subfolder, which contains several of the so-called Code First Migrations.

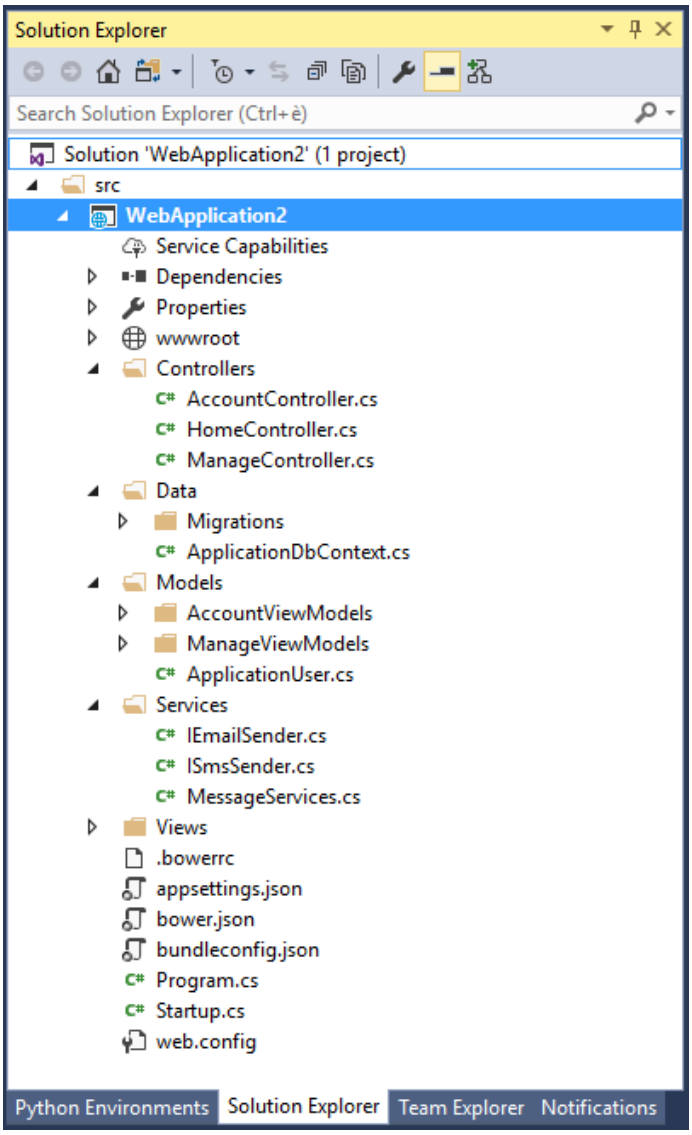

Figure 90: The Structure of an ASP.NET Core Web Application

Note: There is an important breaking change between .NET Core 1.0 and 1.1. With 1.1, Microsoft introduces MSBuild support for .NET Core, which means you now have a solution .sln file and a project .csproj file. In the previous version, there was no solution file and project information was inside project.json. Visual Studio 2017, will help you migrate older projects to the latest version, however.

Behind the scenes, Visual Studio 2017 simply invokes the .NET Core command-line interface and launches the following command that scaffolds a new ASP.NET Core web application:

```
> dotnet new -t web
```

where **-t** means the type of application.

You can then add your own data model, controllers, and views exactly as you would do in an ASP.NET MVC application built on the full .NET Framework. To get a better idea of what Visual Studio 2017 generated with .NET Core, simply press F5. After a few seconds, you will see the web application running in the browser, as shown in Figure 91.

All the powerful debugging tools in Visual Studio 2017 that you already know will be at your disposal. The application has a built-in authentication service, which means users can quickly register and log in to access information secured within controllers. Most importantly, the web application you have can run on Mac OS X, Linux, and Windows.

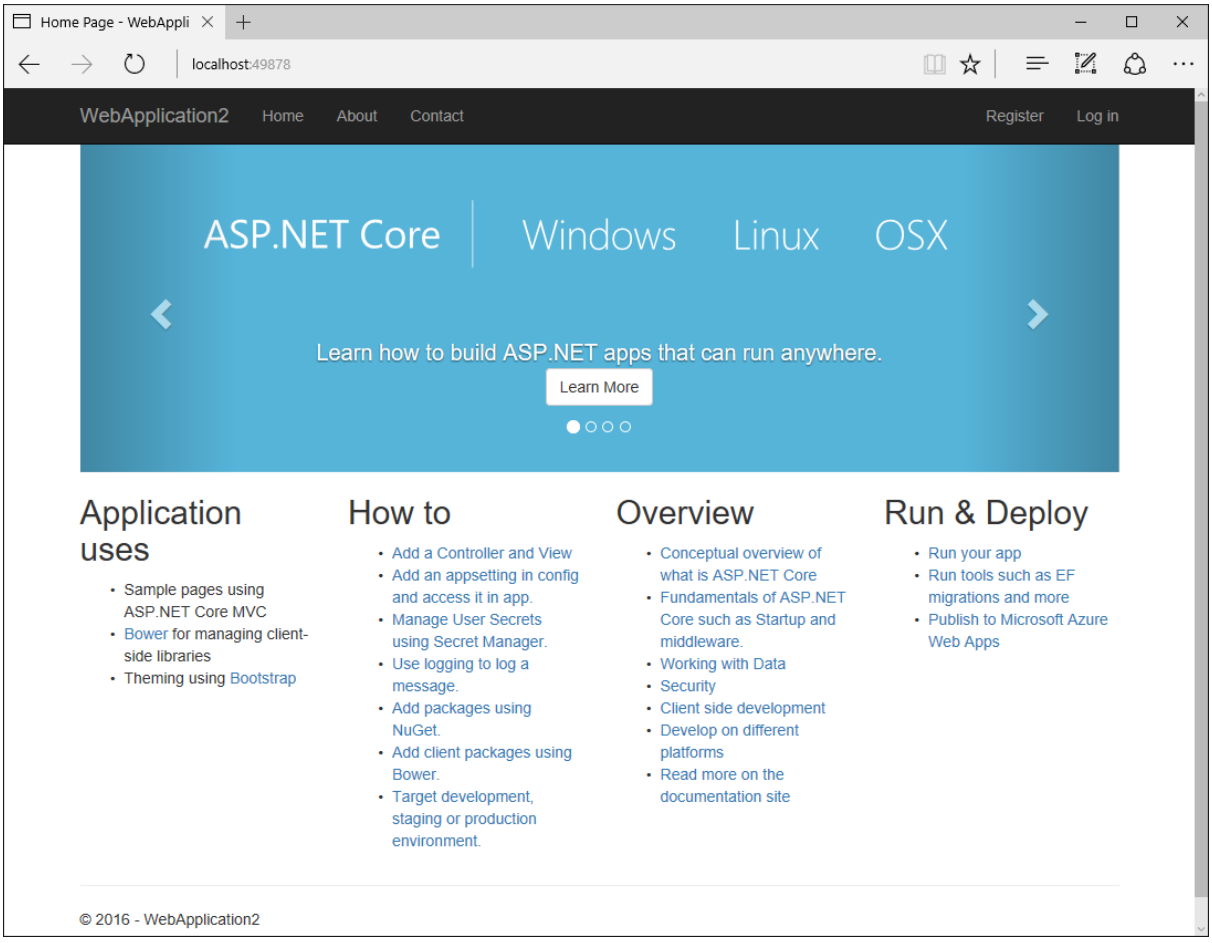

Figure 91: A Cross-Platform ASP.NET Core Web Application Running in the Browser

Do not close the solution, as it will be used in the next section with Docker. Here are a few more considerations about .NET Core 1.1 in Visual Studio 2017:

- NuGet package references are now included in the project .csproj file. This makes it easier to consolidate all packages in one file.
- The Web Publish tool has been moved from PowerShell to MSBuild.
- .NET Core 1.1 supports standard references, which means you can add references to non-.NET Core libraries. This makes it easier for .NET Core projects to interoperate with Xamarin projects.
- Visual Studio 2017 supports continuous delivery to Docker containers directly from within the IDE.

The following is a list of useful resources for further study:

- .NET Core home page.
- Tutorial: Getting started with ASP.NET Core MVC and Visual Studio.
- Entity Framework Core documentation.
- .NET Core command-line interface tools.

Visual Studio 2017 also allows you to quickly package an ASP.NET Core web application into a Docker container, as you will see in the next section.

Introducing tools for Docker containers

Note: This section requires installation of the .NET Core cross-platform development workload.

Containers can be considered units of deployment, and they allow you to package an application, its dependencies, and its environment configuration into an image that is finally deployed to a host operating system—typically Linux or Windows. The biggest benefit of containers is that they isolate applications on a shared operating system (Linux or Windows) and are lighter than virtual machines, because a virtual machine has a host operating system, one or more guest operating systems, and, therefore, a much more complex infrastructure.

Instead, containers use essentially the same, shared operating system, but they isolate applications from one another. In the world of containers, Docker is the most popular platform to package, deploy, and host containerized applications, and it is quickly becoming the standard. Docker can work on the cloud and on-premises, and it has been adopted by many vendors, including Microsoft and the Azure platform. Because of the growing importance and power of Docker, Visual Studio 2017 has added support for containers with the Docker tools for Visual Studio, which easily allow developers to package and deploy containerized applications. This section cannot explain Docker in complete detail, but it will demonstrate how to leverage the Docker integrated tools for development and debugging. Before looking at how Visual Studio 2017 natively supports Docker, it is first necessary to set up your development environment.

Tip: Microsoft has recently released a free e-book called Containerized Docker Application Lifecycle with Microsoft Tools and Platform. This is a very good starting point if you are not familiar with Docker and you want to learn how to use it with Microsoft tools.

Setting up the development environment

This chapter will explain how to package a .NET Core application into a Docker container, and how to publish the container to Microsoft Azure. Before continuing, you need to set up both Docker and Azure.

Note: Publishing a Docker container to Azure is an optional step, so you can skip setting up an Azure workspace and the section titled "Running a Docker container on Azure" if you are not interested in Docker on Azure. However, if you are interested, you need an active Azure subscription. If you do not have one, you can request a free trial at azure.microsoft.com/en-us/free.

Installing and setting up Docker

The first order of business is downloading and installing Docker for Windows for local debugging. When the installation is complete, you will see the Docker icon in the Windows tray bar. Right-click the icon, then select **Settings**. Docker needs you to specify a drive that will provide the shared operating system for containers, so click **Shared Drives** and select the **C** drive. Figure 92 demonstrates this.

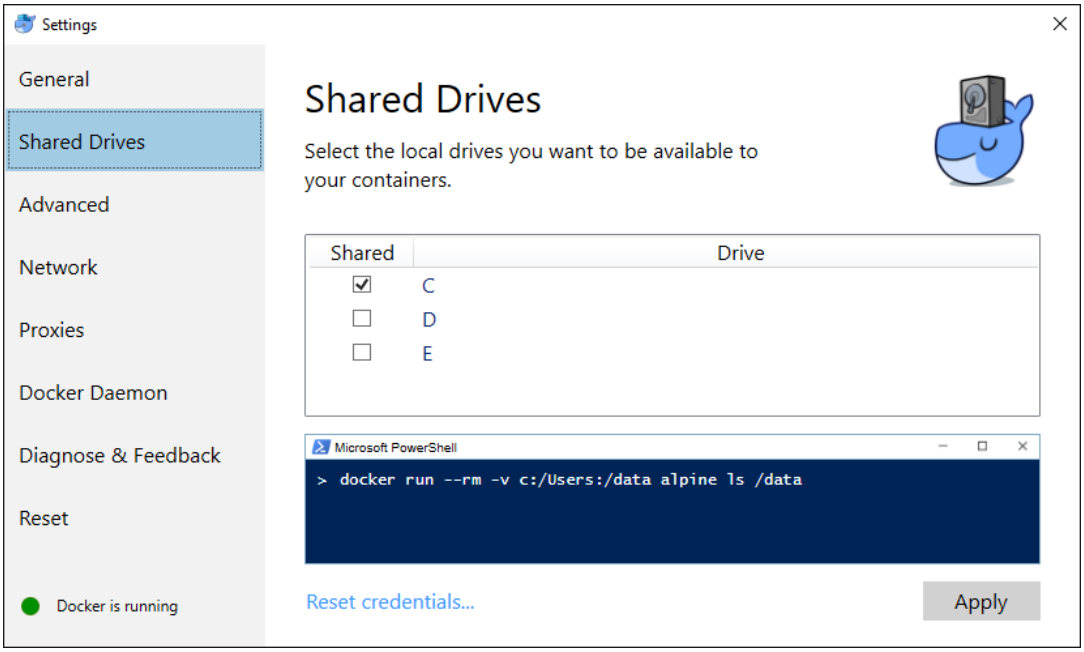

Figure 92: Configuring Shared Drives for Docker

When ready, click **Apply** and close the dialog.

Setting up Azure resources

As you will see shortly, Visual Studio 2017 allows you to quickly and easily publish a containerized application to a Linux system on Microsoft Azure. In order to accomplish this, the IDE needs you to supply the Azure subscription ID and some cloud service instances. You can see Figure 96 as a reference. More specifically, Visual Studio needs the following information:

- Web App Name: Represents the subdomain name in the application URL.
- Subscription: The target Azure subscription.
- Resource Group: A container that holds resources related to an Azure solution. This includes SQL servers, web applications, mobile services, and much more.

- App Service Plan: A collection of physical resources required to host an application.
- Container Registry: A new resource that allows for hosting and managing Docker containers.

The IDE allows you to select existing resources and create new ones. However, if you want to create new resources from Visual Studio, there is a problem—for new resource groups it does not allow for specifying the Azure region. Currently, Docker on Azure is a preview service and only the Western region of the U.S. supports hosting Docker containers on Linux. Also, publishing containers to Azure requires an app service plan and a container registry that must both be located in the Western U.S. and associated with a specific resource group. For these reasons, and until additional regions allow hosting Docker containers, I recommend that you set up the required resources in the Azure Portal rather than with Visual Studio. When you log into the Portal, enable services in the following order (hyperlinks point to the documentation):

1. Create a resource group located in the Western U.S. Once created, it will be visible in the list of resource groups. Click it to view its details.
2. While in the details of the new resource group, create a new App Service plan located in the Western U.S.
3. Go back to the resource group details and add a new Linux-based container registry located in the Western U.S. A storage account located in the same region and associated with the container registry will be created automatically.

You now have everything needed to deploy your application packaged into a Docker container hosted on Azure.

Note: Make sure you delete all your Azure resources when you no longer need them—for instance, at the end of your experiments and tests. This avoids the risk of unexpected charges on your credit card.

Enabling Docker on .NET projects

At the time of writing, the Docker tools for Visual Studio support .NET Core applications (including console applications) and classic ASP.NET applications. To enable Docker support in a project, you have three options:

- In the case of new ASP.NET Core web applications, you can select the **Enable Container (Docker) Support** flag in the **New ASP.NET Core Web Application** dialog, as you can see in Figure 89.
- Select **Project** > **Docker Solution Support** (or **Docker Project Support** if you only need to enable Docker at the project level).
- Right-click the project name in **Solution Explorer** and select **Add** > **Docker Solution Support** (or **Docker Project Support**).

For instance, you can easily enable Docker support for the ASP.NET Core web application created in the previous section with **Project** > **Docker Project Support**. This will add a few files that Docker needs to configure a new container (and that you don't really need to change):

- Dockerfile: This specifies the image and the application output. For ASP.NET Core, the image is called microsoft/aspnetcore and already includes all the native images of the necessary NuGet packages.
- docker-compose.yml: This file defines the collection of images to be built and run within the container.
- docker-compose.override.yml: This file defines the environment for the application (in this case **`ASPNETCORE_ENVIRONMENT`**) and the port (80) that is exposed and mapped to a dynamically assigned port for the development web server (localhost). The port number is determined by the Docker host and can be queried using Docker scripts.
- docker-compose.dev.debug.yml: This file contains additional settings when the build configuration is set to **Debug**. This is used by the Visual Studio development tools.
- docker-compose.dev.release.yml: This file contains settings to optimize the production image for Release configuration.

In the standard toolbar, you will notice the selected host for debugging is **Docker**, and a new button labeled **Docker: Debug Solution** is present (see Figure 93).

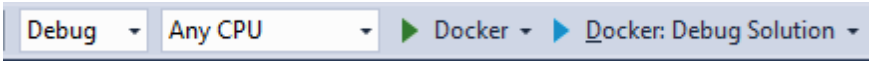

Figure 93: Buttons to Start an Application in a Docker Container

If you click this new button, Visual Studio 2017 will first package the application into a Docker container, invoke Docker to host the container, and then launch the application, offering the usual, powerful debugging support. Behind the scenes, Visual Studio 2017 invokes the Docker command-line interface to produce containers so that, while packaging, the **Output** window shows messages coming from command-line tools. When you switch to the Release configuration and build the solution, Visual Studio and the Docker tools generate an optimized image that is ready for production and for publishing.

Running a Docker container on Azure

Microsoft Azure can host Docker containers on Linux, and Visual Studio 2017 makes it extremely simple to publish a container to your Azure workspace. To accomplish this, right-click the project's name in **Solution Explorer**, then select **Publish**. You will be asked to specify a publish profile (see Figure 94). Assuming no publish profiles have yet been created, click **Create**.

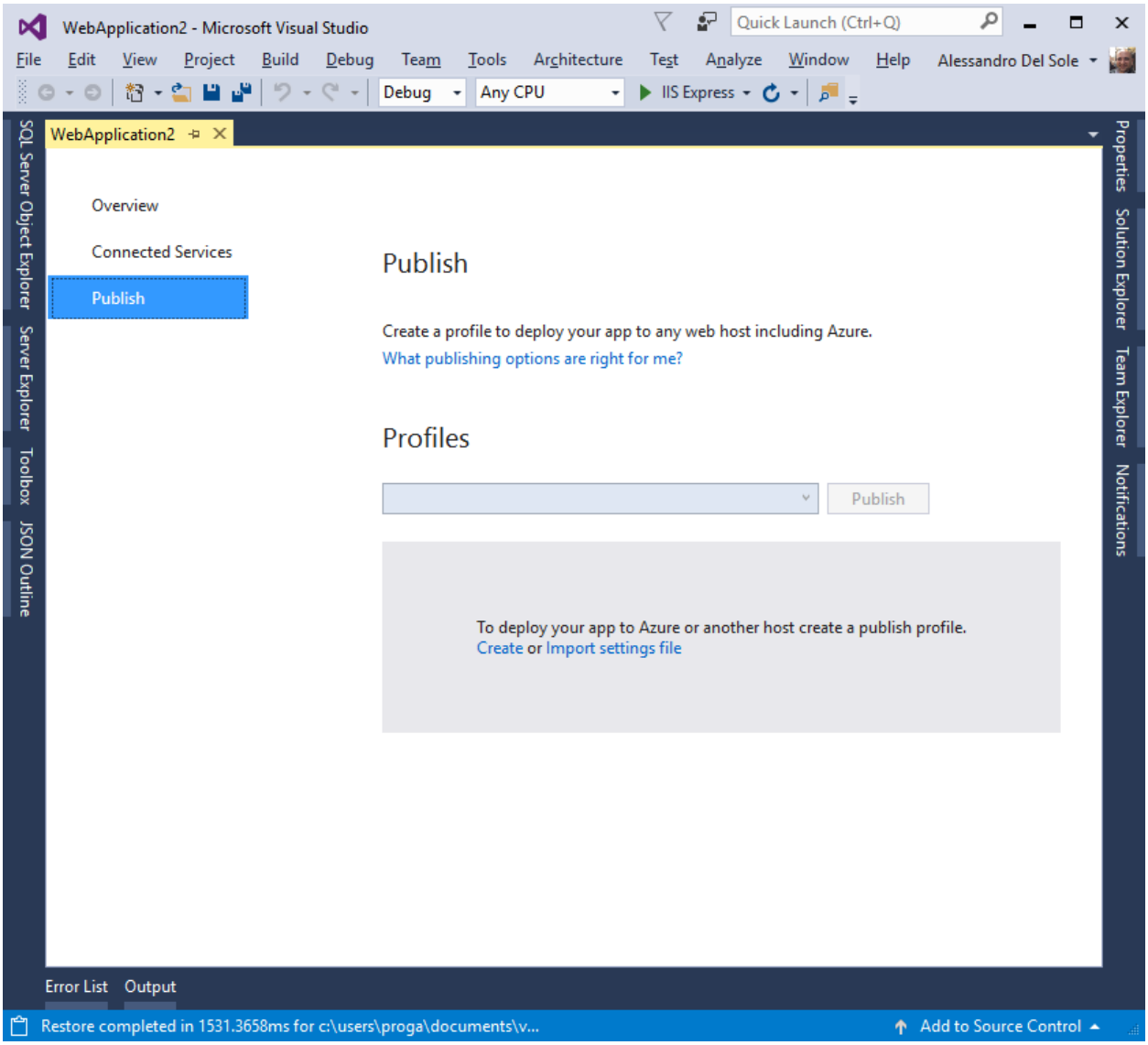

Figure 94: Specifying a Publish Profile

You will be asked to specify a publish target immediately. Select **Azure App Service Linux** (see Figure 95), then click **OK**.

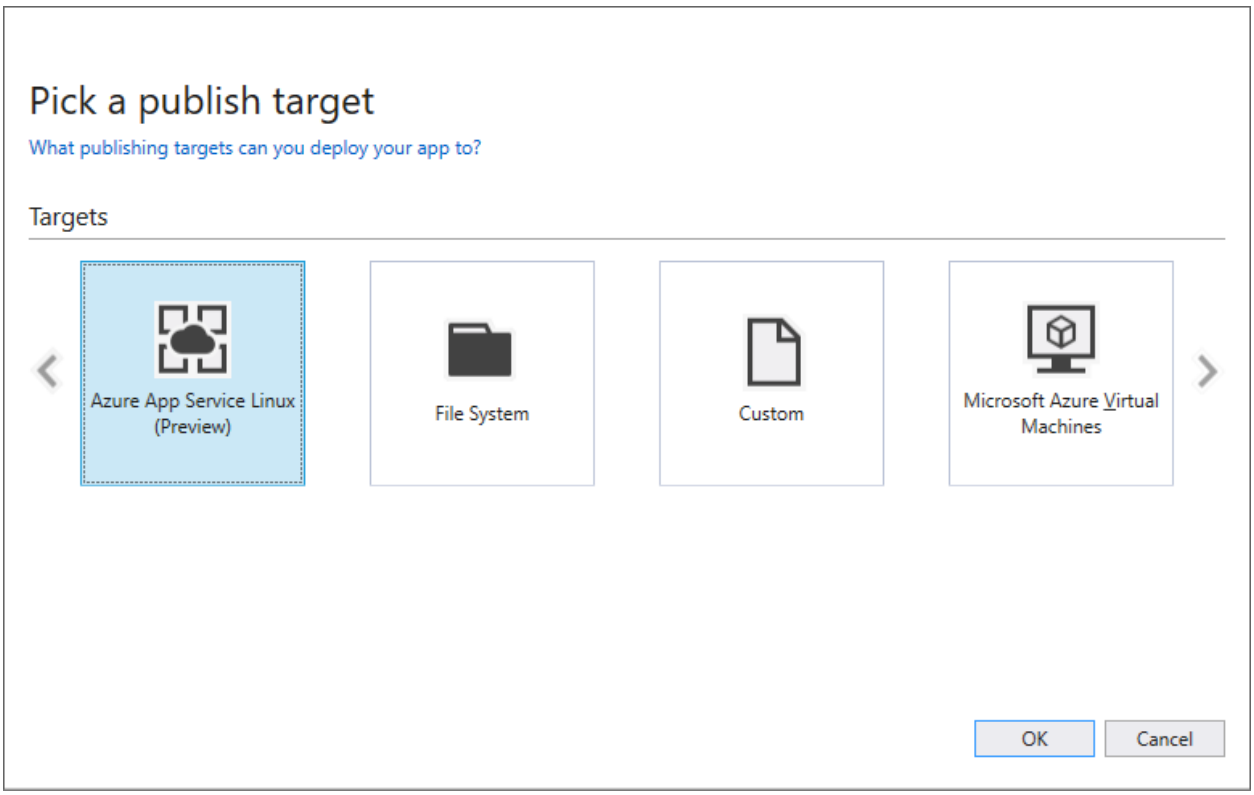

Figure 95: Selecting Linux on Azure as the Publish Target

At this point, the **Create App Service** dialog appears. Here you must supply some important information that Visual Studio 2017 needs in order to publish a Docker container to Azure, such as the web application name, the Azure subscription, a resource group, an App Service plan, and a container registry. Some fields will be automatically filled in, including the Azure subscription if Visual Studio detects one associated with the Microsoft account you used for login (see Figure 96).

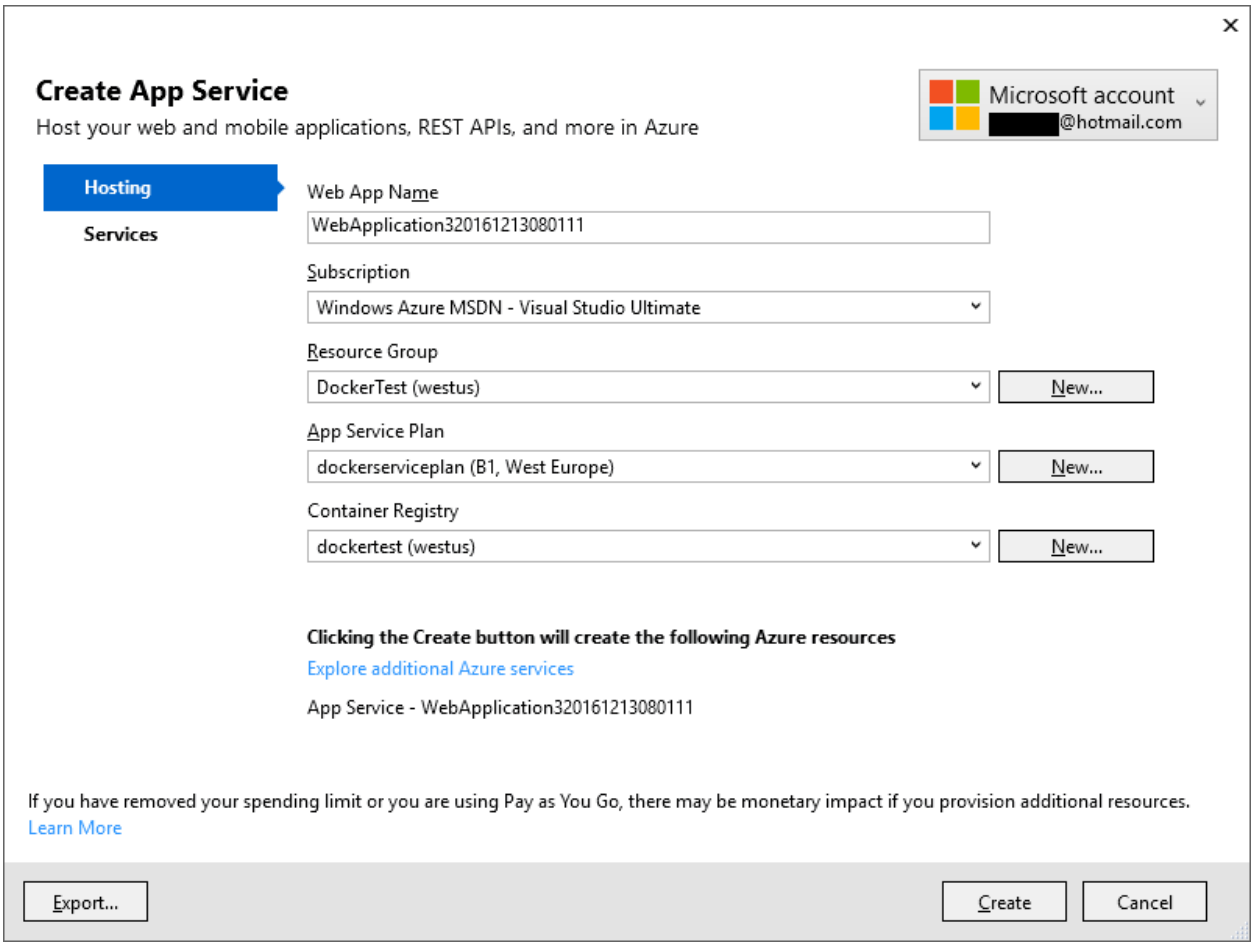

Figure 96: Providing Information Required to Publish a Docker Container to Azure

Select the resources you previously created in the Azure Portal, then click **Create**. At this point, Visual Studio 2017 will set up all the necessary services on Azure. When the setup completes, you will see a summary including the URL that you will use to launch the containerized application (see Figure 97). After reviewing the summary, simply click **Publish**. Visual Studio 2017 will package and deploy the Docker container with your application to your Azure subscription. The progress of publishing will be visible in the **Web Publish** tool window.

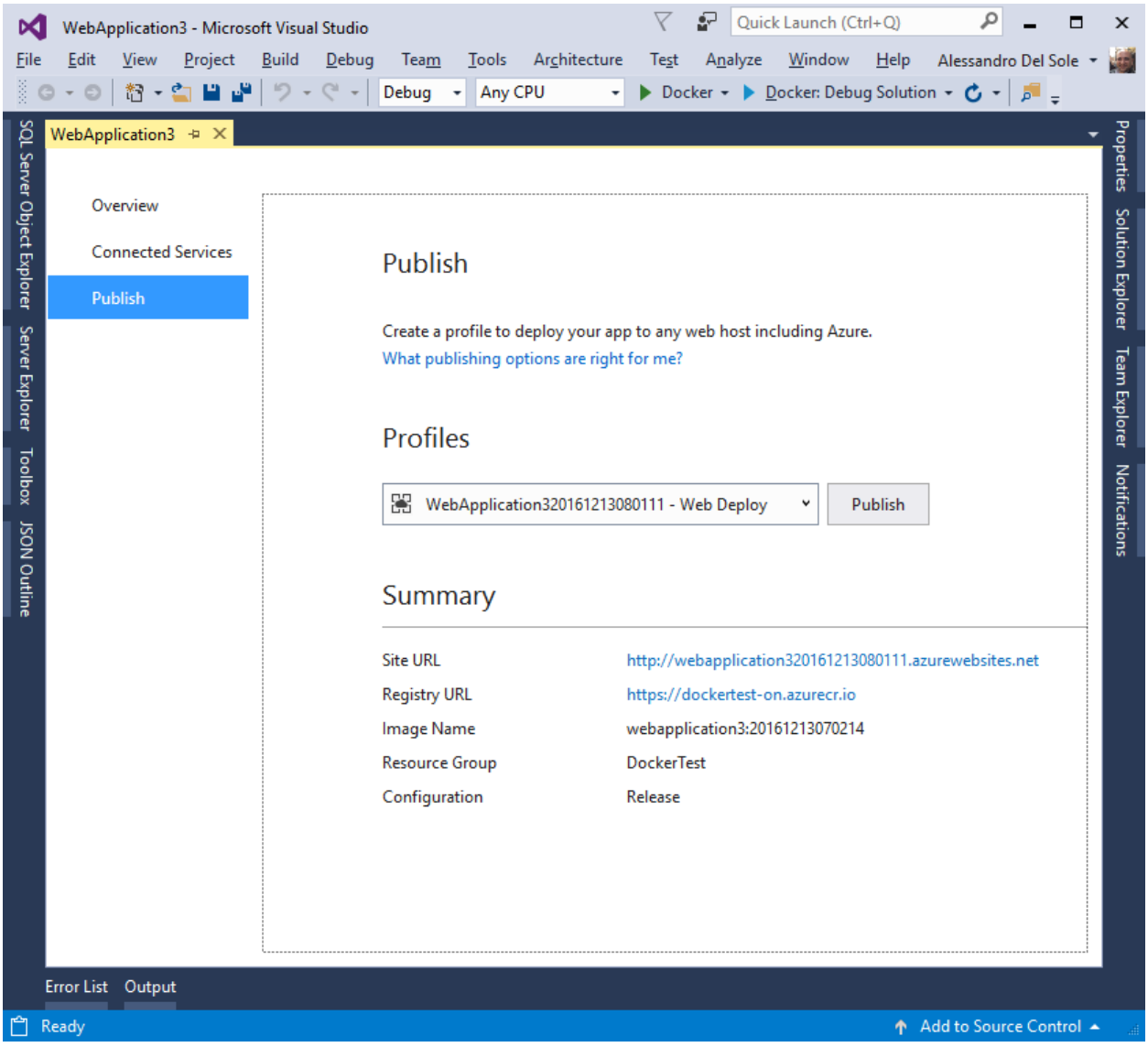

Figure 97: VS Summary Before Publishing the App to Docker

While publishing, Docker will open a console window that you must not close and that will shut down automatically. When Visual Studio 2017 ends publishing your web application, you will be able to launch it in your favorite browser using the URL you received in the summary. You'll do this exactly as you would with any other website or application.

Introducing Service Capabilities

For web and mobile apps, Visual Studio 2017 has introduced a feature called **Service Capabilities**. This is a new way of connecting to a service, and it supplements the older Add Connected Service and Add Service Reference mechanisms. In **Solution Explorer**, you will see Service Capabilities under a project name. Right-click it, then select **Add Service Capability** (you can still use the sample ASP.NET Core web application created previously). The number of available services varies depending on your configuration, but Visual Studio 2017 shows a new tool window called **Service Capabilities** where you can find a list of available services (see Figure 98).

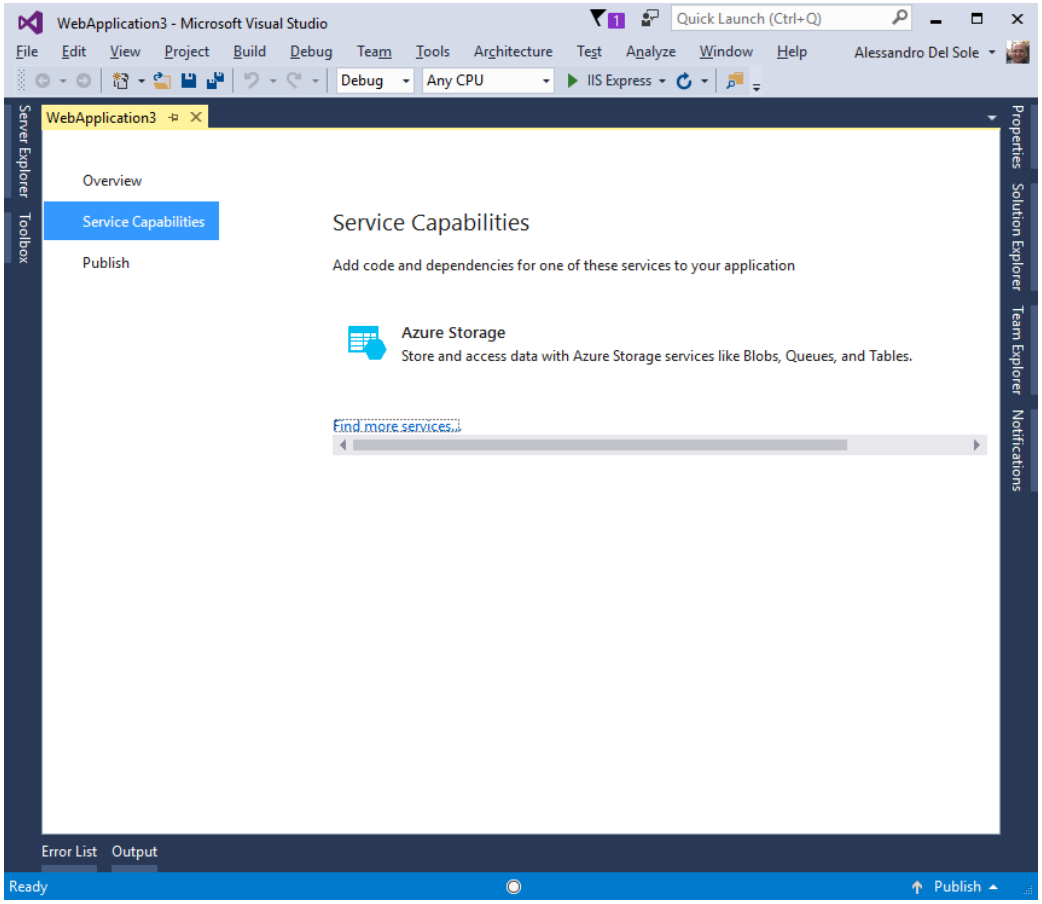

Figure 98: The List of Available Connected Services

If you click the **Find more services** hyperlink, Visual Studio shows the **Extensions and Updates** dialog—more specifically, it opens a new node called **Connected Services** and shows the list of additional connections you can download from either the Visual Studio Marketplace (formerly Visual Studio Gallery) or a vendor's website (see Figure 99).

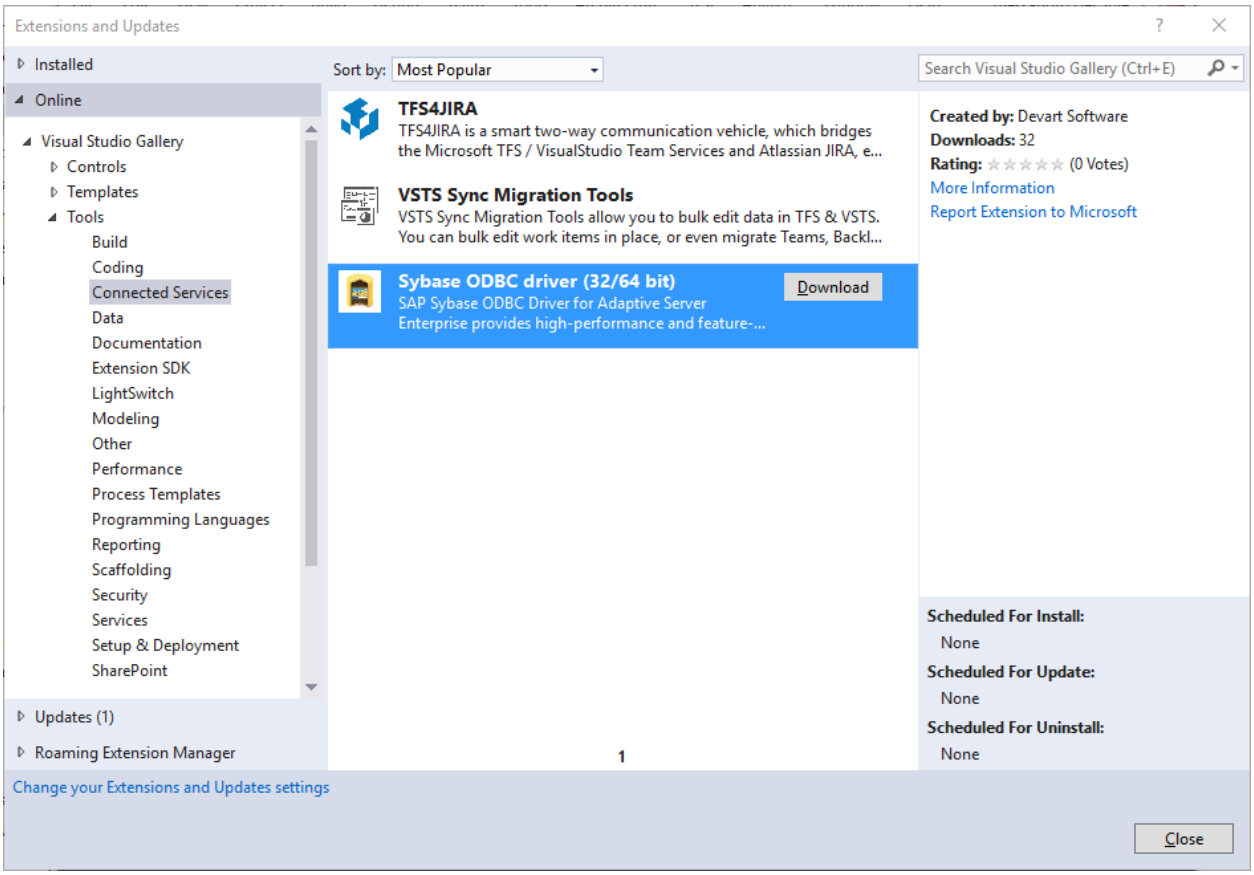

Figure 99: Additional Connected Services to Download as Extensions

You can download and install one or more additional services. Remember that the list shown in Figure 99 is only an example. Go back to the **Service Capabilities** window. In the next steps, I'll show how the connection wizard works against an Azure storage account. This will require you to have an active Azure subscription (the free trial is fine). If you do not have that (or do not want to set one up), you can skip to the next section. Generally speaking, with Service Capabilities, Visual Studio 2017 simplifies establishing a connection to a service, and it takes care of downloading the NuGet packages that are required to work against the selected service. If you click Azure Storage, a wizard appears and requires you to specify whether you want to connect to an existing storage account or if you want to create a new one (see Figure 100).

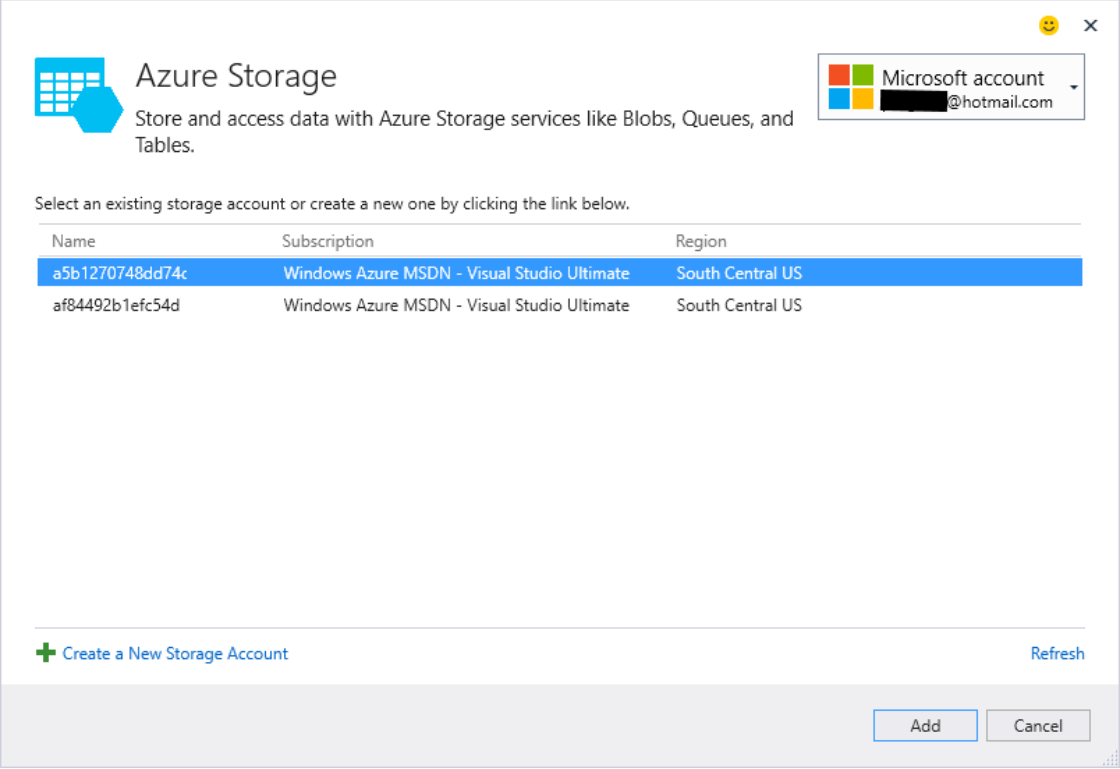

Figure 100: The Azure Storage Wizard

If you click **Create a New Storage Account**, the **Create Storage Account** dialog appears (see Figure 101). Here you will specify your Azure subscription, the storage account name, and required information such as the pricing tier, location, and resource group.

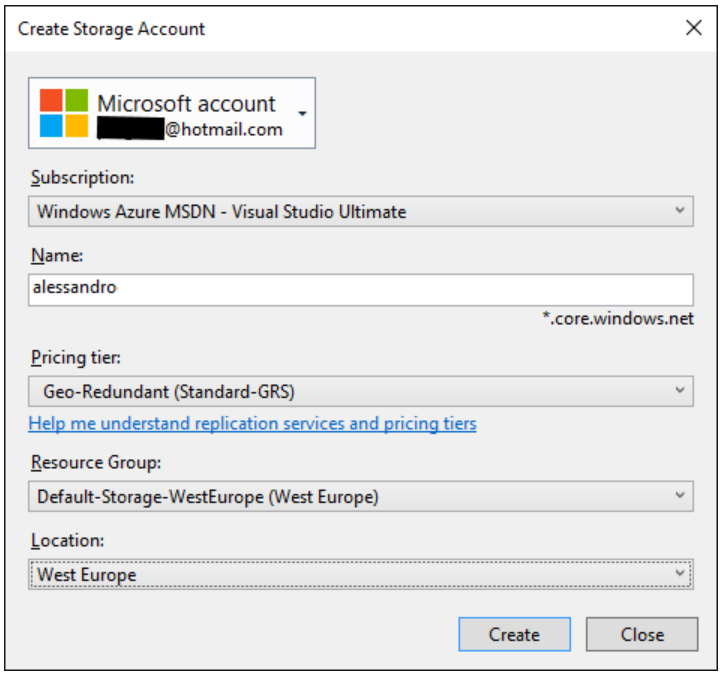

Figure 101: Creating a New Storage Account

When you click **Create**, or if you select **Add** in the **Azure Storage** dialog for an existing account, Visual Studio 2017 will connect to Azure, perform the necessary operations, and download and install the NuGet packages your application needs to interact with the selected service from C#. Figure 102 demonstrates this.

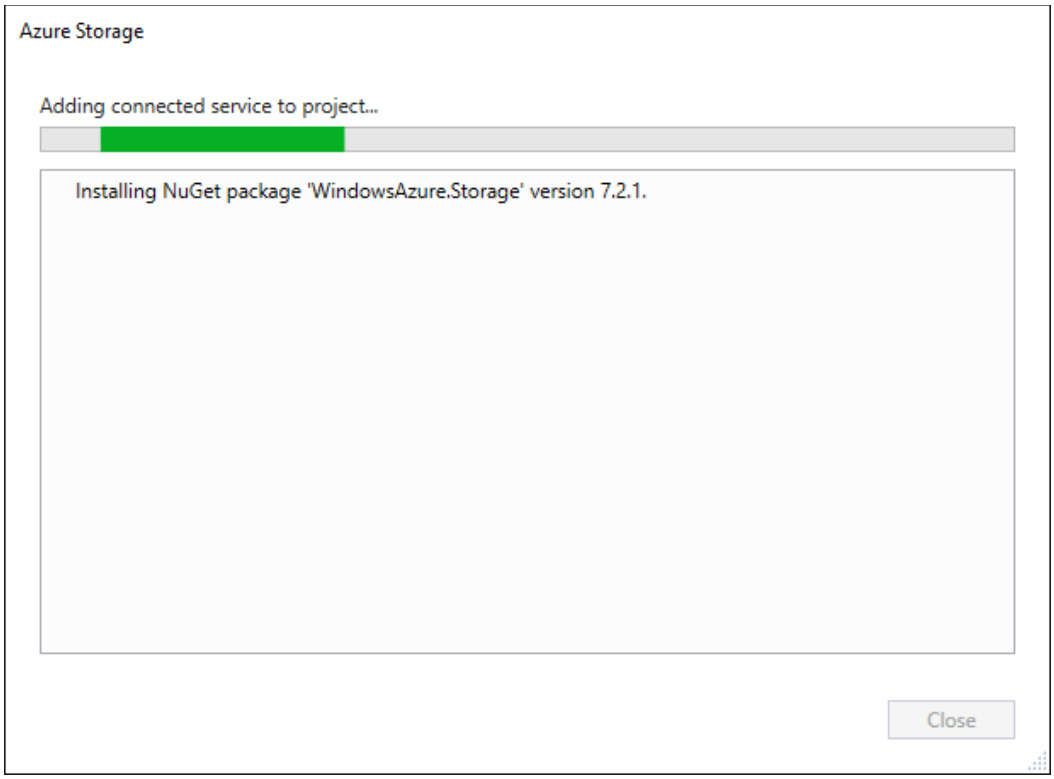

Figure 102: Downloading the Appropriate NuGet Packages

The referenced NuGet packages are also visible in **Solution Explorer**, as expected. Figure 103 shows how **Solution Explorer** displays the downloaded NuGet packages (in this case **WindowsAzure.Storage**) and a new folder, which has the name of your storage account plus the AzureStorage suffix. This folder contains a ConnectedService.json file, which holds dependency information, and a **Getting Started** information file that points to the appropriate documentation page for the selected service.

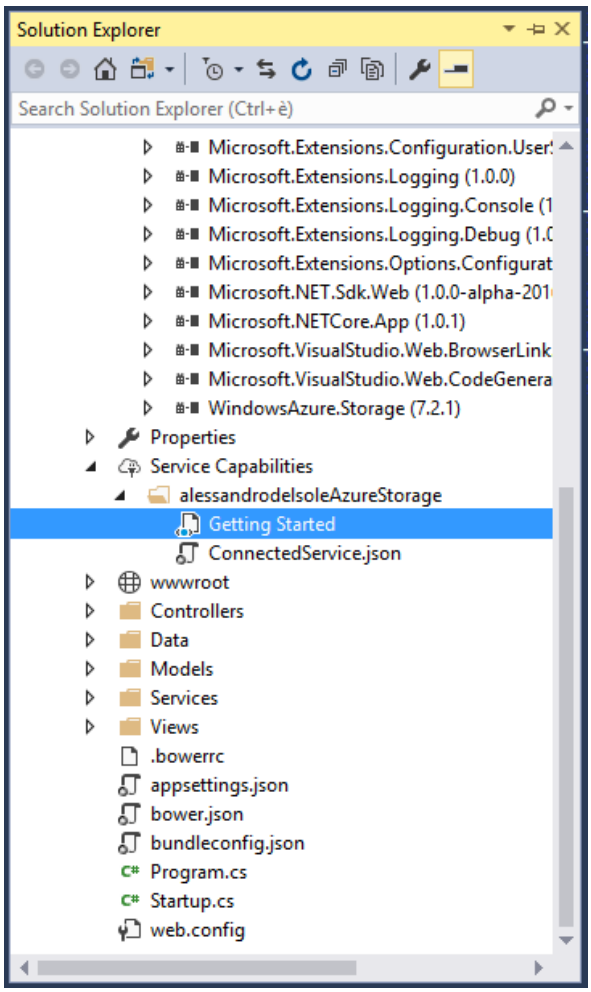

Figure 103: Solution Explorer Displays NuGet Package and a New Support Folder

In this particular case, if you double-click **Getting Started**, Visual Studio will open a page called Get started with Azure blob storage and Visual Studio Connected Services (ASP.NET). Here you will find code examples in C# that you can use to interact with the newly created storage account from within your applications. It is worth remembering that regardless of the connected service you choose, Visual Studio 2017 takes care of setting up a connection and downloading and installing the proper NuGet packages.

Building Node.js applications

Visual Studio 2017 has full support for building Node.js applications. Node.js is a very popular open source, cross-platform, and event-driven JavaScript runtime for developing a variety of applications. Visual Studio 2017 includes project templates you can use to quickly build a number of Node.js apps, including web apps, console apps, and Azure-enabled applications.

Note: Node.js support and integration is only available if you install the Node.js development workload.

Project templates for Node.js are available under the **JavaScript** node in the **New Project** dialog, as shown in Figure 104.

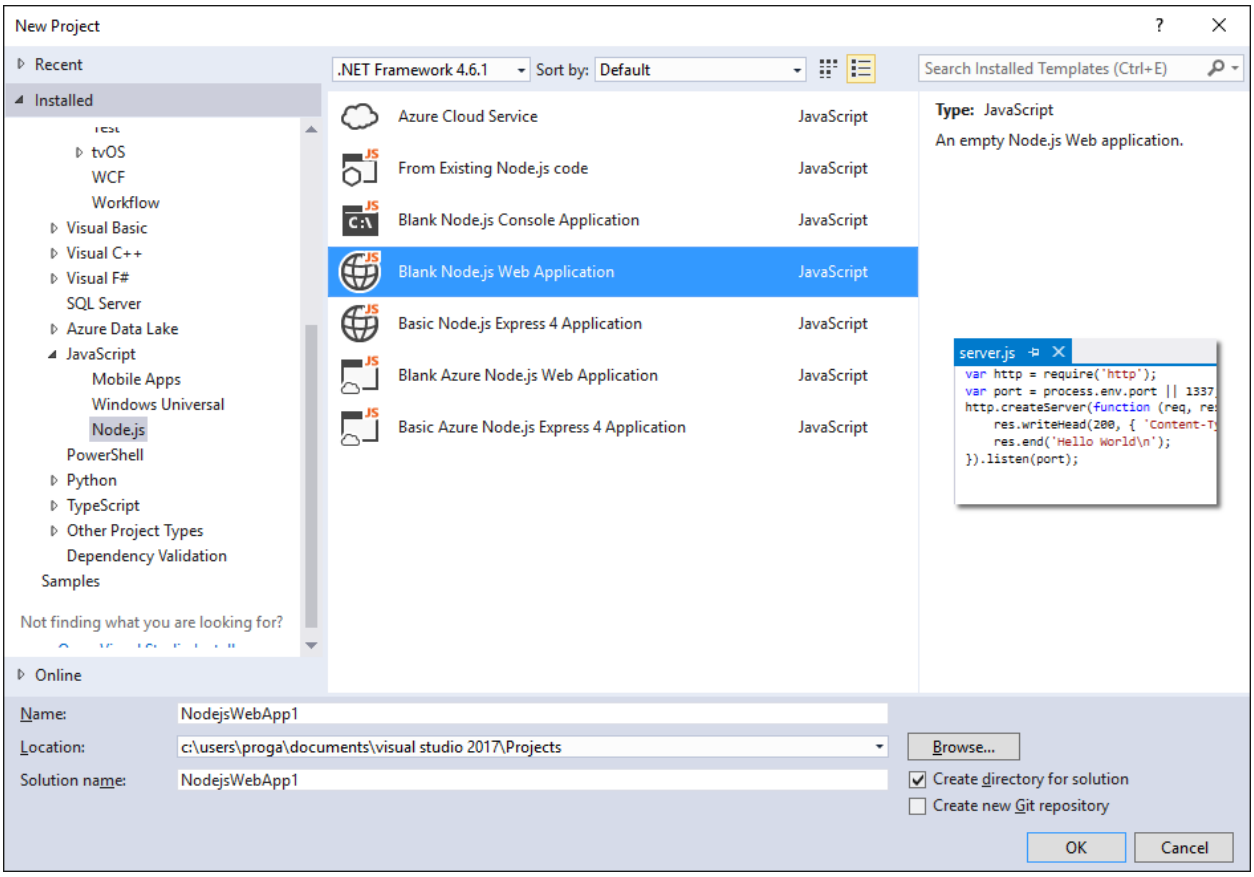

Figure 104: Project Templates for Node.js

The template names are fairly self-explanatory, and the **New Project** dialog provides a good description when you select one. Just a quick note on the template called **From Existing Node.js code**: this allows you to import existing code or assets from files on disk with different extensions. Figure 105 shows the import wizard with the list of supported files. You can include additional extensions in the text box at the bottom, and you can specify the startup item in the second page of the wizard. Finally, you import the specified folder.

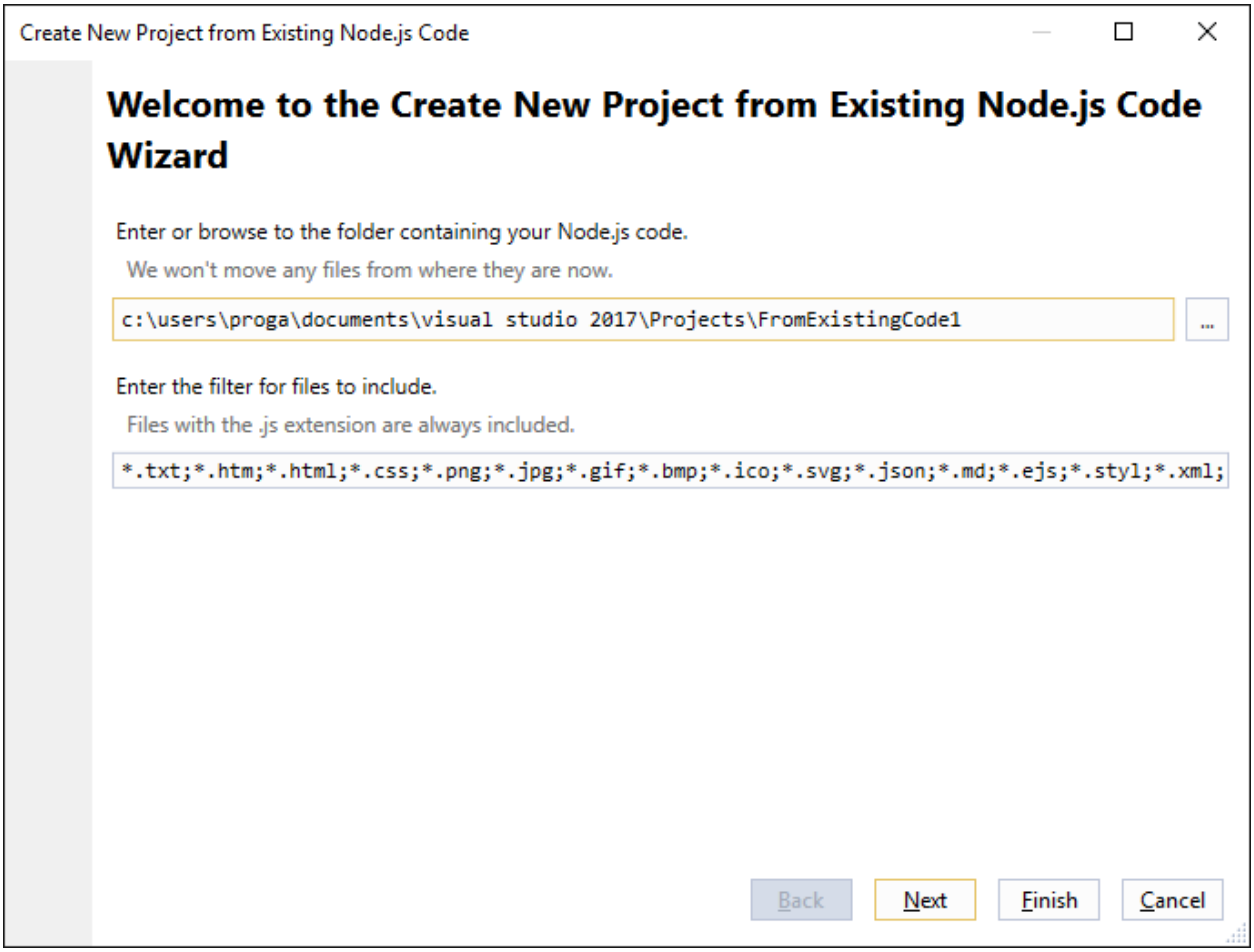

Figure 105: Dialog that Imports Existing Node.js Code

Note: Some project templates support Express 4. This is a Node.js framework for web applications that provides a set of features, utilities, and APIs for web and mobile apps. You can learn more about Express 4 at expressjs.com.

Whatever template you choose, Visual Studio 2017 generates a project that contains the following items visible in **Solution Explorer**:

- A JavaScript file (app.js for console apps or server.js for web apps) that contains minimal startup code.
- A package.json file that contains information about the application.
- An empty README.md file that you can use to write documentation using the Markdown markup language.
- The Node Package Manager (npm) node, which you can right-click to manage, download, and install npm packages for Node.js. The command you select is **Install New npm Packages**.

Tip: npm is the package manager for JavaScript applications. If you are new to JavaScript development, you can compare npm to NuGet in .NET development.

Figure 106 shows the **Install New npm Packages** dialog in action. Start typing package names in the search box and the dialog will list matching packages as you type.

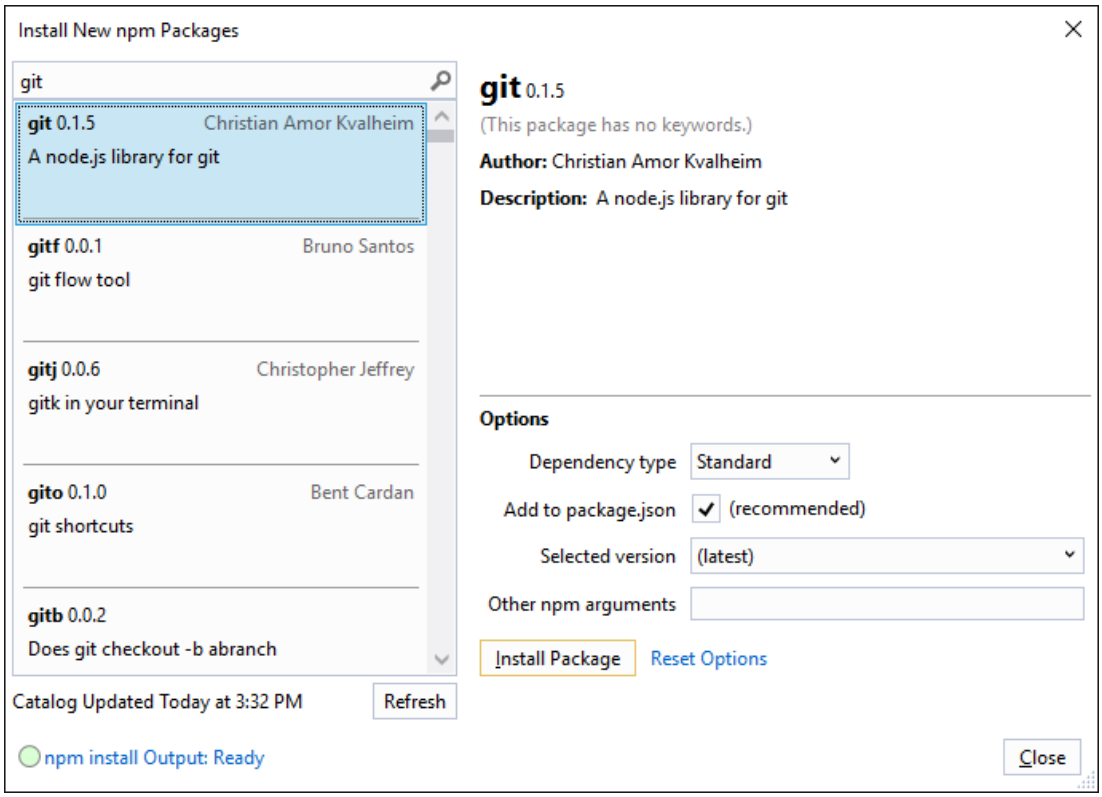

Figure 106: Managing npm Packages

With any template, you can write JavaScript code and take advantage of well-known and powerful integrated tools in Visual Studio such as:

- Full IntelliSense availability.
- Diagnostic and profiling tools.
- Unit testing.
- Source control based on Git.
- Integration with TypeScript.
- Debugging tools, including breakpoints and debugging windows.

As an example, Figure 107 shows the debugger in action over a blank web application in break mode after a breakpoint is hit.

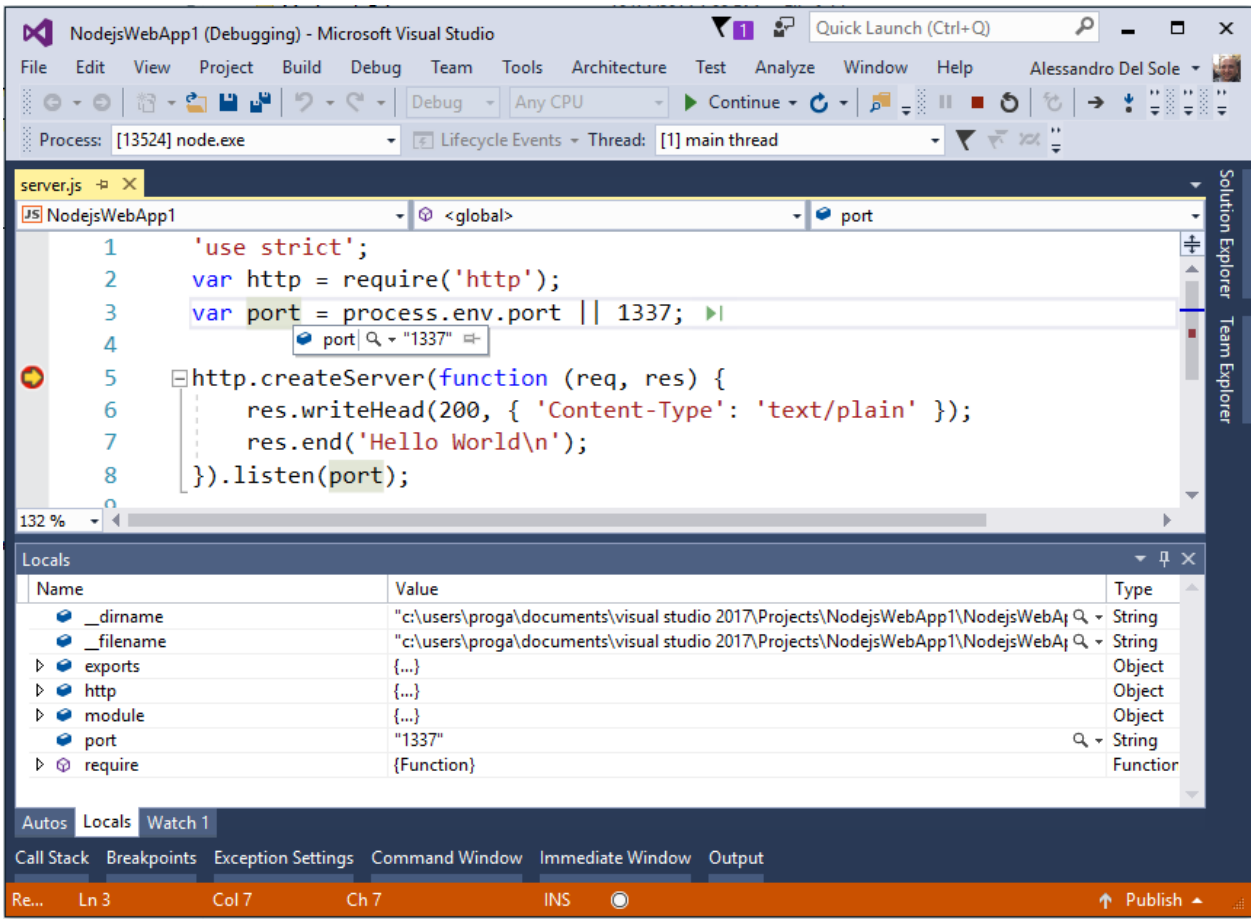

Figure 107: Full Debugging Support in Node.js Applications

As you can see, Node.js receives full debugging support with breakpoints, data tips, and debugging windows such as **Locals**. More about the Visual Studio 2017 tooling for Node.js can be found at www.visualstudio.com/vs/node-js.

Updated tools for Microsoft Azure

Azure is the cloud solution from Microsoft, and it's growing fast with new and updated services. When you select the Azure development workload, the Visual Studio Installer downloads and installs the latest Azure SDK for .NET, which extends Visual Studio 2017 with integrated tools, windows, and project templates that allow developers to work against the majority of the Azure services from within the IDE without opening the Azure portal. **Cloud Explorer** is one of the integrated tools that the SDK brings to Visual Studio 2017. At the time of writing, the Azure SDK for .NET provides the same functionalities to Visual Studio 2015, Visual Studio 2013 Update 4, and Visual Studio 2012 Update 2. Because the number of Azure services that Visual Studio integrates with has grown so much, and because the Azure tools are not specific to Visual Studio 2017, this chapter only summarizes the tools you have with the Azure SDK for .NET. You can later learn more about Azure services and tools in the official documentation. Notice that the list of services and tools may vary in future releases of the Azure SDK and of Visual Studio.

Here's a summary of what's new:

- Support for Azure Data Lake, the new cloud service for big data from Microsoft. This is offered through the Azure Data Lake Tools for Visual Studio. In Visual Studio 2017, these are available when you install the data storage and processing workload. More specifically, the tools allow you to manage your Data Lake resources from Cloud Explorer and offer a number of templates, such as for U-SQL, HDInsight, and Apache Storm projects.
- Support for Azure Service Fabric, a platform that allows you to build, debug, test, and deploy microservice-based applications. In addition to the Azure SDK, you need to install the Azure Service Fabric SDK, which includes integrated tools for Visual Studio. The SDK can be downloaded from the Azure Downloads page and allows you to manage a Service Fabric account from Cloud Explorer.
- Support for the Azure Resource Manager, which is available with the Azure Resource Group project template that you can use to create, configure, and deploy resource groups to Azure. With the Visual Studio tools, you can now manage resource groups within the IDE with the help of the **JSON Outline** tool window. Regarding the Azure Resource Manager tools for Visual Studio, you can follow an official tutorial online.

The previous points highlight, once again, the importance of Cloud Explorer as the integrated tool you use to work with your Azure resources. The next section describes updates to Cloud Explorer in more detail.

Cloud Explorer updates

Cloud Explorer is a tool window that interacts with one or more Azure subscriptions from within Visual Studio. With Cloud Explorer, you can view and manage a number of resources without leaving the IDE. For instance, you can create new storage accounts, blobs, tables, queues, and you can even manage SQL databases with the integration of the **SQL Server Object Explorer** tool window. Starting with the Azure SDK for .NET v. 2.9.6, Cloud Explorer gets important updates. First, you can group the view by resource types or resource groups via the drop-down under the **Microsoft Azure** label. In Figure 108, you can see how the view appears with the **Resource Groups** option selected, while Figure 109 shows how the view appears with the **Resource Types** option selected. Notice that the number and type of items may vary on your machine, depending on which kinds of cloud services you have enabled in your subscription.

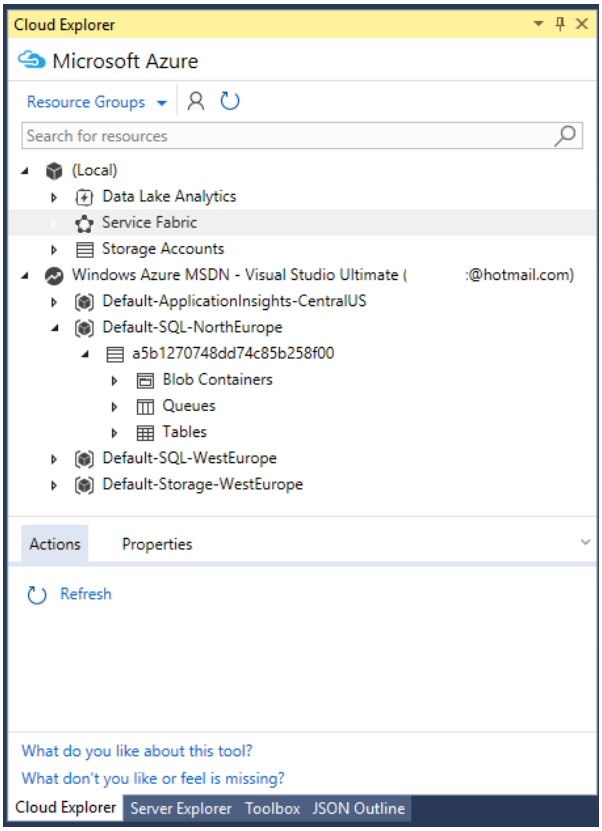

Figure 108: Resource Groups View in Cloud Explorer

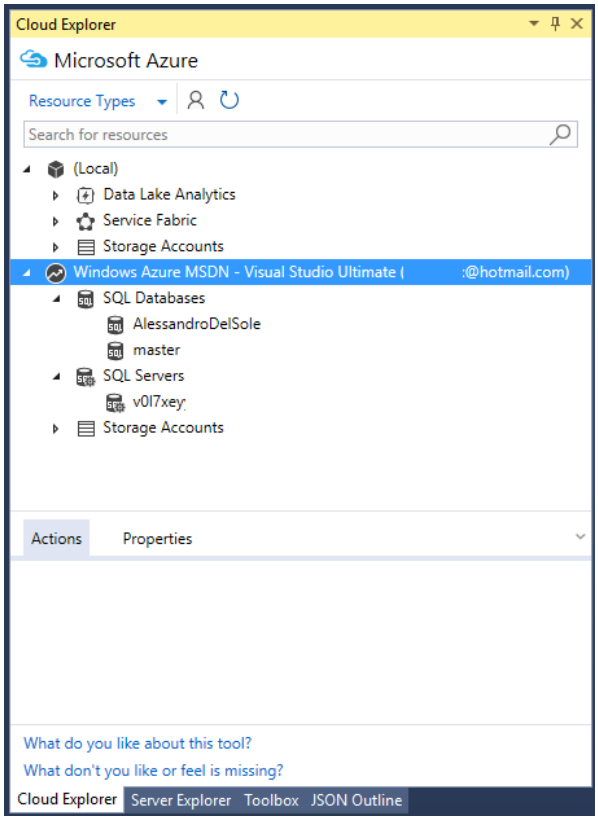

Figure 109: Resource Types View in Cloud Explorer

When you select a resource, the **Actions** tab at the bottom of the window will show a list of available actions against that resource. For instance, if you select a SQL database, available actions will open the database in the Azure portal, then open the database in Visual Studio via **SQL Server Object Explorer**, then refresh the view. Available actions vary depending on the selected resource. The **Properties** tab shows information about the selected resource (if available). If you have installed the data storage and processing workload and you have subscribed the Azure Data Lake service, you will see a node called **Data Lake Analytics**. This allows you to manage Data Lake resources easily, create databases, insert tables, and write scripts. Figure 110 provides a sample view of a database created within Cloud Explorer and a table currently in design mode with specific, integrated tools and editors.

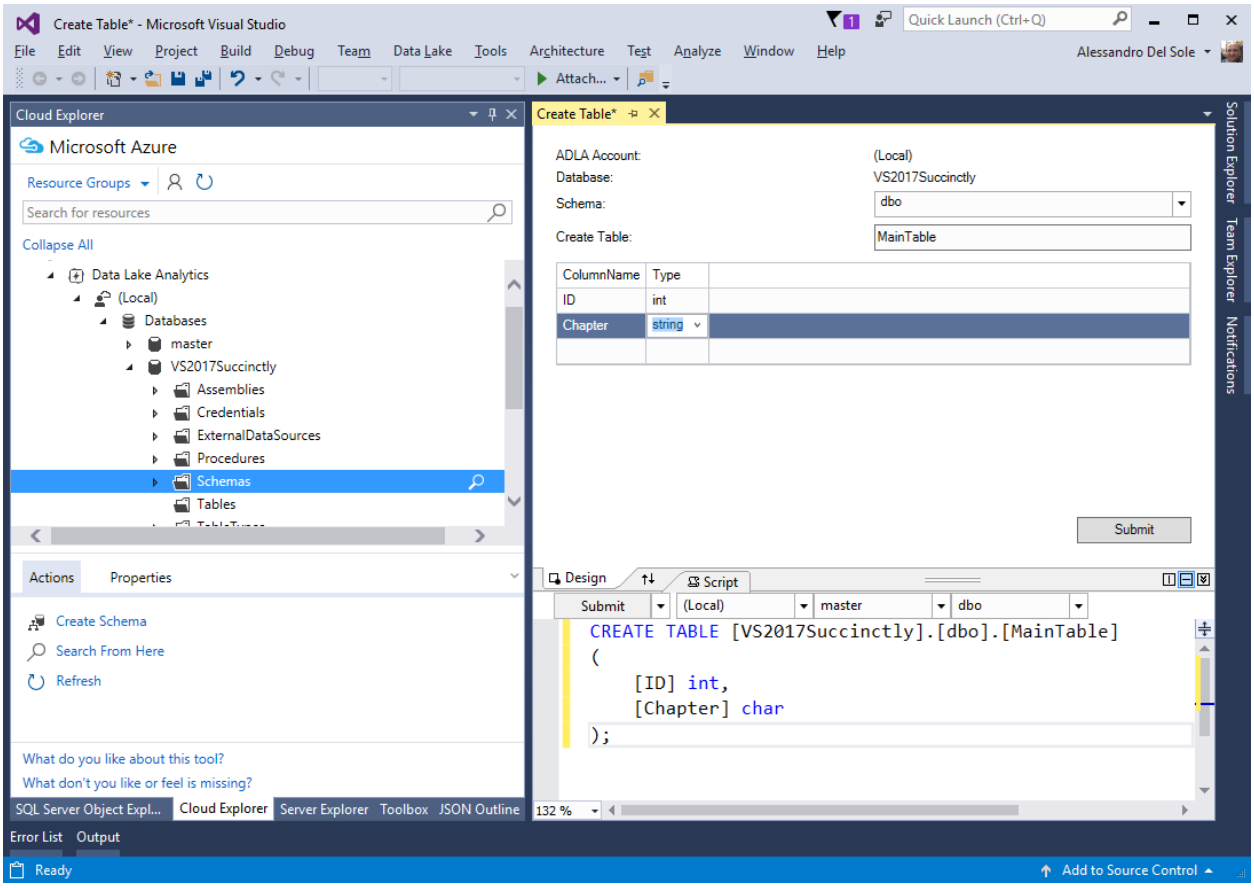

Figure 110: Managing Azure Data Lake Resources

Similarly, if you have installed the Azure Service Fabric SDK, you will be able to manage resources related to this service directly from within Cloud Explorer.

Chapter summary

Visual Studio 2017 introduces important tools for cross-platform development and the cloud—two things that often go hand in hand. Probably the most important addition is tooling for .NET Core, the modular, cross-platform, open source runtime that developers can use to build console and web apps for Linux, Mac, and Windows using C#. With the integrated tools, you can build .NET Core solutions the same way as with classic .NET development. Another fundamental addition is tools for Docker, a *de facto* standard in deploying applications to containers. Docker containers can be hosted on Linux in Azure, and Visual Studio 2017 does the entire job of packaging and deploying a container for you. In conjunction with its aim to be the development environment for any developer on any platform, Visual Studio 2017 has full support for Node.js, including advanced editing, debugging, and testing features. Finally, Visual Studio 2017 supports all the most recent Azure services, including Data Lake and Service Fabric, and it provides an option for interacting with more services from within the IDE through the **Cloud Explorer** tool window. This avoids the need for opening the Azure portal every time.