Visual Studio 2015

Succinctly

by Alessandro Del Sole

Syncfusion | Technology Resource Portal

Visual Studio 2015 Succinctly

By
Alessandro Del Sole

Foreword by Daniel Jebaraj

Technical Reviewer: Matt Duffield

Copy Editor: Benjamin Ball

Acquisitions Coordinator: Hillary Bowling, marketing coordinator, Syncfusion, Inc.

Proofreader: Morgan Cartier Weston, 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 considered a Visual Studio expert and a Visual Basic authority. Alessandro has authored many printed books and e-books on programming with Visual Studio, including *Visual Studio 2013 Succinctly*, *Visual Basic 2015 Unleashed*, *Visual Studio LightSwitch Unleashed*, and *Visual Basic 2010 Unleashed*. He has written tons of technical articles about .NET, Visual Studio, and other Microsoft technologies in Italian and English for many developer portals, including 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. You can follow him on Twitter at @progalex.

Introduction

Microsoft Visual Studio 2015 is the new version of the popular integrated development environment for building modern, high-quality applications for a number of platforms such as Windows, the Web, the cloud, and mobile devices. Continuing on what Microsoft made with Visual Studio 2013, the 2015 version still focuses on increasing developer productivity by improving the code editor, debugging tools, Azure integration with other services requiring a Microsoft account, and much more. Also, Visual Studio 2015 provides support to new strategies at Microsoft, such as Windows 10, open-sourcing .NET, and building cross-platform apps for mobile devices. Visual Studio 2015 ships with the following editions: Enterprise, Professional, and the free Community Edition. Most features described in this book require Visual Studio 2015 Professional, but some of them require Visual Studio 2015 Enterprise, which is the most complete edition available. I will specify where the Enterprise edition is required. For a full comparison of editions, and for a download of a fully-functional, 90-day trial of Visual Studio 2015 Enterprise and other editions, visit the Visual Studio Downloads page. This book contains a comprehensive description of new features in the Visual Studio 2015 IDE which will not only help you to write better code, but will also help you save time while interacting with services in the Visual Studio ecosystem. The focus in Microsoft Visual Studio 2015 is productivity, and this book follows the same perspective in presenting topics.

This book is dedicated to my mom. I miss you.

Alessandro

Chapter 1 Account Management Experience

As you learned from my previous book, *Visual Studio 2013 Succinctly*, Visual Studio 2013 for the first time gave developers the ability to sign into Visual Studio by using their Microsoft account. Although not mandatory, this provides many benefits such as unlocking Visual Studio if you are an MSDN subscriber, permanently unlocking Visual Studio Express editions (now replaced by Visual Studio Community), and extending a trial version period. Most importantly, developers can take advantage of synchronized settings, which roam across multiple installations of Visual Studio; this means that they will automatically have the same settings on any computer they use. As you will learn in this chapter, Visual Studio 2015 further enables additional possibilities via signing in with a Microsoft account.

Account Management

Visual Studio 2015 extends the Sign In experience by supporting multiple Microsoft accounts for signing into the development environment. Having multiple accounts to sign into Visual Studio is common for the following reasons:

- Separate accounts for developing and testing applications (or other purposes).
- Separate accounts for work and home.
- Separate accounts for multiple developers working on a single machine.

To simplify account management, use the **Account Settings** window. You have two options to start it:

- Select **File** > **Account Settings**.
- Click the **Account settings...** hyperlink located under the current profile's information, available on the upper-right corner of the IDE (see Figure 1).

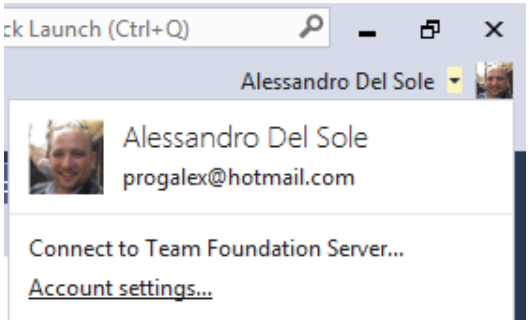

*Figure 1: Locating the **Account settings** hyperlink.*

Supposing that you have only one Microsoft account associated with Visual Studio, the Account Settings window should like Figure 2.

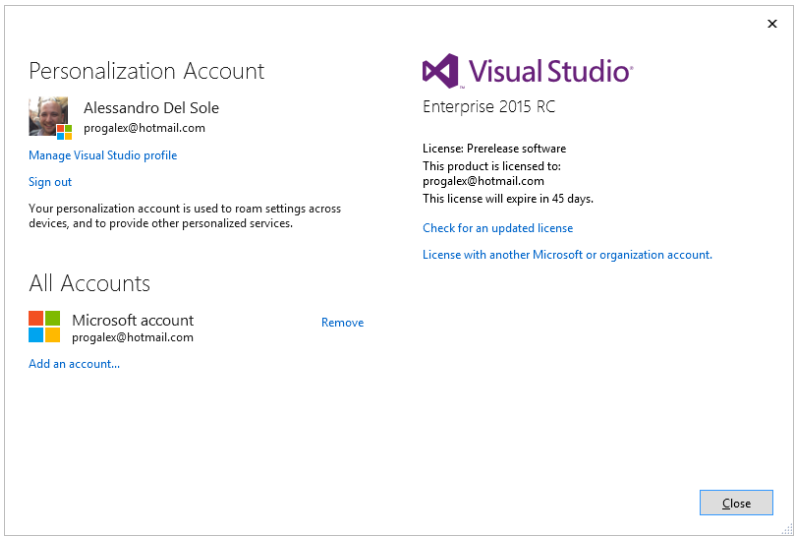

Figure 2: The Account Settings window.

As you can see, Visual Studio 2015 shows summary information about the current user. At the bottom, you can find the **All Accounts** group, which contains the list of associated Microsoft Accounts; the list allows for adding and removing accounts. To add another account, click the **Add an account…** hyperlink. At this point, Visual Studio requires specifying the Microsoft account you want to add, as shown in Figure 3.

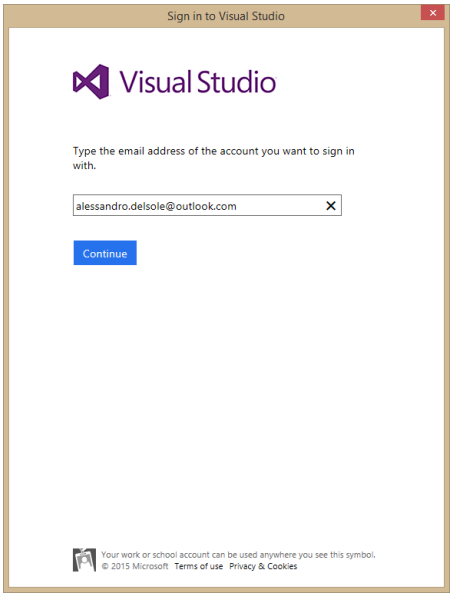

Figure 3: Adding a Microsoft Account to Visual Studio 2015.

If this is the first time that you use the account to sign in, Visual Studio 2015 will also create a new online profile, which is required for synchronizing information and registering an account on Visual Studio Online (formerly known as Team Foundation Service). If this is your case, Visual Studio will ask you to specify basic identity information, as shown in Figure 4.

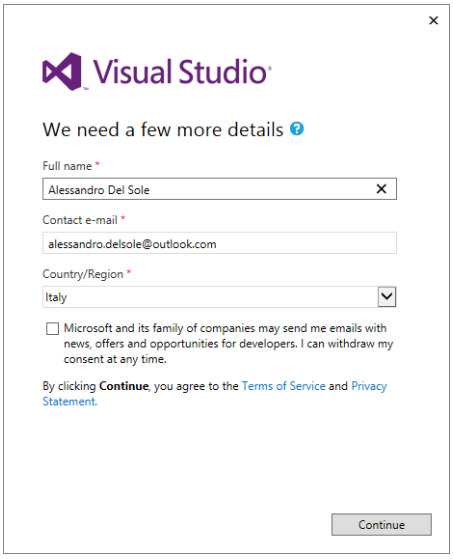

Figure 4: Specifying basic identity info when creating a new online profile.

Tip: You can edit the online profile for a Microsoft account by selecting Manage Visual Studio Profile in the Personalization Account group of the Account Settings window. You will also be able to specify a picture and see the list of Visual Studio Online account memeberships associated with the profile.

Once the online profile has been created, or if it's not the first time you've used the account, the **Account Settings** window will update the list of associated Microsoft accounts, as represented in Figure 5.

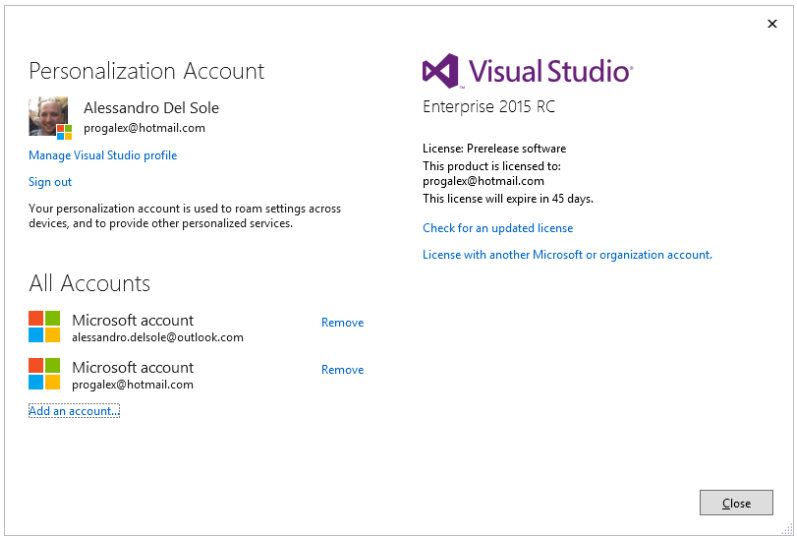

Figure 5: The updated list of available accounts.

In this way, you have a centralized place to manage accounts, and every Microsoft account can take advantage of benefits such as synchronized settings. When you have finished managing accounts, simply click **Close** or sign out from the current account and sign in with a different one.

Tip: You do not need to close an open project when changing accounts.

Automatic Sign In for Microsoft Services

When building an application with Visual Studio 2015, you might need to access many Microsoft services, such as your Azure subscription, Visual Studio Online account, Application Insights (hosted on Azure), Azure Mobile Services, or Windows Store developer subscription.

Note: Application Insights is currently a preview service hosted on your Microsoft Azure subscription, which allows live monitoring and telemetry for availability, performance, and usage of web and mobile applications. You can enable Application Insights when you create a project for a web or mobile app. Support for Application Insights was introduced in Visual Studio 2013 Update 2, so it's not being covered here. You can find additional information here.

All of the previously mentioned services require signing in with a Microsoft account. In Visual Studio 2015, when you sign in, the IDE will automatically connect the current account to any associated Microsoft service; you will not need to enter your credentials every time. For example, if the Microsoft account you use to sign in is also an administrator account on a Microsoft Azure subscription, Visual Studio 2015 will automatically connect the Server Explorer window to the Azure subscription, so that you will be immediately able to interact with services from within the IDE; if you previously set up a Visual Studio Online account, Visual Studio will automatically connect to the account and make source control immediately available. In addition, Visual Studio 2015 makes it easy to select an account when creating connected services or when setting up Application Insights. For example, Figure 6 shows both Team Explorer connected to Visual Studio Online and the option of choosing one of the associated accounts when accessing a subscription.

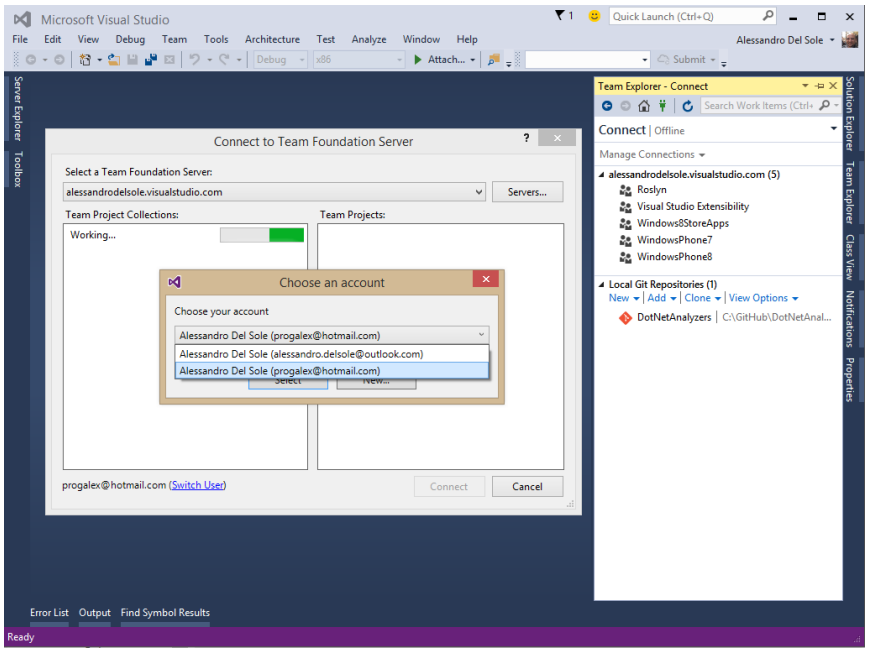

Figure 6: Easy sign into services with Microsoft accounts.

Figure 7 shows how easy it is choosing a Microsoft Account when adding Application Insights telemetry to a new project (this requires an active Azure subscription).

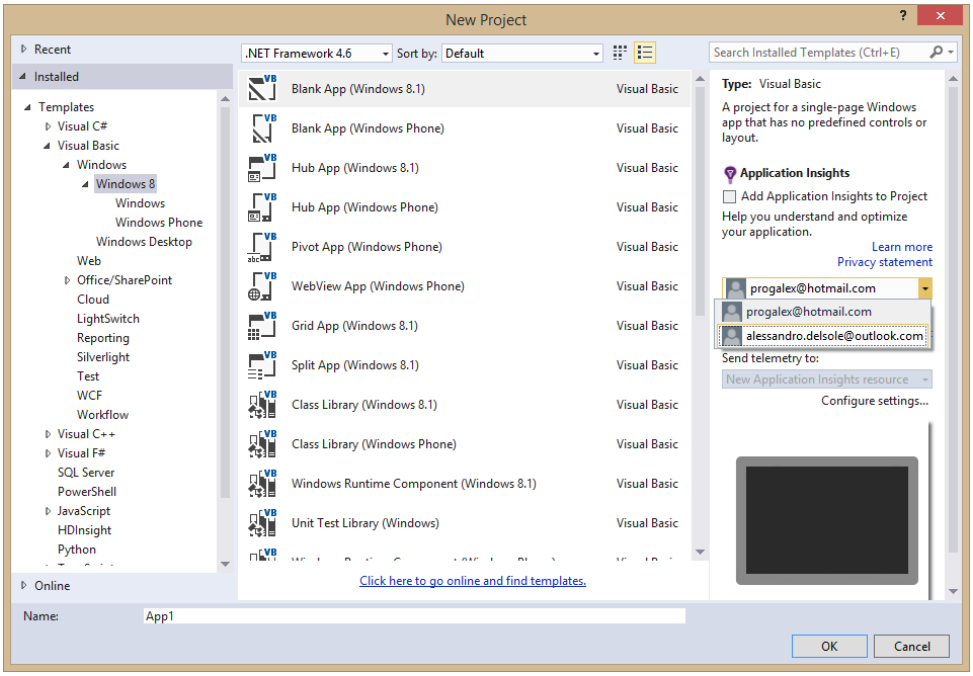

Figure 7: Choosing a Microsoft account for Application Insights.

It is worth mentioning that, when you sign in with a Microsoft Account, Visual Studio 2015 does not store any of your credentials. Authentication happens against a web-based identity provider, which can be the Microsoft Account provider or the Azure Active Directory. If the authentication succeeds, the identity provider gives Visual Studio some authentication tokens, which allows Visual Studio 2015 to handle the authentication tokens securely.

Chapter Summary

Visual Studio 2015 extends and enhances the sign in experience by introducing support for multiple Microsoft Accounts and by enabling automatic connection to Microsoft services that are associated to the account that logged in. Without a doubt, this makes it easier to connect to service and improves productivity by avoiding the need of signing out and then signing in again with different accounts.

Chapter 2 Shared Projects

One of the most desirable features for developers is the ability to write code once and share that code across multiple types of applications. Visual Studio 2015 simplifies the process of sharing code by introducing Shared Projects. If you have created Universal Windows apps in the past with Visual Studio 2013 Update 2 or higher, you have an idea of what a shared project is, but now you can use them to share code between additional project types. This chapter discusses shared projects and explains their structure and usage.

Note: This chapter assumes you have some experience with creating and managing solutions and projects, adding references to other projects, and that you know the terminology for these tasks.

Understanding Shared Projects

Shared Projects allows sharing code, assets, and resources across multiple project types. More specifically, the following project types can reference and consume shared projects:

- Console, Windows Forms, and Windows Presentation Foundation.
- Windows Store 8.1 apps and Windows Phone 8.1 apps.
- Windows Phone 8.0/8.1 Silverlight apps.
- Portable Class Libraries.

Visual Studio 2013 Update 2 first introduced shared projects to provide an easy opportunity to build Universal Windows apps supporting the Windows 8.1/Windows Phone 8.1 pair; now, shared projects are available to all the aforementioned projects and to both C# and Visual Basic. The important thing to underline is that you not only share code but also assets and resources, such as (but not limited to) images and XAML templates. Technically speaking, shared projects are loose assortments of files that are then added, exactly like linked files, to whichever projects reference them. They actually do not target any specific .NET subset and do not produce a .dll library; behind the scenes, when building your solution, the MSBuild tool executes against shared projects the same rules as for linked files.

Creating and Consuming Shared Projects

Shared projects are available through specific project templates in the **New Project** dialog. To understand how they work, create a new solution containing a shared project, a WPF project, and a Windows 8.1 Store app project. In Visual Studio 2015, select **File** > **New** > **Project**. Instead of creating a new project directly, create a new blank solution by selecting **Blank Solution** under **Other Project Types**/**Visual Studio Solutions** (see Figure 8). Name the new solution **SharingSample** and click **OK**.

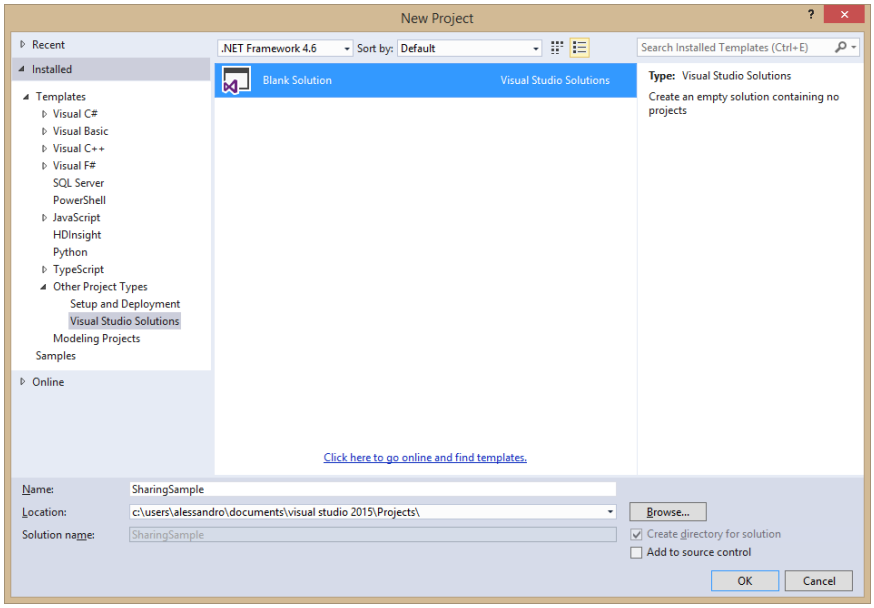

Figure 8: Creating a blank solution.

The next step is adding the shared project. You can either right-click the solution name in Solution Explorer and then select **Add** > **New Project** or select **File** > **Add** > **New Project**. As you can see in the **Add New Project** dialog (see Figure 9), the **Shared Project** template is available in the list. You can select either C# or Visual Basic (there will be an example for both).

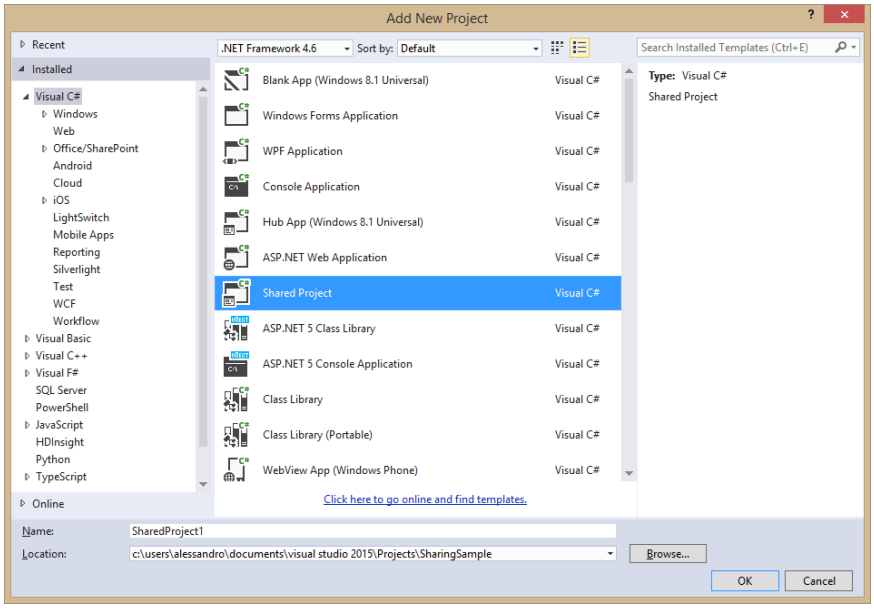

Figure 9: Adding a shared project.

You can choose whatever name you want or leave the default name and click **OK**. As you can see in Solution Explorer, the shared project has a very simple structure; it is represented by a green icon, as shown in Figure 10.

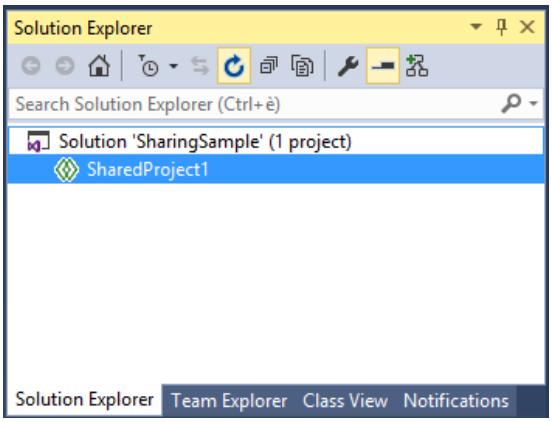

Figure 10: The shared project in Solution Explorer.

Note: Behind the scenes, shared projects are represented by files with the .shproj extension, which basically contains the MSBuild rules required for the compilation process, plus a file with a .projitems extension, which instead contains the list of items to be shared.

The goal of a shared project is to make it easier to share code, resources, and assets. Add three folders to the project called **Code**, **Resources**, and **Assets**. Now, suppose you want to share a XAML template, an image, and some code between a WPF project and a Windows 8.1 Store project. Right-click the **Code** folder and select **Add** > **New Item**. In the **Add New Item** dialog, select the **Class** template and name the new class **Person.cs** (see Figure 11).

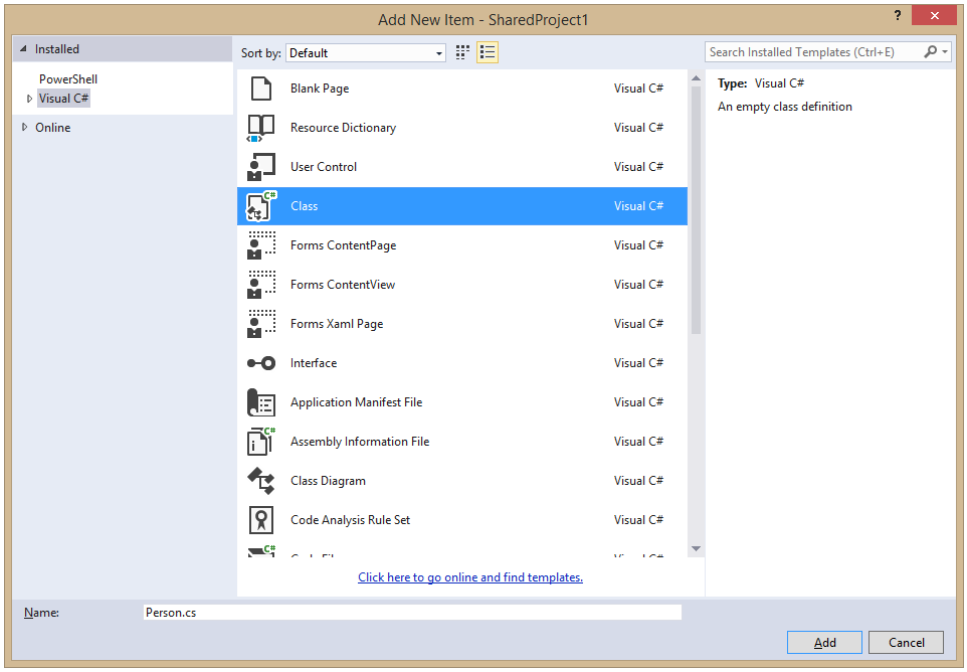

Figure 11: Adding a shared class.

The following code example contains a definition for the **Person** class and a collection called **People**, which contains some sample instances of the **Person** class.

C# Code Example

```
using System.Collections.ObjectModel;

namespace SharedProject1
{
    public class Person
    {
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public int Age { get; set; }
    }

    public class People : ObservableCollection<Person>
    {

        public People()
        {
            //Add sample data
            var p1 = new Person();
            p1.LastName = "Del Sole";
            p1.FirstName = "Alessandro";
            p1.Age = 37;

            var p2 = new Person();
            p2.LastName = "White";
            p2.FirstName = "Robert";
            p2.Age = 40;

            this.Add(p1);
            this.Add(p2);
        }
    }
}
```

Visual Basic Code Example

```
Imports System.Collections.ObjectModel

Public Class Person
    Public Property FirstName() As String
    Public Property LastName() As String
    Public Property Age() As Integer
End Class

Public Class People
```

```
    Inherits ObservableCollection(Of Person)

    Public Sub New()
        'Add sample data
        Dim p1 = New Person()
        p1.LastName = "Del Sole"
        p1.FirstName = "Alessandro"
        p1.Age = 37

        Dim p2 = New Person()
        p2.LastName = "White"
        p2.FirstName = "Robert"
        p2.Age = 40

        Me.Add(p1)
        Me.Add(p2)
    End Sub
End Class
```

The **People** collection is the data that both the WPF and the Windows Store applications will show on screen; since both application types are based on XAML, it is a good idea to create a data template that will be shared in the solution. To accomplish this, right-click the **Resources** folder and select **Add** > **New Item**. In the **Add New Item** dialog, locate the **XAML** node and select **Resource Dictionary**, as shown in Figure 12.

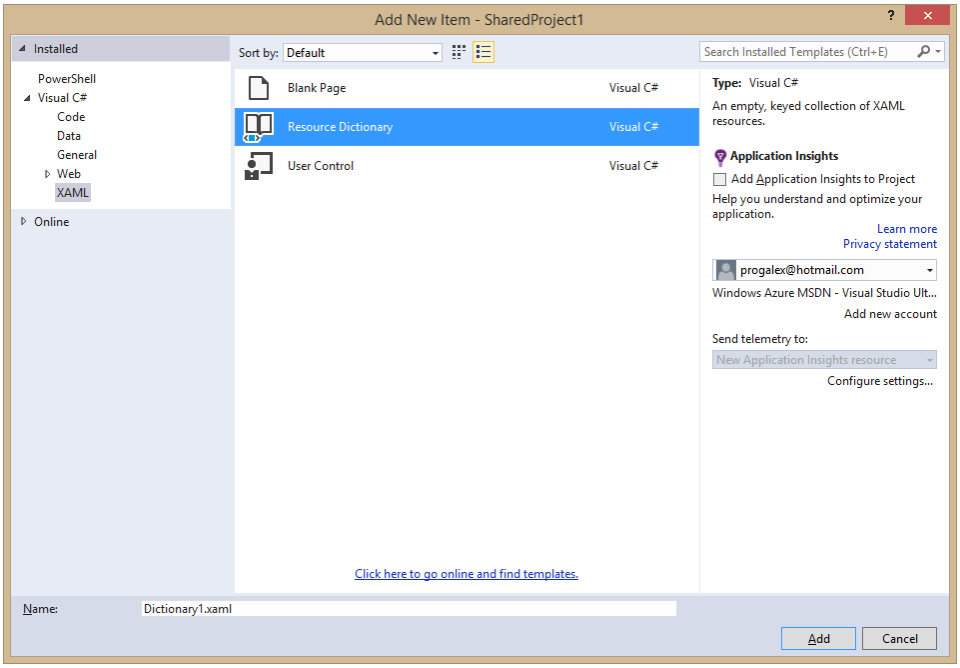

Figure 12: Adding a shared resource dictionary.

In the code you specify a new **DataTemplate** object, which contains some **TextBlock** controls, arranged within **StackPanel** containers, that are bound to the properties of the **Person** class.

```
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

  <DataTemplate x:Key="MyTemplate">
    <Border Width="240" BorderBrush="Black" BorderThickness="2" Margin="3">
      <StackPanel Orientation="Vertical" Margin="3">
        <StackPanel Orientation="Horizontal">
          <TextBlock Text="Last name:" FontWeight="Bold" />
          <TextBlock Text="{Binding LastName}"/>
        </StackPanel>
        <StackPanel Orientation="Horizontal">
          <TextBlock Text="First name:" FontWeight="Bold"/>
          <TextBlock Text="{Binding FirstName}"/>
        </StackPanel>
        <StackPanel Orientation="Horizontal">
          <TextBlock Text="Age:" FontWeight="Bold"/>
          <TextBlock Text="{Binding Age}"/>
        </StackPanel>
      </StackPanel>
    </Border>
  </DataTemplate>
</ResourceDictionary>
```

In order to successfully share the resource dictionary to both targets, you must remove the following XML namespace alias, which is specific to Windows Store and Windows Phone.

```
xmlns:local="using:SharedProject1.Resources"
```

The last step in the shared project is adding a sample image. Right-click the **Assets** folder and select **Add** > **Existing Item**, then select an image file of your choice among one of the supported file formats, such as JPG, PNG, GIF, or BMP. In order to make the image file usable by WPF, in the **Properties** window set the **Copy to Output Directory** property to **Copy if newer**. Figure 13 shows how the shared project appears in Solution Explorer and how to set the property for the image file in the Properties window.

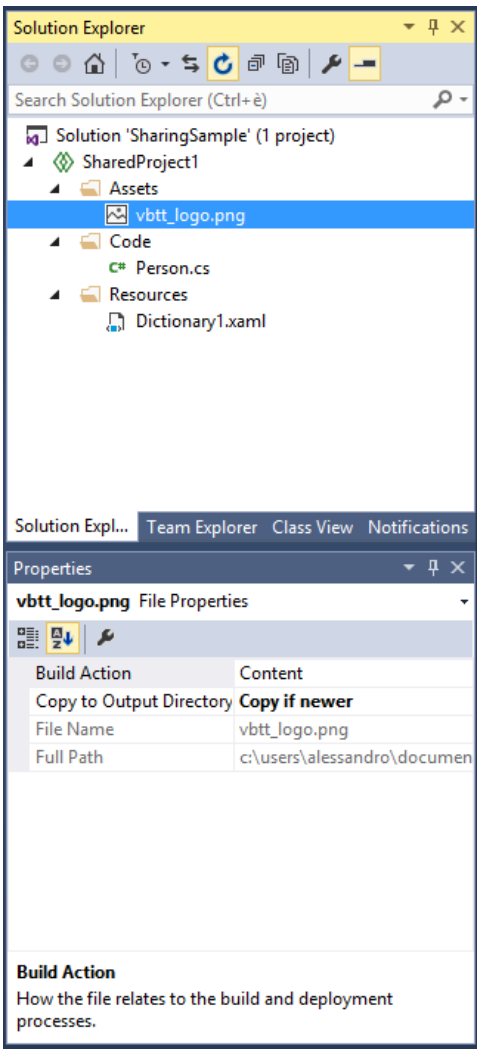

Figure 13: The current appearance for the shared project.

At this point, you want to add the first client project. Select **File** > **Add** > **New Project** and in the **New Project** dialog select the **WPF Application** project template; for the sake of the example's consistency, name the new project as **MyWPFApp** (see Figure 14).

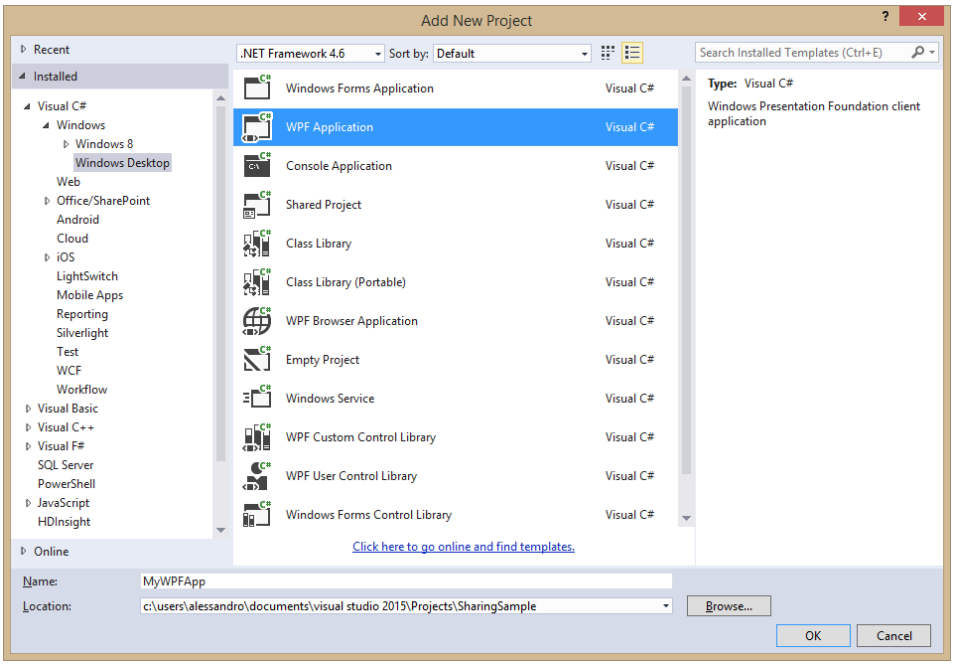

Figure 14: Adding a WPF client project.

Once the project is available, the first thing to do is add a reference to the shared project. As I told you before, shared projects do not produce assemblies, so adding a reference to a shared project means actually linking code files in the shared project to the current project. When you add a reference to a shared project, the **Reference Manager** dialog shows a special node called **Shared Projects**, where you can find a list of shared projects in your solution (see Figure 15).

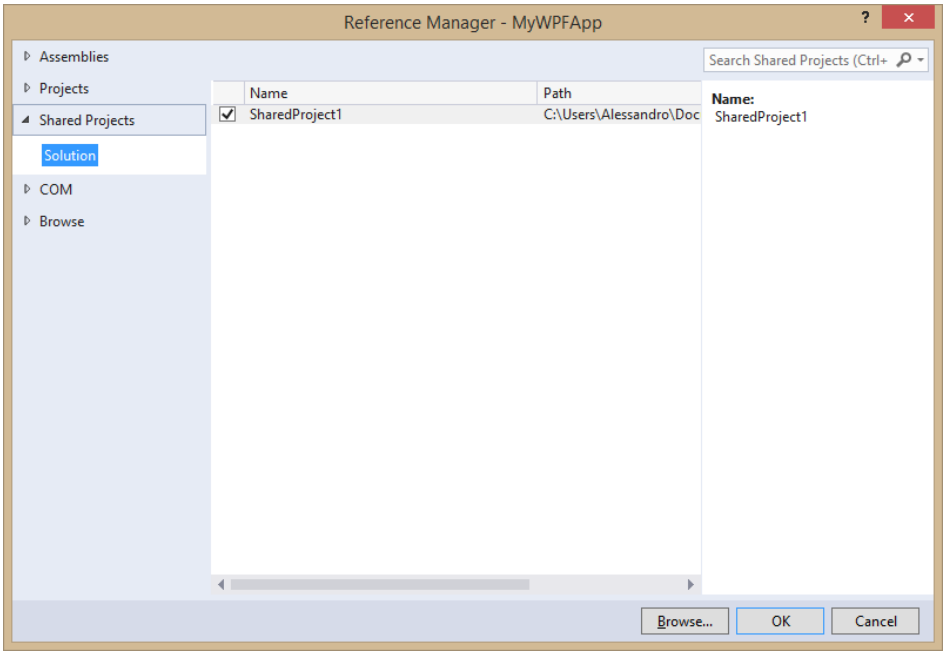

Figure 15: Adding a reference to a shared project.

Select the shared project and click **OK**.

Tip: If you expand the Reference node in the Solution Explorer, you will see that the icon for the shared project is different than the icon for assemblies. This is a reminder that shared projects do not produce an assembly.

In the WPF application you want to use resources residing in the shared resource dictionary. To accomplish this, you need to add a resource dictionary into the App.xaml file (or Application.xaml in Visual Basic) like in the following code.

```
    <Application.Resources>
                <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <ResourceDictionary Source="/Resources/Dictionary1.xaml"/>
            </ResourceDictionary.MergedDictionaries>
        </ResourceDictionary>
    </Application.Resources>
```

Notice how you specify the dictionary location via a relative uniform resource identifier (URI), which includes the subfolder name in the shared project. Do not worry if the `Source` property value is underlined with an error squiggle (semantic error notification), the resource dictionary will definitely be resolved at runtime. The user interface for the sample application is very simple. In the **MainWindow.xaml** file, place a **`Grid`** container; this will show the image file as the application logo and a **`ListBox`** control to present the list of data. Notice how you can use a simplified relative URI to point to the image file; the URI includes the subfolder name as it appears in the shared project.

```
<Window x:Class="MyWPFApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
        xmlns:local="clr-namespace:MyWPFApp" Loaded="Window_Loaded"
        mc:Ignorable="d"
        Title="MainWindow" Height="350" Width="525">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="80"/>
            <RowDefinition/>
        </Grid.RowDefinitions>

        <!--Replace with a file name of your choice...-->
        <Image Source="../SharedProject1/Assets/LogoVBTT.jpg"/>
```

```
        <ListBox ItemTemplate="{StaticResource MyTemplate}" Name="DataBox"
                 Grid.Row="1"/>
    </Grid>
</Window>
```

As highlighted in the code, you also have to provide an event handler for the **`Window.Loaded`** event, which is the place where data is loaded. Concerning the **`ListBox`**, you assign the **`DataTemplate`** called **`MyTemplate`**, defined in the shared resource dictionary, as the item template to present each instance of the **`Person`** class. The **`Window.Loaded`** event handler simply creates an instance of the **`People`** collection; the instance is assigned to the **`ListBox.ItemsSource`** property to populate it.

C# Code Example

```
        //Requires a using SharedProject1.Code directive
        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            this.DataBox.ItemsSource = new People();
        }
```

Visual Basic Code Example

```
    Private Sub Window_Loaded(sender As Object, e As RoutedEventArgs)
        Me.DataBox.ItemsSource = New People()
    End Sub
```

Tip: Remember that C# defines a namespace for each subfolder in the project that contains code. For this reason you have to place a `using SharedProject1.Code` directive to import the subfolder's content.

If you now run the sample project, you will get a result similar to Figure 16 (of course you have probably used a different image file).

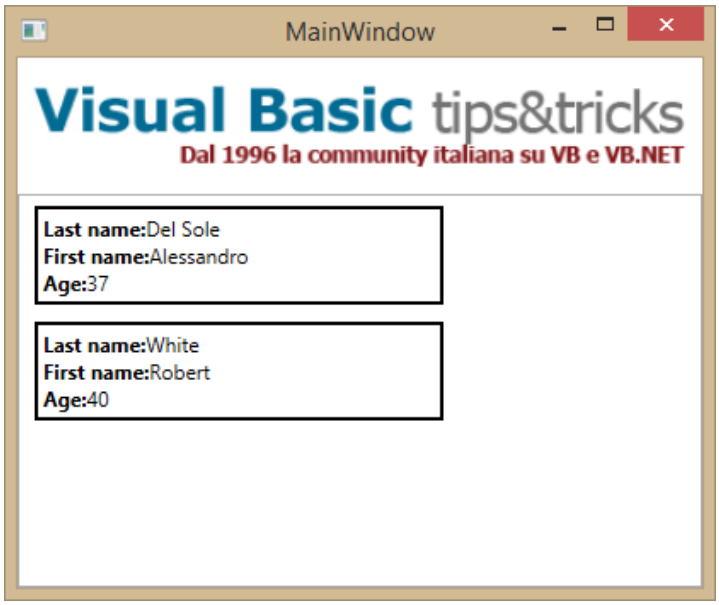

Figure 16: Adding a reference to a shared project.

This is certainly a simple example, but the goal is to understand how to share different type of resources. At this point, add a new Windows Store app by using the **Blank App (Windows 8.1)** template, as shown in Figure 17. Name the new project as **MyWindowsApp** and click **OK**.

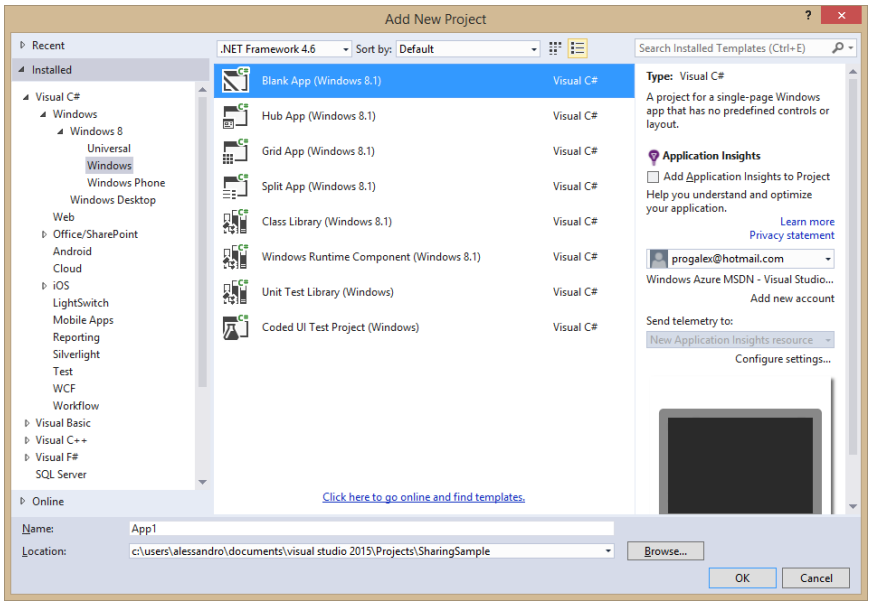

Figure 17: Adding a Windows Store project.

Now do the following:

- Add a reference to the shared project, exactly as you did in the WPF project.
- In the App.xaml file (or Application.xaml in Visual Basic), add the same XAML code that you already wrote in the App.xaml of the WPF project.
- In the MainPage.xaml file, add the same **`Grid`** content definition that you already wrote in the MainWindow.xaml file of the WPF project.

Add an event handler for the **`Page.Loaded`** event. The full XAML code of MainPage.xaml is as follows.

```
<Page
    x:Class="MyWindowsApp.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:MyWindowsApp" Loaded="Page_Loaded"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <Grid.RowDefinitions>
            <RowDefinition Height="80"/>
            <RowDefinition/>
        </Grid.RowDefinitions>
        <Image Source="Assets/vbtt_logo.png"/>
        <ListBox ItemTemplate="{StaticResource MyTemplate}" Name="DataBox"
                 Grid.Row="1"/>
    </Grid>
</Page>
```

The **`Page.Loaded`** event handler is identical to its WPF counterpart, except that here you have a **`Page`** rather than a **`Window`**.

C# Code Example

```
        //Requires a using SharedProject1.Code directive
        private void Page_Loaded(object sender, RoutedEventArgs e)
        {
            this.DataBox.ItemsSource = new People();
        }
```

Visual Basic Code Example

```
    Private Sub Page_Loaded(sender As Object, e As RoutedEventArgs)
        Me.DataBox.ItemsSource = New People()
    End Sub
```

Set the Windows Store project as the startup one and press **F5**. As you can see, the app shows the same image and data, which has been presented by using the same data template as in the WPF counterpart (see Figure 18).

Figure 18: The Windows Store app running.

It is easy to understand how you can leverage shared projects to share a variety of items between multiple project types, including code, assets, and resources. Sharing XAML and C#/VB code is probably the most interesting opportunity that shared projects offer.

Note: both shared projects and portable class libraries (PCL) allow sharing code, XAML resources, and assets, but of course there are some differences that might be summarized as follows. A shared project does not produce a reusable assembly, so it can only be consumed from within the solution. A shared project has support for platform-specific code, because it supports environment variables such as WINDOWS_PHONE_APP and WINDOWS_APP that you can use to detect which platform your code is running on. Finally, shared projects cannot have dependencies on third-party libraries. By comparison, a PCL produces a reusable .dll library and can have dependencies on third-party libraries, but it does not support platform environment variables.

Chapter Summary

First introduced in Visual Studio 2013 Update 2 for building Universal Windows apps, Visual Studio 2015 shared projects are available to a variety of project types, such as WPF, Windows Forms, Windows Store, Windows Phone, and Portable Libraries. With shared projects, you can write C#/VB code once and share it to multiple projects in your solution, and you can easily share XAML resources like styles, control templates, and data templates, as well as assets such as image files. Shared projects are not always the ultimate solution for sharing items; in fact, there are situations in which you might want to prefer Portable Libraries, such as if you need to create a reusable .dll that targets a specific .NET subset. Regardless, shared projects are definitely an easy and lightweight opportunity to save time.

Chapter 3 Code Editor Improvements

Continuing to the same goal of Visual Studio 2013, Microsoft has made additional, important improvements to a fundamental area of the IDE: the code editor. This chapter describes new features in the so-called code-focused IDE experience, which brings productivity to the maximum and really helps for writing high-quality, fast, and efficient code.

Note: Except where explicitly stated, all the new features described in this chapter apply to both C# and Visual Basic.

Touch Gestures

The code editor has been enhanced to support common gestures on touch screens. Touch already worked in previous versions but the IDE was not optimized; touch support was only available previously because of hardware capabilities. In Visual Studio 2015, you can:

- Double-tap a word or an identifier to select it.
- Press and hold on screen to open the context menu.
- Tap the margin of the code editor to select the corresponding line of code.
- Tap and hold anywhere on the code editor's surface to scroll up and down.
- "Pinch and zoom" to zoom in and out with two fingers.

You will find these features very useful if you use Visual Studio 2015 on touch-enabled devices such as tablets or laptops.

Colorized Tooltips

The code editor automatically creates collapsible regions for many language constructs, such as (but not limited to) class definitions, structure definitions, and method bodies. Starting from Visual Studio 2013, when a region is collapsed, you can get a preview of the full code by passing the mouse pointer over the collapsed construct. Visual Studio 2015 provides an enhanced experience providing colorized tooltips and the full object or member declaration, as shown in Figure 19.

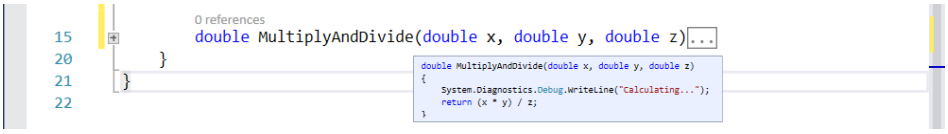

Figure 19: Colorized Tooltips.

This is much better if compared to Visual Studio 2013, where you do not have the object/member declaration and the tooltip's color is only black.

Light Bulbs and Quick Actions

As you might know, in Visual Studio 2015 and .NET Framework 4.6, the managed compilers have been completely rewritten, released as an open source project, and brought to the development experience by the .NET Compiler Platform (formerly known as Project Roslyn).

Note: Talking about the .NET Compiler Platform is out of the scope of this book. You can get specific information and learn how to create custom code analyzers at Roslyn.

Among the other benefits, compilers expose APIs, have even better integration with the code editor and the rest of the IDE, and they provide full, real-time, static code analysis to find issues and provide suggestions as you type. Improvements center primarily on redundant code, finding and fixing errors as you type, and refactoring code.

When Visual Studio 2015 detects redundant code, errors, or code that should be refactored, it shows the Light Bulb. The Light Bulb is a container of actions, called Quick Actions, which you can take to solve a particular problem. Figure 20 shows an example of the Light Bulb and the available actions when you invoke a method that has not been declared in your code.

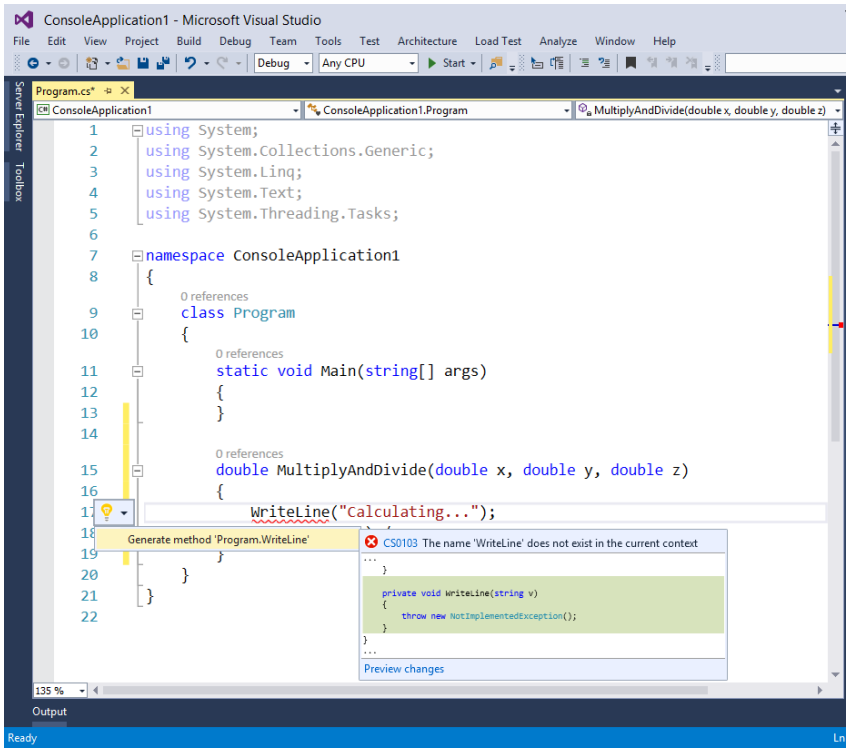

Figure 20: The Light Bulb provides quick actions to suggest fixes for errors or to refactor your code.

Usually the Light Bulb appears automatically when you place the cursor on a piece of code that needs some fixes or when you hover over error squiggles; however, you can manually enable the Light Bulb by pressing **CTRL + <Period>**. You got just a quick introduction about the Light Bulb and Quick Actions, but in the next paragraphs you will learn more about this powerful new feature.

Finding and Fixing Redundant Code

Live static code analysis is able to detect redundant code such as unused **`using`** directives (**`Imports`** for Visual Basic) or unneeded object invocations. Redundant code is grayed out so that you can easily see it in your code. Figure 21 shows an example of redundant code, made of unused **`using`** directives and of the **`this`** object invocation.

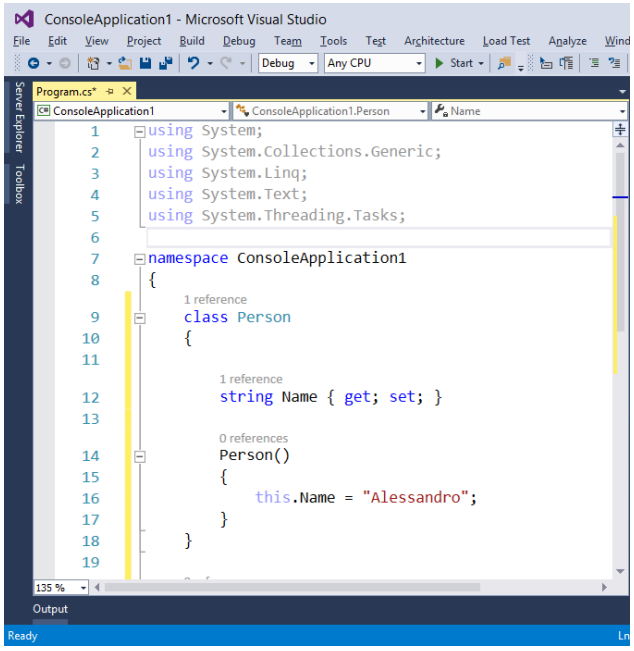

Figure 21: Detecting redundant code.

If you click one of the **`using`** directives, the Light Bulb appears. You can expand it and see how Visual Studio suggests a quick action for removing unnecessary directives by highlighting in red the code that will be removed (see Figure 22).

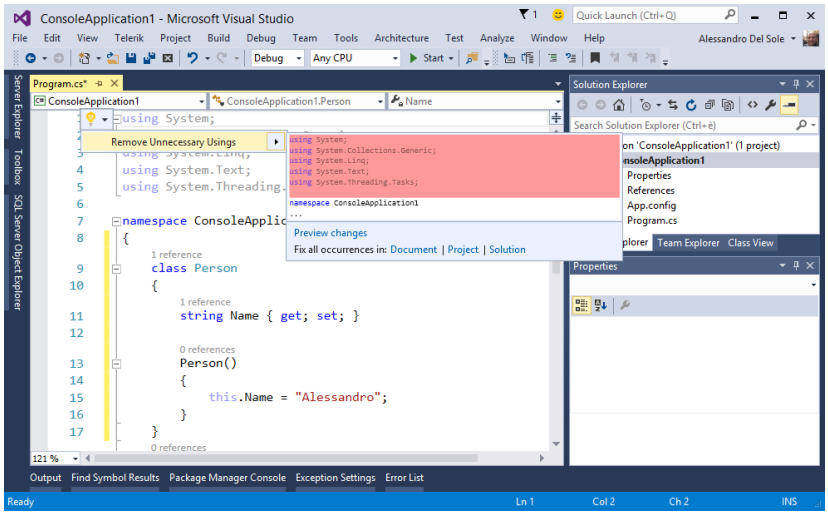

Figure 22: The Light Bulb suggests removing unnecessary code.

You can also apply these changes to the current project or to the entire solution, not just the current code file. As you can also see, you have the option of previewing changes. This is explained in more detail in the next paragraph. For now, click the **Remove Unnecessary Using** suggestions and see how the IDE removes the redundant **`using`** directives. You can also repeat the same steps on the **`this`** object.

Tip: The option for removing unnecessary using is also available by right-clicking in the code editor, navigating to Organize Usings, and selecting Remove Unnecessary Usings. In Visual Basic, the same option is available under Organize Imports > Remove Unnecessary Imports.

Fixing Errors

Live static code analysis and Light Bulbs make it easier to detect and fix errors as you type. For example, consider the class definition shown in Figure 23.

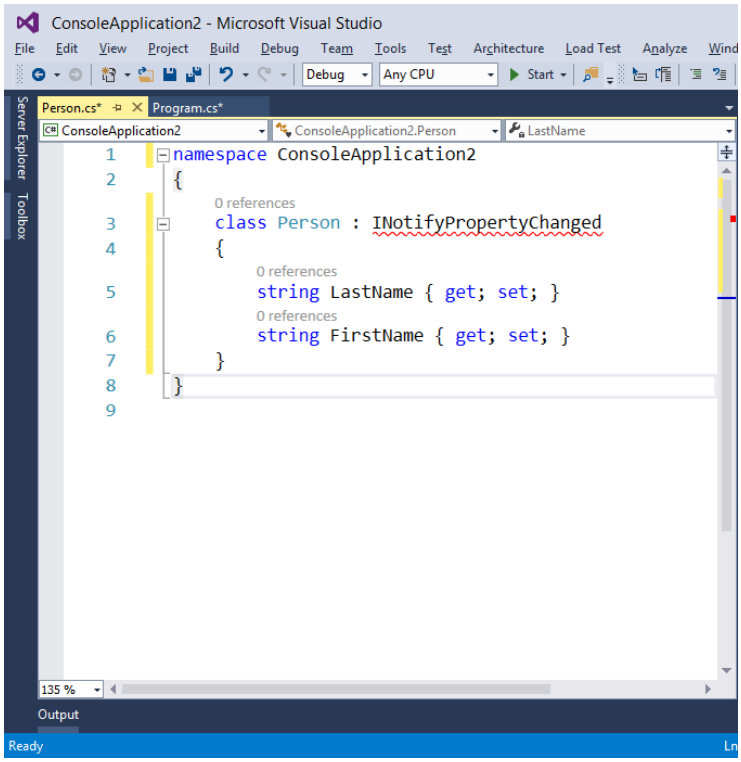

Figure 23: A sample class with errors.

The code has several errors, in that it is missing a **`using System.ComponentModel`** directive and the class does not actually implement the **`INotifyPropertyChanged`** interface, as highlighted by the error squiggle.

Tip: In Visual Studio 2015, you no longer need to place the cursor outside a line of code to get error squiggles. In fact, the new live static code analysis will show them immediately.

You can use the Light Bulb to fix errors. In this case, the Light Bulb first suggests quick actions to resolve the **`INotifyPropertyChanged`** interface, as shown in Figure 24.

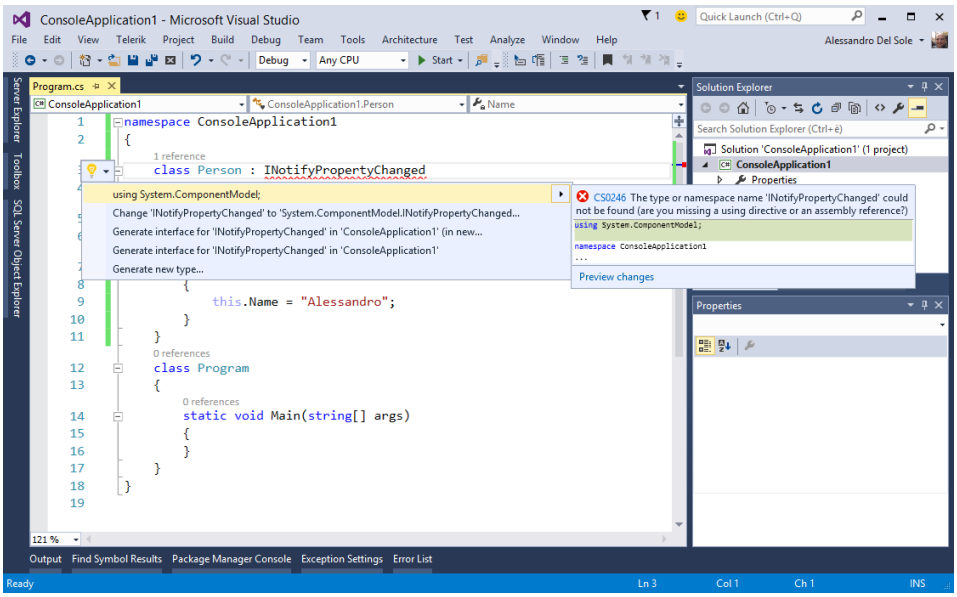

Figure 24: Quick Actions for a missing namespace.

A tooltip provides a detailed description about the error message and shows a preview of changes in the code when you hover over one of the suggested quick actions. A good option at this point is adding a **`using`** directive; select the first solution to do it. At this point, the compiler detects that the class is not implementing the interface yet, so the Light Bulb provides specific quick actions to fix this error, as demonstrated in Figure 25.

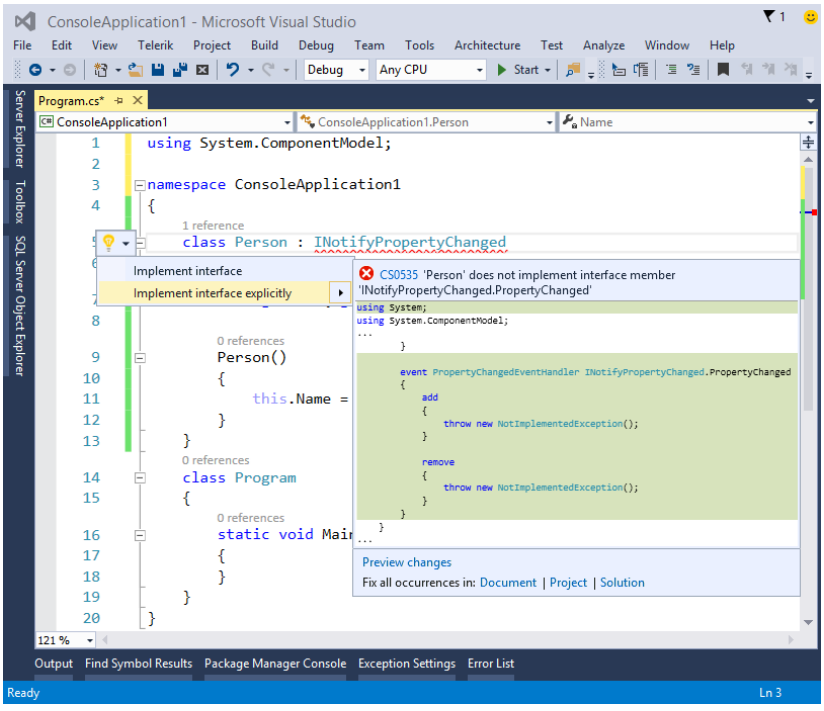

Figure 25: Quick Actions for a not implemented interface.

Suppose you want to implement the interface implicitly; select the first solution (**Implement interface**) and in the preview click the **Preview changes** hyperlink. At this point, the **Preview Changes** dialog appears, showing one or more code files where the change is going to be applied and a full preview of the code with changes. These changes are highlighted so that you can immediately have a look, as demonstrated in Figure 26.

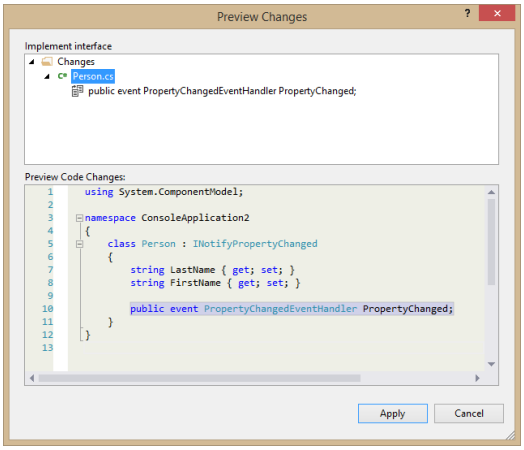

Figure 26: Previewing changes before making edits.

If you are fine with the changes, simply click **Apply** or, if you want to choose a different quick action, click **Cancel**. Compilers are intelligent enough to suggest additional solutions based on the object you are going to use. For instance, if you want an object to implement the **`IDisposable`** interface, the Light Bulb will show additional quick actions specific to the Dispose pattern, as represented in Figure 27.

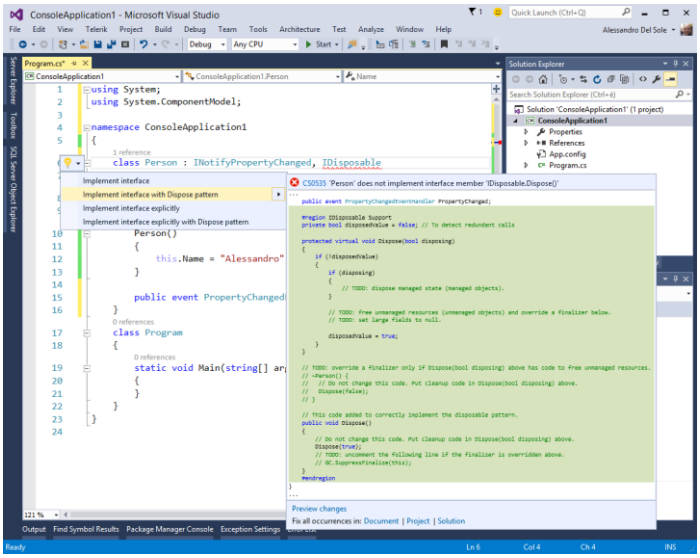

Figure 27: Quick Actions suggesting different ways to implement a pattern.

Light Bulbs and Quick Actions are very powerful, not only because they help you fix errors quickly, but also because they usually provide the most appropriate solutions for the current context.

Applying Refactorings

Refactoring is a technique for rewriting pieces of code in a more efficient and cleaner way, while keeping the existing code behavior and results. During the years, Visual Studio has been offering built-in support to certain refactoring techniques for C#, while Visual Basic developers had to use third-party IDE extensions. In Visual Studio 2015, refactoring has been dramatically enhanced in several ways. Refactoring techniques are now part of the Light Bulb's quick actions; in addition, new features have been added, and finally Visual Basic has support for refactoring. You'll learn more about this in the next few paragraphs.

Extract Method

Consider the following code, which simply calculates the area of a circle, given the radius.

```
        static void Main(string[] args)
        {
            double radius = 4;
            double circleArea = radius * radius * Math.PI;
            Console.WriteLine(circleArea);
            Console.ReadLine();
        }
```

The math operation happens inside a method, but it would be useful to separate the logic and make that code reusable. If you select the first two lines of the method body and enable the Light Bulb by pressing CTRL+<Period>, you will see an option called **Extract Method**, as shown in Figure 28.

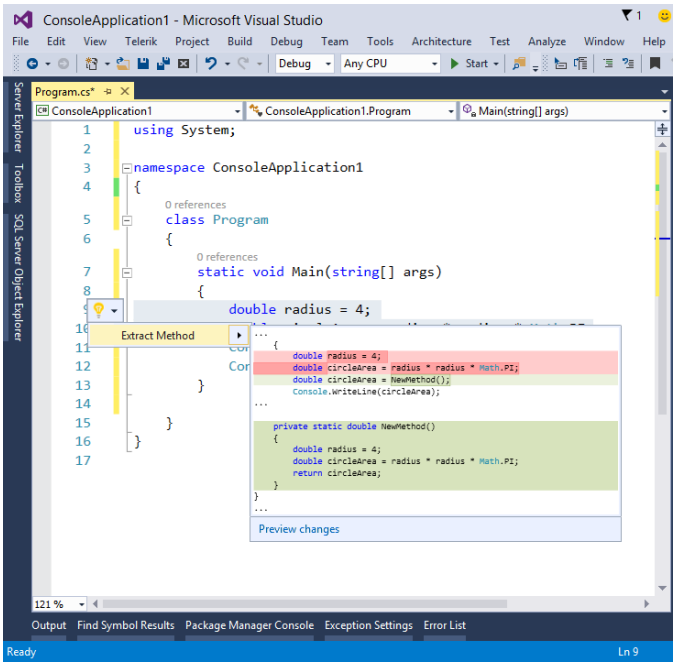

Figure 28: Quick Actions suggesting to extract a new method.

As usual, you will get an inline preview of changes and you can click the **Preview Changes** hyperlink for a more detailed view. If you click **Extract Method**, Visual Studio will change your code to the following:

```
        static void Main(string[] args)
        {
            double circleArea = NewMethod();
            Console.WriteLine(circleArea);
            Console.ReadLine();
        }

        private static double NewMethod()
        {
            double radius = 4;
            double circleArea = radius * radius * Math.PI;
            return circleArea;

        }
```

Not only the code is organized better, but also the new method is reusable. Of course, the best option would be making the new method accept an argument and use this instead of hard-coding the **`radius`** variable value. When the new method is generated, you can replace the green-highlighted default name; this is accomplished via a new feature called **Inline Rename**, which deserves some more detailed information.

Tip: In C#, Extract Method refactoring is still available in the context menu under Refactorings.

Inline Rename

One of the most common tasks while writing code is renaming identifiers, such as method or variable names. In the past, you could simply right-click the identifier, select **Rename**, and rename the identifier inside a modal dialog. In C#, you can also do this by pressing Ctrl + R to rename the variable or method the cursor is on. This feature has been completely redesigned in Visual Studio 2015: when you right-click an identifier and select **Rename**, you can rename directly inside the code editor and the changes are reflected to all the other occurrences of the identifier. With this approach, you never lose focus on your code. In addition, you can automatically rename occurrences inside comments and in string literals. Continuing the previous example based on the Extract Method refactoring, Figure 29 shows how Visual Studio 2015 allows you to rename a newly generated method.

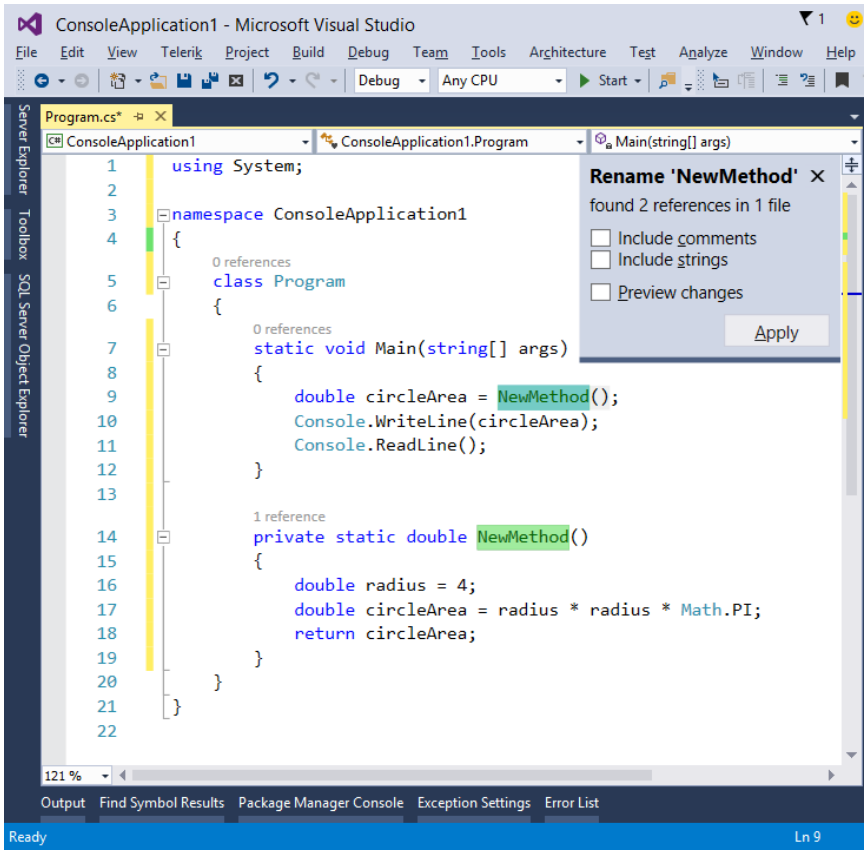

Figure 29: Renaming an object with the new Inline Rename.

A non-modal popup at the upper-right corner shows how many occurrences of the identifier the IDE found and the total number of code files containing the identifier. To rename an object, simply type the new name over one of the highlighted occurrences. Changes will be immediately reflected to all other occurrences. If you are satisfied with changes, click **Apply**, otherwise just close the popup or press **ESC**. Notice that you can still preview changes as you did with when fixing errors (see the *Fixing Errors* section earlier in this chapter). The good news is that you can automatically rename occurrences inside comments and string literals. Figure 30 shows the same code extended with a comment and a string literal within a **`Console.WriteLine`** statement.

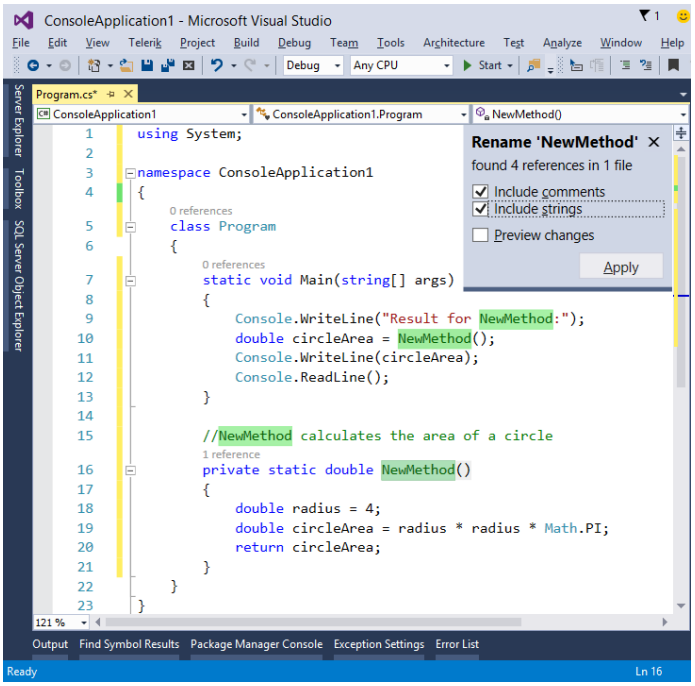

Figure 30: Inline Rename can replace occurrences inside comments and string literals.

If you select both the **Include comments** and **Include strings** option in the **Rename** box, you will see how occurrences of the identifier in comments and string literals are automatically selected. If you type a new name and click **Apply**, all the occurrences will be replaced, as demonstrated in Figure 31 which shows how the **`NewMethod`** identifier has been renamed into **`CalculateCircleArea`**.

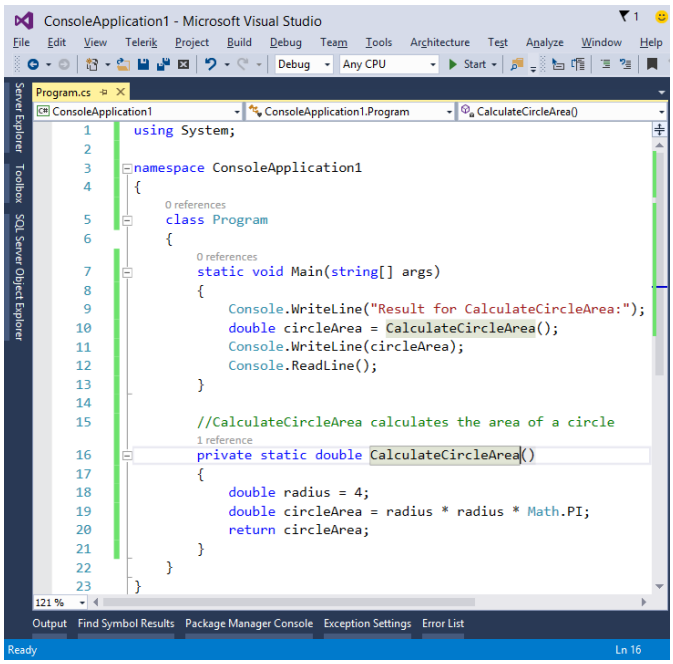

Figure 31: All occurrences of an identifier renamed with Inline Rename.

Inline Rename is another helpful feature that will help you save a lot of time when refactoring your code.

Introduce Constant and Introduce Local Constant

The Introduce Constant and Introduce Local Constant refactoring techniques allow replacing hard-coded values with constants; these can be at the class level or within method bodies. For example, consider the following code.

```
class Program
{
    static void Main(string[] args)
    {
        double circleArea = CalculateCircleArea(4.2);
    }

    private static double CalculateCircleArea(double radius)
    {
        double circleArea = radius * radius * Math.PI;
        return circleArea;
    }
}
```

This is very simple code: the **`CalculateCircleArea`** method returns to the caller the area of a circle, given the radius, which is passed as an argument. The **`Main`** method invokes **`CalculateCircleArea`** passing a value. If you now select **`4.2`** and enable the Light Bulb, you will see four options:

1. Introduce constant for '4.2',
2. Introduce constant for all occurrences of '4.2' (see Figure 32)
3. Introduce local constant for '4.2'
4. Introduce local constant for all occurrences of '4.2' (see Figure 33)

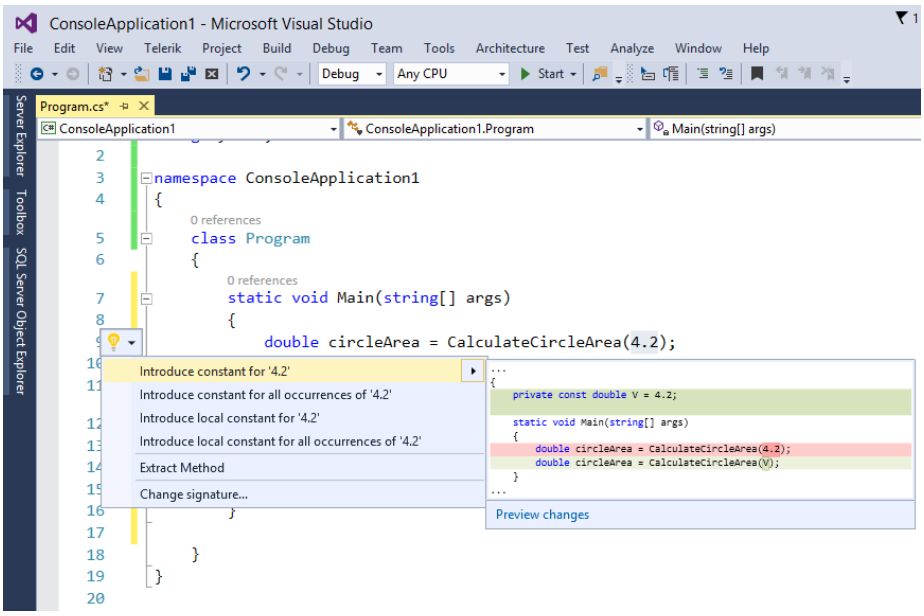

Figure 32: Introducing a constant at the class level.

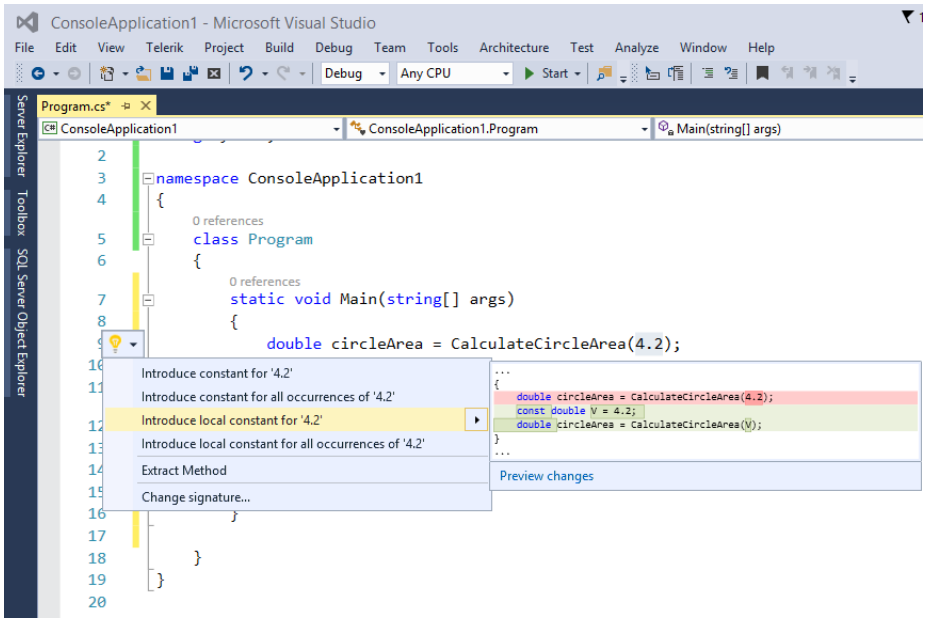

Figure 33: Introducing a local constant into the method body.

The first two options will introduce a constant declaration at the class level, assigning 4.2 as the value; in particular, the second option will do the same for all occurrences (if any other) of 4.2 in the code. The other option will instead introduce a local constant into the method body, for the current occurrence or for all occurrences respectively. Whatever option you choose, Visual Studio will apply the change and will enable **Inline Rename** to give you an opportunity of choosing a different identifier, as represented in Figure 34, which shows both the applied constant and the Inline Rename in action.

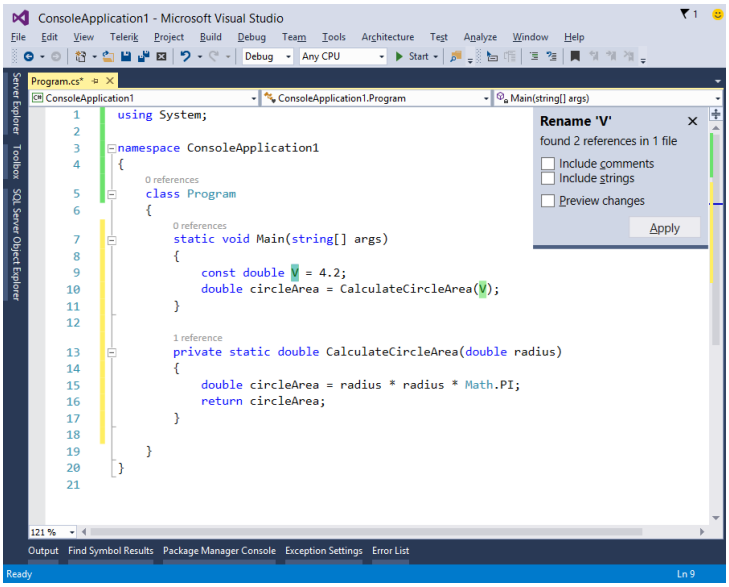

Figure 34: Applying and renaming a constant.

Inline Temporary Variable

Inline Temporary Variable is a new refactoring technique that allows simplifying your code by removing unnecessary variable declarations. To understand how it works, consider the **`CalculateCircleArea`** method described in the previous paragraph. If you select the **`circleArea`** variable in the method body and then enable the Light Bulb, Visual Studio will show a simplified solution, as represented in Figure 35.

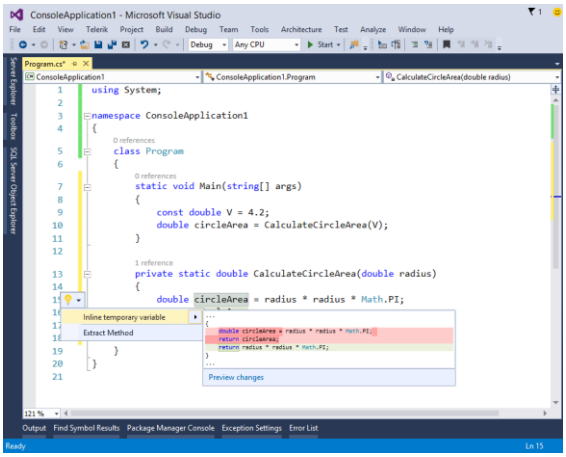

Figure 35: Inline Temporary Variable.

In the preview, Visual Studio highlights in red the current code and then shows the new result below. Applying this quick action results in the following code.

```
        private static double CalculateCircleArea(double radius)
        {
```

```
            return radius * radius * Math.PI;
        }
```

This code is certainly simpler and more readable.

Introduce Local

Introduce Local is a new refactoring technique that simplifies complex expressions by introducing local variables. As an example, consider the following code.

```
private static double CalculateCircleArea(object radius)
{
    double circleArea =
    Convert.ToDouble(radius) * Convert.ToDouble(radius) * Math.PI;
    return circleArea;
}
```

This new version of the **`CalculateCircleArea`** method takes an argument of type **`object`**, and then performs the same conversion twice. With Introduce Local, you can definitely improve this code. Select **`Convert.ToDouble(radius)`** and then enable the Light Bulb. You will get two Quick Actions: the first allows introducing a local variable for the selected code, whereas the second one allows introducing a local variable for all other occurrences of the selected code (see Figure 36).

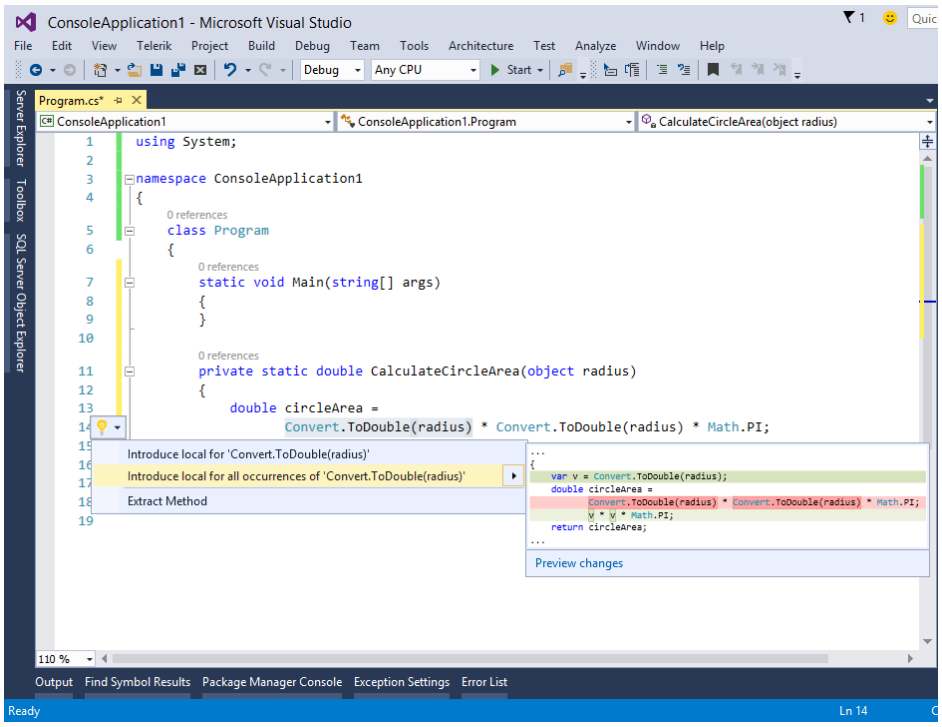

Figure 36: Introducing a local variable to simplify the code.

When you apply your changes, Inline Rename still appears so that you can choose a different identifier for the local variable. Applying the second quick action will result in the following code.

```
private static double CalculateCircleArea(object radius)
{
      var v = Convert.ToDouble(radius);
      double circleArea = v * v * Math.PI;
      return circleArea;

}
```

As you can see, the code is now much simpler and more readable. You typically use Introduce Local to separate complex expressions that are nested into other expressions.

Visual Basic Support

For the first time in the history of Visual Studio, refactoring support has been added to Visual Basic. This means that all the refactoring techniques described in this chapter are available to VB, including quick actions for fixing errors and redundant code. As examples, Figure 37 shows how you can extract a method and Figure 38 shows how you can introduce an inline temporary variable.

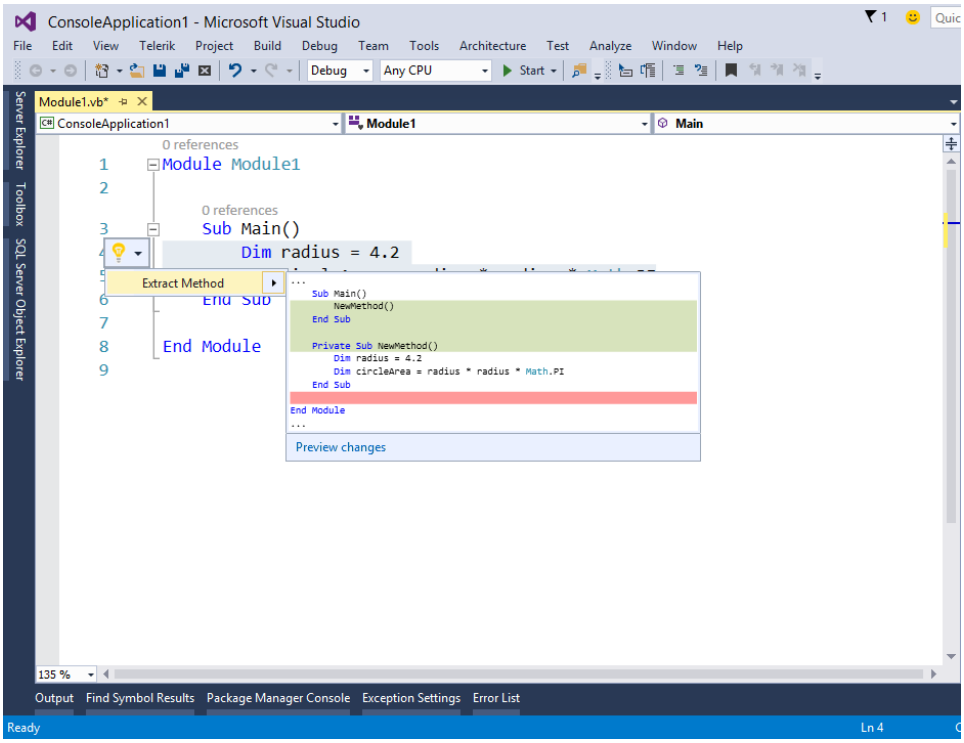

Figure 37: Visual Basic support for refactoring: extracting a method.

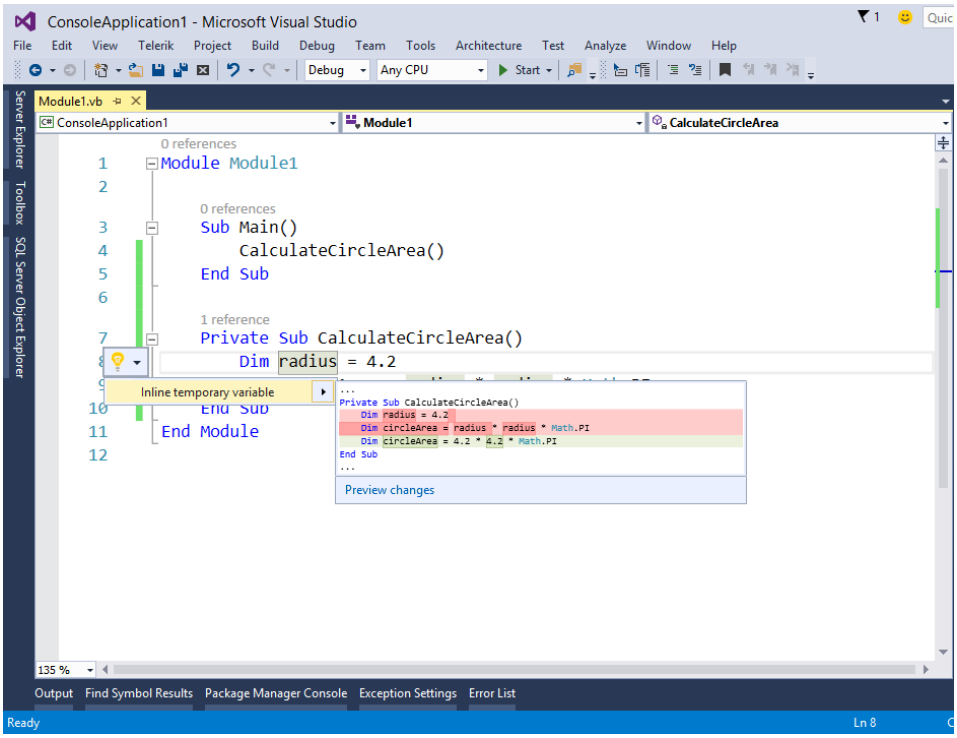

Figure 38: Visual Basic support for refactoring: introducing an inline temporary variable.

Chapter Summary

This chapter introduced some new key features in the code editor. You met colorized tooltips, which show cool previews of collapsed code regions. You learned how you can successfully interact with the code editor on touch screen devices. Most importantly, you learned about the Light Bulbs and Quick Actions, which will help you fix errors as you type and write better code with new and enhanced refactoring options, now available for Visual Basic too.

Chapter 4 XAML Editor Improvements

Many important development platforms are based on the XAML markup language, including Windows Presentation Foundation, Windows Store apps, and Windows Phone apps. XAML therefore plays a key role for thousands of developers. In addition, despite the availability of specialized design tools, writing, editing, and adjusting XAML code manually is a very common task. Because of this, Microsoft is still investing to improve the XAML code editor in Visual Studio; version 2015 brings new goodies that you will definitely love, especially if you need to edit parts of the user interface.

Peek Definition in the XAML Code Editor

Do you remember Peek Definition? I talked about this amazing feature in my book *Visual Studio 2013 Succinctly*, describing how you can show and edit pieces of C# or Visual Basic code inside an interactive popup window which appears in the context of your code. Visual Studio 2015 takes an important step further by bringing Peek Definition to the XAML code editor.

Note: In this chapter, I will mention several concepts related to XAML, such as resource dictionaries, styles, and control templates. I assume you are familiar with those concepts and with XAML in general. If you are not, refer to the MSDN documentation.

This is an important addition, because it makes it easier to edit styles, control templates, and data templates that reside in external resource dictionaries from the context of their usage. To understand how the XAML code editor can help you save time, create a new WPF project. Of course the techniques described in this chapter work fine with other environments, such as Windows Store apps and Windows Phone apps, not just WPF. When the new project is ready, add a new resource dictionary. You already learned in Chapter 2 how to add resource dictionaries to a WPF project, so I will not describe this again. Suppose you want to provide a custom control template for the **`Button`** control. The following code demonstrates how to define a control template that represents buttons with an ellipse, and where each state (normal, mouse over, pressed) is intercepted with a **`Trigger`** and represented by a different gradient background, via **`LinearGradientBrush`** objects.

```
<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                    xmlns:local="clr-namespace:WpfApplication1">

      <Style x:Key="ButtonStyle1" TargetType="Button">
          <Setter Property="Template">
              <Setter.Value>
                  <ControlTemplate TargetType="Button">
```

```
                        <Grid>
                            <Ellipse x:Name="ellipse" Stroke="Black">
                                <Ellipse.Fill>
                                    <LinearGradientBrush
                                     EndPoint="0.5,1" StartPoint="0.5,0">
                                        <GradientStop Color="DarkBlue"
                                         Offset="0"/>
                                        <GradientStop Color="Violet"
                                         Offset="1"/>
                                    </LinearGradientBrush>
                                </Ellipse.Fill>
                            </Ellipse>
                            <ContentPresenter
HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}"

VerticalAlignment="{TemplateBinding VerticalContentAlignment}"

SnapsToDevicePixels="{TemplateBinding SnapsToDevicePixels}"
                                              RecognizesAccessKey="True"/>
                        </Grid>
                        <ControlTemplate.Triggers>
                            <Trigger Property="IsMouseOver" Value="True">
                                <Setter Property="Fill"
                                        TargetName="ellipse">
                                    <Setter.Value>
                                        <LinearGradientBrush
                                         EndPoint="0.5,1"
                                         StartPoint="0.5,0">
                                            <GradientStop Color="Violet"
                                             Offset="0"/>
                                            <GradientStop Color="DarkBlue"
                                             Offset="1"/>
                                        </LinearGradientBrush>
                                    </Setter.Value>
                                </Setter>
                            </Trigger>
                            <Trigger Property="IsPressed" Value="True">
                                <Setter Property="Fill"
                                        TargetName="ellipse">
                                    <Setter.Value>
                                        <LinearGradientBrush
                                         EndPoint="0.5,1"
                                         StartPoint="0.5,0">
                                            <GradientStop Color="#FF290664"
                                             Offset="0"/>
                                            <GradientStop Color="#FF290664"
                                             Offset="1"/>
                                            <GradientStop Color="Violet"
                                             Offset="0.483"/>
```

```
                                        </LinearGradientBrush>
                                    </Setter.Value>
                                </Setter>
                            </Trigger>
                            <Trigger Property="IsEnabled" Value="False"/>
                        </ControlTemplate.Triggers>
                    </ControlTemplate>
                </Setter.Value>
            </Setter>
        </Style>
</ResourceDictionary>
```

Before you can use resources from a dictionary, you must tell the application what dictionaries you will use and where they are located. This is accomplished in the App.xaml (or Application.xaml in other project types) file, more specifically you use a collection called **`ResourceDictionary.MergedDictionaries`** that collects required dictionaries.

```
    <Application.Resources>
        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <ResourceDictionary Source="/Dictionary1.xaml"/>
            </ResourceDictionary.MergedDictionaries>
        </ResourceDictionary>
    </Application.Resources>
```

You can now use resources provided by the resource dictionary in your user interface elements. For example, in the main window of your project you can draw a button and apply the custom **`ButtonStyle1`** like this.

```
    <Grid>
        <Button Width="120" Height="120" Content="Click me!"
                Style="{StaticResource ButtonStyle1}" />

    </Grid>
```

Tip: IntelliSense will help you assign an appropriate resource according to the UI element you are working on. For example, if you have styles for buttons and for text boxes, IntelliSense will show only styles for buttons. This has been introduced in Visual Studio 2013 and is particularly useful when you have dozen styles or templates.

The result of this assignment is immediately visible on the designer, where an elliptical colored button appears. You can now right-click on the **`ButtonStyle1`** identifier and select **Peek Definition** from the popup menu. Figure 39 shows how the code editor appears at this point.

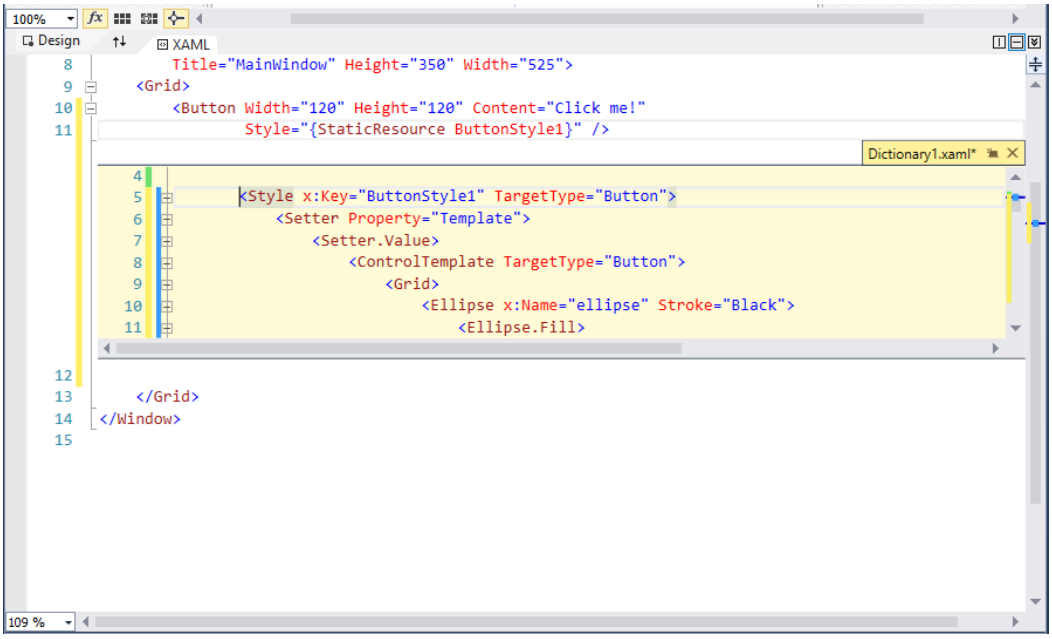

Figure 39: Peek Definition in the XAML code editor.

As you can see, a convenient popup editor appears and allows you to edit the control template within the context of its usage. The popup is interactive, which means that you can change the code directly, without the need for opening an external window that points to the specified file. The designer immediately reflects changes you make in the Peek Definition's window. When you are done, you can simply close the popup the usual way. This feature works in the XAML editor exactly as it does for C# and Visual Basic. You can also invoke Peek Definition against managed code. For example, if you have a `Click` event handler for your button, right-clicking the event handler's name in the XAML editor and selecting Peek Definition will cause Visual Studio to open the peek window on the C#/VB code for the event handler.

Additional Considerations for Peek Definition In XAML

Peek Definition doesn't work with just resource dictionaries; it also works with styles and templates you define elsewhere in your solution. It allows you to quickly edit styles, control templates, data templates, and any other element definable as a resource. This feature becomes very helpful with real world applications where you might have dozen dictionaries and resources, with many of them stored inside external libraries. Without a doubt, it helps to save time because it avoids the need of searching for the resource definition, opening the resource inside an external editor window, making your changes, and then going back to the resource context of usage.

Chapter Summary

Since it's common to edit XAML code manually, Visual Studio 2015 brings Peek Definition into the XAML code editor. This feature enables developers to edit styles, control templates, data templates, and any other kind of resource directly from the context of usage, with edits immediately reflected on the designer surface.

Chapter 5 IDE Customizations: Window Layout

Visual Studio has been offering a huge number of ways to customize the development environment for many years. This includes customizing toolbars, settings, menu commands, the code editor, and other parts of the workspace. After all, as a developer, you spend so much time in Visual Studio and so you must be able to adjust the environment according to your personal preferences and needs. Visual Studio 2015 introduces a new customization called Custom Window Layout, which makes it easier to organize your workspace.

Custom Window Layout

Visual Studio 2015, as well as its predecessors, has many tool windows that can be arranged on the workspace and even be docked to the four sides of the development environment. Solution Explorer, Toolbox, Error List, and Properties are all examples of related tool windows. You typically organize your workspace according to the kind of project you are working on. However, when you switch to another project type, you will probably use different tool windows and reorganize your workspace. Switching between different project types, and consequently to different window layouts, would then require manually rearranging all the necessary tool windows. In addition, if you work with Visual Studio on multiple machines, you have to arrange your window layout on each installation. To make your life easier, Visual Studio 2015 introduces a new way of customizing your workspace, called Custom Window Layout. This new feature allows saving, managing, reapplying, and roaming your window layouts with convenient tools.

Saving Window Layouts

To understand how Custom Window Layout works, first suppose you have specific tool windows opened when working on a WPF project for Windows. Figure 40 provides an example where you can see the Toolbox, Document Outline, Solution Explorer, and Properties tool windows.

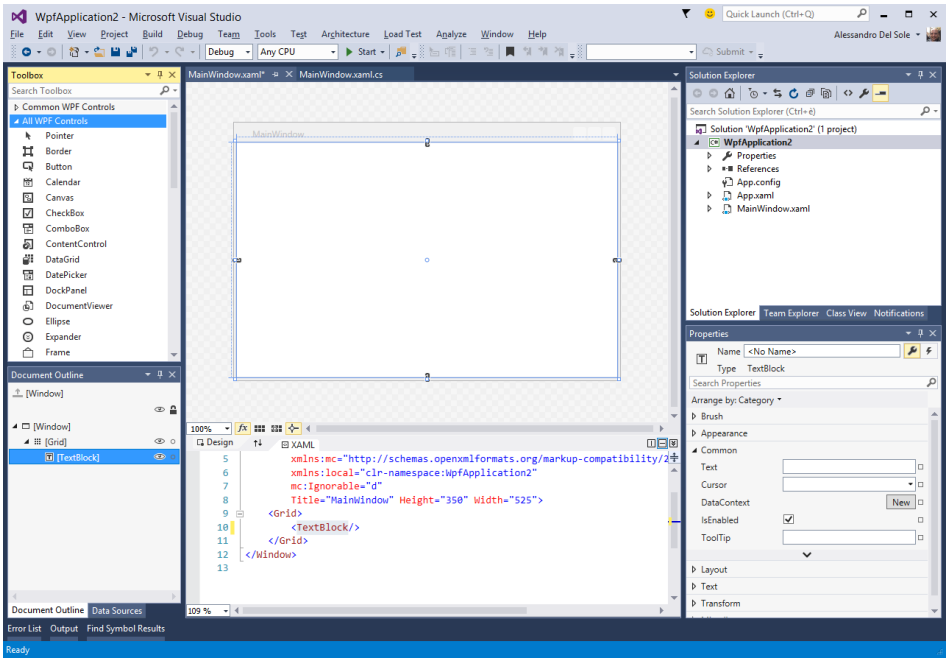

Figure 40: A sample window layout for WPF projects.

Suppose you now need to change project type, switching to a web project for the Azure platform. You will need to open completely different tool windows, which means losing the current layout now and for future work on the project. Visual Studio 2015 now provides an easy opportunity to save and restore window layouts. The **Window** menu offers a new command called **Save Window Layout**. This will open the **Save Window Dialog**, where you can enter a name for the current layout, as demonstrated in Figure 41.

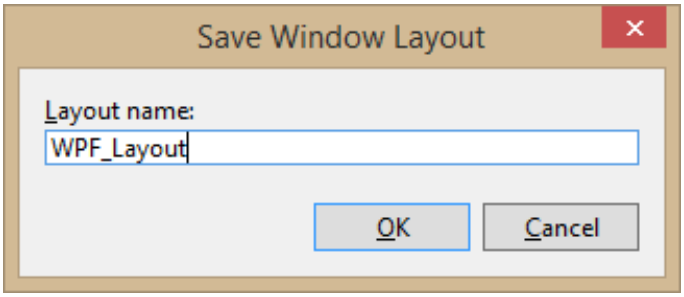

Figure 41: Supplying a name for the window layout.

At this point, your window layout will be saved for later reuse.

Tip: When you sign in with your Microsoft Account, custom window layouts are synchronized across all your Visual Studio 2015 installations. This means that you can run Visual Studio 2015 on a different machine and find all your saved layouts.

Applying Window Layouts

Imagine that, after working on a WPF project, you need to switch to a project for the Azure platform, for example a Mobile Service project. You will definitely use different tool windows such as Server Explorer, SQL Server Object Explorer, and HDInsight Task List. Figure 42 shows an example of window layout for this kind of project, but your effective preferences might be different.

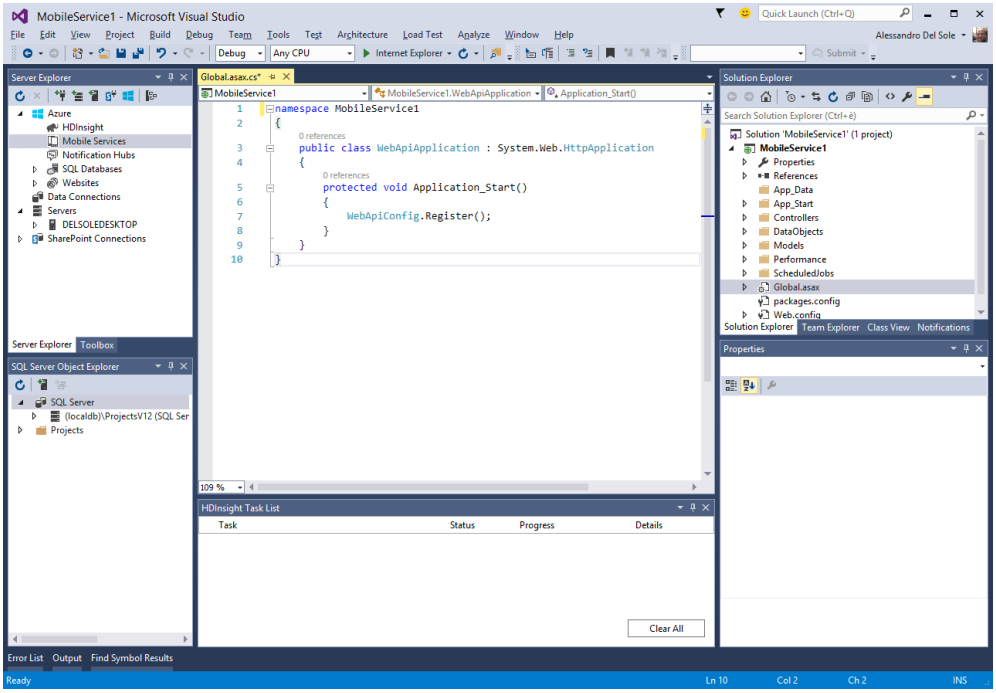

Figure 42: Using another window layout for a different project type.

You can save the current window layout as you saw in the previous paragraph, but the most important consideration comes when you return to working on your WPF project. For a better understanding, close the Azure project and open a new or existing WPF project. The **Window** menu provides a command called **Apply Window Layout**, which allows selecting a different window layout among a list of saved layouts, as Figure 43 demonstrates.

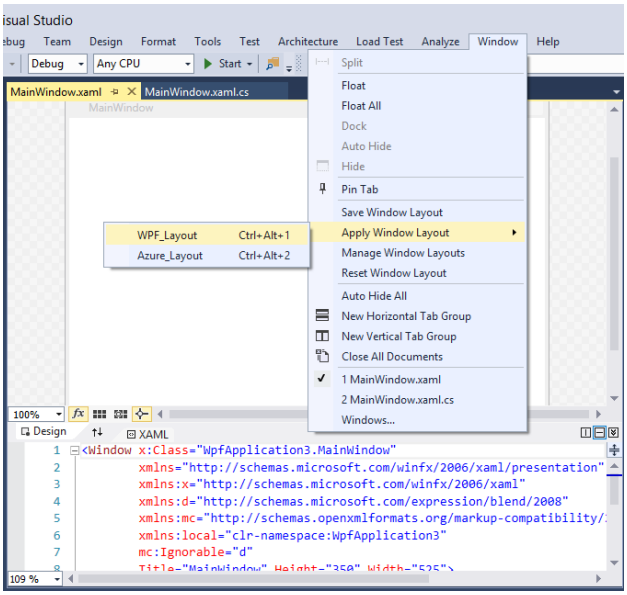

Figure 43: Applying window layouts.

When you choose one from the list, Visual Studio 2015 will ask for a confirmation before applying a different layout. At this point, you will get your original workspace restored, showing all the necessary tool windows for working with WPF.

Note: Visual Studio 2015 will be able to completely restore a window layout only if the current project type supports all tool windows in the layout. For instance in the current example, if you try to apply the WPF layout when the project for Azure is still open, the Document Outline tool window will not be restored, because it is not supported by a project for Azure. More generally, it is not supported by projects different than XAML-based ones. This is the reason why you should first open a specific project and then apply a layout.

Managing Custom Window Layouts

Visual Studio 2015 also provides an easy way to manage your custom window layouts. The Manage Window Layouts command in the Window menu opens the Manage Window Layouts dialog (see Figure 44), where you will be able to delete layouts, rename layouts, and move layouts on the list.

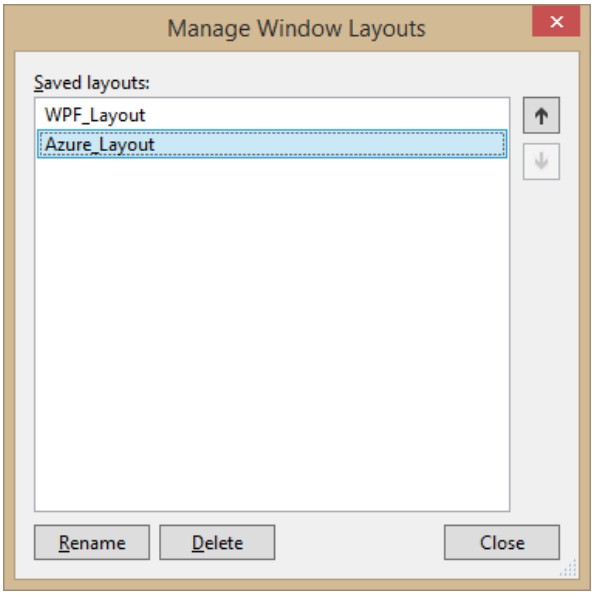

Figure 44: Managing window layouts.

Restoring The Default Layout

Visual Studio 2015 ships with a default window layout. Restore the default layout by using the **Reset Window Layout** command in the **Window** menu. This option is useful when you want to completely clean your environment preferences and reapply saved layouts.

Chapter Summary

Custom Window Layout in Visual Studio 2015 provides a new way of customizing the IDE by saving, managing, and restoring tool window arrangements in the workspace for different projects. This feature helps you keep a very well organized development environment.

Chapter 6 Error List Revisited and Debugging Improvements

Discovering and fixing errors is just as important as writing code. Visual Studio has always offered sophisticated debugging tools; the Error List window is a great companion for examining errors when writing or compiling code. Diagnosing and debugging errors is an area that Microsoft improves tooling for with every new version of the IDE. This chapter describes the Error List window in its new look, and introduces improved debugging and diagnostic tools that will help you solve problems in a more productive way.

The Error List Revisited

The Error List tool window shows the list of compile-time errors that Visual Studio detects while writing code or when you compile your solution. In Visual Studio 2015, the Error List has been revisited and now provides additional useful features.

Tip: The Error List window should open up automatically when compilers encounter any error. However, you can open it up anytime by selecting View > Error List or by pressing CTRL+\, E.

The limit of 100 messages has been removed and now the window shows the full list of errors. To understand the other new features, first consider the code represented in Figure 45 and Figure 46, which contains some errors in different code files. The reason why errors are exemplified in two different files will be clear shortly.

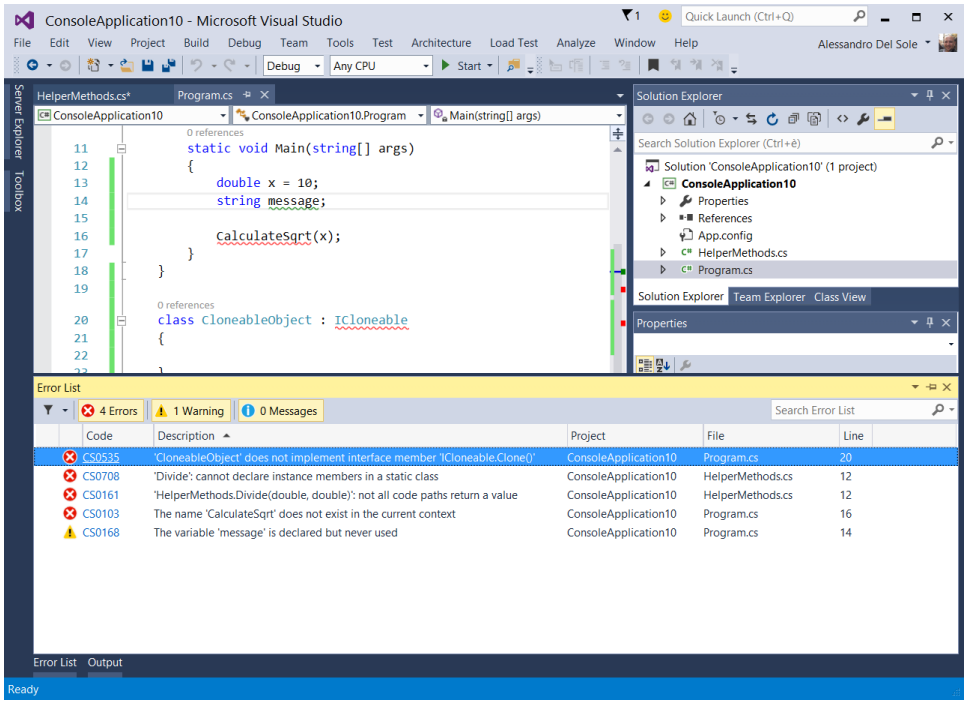

Figure 45: Simulating some errors—Part 1.

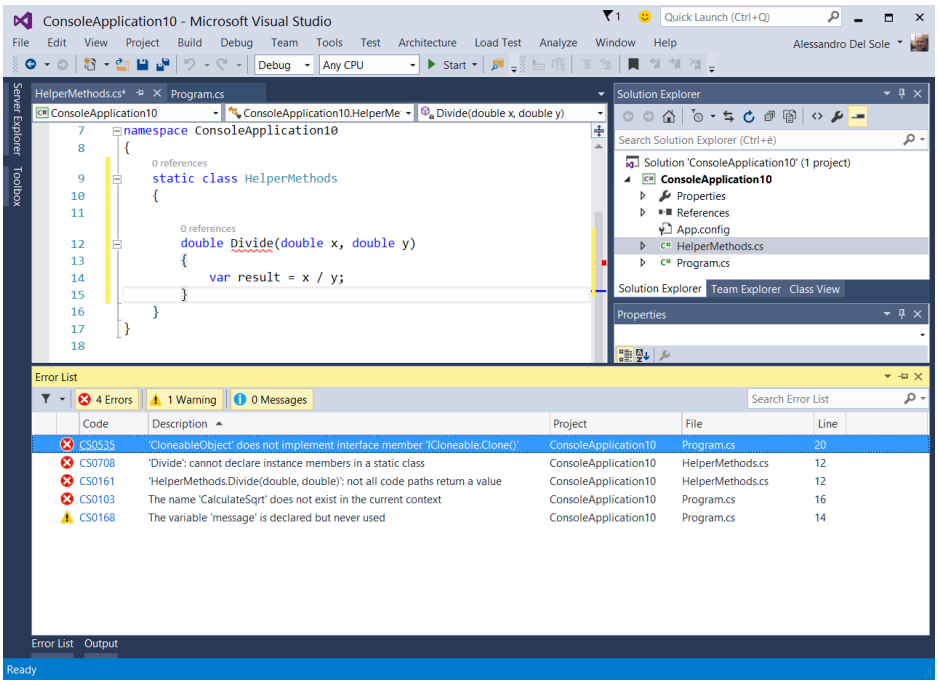

Figure 46: Simulating some errors—Part 2.

Focus on the Error List; Figure 47 shows a magnified view at this point.

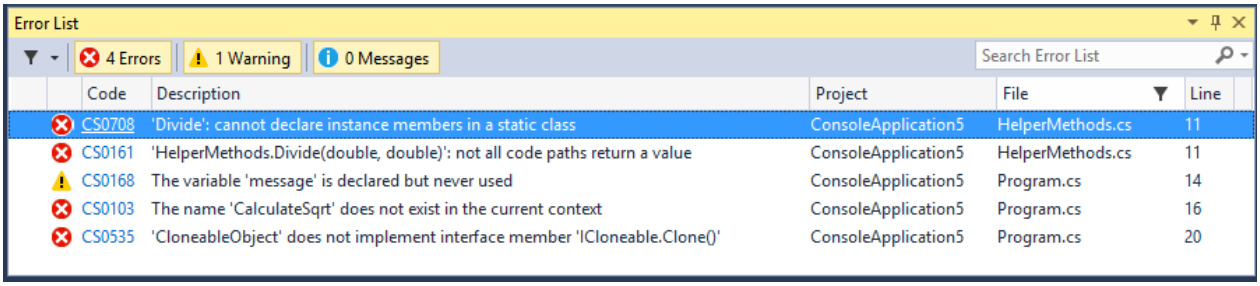

Figure 47: A more specific view of the Error List window.

The first thing you will notice is an error code for each message. Such an error code represents a specific compiler error; it starts with **CS** if it is raised by the C# compiler or with **BC** if it is raised by the Visual Basic compiler.

Note: Additional error codes can be shown if you install (or create) other analyzers, like the Code Analysis for Azure described in Chapter 8.

Error codes are presented in the form of hyperlinks, so if you click any error code, Visual Studio 2015 will open up your default web browser and will search the Internet with Bing to find information about both the error code and the error message. Another new feature is filters. You can basically filter the error list by the following columns: code, project, and file. For instance, suppose you want to view errors in some specific files only; click the filter symbol on the **File** column and select the desired code files, as demonstrated in Figure 48.

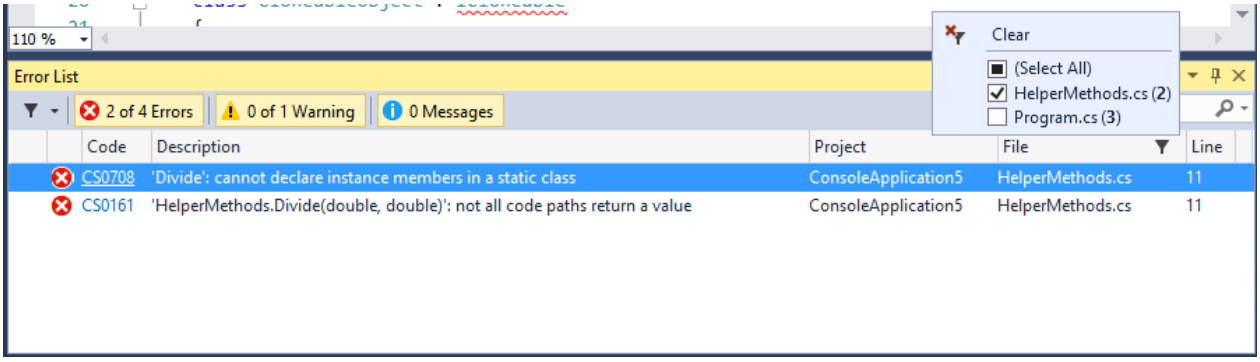

Figure 48: Filtering errors by code file.

Similarly, you could filter the error list by error code or by project; the latter can be very useful with large solutions. The filter symbol on the upper-left corner of the Error List window provides additional ways of filtering the list, as shown in Figure 49. With this option, you can browse errors for the current document, for all open documents, or for the current project, which is also the default filter.

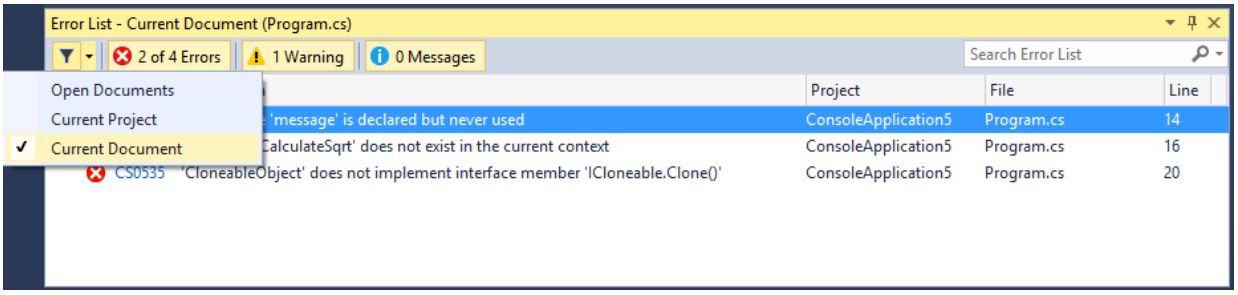

Figure 49: Viewing only errors in the current document.

You can also filter the list by entering phrases in the search box, but that was introduced in Visual Studio 2013.

Debugging Improvements

Debugging is more efficient when you have the right tools. Microsoft has always paid particular attention to offering sophisticated and advanced debugging tools, and Visual Studio 2015 is no exception. This chapter describes what's new in the debugging experience with Visual Studio 2015.

Support For Lambda Expressions Debugging

Visual Studio 2015 introduces support for debugging lambda expressions. The community asked for this feature so loudly that Microsoft worked hard to make it available in the new version of the IDE. This feature allows the use of the **Watch** and **Immediate** windows to evaluate lambda expressions; it also allows you to step into the code of a lambda expression with proper debugging commands such as Step Into (F11). To understand how it works, consider the following code.

```
class Program
{
    static void Main(string[] args)
    {
        List<string> Names = new List<string>();
        Names.Add("Alessandro");
        Names.Add("Renato");
        Names.Add("Anthony");
        Names.Add("Diego");

        Debugger.Break();
    }
}
```

The code example is a simple collection of strings containing some names. If you run the code, the execution breaks at **`Debugger.Break();`**. Right-click the **`Names`** variable and select **Add Watch**. At this point, the **Watch** debug window is opened and an instance of the selected object is available for evaluation. The good news is that you can type a lambda expression over the object instance and evaluate it to discover if it works as expected. Figure 50 shows an example where a lambda expression extracts all names beginning with “A.”

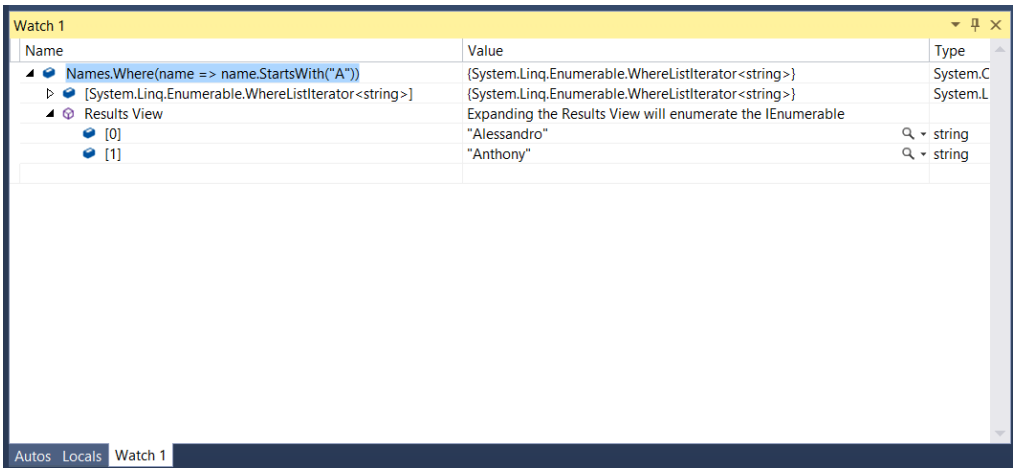

Figure 50: Debugging a lambda expression.

IntelliSense will also work as you type, which makes writing a lambda inside the Watch window a really easy task. Not limited to this, you can also write LINQ queries against supported data sources, like .NET objects, Entity Data Models, LINQ to SQL, and so on. Figure 51 provides an example based on the previous code.

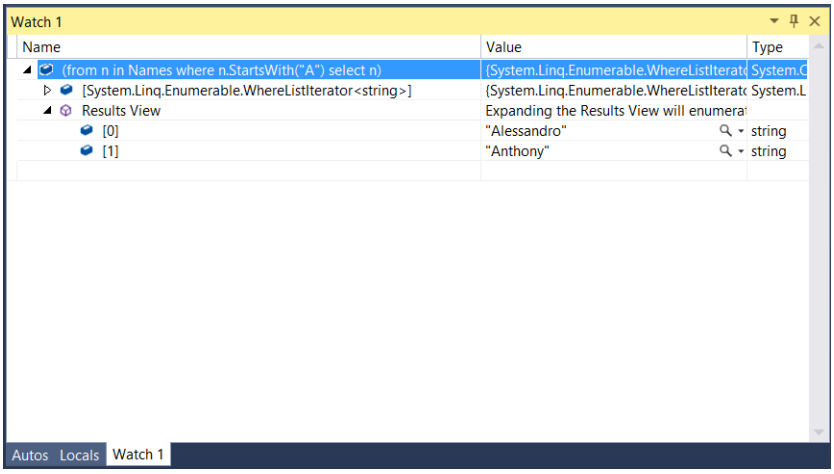

Figure 51: Debugging a LINQ query.

Similarly, you can evaluate expressions in the **Immediate Window**, both lambdas and LINQ queries, as shown in Figure 52.

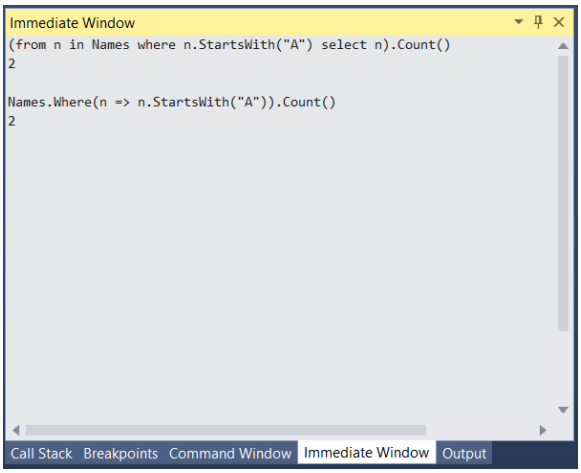

Figure 52: Evaluating lambdas and LINQ queries in the Immediate Window.

Most developers will love this new feature because it saves time when you need to debug these kinds of expressions.

Visualizing Performance Tooltips: PerfTips

Visual Studio 2015 introduces a new feature called PerfTips (Performance Tooltips), which shows how long it takes to run a specific piece of code while debugging. To understand how PerfTips work, you need a better sample. To do this, create a WPF application that loads an RSS feed from the Internet and then measure how long it takes to execute specific code.

Creating a Sample Application

In Visual Studio 2015, create a new WPF application with either C# or Visual Basic. The goal of the sample application is to download the RSS feed from the official Visual Studio Blog and present some information in the user interface. The first thing you want to do is add a class that represents a single item in the RSS feed; we call such a class **`Item`**.

C# Code Example

```
    public class Item
    {
        public string Title { get; set; }
        public Uri Link { get; set; }
        public string Author { get; set; }
        public System.DateTime PubDate { get; set; }
    }
```

Visual Basic Code Example

```
Public Class Item
    Public Property Title As String
```

```
    Public Property Link As Uri
    Public Property Author As String
    Public Property PubDate As Date
End Class
```

As you can see, this is a simple class that stores information like the blog post author, title, link, and publish date. Next is to write the XAML code for the user interface, which provides a button for canceling the operation and a **ListBox** control, whose **ItemTemplate**'s **DataTemplate** arranges some **TextBlock** controls vertically inside a **StackPanel** container. **TextBlock** controls are bound to properties of the **Item** class.

```
<Window x:Class="OpenFeed.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Loaded="Window_Loaded"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
        xmlns:local="clr-namespace:OpenFeed"
        mc:Ignorable="d"
        Title="MainWindow" Height="350" Width="525">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="40"/>
            <RowDefinition/>
        </Grid.RowDefinitions>
        <Button Width="120" Height="30" Name="CancelButton"
                Content="Cancel" Click="CancelButton_Click"/>
        <ListBox Name="ContentBox" Grid.Row="1"
                 ItemsSource="{Binding}">
            <ListBox.ItemTemplate>
                <DataTemplate>
                    <StackPanel>
                        <TextBlock Text="{Binding Title}"
                         TextWrapping="Wrap" FontSize="16"
                                   Grid.Row="1" Tag="{Binding Link}"
                                   Foreground="Blue"
                                   FontWeight="Bold"

                          MouseLeftButtonUp="TextBlock_MouseLeftButtonUp"/>
                        <TextBlock Text="{Binding PubDate,
                         StringFormat=d}" Foreground="Red"
                                   Grid.Row="2"/>
                        <TextBlock Text="{Binding Author}" Grid.Row="3"/>
                    </StackPanel>
                </DataTemplate>
            </ListBox.ItemTemplate>
```

```
        </ListBox>
    </Grid>
</Window>
```

Now you need some code that downloads the RSS feed content, parses the corresponding XML, and populates the user interface with the appropriate information (see comments).

Note: You need to add a reference to System.Net.Http.dll in order to use the HttpClient class.

C# Code Example

```
        //The RSS feed URL.
        private const string FeedUri =
            "http://blogs.msdn.com/b/visualstudio/rss.aspx";

        //Cancellation token required to cancel the operation.
        private CancellationTokenSource cancellationToken;

        private void CancelButton_Click(object sender, RoutedEventArgs e)
        {
            if (this.cancellationToken != null)
            {
                //Cancellation request.
                this.cancellationToken.Cancel();
            }
        }

        //Open the selected post in the default browser.
        private void TextBlock_MouseLeftButtonUp(object sender,
            MouseButtonEventArgs e)
        {
            var content = (TextBlock)sender;
            var link = (Uri)content.Tag;
            Process.Start(link.AbsoluteUri);
        }

        private async void Window_Loaded(object sender,
            RoutedEventArgs e)
        {
            //Create an instance of the cancellation token.
            this.cancellationToken = new CancellationTokenSource();

            //Load and assign data to the Window's DataContext.
            this.DataContext = await
QueryRssAsync(this.cancellationToken.Token);
```

```
        }

        private async Task<IEnumerable<Item>>
QueryRssAsync(CancellationToken token)
        {
            try
            {
                //Access via HTTP to the feed and download data as a
string.
                var client = new HttpClient();
                var data = await client.GetAsync(new Uri(FeedUri), token);

                var actualData = await data.Content.ReadAsStringAsync();

                //Execute a LINQ to XML query against the feed.
                var doc = XDocument.Parse(actualData);
                var dc =
XNamespace.Get("http://purl.org/dc/elements/1.1/");

                var query = (from entry in doc.Descendants("item")
                             select new Item
                             {
                                 Title = entry.Element("title").Value,
                                 Link = new
Uri(entry.Element("link").Value),
                                 Author = entry.Element(dc +
"creator").Value,
                                 PubDate =
DateTime.Parse(entry.Element("pubDate").Value,

System.Globalization.CultureInfo.InvariantCulture)
                             });

                return query;
            }
            catch (OperationCanceledException)
            {
                MessageBox.Show("Operation canceled by user");
                return null;
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
                return null;
            }
        }
```

Visual Basic Code Example

```
Imports System.Net.Http
Imports <xmlns:dc="http://purl.org/dc/elements/1.1/">
Imports System.Threading

…



    Private Const FeedUri As String = _
        "http://blogs.msdn.com/b/visualstudio/rss.aspx "

    'Cancellation token required to cancel the operation.
    Private cancellationToken As CancellationTokenSource

    Private Async Function QueryRssAsync(token As CancellationToken) As  _
        Task(Of IEnumerable(Of Item))
        Try
            'Access via HTTP to the feed and download data as a string.
            Dim client As New HttpClient

            Dim data = Await client.GetAsync(New Uri(FeedUri), token)

            Dim actualData = Await data.Content.ReadAsStringAsync

            'Execute a LINQ to XML query against the feed.
            Dim doc = XDocument.Parse(actualData)

            Dim query = From feed In doc...<item>
                        Select New Item With {
                              .Title = feed.<title>.Value,
                              .Author = feed.<dc:creator>.Value,
                              .Link = New Uri(feed.<link>.Value),
                              .PubDate = Date.Parse(feed.<pubDate>.Value,
                               Globalization.CultureInfo.InvariantCulture)}

            Return query

        Catch ex As OperationCanceledException
            MessageBox.Show("Operation canceled by user.")
            Return Nothing
        Catch ex As Exception
            MessageBox.Show(ex.Message)
            Return Nothing
        End Try
    End Function

    Private Async Sub MainWindow_Loaded(sender As Object,
```

```
        e As RoutedEventArgs) Handles Me.Loaded
        'Create an instance of the cancellation token.
        Me.cancellationToken = New CancellationTokenSource

        'Load and assign data to the Window's DataContext.
        Me.DataContext = Await QueryRssAsync(Me.cancellationToken.Token)
    End Sub

    Private Sub CancelButton_Click(sender As Object, e As RoutedEventArgs)
        If Me.cancellationToken IsNot Nothing Then
            'Cancellation request.
            Me.cancellationToken.Cancel()
        End If
    End Sub

    'Open the selected post in the default browser.
    Private Sub TextBlock_MouseLeftButtonUp(sender As Object,
        e As MouseButtonEventArgs)
        Dim content = CType(sender, TextBlock)
        Dim link = CType(content.Tag, Uri)

        Process.Start(link.AbsoluteUri)
    End Sub
```

Figure 53 shows an example of how the data is retrieved and presented in the UI.

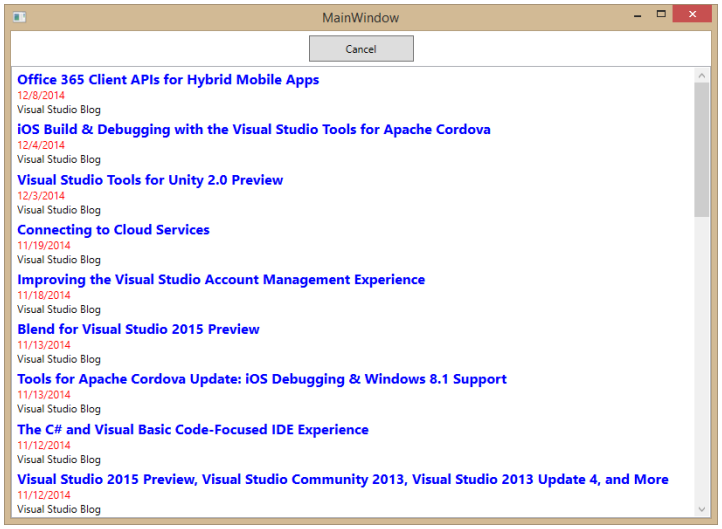

Figure 53: Downloading and presenting the RSS feed contents.

Debugging With PerfTips

To understand PerfTips, place three breakpoints on the following (see Figure 54):

- The instance declaration of the **`HttpClient`** class.
- The line that invokes **`XDocument.Parse`**.
- The line that returns the query result.

Note: For the sake of simplicity, from now on all figures will be based on the C# version of the sample application. However, the same concepts, steps, and results apply to Visual Basic as well.

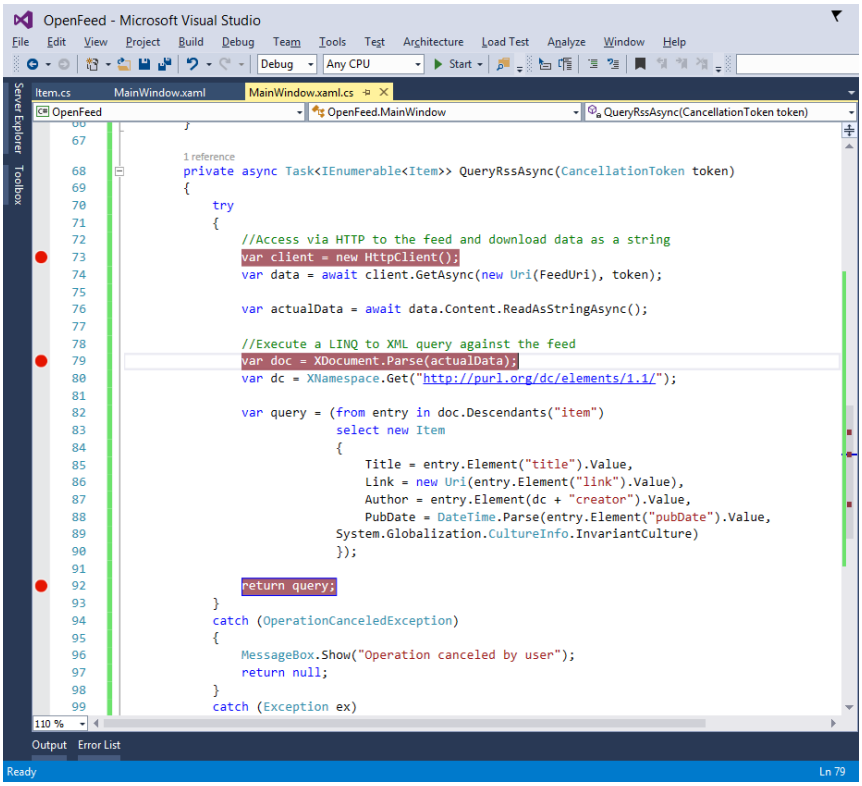

Figure 54: Adding some breakpoints.

Press **F5** to start debugging. The debugger will break before accessing the RSS feed. With PerfTips you can now measure how long it takes between downloading data from the feed and before the application starts analyzing downloaded data. Press **F5** again; when the debugger encounters the second breakpoint, it breaks the execution and shows a small piece of information regarding the elapsed time in milliseconds. If you hover over that information, a more detailed tooltip appears. Figure 55 shows both features.

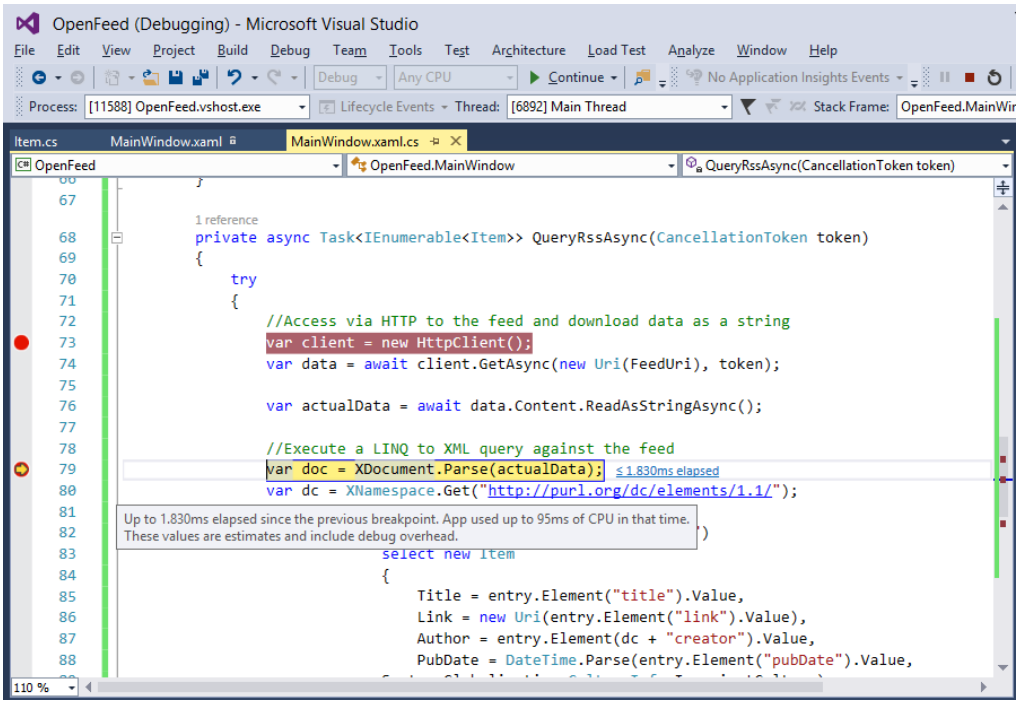

Figure 55: Measuring time of code execution with PerfTips.

If you now continue the execution by pressing **F5**, you will be able to measure the code execution until the next breakpoint (see Figure 56). In this particular case, you can see how long it took to execute a LINQ query against the downloaded feed data.

Note: Remember that the elapsed time also includes some debugger overhead. This is because the debugger execution affects performances in some way, so the elapsed time is always a little bit higher than the real execution time.

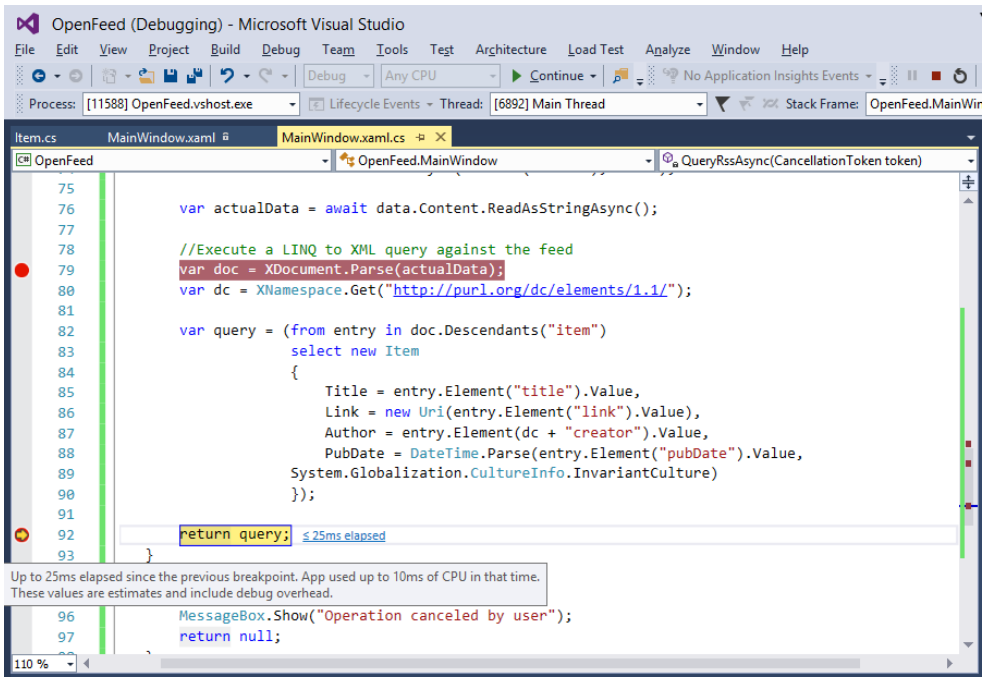

Figure 56: Measuring another piece of code with PerfTips.

You are not limited to use PerfTips with breakpoints. In fact, you can get performance information with other debugging options, such as executing code line by line with **Step Into** by pressing **F11**, or when stepping through the code with **Step Over** (F10), **Step Out** (Shift + F11), or **Run to Cursor** (CTRL+F10).

Diagnostic Tools

Visual Studio 2015 Enterprise edition includes a new tool called Diagnostic Tools, which allows analyzing where and how your application consumes resources. Diagnostic Tools is available as a tool window that automatically shows up when you start debugging an application. Among the others, you can analyze CPU utilization, memory usage, and debugger events. Figure 57 shows an example of this.

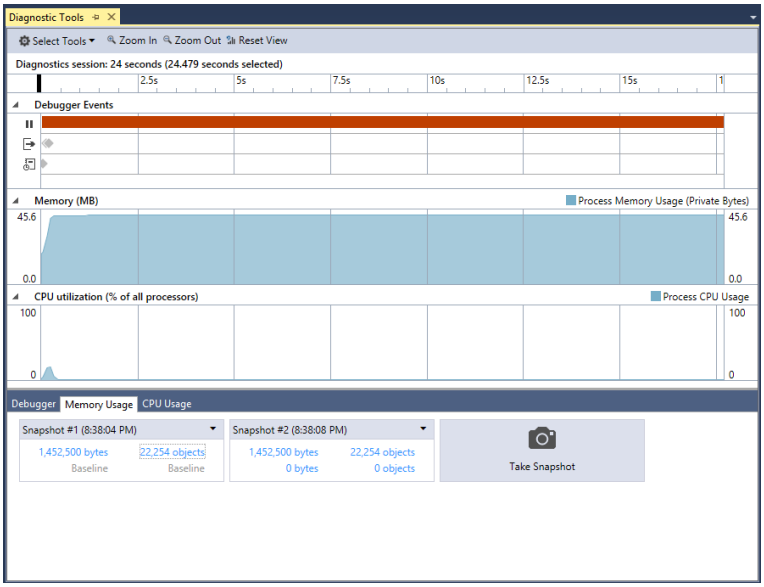

Figure 57: Analyzing application performances with Diagnostic Tools.

The Diagnostic Tools window has a timeline that is populated with information as the application's execution goes on. It shows debugger events such as exceptions and breakpoints, but it also integrates with IntelliTrace events. You can also perform advanced memory analysis by selecting the **Memory Usage** tab (see the bottom of Figure 57) and taking memory snapshots. When you get some, you can click on the total number of objects instantiated at a certain time to see the full list of objects and how many instances are active for each object. You can disable (or enable) the Diagnostic Tools window at any time by selecting **Tools > Options > Debugging > Diagnostic Tools**. Additionally, you can manually open Diagnostic Tools via **Debug > Show Diagnostic Tools**. At the moment of writing this chapter, Diagnostic Tools does not support ASP.NET 5 Core projects, JavaScript 64-bit Windows Store apps, and remote debugging.

Managing Breakpoints with Breakpoint Settings

When placing breakpoints in the preview sample applications, you probably noticed a small toolbar appearing near the red glyph. This small toolbar allows access to breakpoint settings with a completely new experience, as shown in Figure 58.

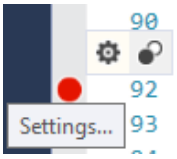

Figure 58: The new toolbar for breakpoint settings.

This toolbar has a button for disabling breakpoints but, most importantly, it offers another button for quickly customizing breakpoints. Breakpoints settings now appear as a non-modal, peek window directly inside the code editor, as represented in Figure 59.

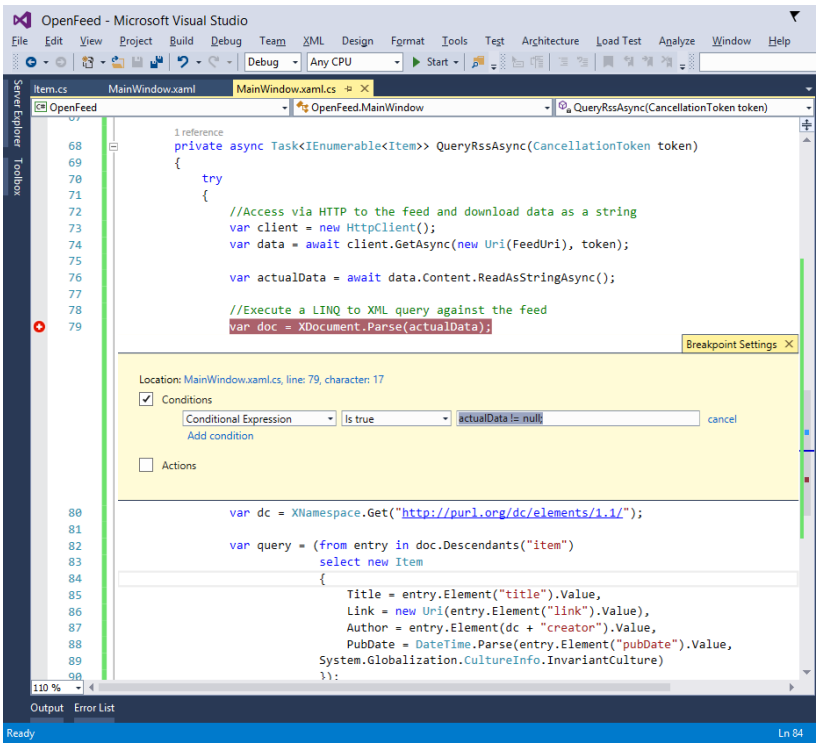

Figure 59: Breakpoint settings now appear as a peek window.

The settings you can edit are largely the same, but what Visual Studio 2015 changes is the way you access and edit settings, which is directly inside the code editor; this is a new, non-blocking approach that allows you to write code with breakpoint settings open. You can still set **Conditions** and **Actions**. Conditions control when the breakpoint must be hit; the default condition is that the breakpoint is hit when the code execution reaches its location. Actions control what the debugger should do when all conditions are satisfied. The default action is that the debugger stops the code execution. For example, you might want to stop the execution of the sample application only when the result of downloading data from the RSS feed is null and, if it happens, you might want to send to the output window the call stack. You can use a condition like the one shown in Figure 60 where the **`actualData == null`** expression must be equal to **`true`**. It is worth mentioning that you will have IntelliSense available when writing your conditions. You can set an action invoking the **`$CALLSTACK`** function which will print the entire call stack into the Output window.

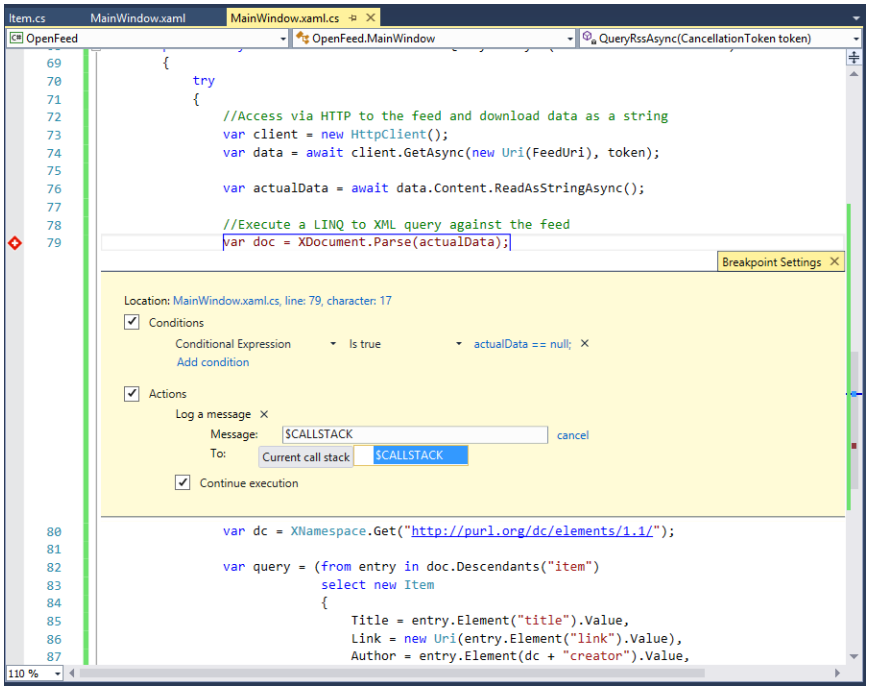

Figure 60: Specifying conditions and actions.

Tip: When providing actions, a specialized IntelliSense will help you choose among available functions and a tooltip will describe each of them.

Concerning conditions, you can select one among Conditional Expression, Hit Count, and Filter. As the name implies, a Conditional Expression will make the breakpoint to be hit only when the specified expression evaluates to True or When Changed. If you instead have iterations (e.g. **for** loops) and you want a breakpoint to be hit only at the specified iteration, you can use Hit Count and specify that the iteration number must be **=**, **Multiple of**, or **>=** than the specified value. This is useful when you discover that an error occurs only from a certain point in the application execution. You can instead use Filter to specify devices, processes, or threads that a breakpoint executes on.

Note: Breakpoint settings are not different from previous editions. The difference is in the way you access and edit settings. In this book, we focus on what's new in the developer experience, so you might want to check the MSDN documentation for more details about available settings and conditions.

Conditions and actions can either work together or separately. Without a doubt, the new experience is awesome because you can easily manage breakpoint settings without ever losing focus on your code.

Chapter Summary

In this chapter you learned about how the Error List tool window has been revisited with error codes, hyperlinks, and advanced filtering options. Then you saw how many investments have been made to the debugging tools, including the option of debugging lambda expressions and LINQ queries in Watch and Immediate windows; you also learned how you can measure code performances with PerfTips and Diagnostic Tools, and how to control breakpoint settings with the new non-blocking, non-modal, and peek window-based experience. All these important features will increase your productivity and will definitely help you write better and more performant code.

Chapter 7 Managing NuGet Packages

NuGet is the package manager for Microsoft Visual Studio. With NuGet, you can easily search for libraries that your application might need. The IDE tooling for NuGet automatically downloads packages to your solution, adds references, and resolves all the required dependencies. NuGet is also a huge repository of libraries that you can consume, but you can also publish your own libraries to NuGet that other developers can use in their projects. NuGet has been around for a few years, starting as a stand-alone project and then becoming a fundamental part of Visual Studio 2012. For Microsoft, NuGet is so important that many libraries are available only via NuGet, such as the Entity Framework or the ASP.NET 5 composable stack. More precisely, in .NET Core 5 all required libraries come from NuGet and the IDE no longer adds references to any library from the .NET Framework. Visual Studio 2015 enhances support for NuGet, making it an even better tool.

A New NuGet

Visual Studio 2015 offers a revisited NuGet experience. Now the NuGet tools appear inside a dockable tool window with a non-modal and non-blocking approach. To understand the new NuGet, suppose you have a WPF project and that you want to install the Entity Framework libraries. In Visual Studio 2015, you access NuGet tools by selecting **Tools** > **NuGet Package Manager** > **Manage NuGet Packages for Solution** or by right-clicking the project name in **Solution Explorer** and selecting **Manage NuGet Packages**. The NuGet window appears as represented in Figure 61.

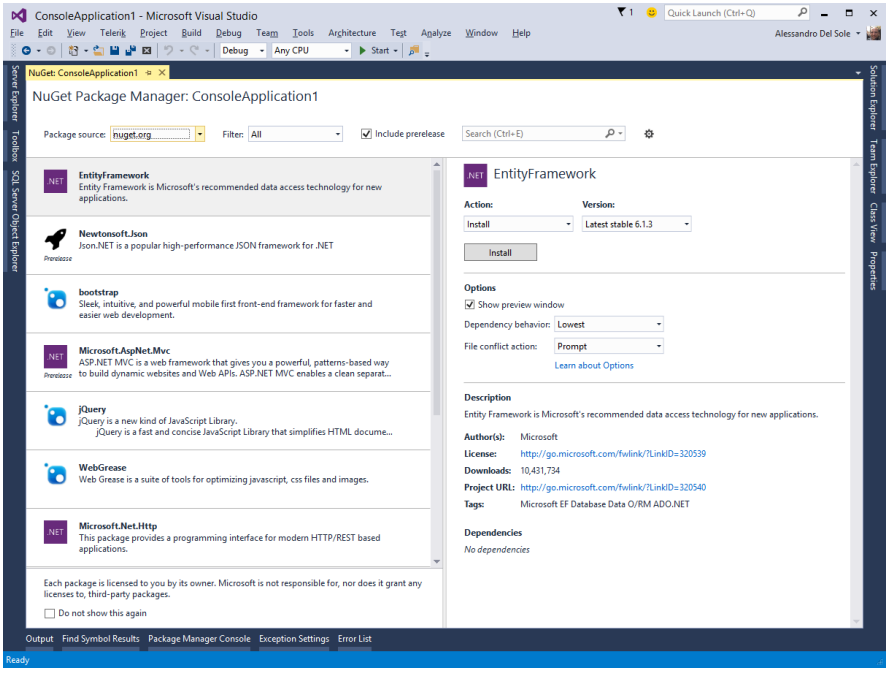

Figure 61: The new NuGet window.

As you can see, you no longer have a modal dialog; instead, you are free to switch between tool windows and code editor windows. The left side of the window shows the list of available packages, based on the **Package source**, **Filter**, and **Include Prerelease** boxes. The **Package source** box allows selecting the online repository through which to search for packages. The default value is **nuget.org**, while other available options are **Microsoft and .NET** and **preview.nuget.org**; the later option specifically searches for preview releases. The **Filter** box filters the list of packages in case you want to see **All** packages (default value), **Installed** packages, and packages with an **Update available**. You can easily search packages by keyword via the **Search** text box. On the right side of the window, you get detailed information about the selected package (see Figure 61). You get information like the package description, author, license, and downloads number, but also you have an option of choosing among multiple versions. If your package has dependencies on other libraries, then these are also listed at the bottom of the window. It is also interesting how you can specify what action to take in case of file conflict (**Prompt**, **Ignore all**, **Overwrite all**). When ready, simply click **Install**. Accept any license agreement required by the package owner and wait for a few seconds in order to get the packages installed. If the installation succeeds, Visual Studio shows a green check box near the package name, as shown in Figure 62.

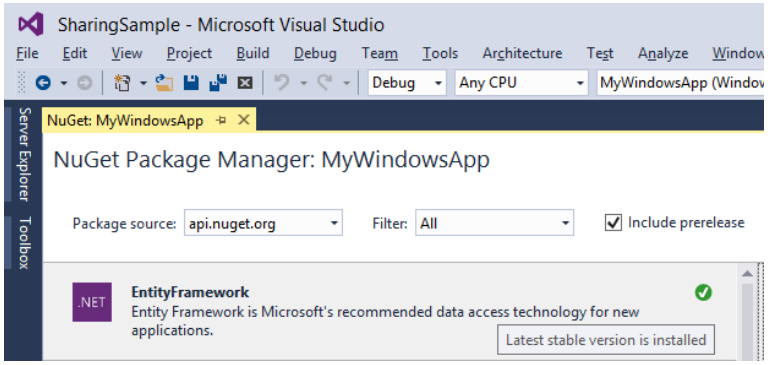

Figure 62: Package installation succeeded.

Tip: Except for the license agreement acceptance, which is a modal dialog, the NuGet window never blocks the UI, so you can switch to other windows, including the code editor, while installing packages.

You will be able to see the downloaded packages in the **References** node for your project in Solution Explorer. A nice addition to NuGet support in Visual Studio 2015 is the option to see a dependency graph for packages. Select **Tools** > **NuGet Package Manager** > **Package Visualizer** to get a visual representation for installed packages and their dependencies. Figure 63 shows an example in which you have a main project with two packages installed and one package has dependencies on the other one.

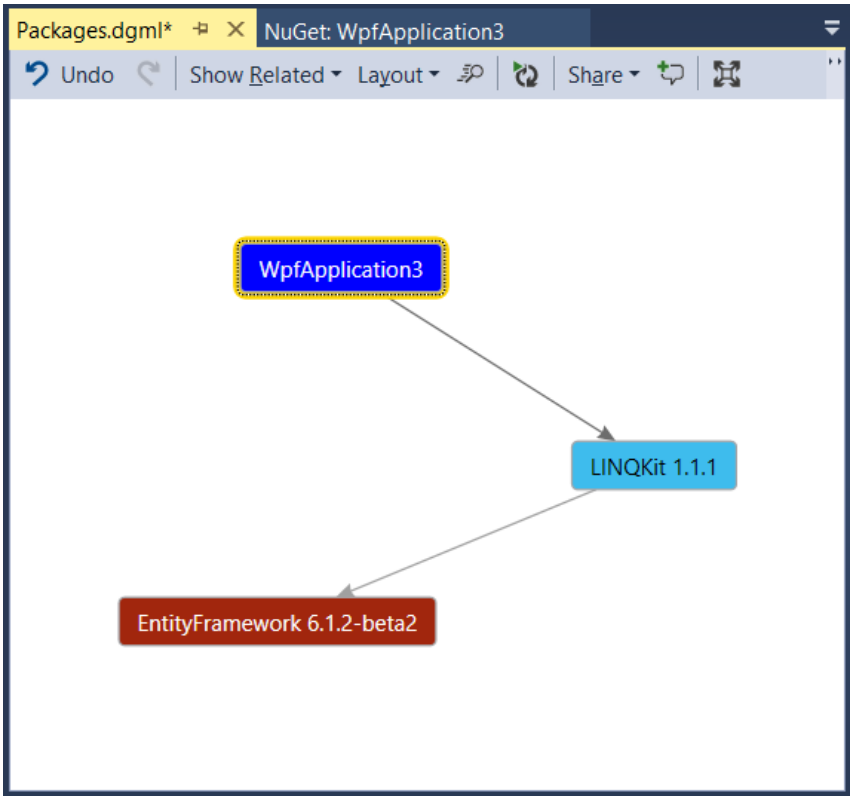

Figure 63: Package Visualizer.

The dependency graph is based on the Code Map tooling, which was introduced into the IDE in Visual Studio 2012 and enhanced in Visual Studio 2013. It inherits all of its features, such as exporting the graph to image, grouping items, choosing different colors, adding comments, and so on.

Setting NuGet Preferences

The new NuGet window has as shortcut near the Search box (see Figure 61) to invoke the **Options** dialog for NuGet. Of course, you can still use **Tools** > **NuGet Package Manager** > **Package Manager Settings** to open NuGet settings. Without a doubt, the most interesting option is that you can specify additional package sources, which also means that you can create your NuGet-based repository. Figure 64 shows the **Package Sources** options.

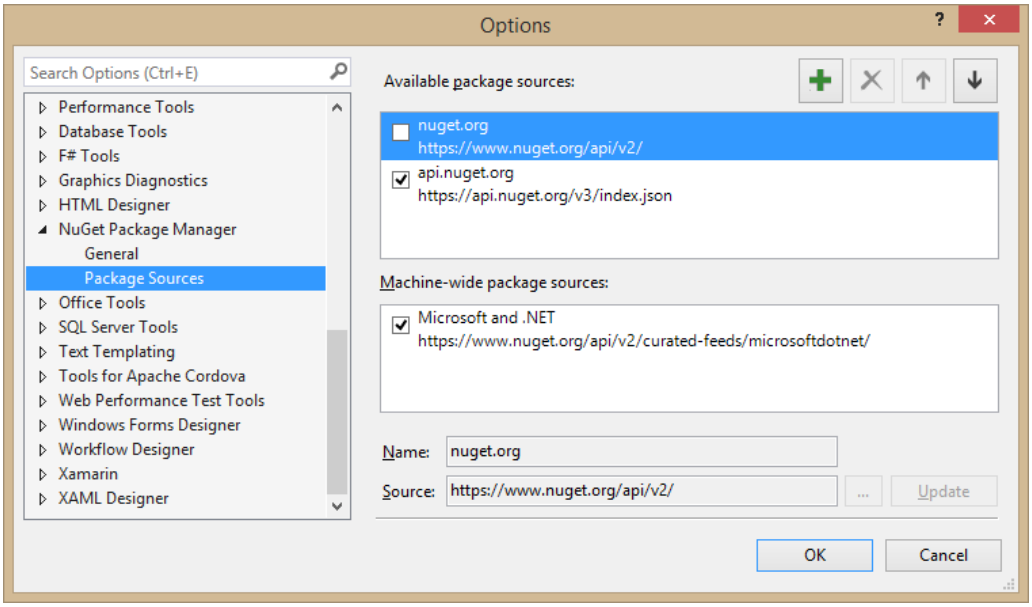

Figure 64: Package Sources options.

While you cannot edit the **Machine-wide package sources** contents, which typically points to packages of the .NET Framework, you can definitely add custom sources by clicking the green **+** symbol. The **Name** and **Source** text boxes will be enabled, accepting a friendly name for your repository and the URL. It is important to mention that you can also specify a local folder on your machine, not just an online repository.

Note: In case you are interested in creating custom NuGet packages and repositories, the NuGet foundation has specific documentation.

Chapter Summary

In this chapter, you saw how NuGet tooling has been revisited in order to be non-blocking when managing packages in your solution. You also saw how to create dependency graphs for your packages and how to specify additional settings. It is important to remember that NuGet tools are actually offered by a Visual Studio extension, and Microsoft provides updates for this extension very often.

Chapter 8 Visual Studio 2015 for ASP.NET and Azure

Building applications for web and cloud environments is the core business for many thousands of developers. As the web and cloud environments evolve continuously and concurrently, Microsoft is aware that developers need appropriate tools to be on the market quickly and efficiently. Based on .NET Core 5, the recent release of ASP.NET 5 marks a new era for web development; ASP.NET is now an open source platform able to run on multiple operating systems, such as Linux and Mac OSX, and projects you create only use the necessary libraries via NuGet packages, rather than relying on the full .NET Framework. This is known as cloud-optimized ASP.NET; it makes it easier to deploy web applications to Microsoft Azure. Moreover, Microsoft Azure offers powerful new services and tools with every update. All these important improvements and new features make Visual Studio 2015 the perfect place to build web applications with ASP.NET and Microsoft Azure. This chapter explains what's new from a development environment perspective to help you increase your productivity.

Note: Explaining ASP.NET 5 and Microsoft Azure in detail would require entire books, so providing guidance on how you build applications is not possible in this context. ASP.NET has a totally revisited infrastructure and many new features that make a brief summary impossible in this book. Instead, learn more at the official sites for ASP.NET and Microsoft Azure, including full documentation about both technologies.

Chapter Prerequisites

All you need to work with ASP.NET 5 is already included in Visual Studio 2015, assuming you selected the web development tools among the available installation options. In order to complete this chapter, you need both of the following for Microsoft Azure:

- A Microsoft Azure subscription—If you are an MSDN subscriber, you can activate an Azure subscription included in your plan; if you do not have an Azure subscription, you can get a free trial for 30 days.
- The Azure SDK for Visual Studio 2015, which can be obtained via the free Microsoft Web Platform Installer tool. At publication, the current version for the Azure SDK is 2.5.1.

Assuming you have installed all the necessary components and activated a Microsoft Azure subscription, you are ready to discover the new features in Visual Studio 2015.

Visual Studio 2015 for ASP.NET 5

The most interesting tools added to support ASP.NET 5 on .NET Core 5 in Visual Studio 2015 are basically new project templates, a new project system, an enhanced IntelliSense to support multi-framework targeting, and the updated NuGet manager.

Tip: This chapter doesn't cover how to manage NuGet packages since it was discussed in Chapter 7. Just remember that the previous chapter applies to ASP.NET as well.

ASP.NET 5 Core Project Templates

Visual Studio 2015 adds five new project templates to support ASP.NET 5 Core. In particular, two new project templates have been added to the **Web** node of the **New Project** dialog; they are the **Class Library (Package)** and the **Console Application (Package)** (see Figure 65).

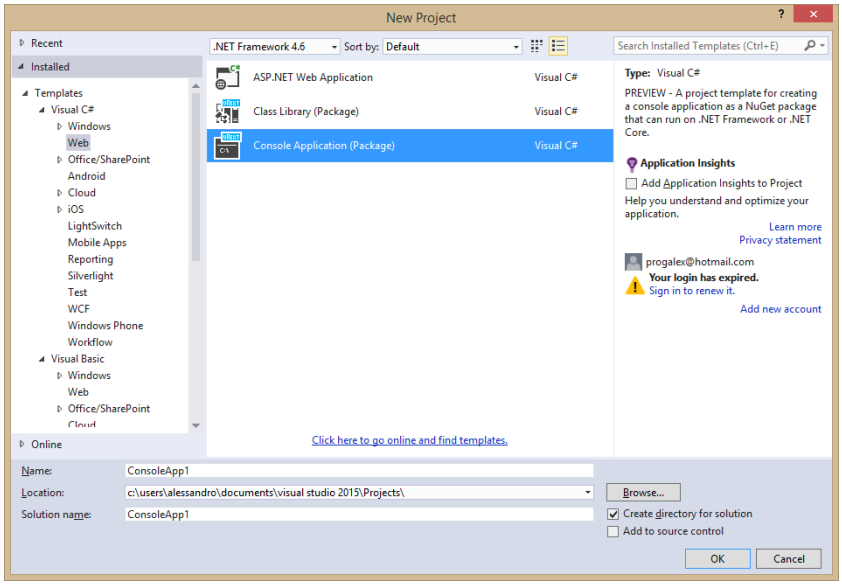

Figure 65: ASP.NET 5 Class Library and Console Application new project templates.

Both the Class Library and Console Application templates produce assemblies that can run on multiple platforms when you use ASP.NET 5 Core. Three other project templates have been added to the **New ASP.NET Project** dialog, which appears when you select the **ASP.NET Web Application** template in the **New Project** dialog. Specifically, you now have a new group called ASP.NET 5 Preview Templates, containing the **Empty**, **Web Site**, and **Web API** project templates, as shown in Figure 66.

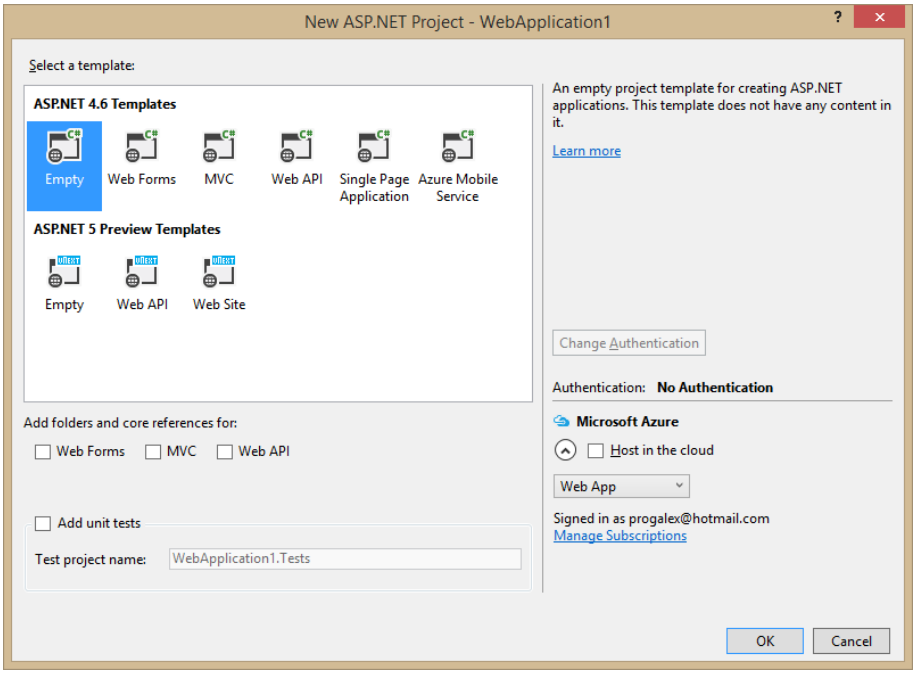

Figure 66: ASP.NET 5 new project templates.

Tip: When you install the Azure SDK 2.5.1 or higher, another project template called Azure Mobile App is added to the ASP.NET 4.6 templates.

What the ASP.NET 5 Starter Web template does is create a basic web application based on MVC 6, including some identity models, controllers, views, and support for Entity Framework migrations. Figure 67 shows what happens when you create a new Starter Web project. You can see how Visual Studio 2015 provides an important list of links for learning what's new; it also displays a new project system in the Solution Explorer.

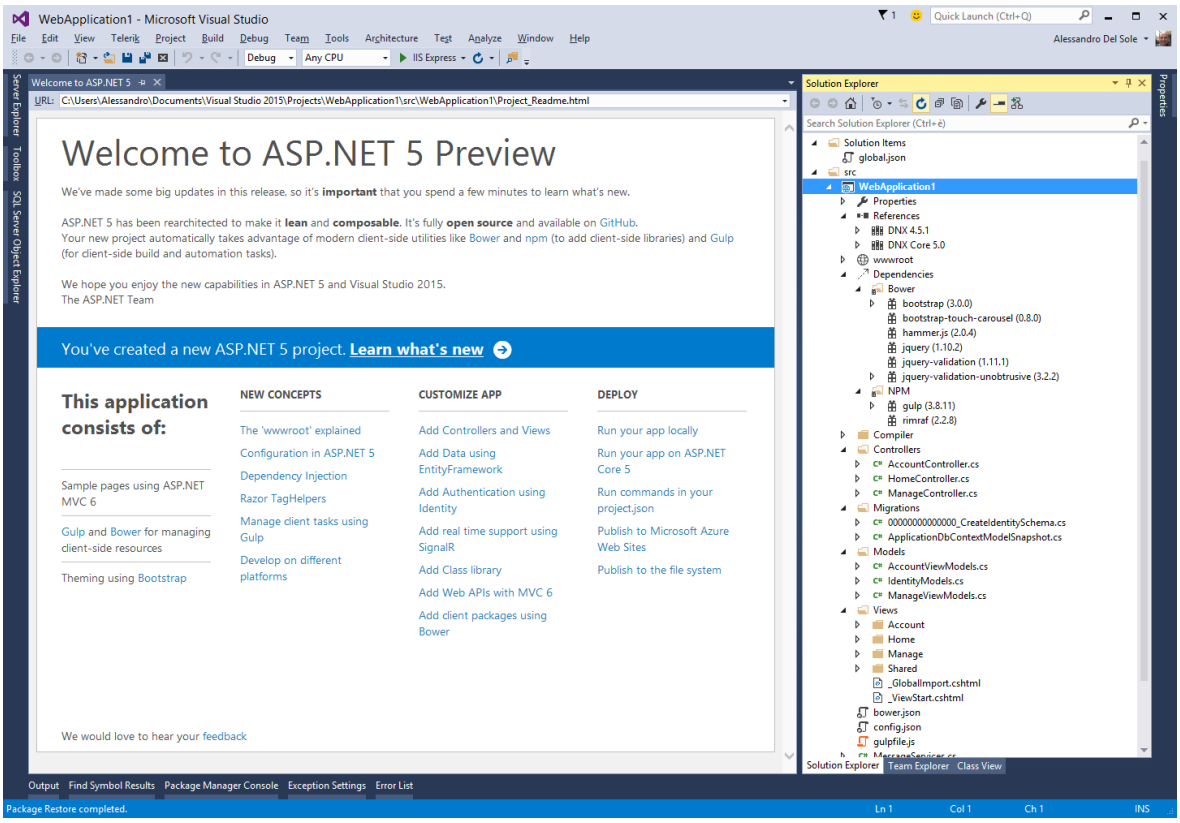

Figure 67: Creating a Starter Web project.

If you are familiar with ASP.NET MVC, you will be able to reuse your skills about models, views, and controllers. This project template also supports Web API to build RESTful HTTP services. The big differences are instead related to the new project system.

A New Project System

As you can see by expanding all project items in Solution Explorer, ASP.NET 5 Core projects have a new structure. This new project system allows cleaner separation between static files and code files, automated project management, and better package management.

The wwwroot Node

The **wwwroot** node contains all static code files, such as assets, fonts, CSS files, JavaScript files, and HTML files. Code files, like C# and Razor files, should never go in this folder because wwwroot provides a clean separation between code files and static files.

The Dependencies Node

The **Dependencies** node contains dependencies for Bower and NPM. Bower is basically a client package manager for the Web and allows installing and restoring client-side packages, including JavaScript and CSS libraries. Bower works on the client, so you will still use NuGet on server-side frameworks like MVC 6. NPM is instead a package manager originally built for Node.js. Visual Studio 2015 automatically uses Bower to install packages and generates a list of packages in the **Bower.json** file that you can find in Solution Explorer. You can manually edit Bower.json by adding dependencies to the **dependencies** node. Supposing you want to use the **RequireJS** library, you can start typing and get IntelliSense help to choose among available libraries, as shown in Figure 68. Not limited to this, you get IntelliSense support to choose one of the available versions (see Figure 69).

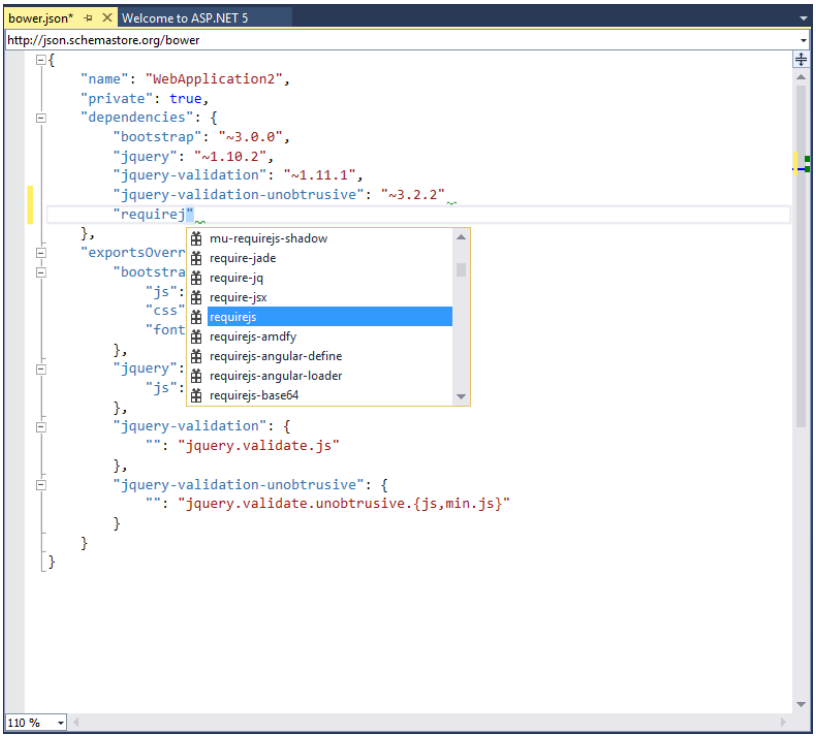

Figure 68: Editing the Bower.json file with IntelliSense support.

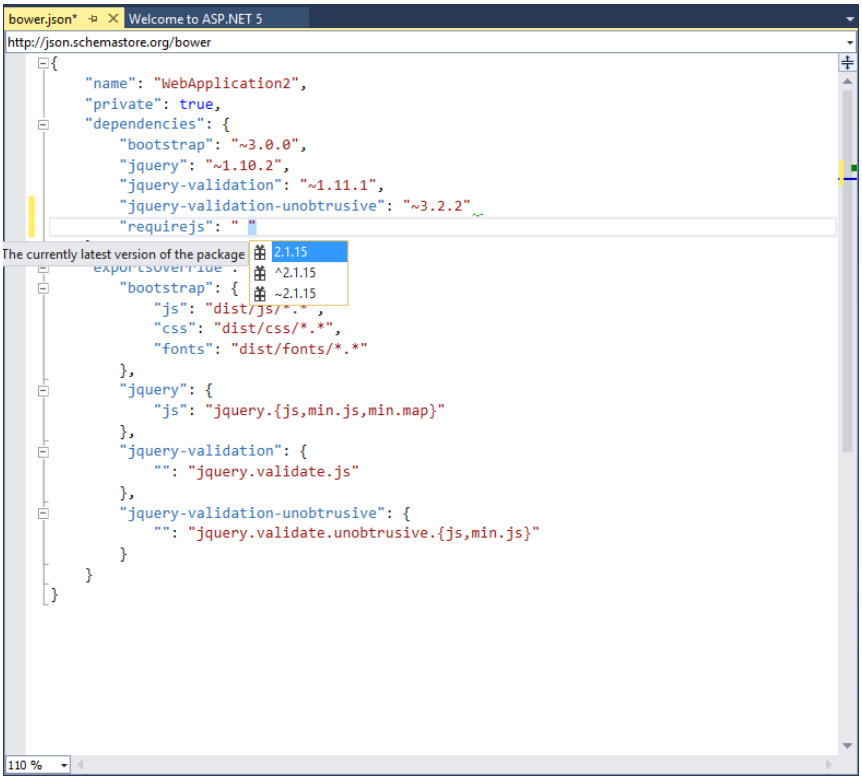

Figure 69: IntelliSense helps you choose one of the available versions.

Once you have completed your edits, right-click the **Bower** subfolder in Solution Explorer and select **Restore Packages** (see Figure 70).

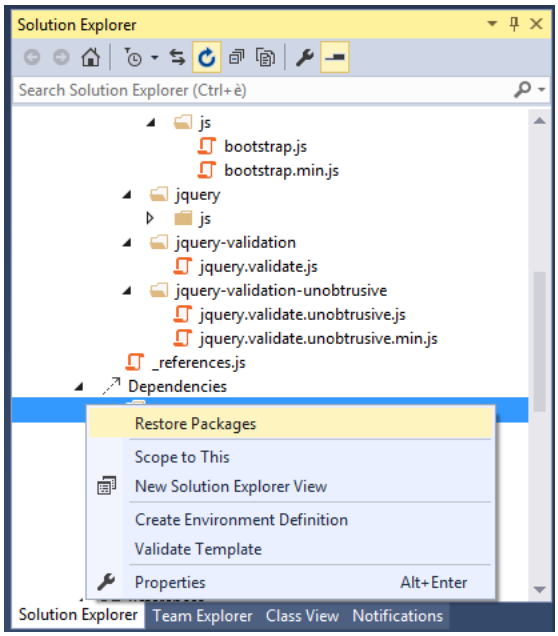

Figure 70: Restoring the appropriate packages after editing Bower.json.

In Solution Explorer you will also see a file called **package.json**, which contains the list of dependencies for NPM. The NPM subfolder contains additional JSON files specific to Grunt. Grunt is a JavaScript-based task runner that helps automate routine development tasks. The official ASP.NET web site has an interesting example showing how to use Grunt with the Starter Web project template. Such an example also explains how to use additional tooling specific to Grunt, such as the Task Runner Explorer window.

New Project Files and JSON Files

ASP.NET 5 Core projects have different organization for files and configuration information. In particular, the differences include the following:

- Project files have the **.kproj** extension. The project file no longer contains any directory items or references because Visual Studio 2015 automatically includes the necessary files.
- The **project.json** file contains the list of reference and package dependencies, version definitions, framework configurations, compile options, build events, package creation meta-data, and commands. Having all this information in a JSON file is very useful because it can be edited manually on projects that run on Linux and MacOS and that do not have Visual Studio 2015 installed.
- The **config.json** file is used instead of Web.config and contains all the application and server configuration information. Of course, Web.config is still there for ASP.NET 4.5 projects.

The References Node

The Reference node displays all frameworks listed in project.json. By default, selected (and displayed) frameworks are **DNX 4.5.1** and **DNX Core 5.0**. Each framework contains the code required to bootstrap and run an application within a .NET execution environment, and is represented by a node that can be expanded to show the list of packages per framework.

The Compiler Node

The Compiler node contains code files for pre-compilation directives based on the Razor engine.

IntelliSense Improvements

IntelliSense is another place that has improvements in Visual Studio 2015 for ASP.NET 5 Core. For instance, since you may target multiple frameworks, such as ASP.NET 5 and ASP.NET Core 5, IntelliSense can mix suggestions to show a warning when a member is not supported in one framework, as demonstrated in Figure 71.

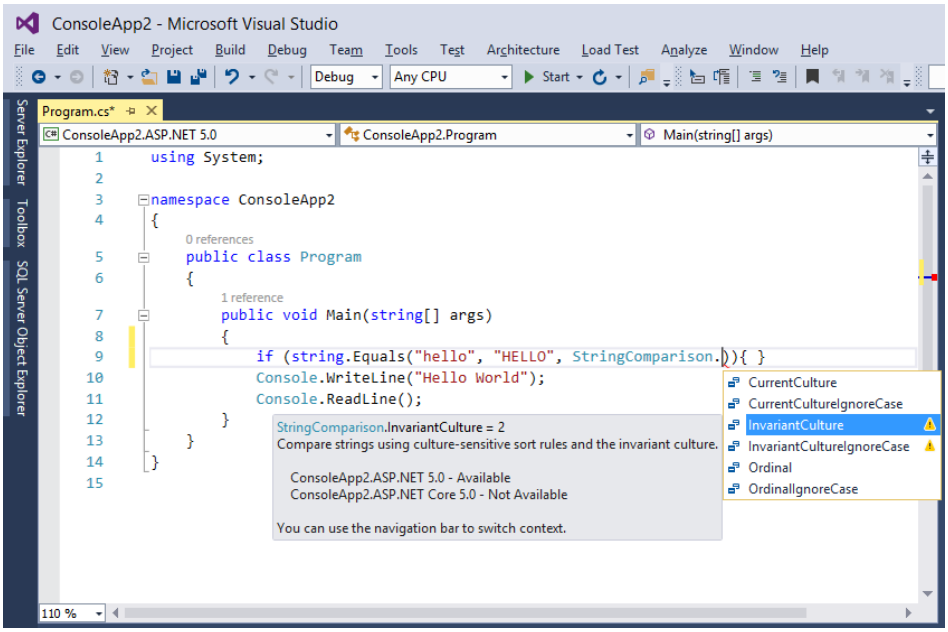

Figure 71: IntelliSense can target multiple frameworks.

Not limited to this, the Error List window can show errors for specific frameworks, as demonstrated in Figure 72, which reflects the code shown in Figure 71.

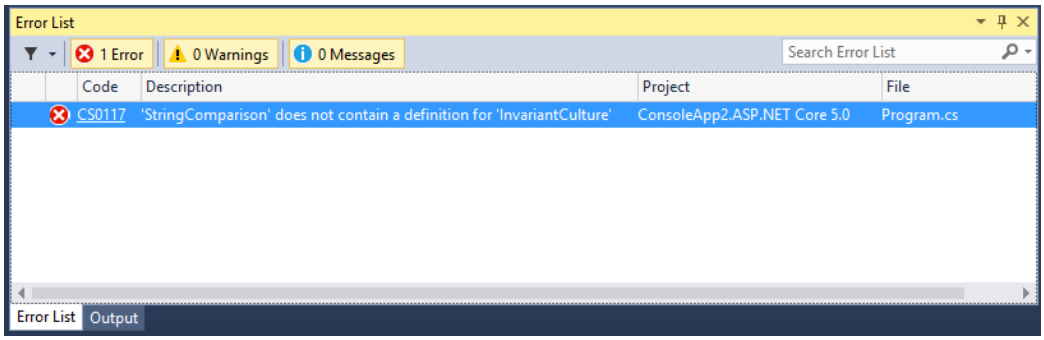

Figure 72: The Error List shows per-framework errors.

IntelliSense has specific support for bower.json and package.json files and introduces support for the **`<link rel="import"/>`** element in the HTML editor, which is part of the Web Components Standard, as demonstrated in Figure 73.

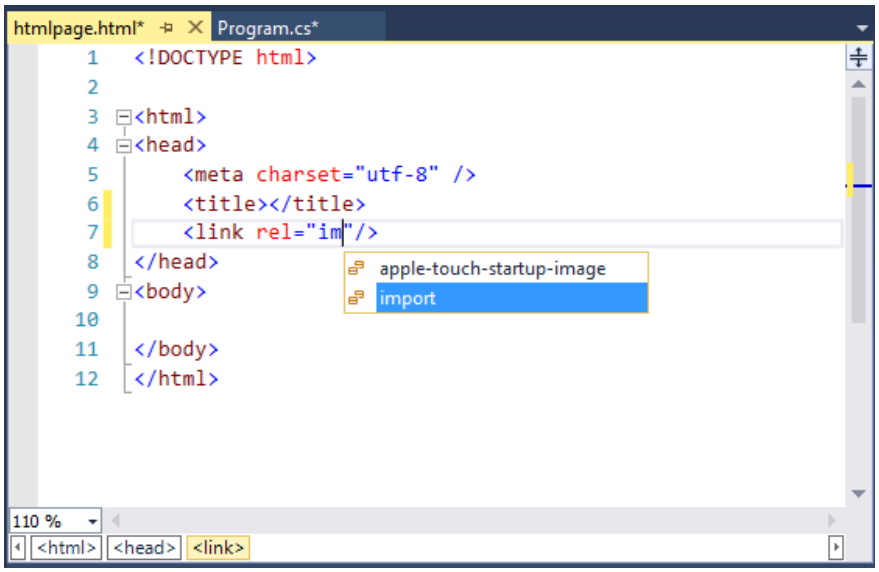

Figure 73: IntelliSense updates for the HTML editor.

Visual Studio 2015 For Microsoft Azure

At the beginning of this chapter, you were invited to install the latest SDK for Microsoft Azure, which extends Visual Studio 2015 to interact with the Azure platform from within the IDE. Now you will get guidance on the most relevant new features that will make your life easier when working with Azure from Visual Studio.

Note: As for previous versions, the current Azure SDK adds not only many features to Visual Studio, but it provides additional tools and services. In this book we focus on features that are available in Visual Studio 2015's IDE. For a comprehensive list of new features, you can read the Azure SDK 2.5.1 for .NET Release Notes. Also, this chapter mentions many services and platforms that Microsoft Azure exposes; in this case, hyperlinks to official pages and documentation are provided, while the book focuses on new tooling.

Quick Starts

Quick Starts provide a collection of sample projects, which demonstrate the usage of various Azure services and APIs. They are available in the **Cloud** > **QuickStarts** node in the **New Project** dialog as demonstrated in Figure 74.

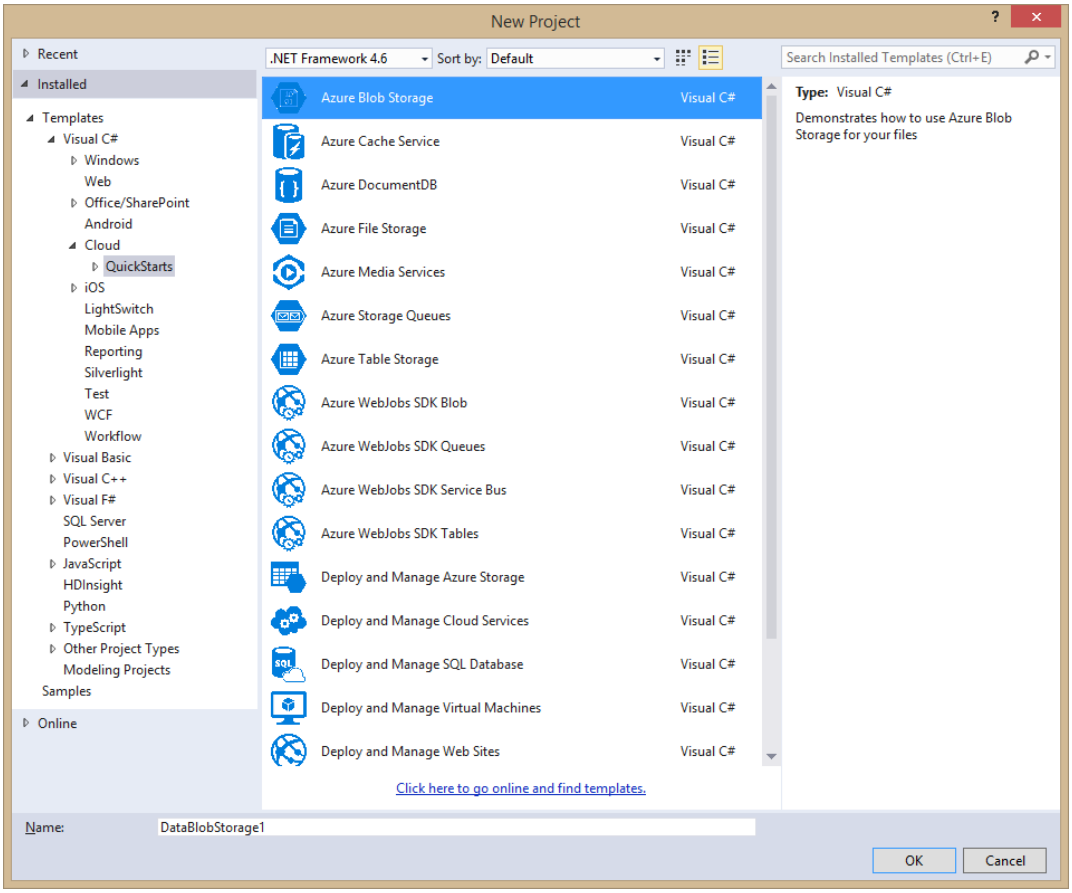

Figure 74: Quick Starts projects for Azure.

Quick Starts projects are also organized in sub-nodes, called **AppServices**, **Compute**, and **DataServices**. Each project provides a fully functional sample, demonstrating how to use APIs that are specific to the targeted service. This means that the sample code contains very detailed comments and examples on how to interact with the desired service in code. For instance, the following listing is a significant excerpt from the code generated by the **Azure Blob Storage** template, and shows how to interact with your Azure Blob Storage account.

```
namespace DataBlobStorage3Sample
{
    using Microsoft.WindowsAzure;
    using Microsoft.WindowsAzure.Storage;
    using Microsoft.WindowsAzure.Storage.Blob;
    using System;
    using System.Collections.Generic;
    using System.IO;
    using System.Linq;
    using System.Threading.Tasks;

    /// <summary>
```

```
    /// Azure Storage Blob Sample - Demonstrate how to use the Blob Storage
service.
    /// Blob storage stores unstructured data such as text, binary data,
documents or media files.
    /// Blobs can be accessed from anywhere in the world via HTTP or HTTPS.
    ///
    /// Note: This sample uses the .NET 4.5 asynchronous programming model
to demonstrate how to call the Storage Service using the
    /// storage client libraries asynchronous API's. When used in real
applications, this approach enables you to improve the
    /// responsiveness of your application. Calls to the storage service
are prefixed by the await keyword.
    ///
    /// Documentation References:
    /// - What is a Storage Account - http://azure.microsoft.com/en-
us/documentation/articles/storage-whatis-account/
    /// - Getting Started with Blobs - http://azure.microsoft.com/en-
us/documentation/articles/storage-dotnet-how-to-use-blobs/
    /// - Blob Service Concepts - http://msdn.microsoft.com/en-
us/library/dd179376.aspx
    /// - Blob Service REST API - http://msdn.microsoft.com/en-
us/library/dd135733.aspx
    /// - Blob Service C# API -
http://go.microsoft.com/fwlink/?LinkID=398944
    /// - Delegating Access with Shared Access Signatures -
http://azure.microsoft.com/en-us/documentation/articles/storage-dotnet-
shared-access-signature-part-1/
    /// - Storage Emulator - http://msdn.microsoft.com/en-
us/library/azure/hh403989.aspx
    /// - Asynchronous Programming with Async and Await  -
http://msdn.microsoft.com/en-us/library/hh191443.aspx
    /// </summary>
    public class Program
    {
        //
**************************************************************************
*****************************************
        // Instructions: This sample can be run using either the Azure
Storage Emulator that installs as part of this SDK - or by
        // updating the App.Config file with your AccountName and Key.
        //
        // To run the sample using the Storage Emulator (default option)
        //      1. Start the Azure Storage Emulator (once only) by pressing
the Start button or the Windows key and searching for it
        //         by typing "Azure Storage Emulator". Select it from the
list of applications to start it.
        //      2. Set breakpoints and run the project using F10.
        //
        // To run the sample using the Storage Service:
```

```
        //      1. Open the app.config file and comment out the connection
string for the emulator (UseDevelopmentStorage=True) and
        //         uncomment the connection string for the storage service
(AccountName=[]...)
        //      2. Create a Storage Account through the Azure Portal and
provide your [AccountName] and [AccountKey] in
        //         the App.Config file. See
http://go.microsoft.com/fwlink/?LinkId=325277 for more information
        //      3. Set breakpoints and run the project using F10.
        //
        //
**************************************************************************
*****************************************
        static void Main(string[] args)
        {
            Console.WriteLine("Azure Storage Blob Sample\n ");

            // Block blob basics.
            Console.WriteLine("Block Blob Sample");
            BasicStorageBlockBlobOperationsAsync().Wait();

            // Page blob basics.
            Console.WriteLine("\nPage Blob Sample");
            BasicStoragePageBlobOperationsAsync().Wait();

            Console.WriteLine("Press any key to exit");
            Console.ReadLine();
        }

        /// <summary>
        /// Basic operations to work with block blobs.
        /// </summary>
        /// <returns>Task<returns>
        private static async Task BasicStorageBlockBlobOperationsAsync()
        {
            const string ImageToUpload = "HelloWorld.png";

            // Retrieve storage account information from connection string.
            // How to create a storage connection string -
http://msdn.microsoft.com/en-us/library/azure/ee758697.aspx
            CloudStorageAccount storageAccount =
CreateStorageAccountFromConnectionString(CloudConfigurationManager.GetSetti
ng("StorageConnectionString"));

            // Create a blob client for interacting with the blob service.
            CloudBlobClient blobClient = storageAccount.
                                CreateCloudBlobClient();
```

```
            // Create a container for organizing blobs within the storage
account.
            Console.WriteLine("1. Creating Container");
            CloudBlobContainer container = blobClient.
            GetContainerReference("democontainerblockblob");
            try
            {
                await container.CreateIfNotExistsAsync();
            }
            catch (StorageException)
            {
                Console.WriteLine("If you are running with the default
configuration please make sure you have started the storage emulator. Press
the Windows key and type Azure Storage to select and run it from the list
of applications - then restart the sample.");
                Console.ReadLine();
                throw;
            }

// To view the full Quick Start code, create a new Azure Blob Storage Quick
// Start in Visual Studio 2015.

}
```

As you can see, the code offers samples about the most important tasks that you need to accomplish with the selected scenario. All available Quick Starts behave this way.

Azure Resource Manager Tools

The Azure SDK introduces a new tool called Azure Resource Manager. This new tool basically helps you create both a web project and a deployment project that contains the necessary artifacts to provision Azure resources, creating the appropriate environment for the application on your behalf. You use the Azure Resource Manager by taking advantage of a new project template called **Cloud Deployment Project.** This project template is available in the **Cloud** node of the **New Project** dialog, as shown in Figure 75. Notice the rich template description on the right side of the dialog.

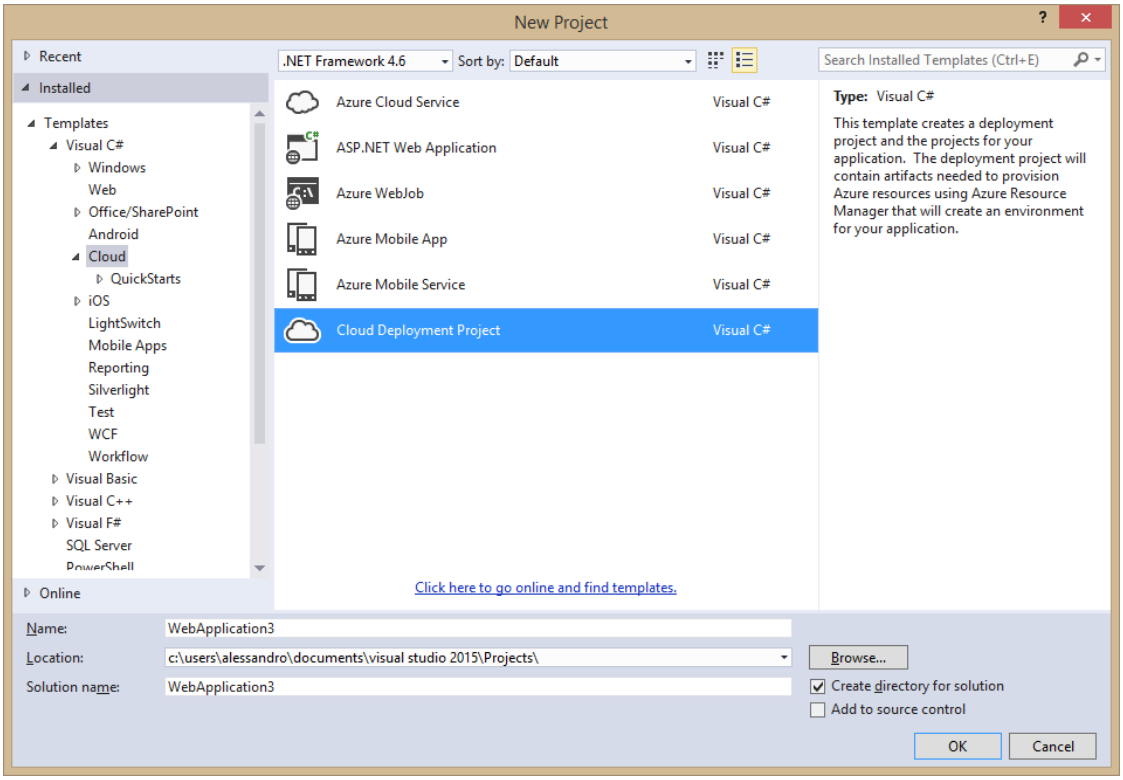

Figure 75: The Cloud Deployment Project template.

The Cloud Deployment Project uses templates on the Azure Gallery, which allows deep customizations but also reduces cost and effort when moving between environments. Such templates make it easy to configure the application deployment by simply specifying some information from your subscription. To understand how this works, create a new project using this template. You will be asked to select one of the available Azure Gallery templates, as shown in Figure 76.

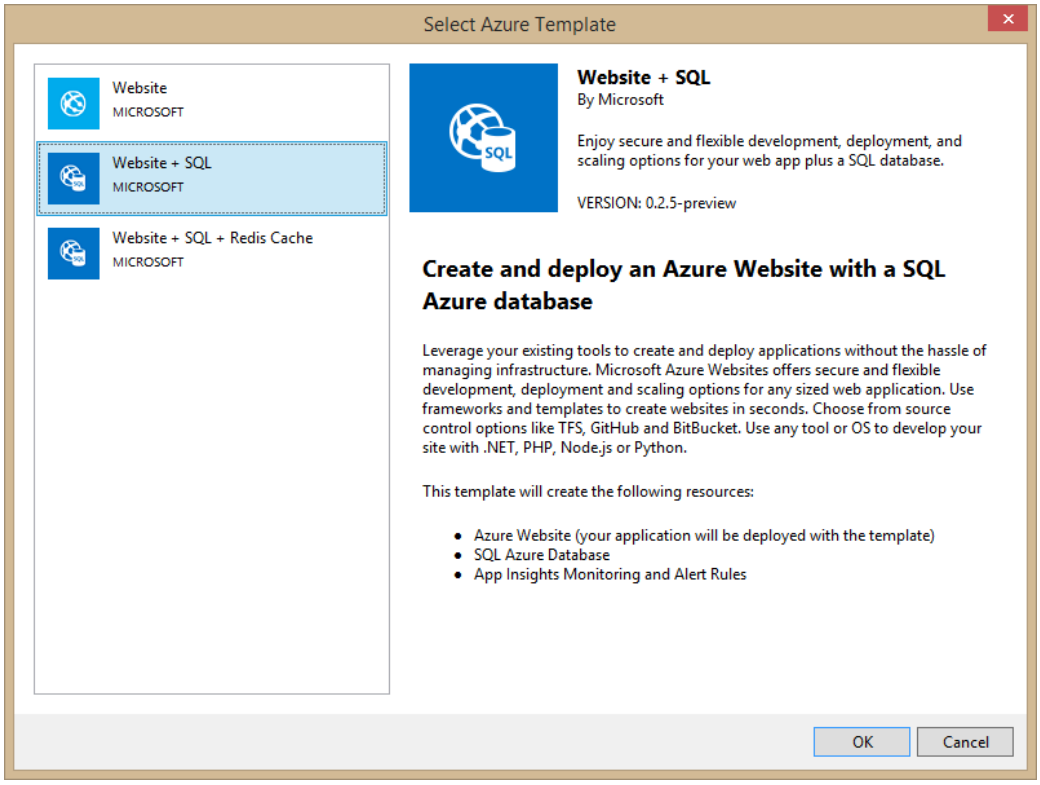

Figure 76: The Azure Gallery templates.

Note: In future releases of the Azure SDK, Microsoft should also add more templates to target services like networking, storage, and virtual machines.

For the current example, select the **Website + SQL template** (see Figure 76). Next, you will be asked to specify an ASP.NET project template. Just select the **Empty** one, as it serves as an example. Then the project is ready; in Solution Explorer you will see two projects: the ASP.NET project and an Azure Resource Manager Deployment project, whose name ends with **.Deployment**. If you expand subfolders in this project you will see the following:

- A reference to the ASP.NET project, a PowerShell script which is responsible for publishing resources to Azure (**Publish-AzureResourceGroup.ps1**).
- A deployment template (**WebSiteDeploy.json**).
- A parameter file containing values that the configuration file uses (**WebSiteDeploy.param.dev.json**).
- A deployment tool (**AzCopy.exe**).

Supposing you have your ASP.NET project ready to do some tasks, you can now prepare for deployment to Azure. When you deploy a Cloud Deployment Project, you deploy it to a Azure Resource Group, which is a logical grouping of resources including (but not limited to) web sites, databases, and storages. Right-click the **.Deployment** project name and select **Deploy** > **New Deployment**. In the **New Deployment** dialog (see Figure 77), select the **Resource group** option, which is specific to the current scenario, and click **OK**.

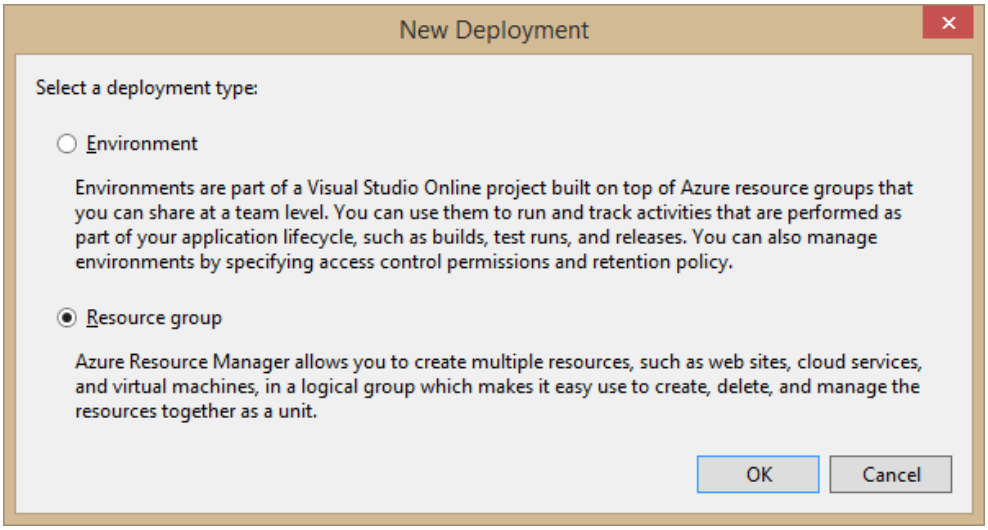

Figure 77: The New Deployment dialog.

You will now see the **Deploy to Resource Group** dialog (see Figure 78), whose fields will be automatically filled by Visual Studio with appropriate values, except for the **Resource group** box, which requires specifying a resource group. This can be one of the default or previously created groups.

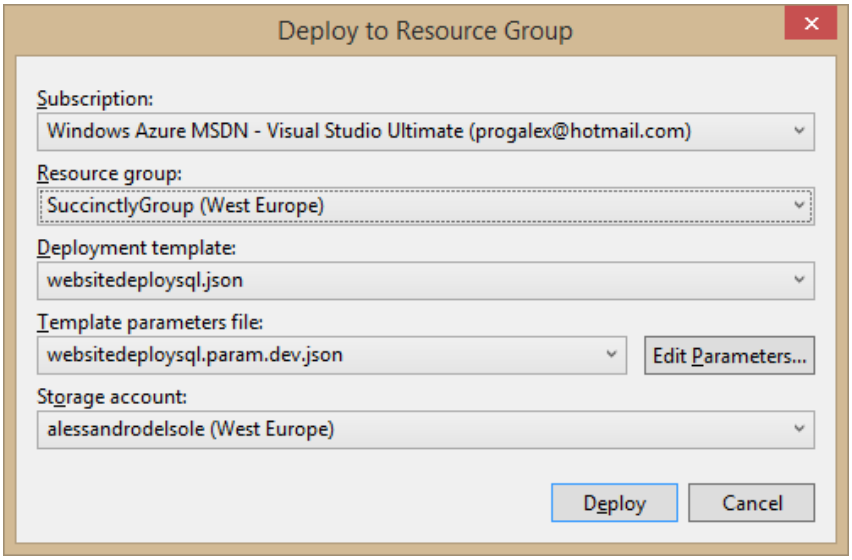

Figure 78: The Deploy to Resource Group dialog.

To create a resource group, click the **Create New…** option in the combo box and in the **Create Resource Group** dialog specify a custom name and region. This must represent the nearest location to you (see Figure 79).

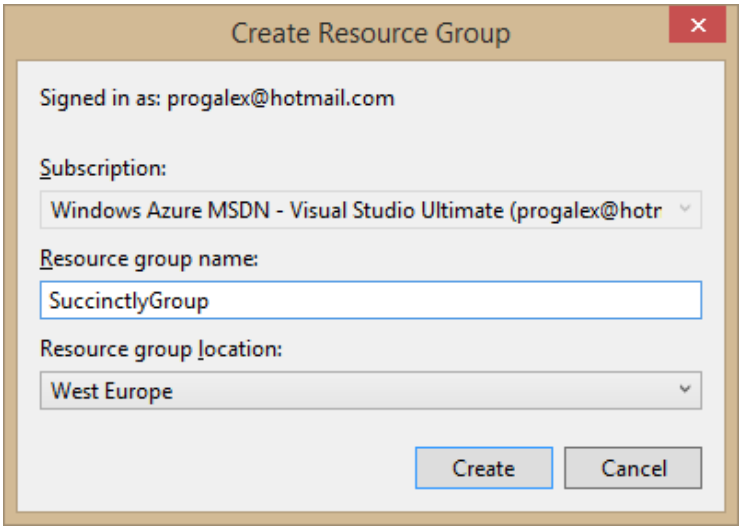

Figure 79: Creating a new resource group.

Before deploying the application, you will need to edit parameters for the WebSiteDeploy.param.dev.json file. By clicking **Edit Parameters**, you will be able to edit parameters visually inside the **Edit Parameters** dialog, as shown in Figure 80.

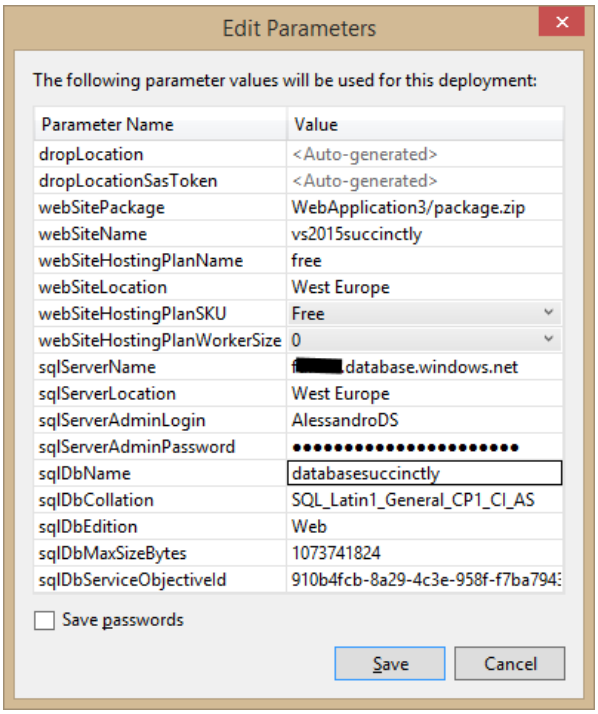

Figure 80: Editing parameters for deployment.

The number of settings varies depending on what Azure Gallery template you select at the beginning. You will see many empty fields and all of them are mandatory; you will see a red glyph with a white exclamation mark that highlights mandatory parameters. Common mandatory parameters are summarized in Table 1.

Table 1: Common Mandatory Parameters for Deployment

Parameter	Description	Value
webSiteName	The first part of the Web site URL.	For instance, if your site is called **vs2015succinctly.azurewebsites.net**, here you specify **vs2015succinctly**.
webSiteHostingPlanName	Web hosting plans represent a set of features and capacity that you can share across websites; it supports the four Azure Websites pricing tiers (Free, Shared, Basic, and Standard).	A web hosting plan name. You are strongly encouraged to read the official page from Microsoft about creating Web Hosting Plans.
webSiteLocation	The web site geographic location.	One of the supported Azure Regions.

Additional parameters are required when you use the **Web site + SQL** template, as in the current example. SQL-related parameters are summarized in Table 2.

Table 2: SQL Mandatory Parameters

Parameter	Description	Value
sqlServerName	The name of your database server, which can be retrieved in the Management Portal.	The server name is like **yourserver.database.windows.net**, where **yourserver** stands for your server's name.
sqlServerLocation	The server geographic location. This will be used if the server does not exist yet.	One of the supported Azure Regions.

Parameter	Description	Value
sqlServerAdminLogin	The database administrator's user name.	See *Description.*
sqlServerAdminPassword	The administrator's password.	See *Description.*

When you are ready, click **Deploy**. Visual Studio 2015 will publish the application to your Microsoft Azure subscription and create the appropriate resource group, including the database if it does not exist. The operation progress is displayed in the Output window as usual. You will then be able to run and manage the application.

Support for Multiple Microsoft Accounts

As you learned in Chapter 1, Visual Studio 2015 introduces support for multiple Microsoft Accounts. This is even more useful when talking about Microsoft Azure, because you can simultaneously sign into multiple subscriptions and leverage the **Server Explorer** window to manage your resources on the cloud. In Server Explorer, right-click the **Azure** node and then select **Manage and filter Subscriptions**. This will open the **Manage Microsoft Azure Subscriptions** dialog, which is represented in Figure 81.

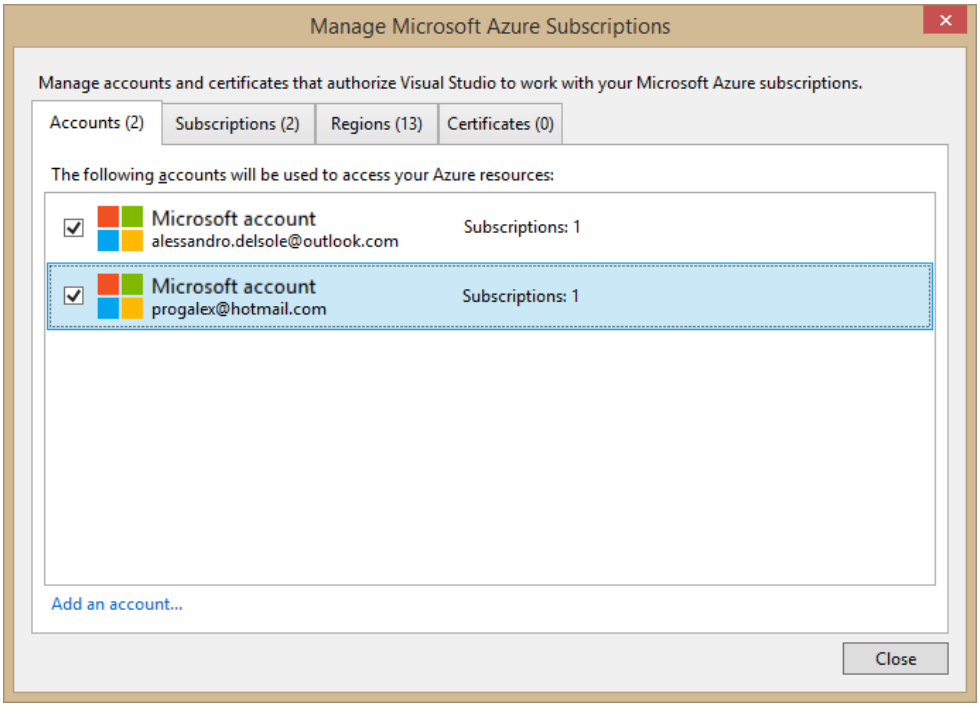

Figure 81: Managing Azure subscriptions.

Here you can select or unselect accounts that will be automatically signed in for Server Explorer; in the **Subscriptions** tab you can select or unselect subscriptions so that Server Explorer shows resources only for the selected ones.

Blob Storage Folders

Visual Studio 2013 and the Azure SDK 2.2 introduced the ability to interact with your Azure Storage Account, which includes blobs, tables, and queues, directly from within the IDE via the Server Explorer window and its Storage node. With regard to the Blob storage, these tools allow creating and managing containers, and uploading/downloading files to/from containers. Visual Studio 2015 and the Azure SDK 2.5.1 take it one step further by providing support for blob folders. These allow blobs to be grouped into logical units, which is very useful when you are using only one blob container. Blob folders are actually a prefix for the blob name; these prefixes include the / delimiter. For instance, say you have a storage account called *vs2015succinctly*, with a container called *documents* and a blob folder called *personal*, and here you have a Word file called *mydocument.docx*. The URL will look like this:

https://vs2015succinctly.blob.core.windows.net/documents/personal/mydocument.docx

This is very intuitive and recalls how you reach folders on a web site (which is actually what you do).

While you can navigate existing blob folders, you cannot create folders from Server Explorer; therefore you create a blob folder when uploading the first file to it. For a better understanding, follow these steps:

1. Right-click the **Blobs** node in Server Explorer for one of your storage accounts (I'm using the local development storage for this example) and select **Create Blob Container**.
2. In the **Create Blob Container** dialog, enter **documents** as the container name (see Figure 82).
3. Double-click the newly created container so that Visual Studio opens the **Container** tool window.
4. Click **Upload** and specify both the file name you want to upload and the folder name, e.g. **personal**. Figure 83 shows an example.
5. When ready, click **OK**. At this point, Visual Studio shows the content of the **personal** blob folder and its content (see Figure 84). Here you can manage blobs the usual way.
6. Click **Open Parent Directory**, which is located near the blob folder name. This will bring you to the upper level and show the list of blob folders in the current container.

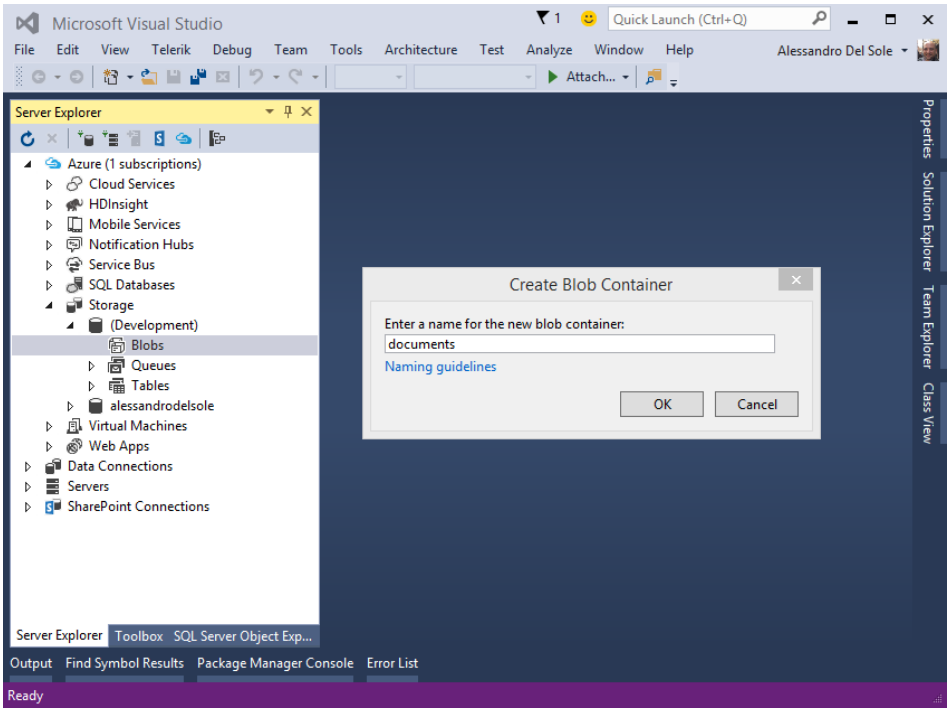

Figure 82: Creating a blob container.

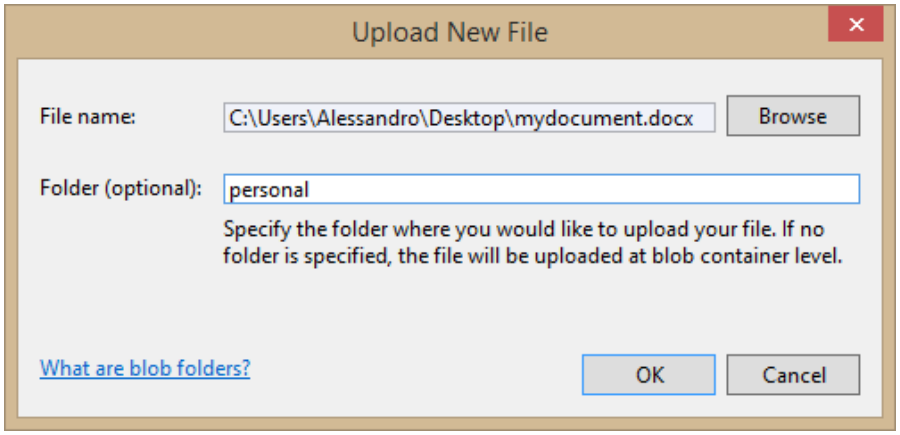

Figure 83: Uploading a file and creating a blob folder.

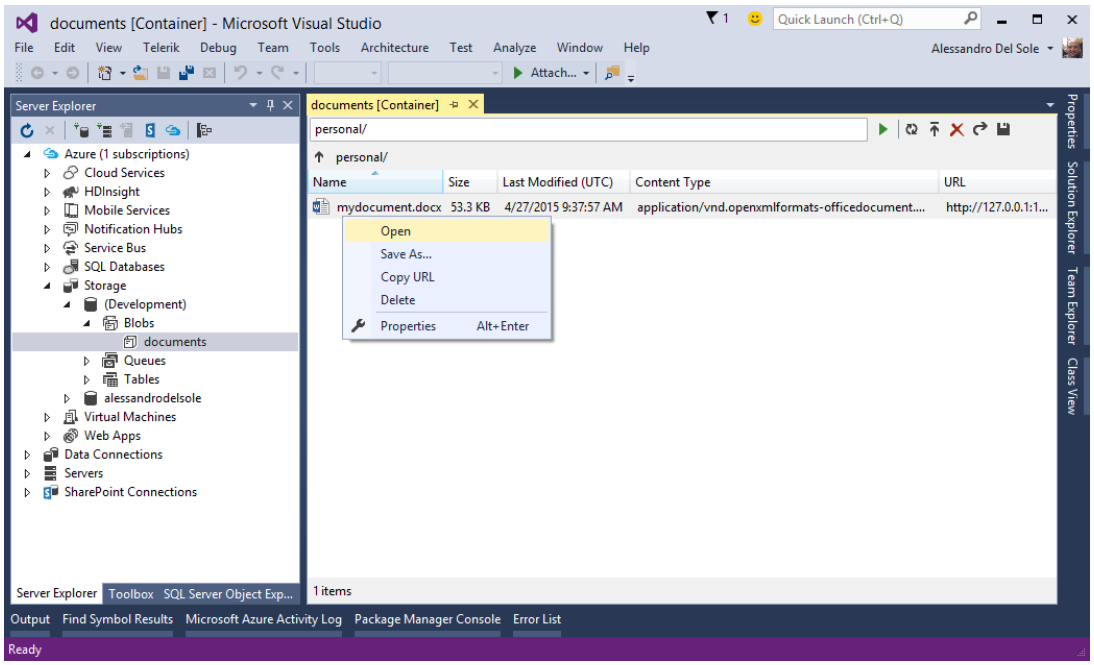

Figure 84: Managing blobs in the blob folder.

It is very important to underline that deleting all blobs from a specific folder will also result in folder deletion. The reason is that a blob folder is just a logical grouping, not a physical location, so when you remove all blobs from a folder, this has no reason to exist anymore.

Adding Connected Services

Visual Studio 2015 and the Azure SDK 2.5.1 make it easier to connect to Azure-based services like Azure Mobile Services, Azure Storage, Office 365, and Salesforce. To understand the improvements, suppose you have an ASP.NET web application. In Solution Explorer, right-click the project name and select **Add** > **Connected Service**. The **Add Connected Service** dialog will show up, offering choices that you can see in Figure 85.

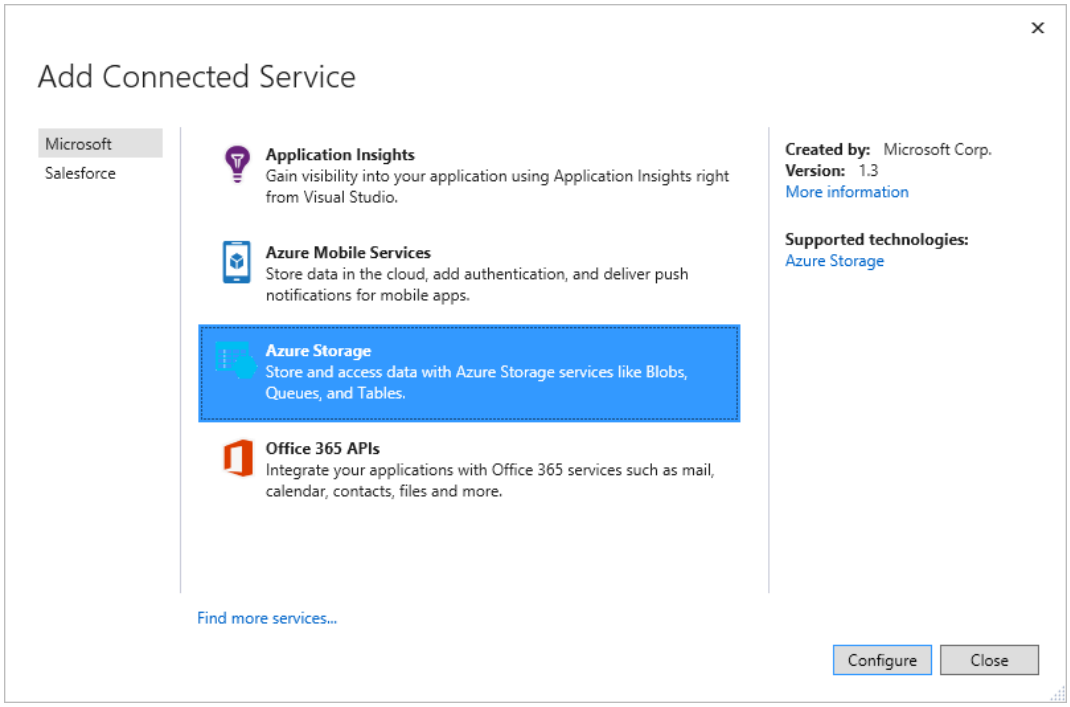

Figure 85: Adding a Connected Service.

Say you want to connect to Azure Storage; select **Azure Storage** and click **Configure**. At this point you will be prompted to select one of your storage accounts in the Azure Storage dialog (see Figure 86), where you will also have an option to switch Microsoft accounts easily.

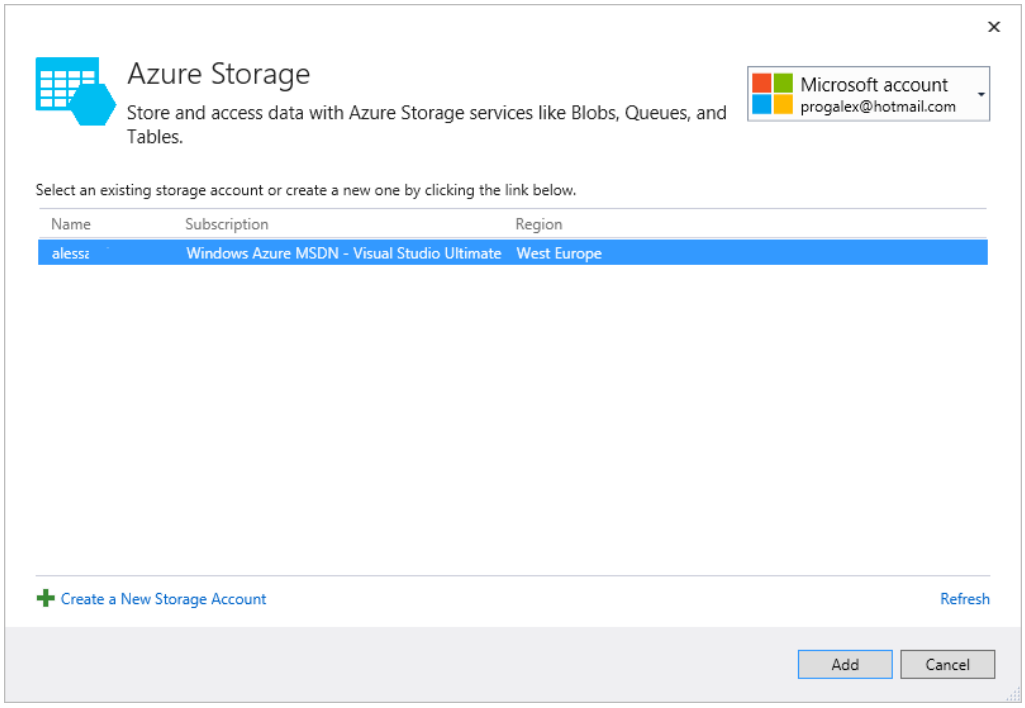

Figure 86: Selecting a storage account.

When you click **Add**, Visual Studio will launch your default Web browser pointing to a “getting started” web page where you get guidance about the code that you need to interact with in the connected service of your choice. Figure 87 shows an example based on the current selection.

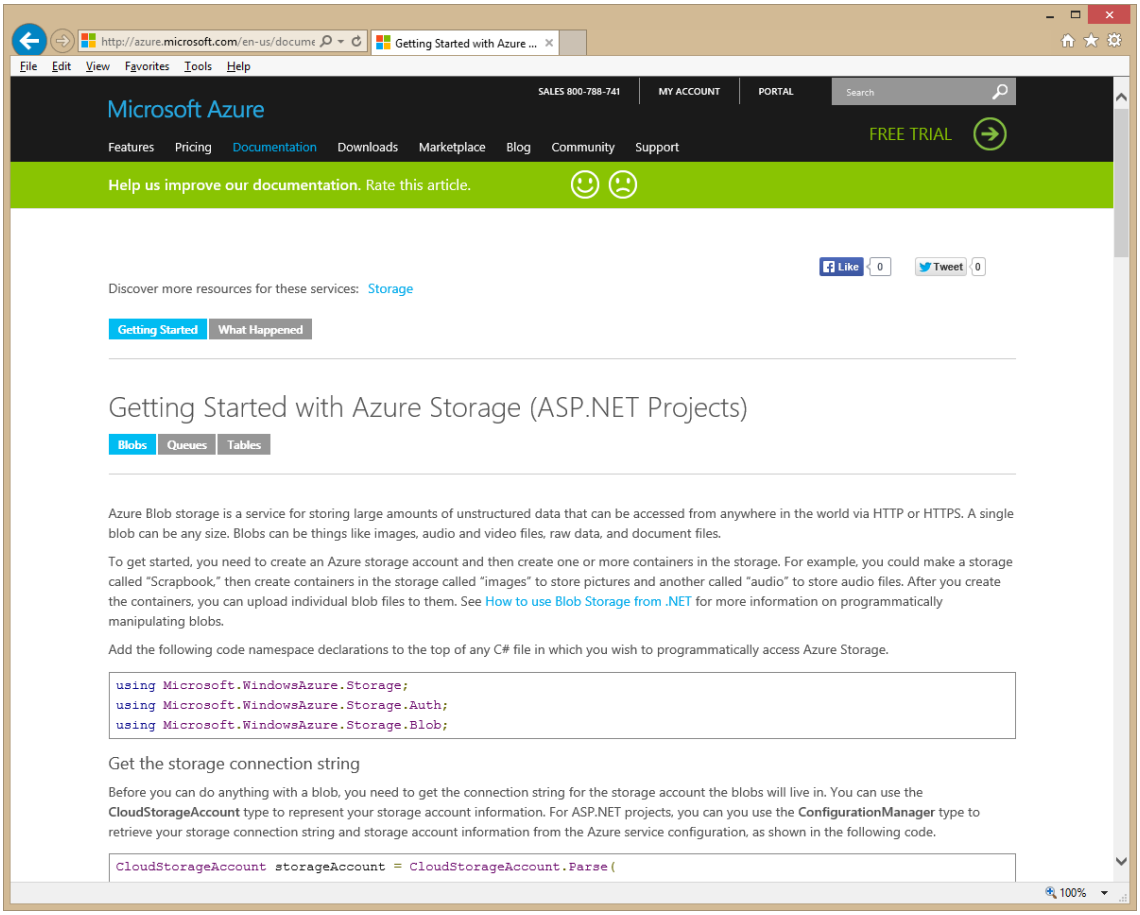

Figure 87: Getting guidance about code you need in the current scenario.

Most importantly, Visual Studio 2015 automatically adds references to the necessary libraries invoking NuGet and the connection string in the Web.config file. This happens with any connected service you choose in the **Add Connected Service** dialog. Describing how each service works is impossible in this book, so this is left to your further studies. What it is important to underline here is how the new way of referencing services makes things faster and easier.

Code Analysis for Azure

The .NET Compiler Platform, formerly known as Project Roslyn, provides compilers with real-time code analysis and issue detection as you type. These capabilities were illustrated by examples in Chapter 3, with regard to Light Bulbs and Quick Actions. Microsoft has also created an additional rules set specifically for Azure, which analyzes issues in your code and suggests possible fixes. The code analysis for Azure is available as a NuGet package. That said, supposing you have an Azure project like a Web Role or a Worker Role, search for the Azure Code Analysis package as shown in Figure 88.

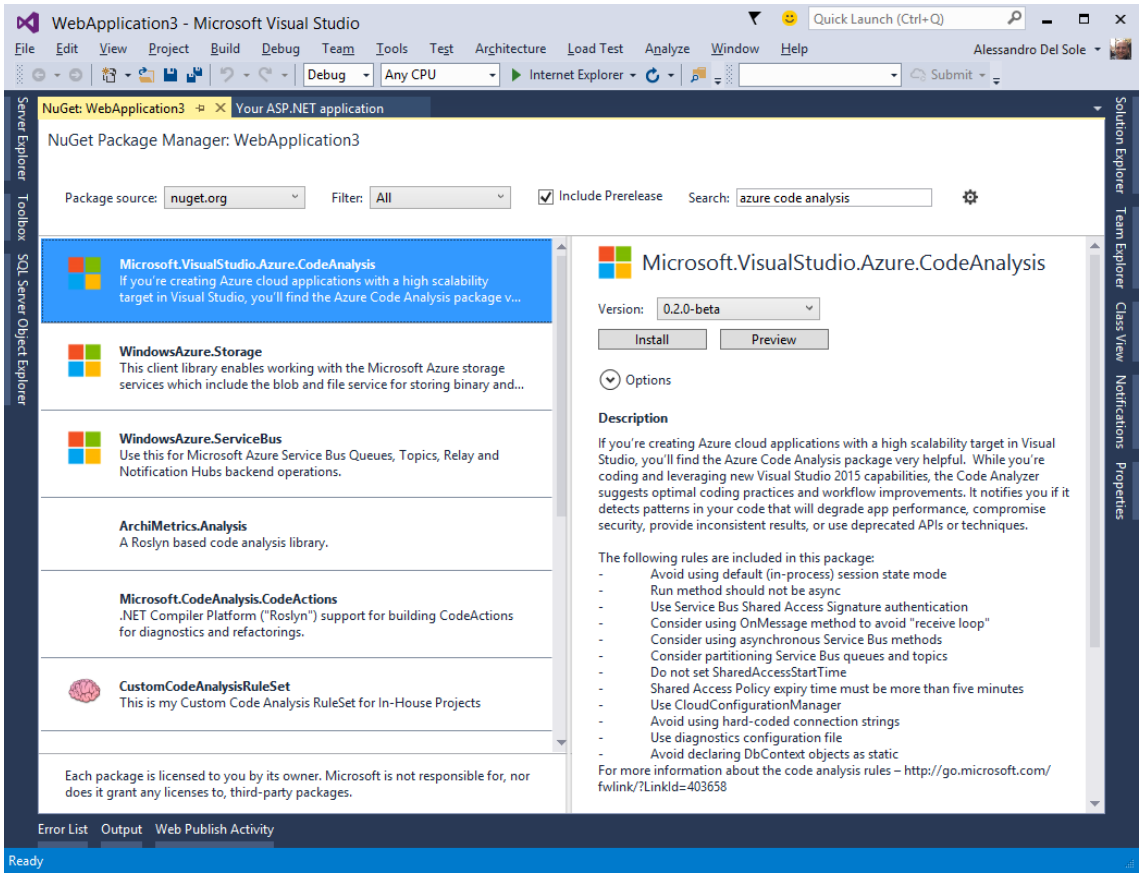

Figure 88: Adding Azure Code Analysis to your projects.

When the package is installed, compilers are able to check for code compliance to the Azure rules and best practices in real time. For instance, Figure 89 shows an issue with the **Run** method in a Worker Role, which has been marked as **async** but is not compliant to the Azure rules; Figure 89 also shows the Light Bulb in action.

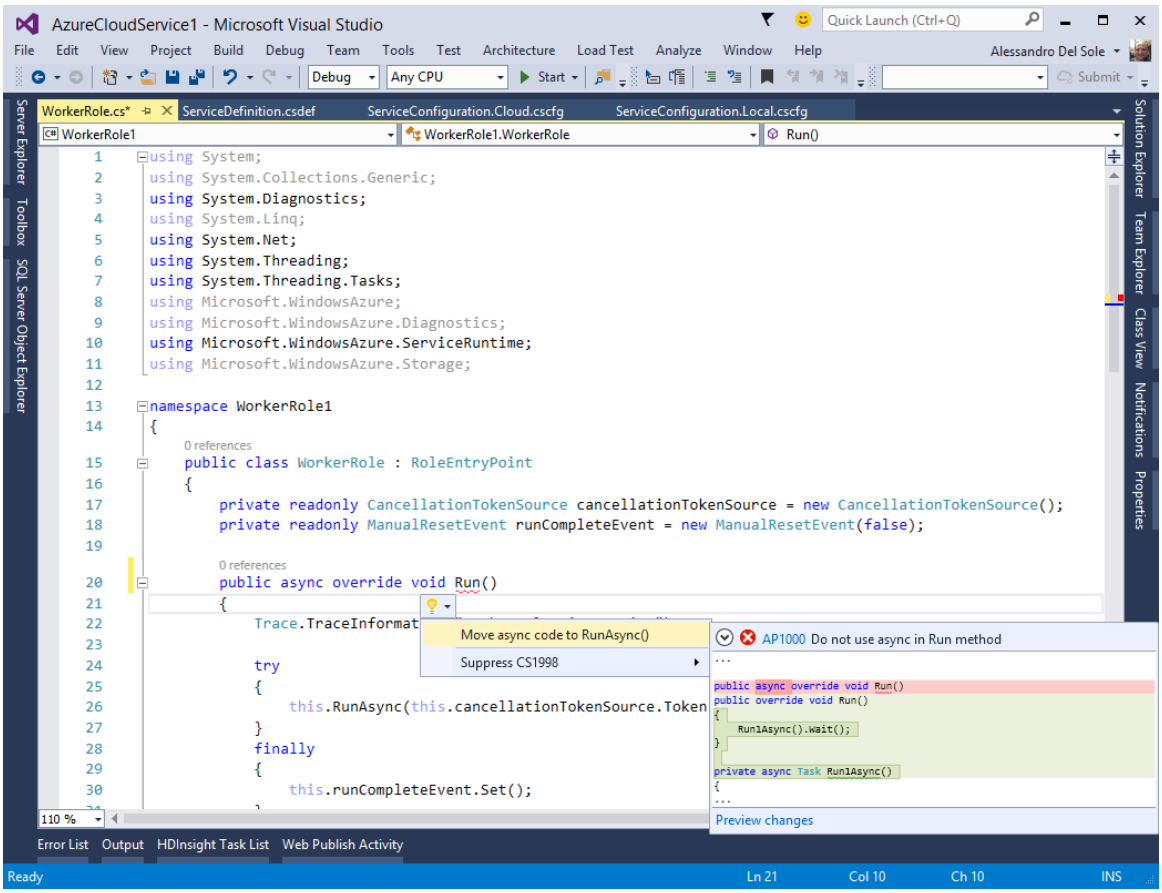

Figure 89: Adding Azure Code Analysis to your projects.

The Azure Code Analysis is particularly useful when you build applications with a high-scalability target, because it suggests best practices and helps you write better and more efficient code.

HDInsight Support

The Azure SDK 2.5.1 and Visual Studio 2015 have support for HDInsight services for Big Data and for Hive to query datasets. Support is made of project templates (see Figure 90) including a Hive sample, which comes with some demo code, and a Server Explorer interaction with HDInsight items, exactly as you would do with any other subscription service.

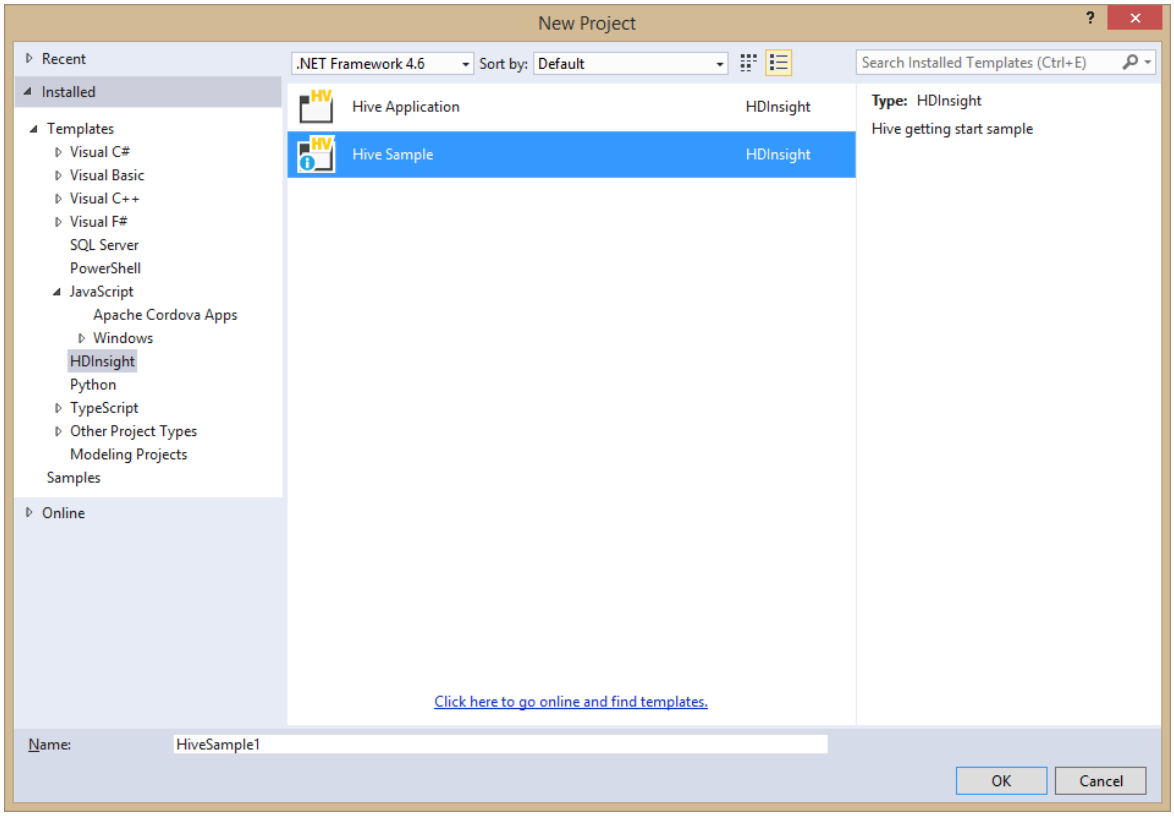

Figure 90: Creating Hive projects with HDInsight support.

The Hive Sample template (see Figure 90) gives you an idea of how HDInsight works. If you double-click a template to create a sample project, you will see how Visual Studio generates two .hql files, each containing specific SQL queries against Azure Websites (WebLogAnalysis.hql) and devices' sensors (SensorDataAnalysis.hql) respectively. The following listing reproduces the sample code for WebLogAnalysis.hql. Notice that the comments explain how each code snippet works and how the code is made of familiar syntax nodes based on SQL.

```
--Part 1: Introduction
-- In this sample you will use an HDInsight query that analyzes website log files
-- to get insight into how customers use the website. With this analysis, you can see
-- the frequency of visits to the website in a day from external websites, and a
-- summary of website errors that the users experience.
-- In this tutorial, you'll learn how to use HDInsight to:
-- *Connect to an Azure Storage Blob containing website log files.
-- *Create Hive tables to query those logs.
-- *Create Hive queries to analyze the data.

--Part 2: Prerequisites
-- The script here is only for your reference. You need to log in to
-- https://yourclusteraddress.azurehdinsight.net/ in order to trigger the HDInsight
-- service to load the sample data.
-- For clusters provisioned before 10/15/2014, go to
-- http://azure.microsoft.com/en-us/documentation/services/hdinsight/ for more details
-- since those clusters do not have the sample data installed.
```

```
--Part 3: Website Log Data Loaded into Windows Azure Storage Blob
-- The data stored in Windows Azure Storage Blob can be accessed by expanding a
-- HDInsight cluster and double-clicking the default container of your default storage
-- account. The data for this sample can be found under the
-- /HdiSamples/WebsiteLogSampleData/SampleLog path in your default container.

--Part 4: Creating Hive table to Query Website Log Data
-- The following Hive statement creates an external table that allows Hive to query
-- data stored in Azure Blob Storage. External tables preserve the data in the original
-- file format, while allowing Hive to perform queries against the data within the
file.
-- The Hive statement below creates a new table, named weblogs, by describing the
-- fields within the files, the delimiter between fields, and the location of the file
-- in Azure Blob Storage. In the Creating Hive Queries to Analyze Data section of this
-- tutorial, you will perform queries on the data stored in this table.
-- You could also create a table by right-clicking on a database and selecting
-- "Create Table". We will provide you with a UI to help you to create this table.

DROP TABLE IF EXISTS weblogs;
-- Create table weblogs on space-delimited website log data.
-- In this sample we will use the default container. You could also use
-- 'wasb://[container]@[storage account].blob.core.windows.net/Path/To/Data/' to access
-- the data in other containers.
CREATE EXTERNAL TABLE IF NOT EXISTS weblogs(s_date date,
                s_time string, s_sitename string, cs_method string, cs_uristem string,
                cs_uriquery string, s_port int, cs_username string, c_ip string,
                cs_useragent string,
                cs_cookie string, cs_referer string, cs_host string,
                sc_status int, sc_substatus int,
                sc_win32status int, sc_bytes int, cs_bytes int, s_timetaken int )
ROW FORMAT DELIMITED FIELDS TERMINATED BY ' '
STORED AS TEXTFILE LOCATION '/HdiSamples/WebsiteLogSampleData/SampleLog/'
TBLPROPERTIES ('skip.header.line.count'='2');


-- The following HIVE queries create two new tables based on the queries run on the
-- weblogs table. The new tables are called clienterrors and refersperday.
-- The query for clienterrors extracts data from the weblogs table for HTTP status
-- codes between 400 and 500, and groups them by the users facing those errors and the
-- type of error codes. The range of status code between 400 and 500, represented by
-- sc_status column in the weblogs table, corresponds to the errors clients get while
-- accessing the website. The extracted data is then sorted on the number of
-- occurrences of each error code and written to the clienterrors table.
-- The query for refersperday extracts data from the weblogs table for all external
-- websites referencing this website. The external website information is extracted
-- from the cs_referer column of the weblogs table. To make sure the referring links
did
-- not encounter an error, the table only shows data for pages that returned an HTTP
-- status code between 200 and 300. The extracted data is then written to the
-- refersperday table.

DROP TABLE IF EXISTS ClientErrors;
-- Create table ClientErrors for storing errors users experienced and their
frequencies.
CREATE EXTERNAL TABLE ClientErrors(sc_status int, cs_referer string, cs_page string,
cnt int)
```

```
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

-- Populate table ClientErrors with data from table weblogs.
INSERT OVERWRITE TABLE ClientErrors
SELECT sc_status, cs_referer,
             concat(cs_uristem,'?', regexp_replace(cs_uriquery,'X-ARR-LOG-ID=[0-9a-
f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}','')) cs_page,
             count(distinct c_ip) as cnt
FROM weblogs
WHERE sc_status >=400 and sc_status < 500
GROUP BY sc_status, cs_referer, concat(cs_uristem,'?', regexp_replace(cs_uriquery,'X-
ARR-LOG-ID=[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}',''))
ORDER BY cnt;

DROP TABLE IF EXISTS RefersPerDay;
-- Create table RefersPerDay for storing references from external websites.
CREATE EXTERNAL TABLE IF NOT EXISTS RefersPerDay(year int, month int, day int,
cs_referer string, cnt int)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

-- Populate table RefersPerDay with data from the weblogs table.
INSERT OVERWRITE TABLE RefersPerDay
SELECT year(s_date), month(s_date), day(s_date), cs_referer, count(distinct c_ip) as
cnt
FROM weblogs
WHERE sc_status >=200 and sc_status <300
GROUP BY s_date, cs_referer
ORDER BY cnt desc;

--Part 6: Executing Queries and Viewing the Results
-- Select Submit/Submit(Advanced) in the HDInsight toolbar to execute the queries. You
-- can also use Alt+Shift+S for a quick submission. After submitting the job, you can
-- view details by right-clicking on the cluster and selecting "View Hive Jobs".
-- You can also expand the Hive databases and right-click on the tables you just
created, select "View Top 100 Rows" and sample the table you just created.
```

Of course, you will need to replace sample web site names with yours, and to adjust the code based on your services and needs, but it is a nice example to understand the purpose of HDInsight.

Introducing WebJobs

Azure WebJobs provides the ability to run programs, such as .exe and .cmd files, on Azure web sites. You may have many reasons to run a program, such as executing background tasks, CPU-Intensive work, image processing when uploading blobs, RSS aggregation, and others. To make WebJobs programming easier, Microsoft also released a WebJobs SDK, which provides a simplified framework that minimizes your effort and reduces the amount of code you need to write. The Azure SDK 2.5.1 for Visual Studio 2015 provides support to the IDE in several ways. First, you have project templates. The **Cloud** node of the **New Project** dialog contains a project template called **Azure WebJob**, as shown in Figure 91.

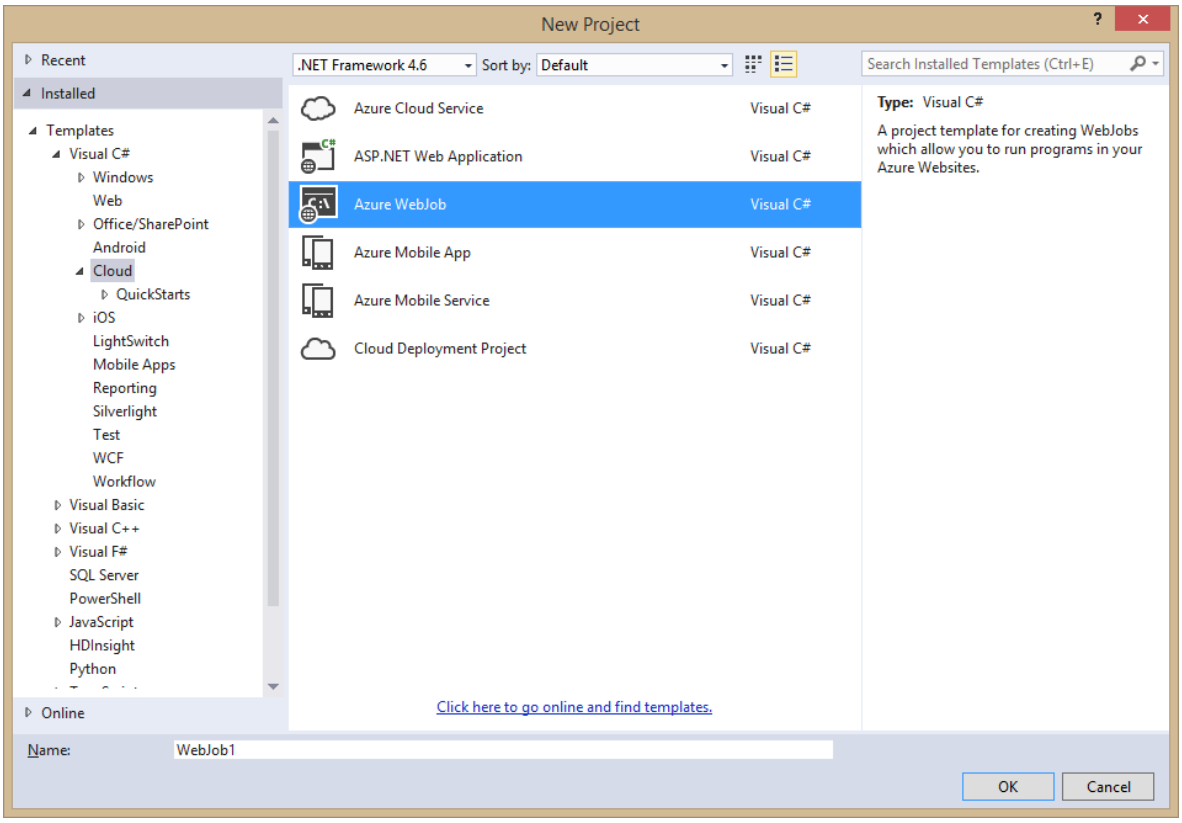

Figure 91: Azure WebJob project template.

However, this is the most basic project template and provides no code. Instead, you can use the Azure Quick Starts to get started with the SDK examples; Figure 92 shows WebJobs Quick Starts.

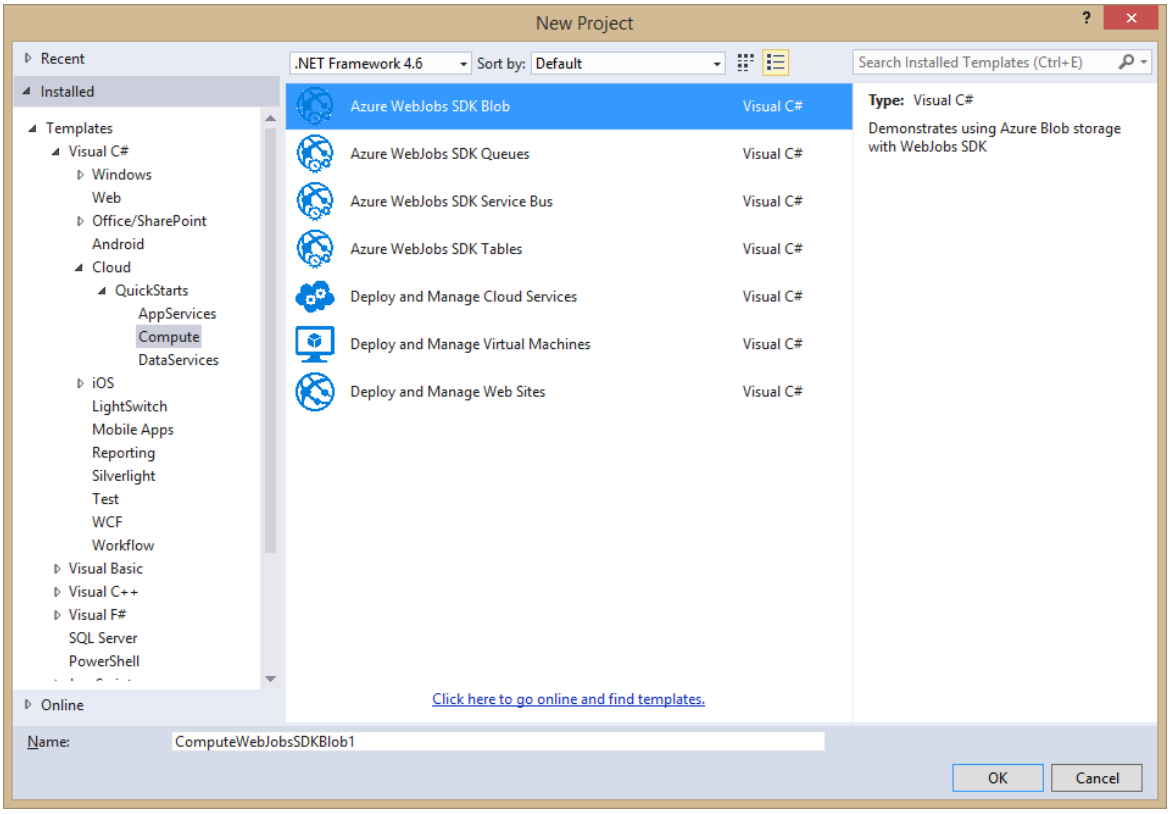

Figure 92: Azure WebJobs Quick Starts from the SDK.

For instance, the Azure WebJobs SDK Blob template demonstrates how to use the Azure Blob storage with WebJobs. More specifically, this example demonstrates how to read/write objects in a queue and how to write information into a text file. As you can see from Figure 92, similar examples are available for other Azure services. In order to publish WebJobs to an Azure Website, you right-click the project name in Solution Explorer and select **Publish as Azure WebJob**. At this point the **Add Azure WebJob** dialog appears (see Figure 93) and allows specifying information such as the name, run mode (**Run Continuously**, **Run on a Schedule**, **Run on Demand**), plus starting and end time in case your WebJob represents a recurring operation.

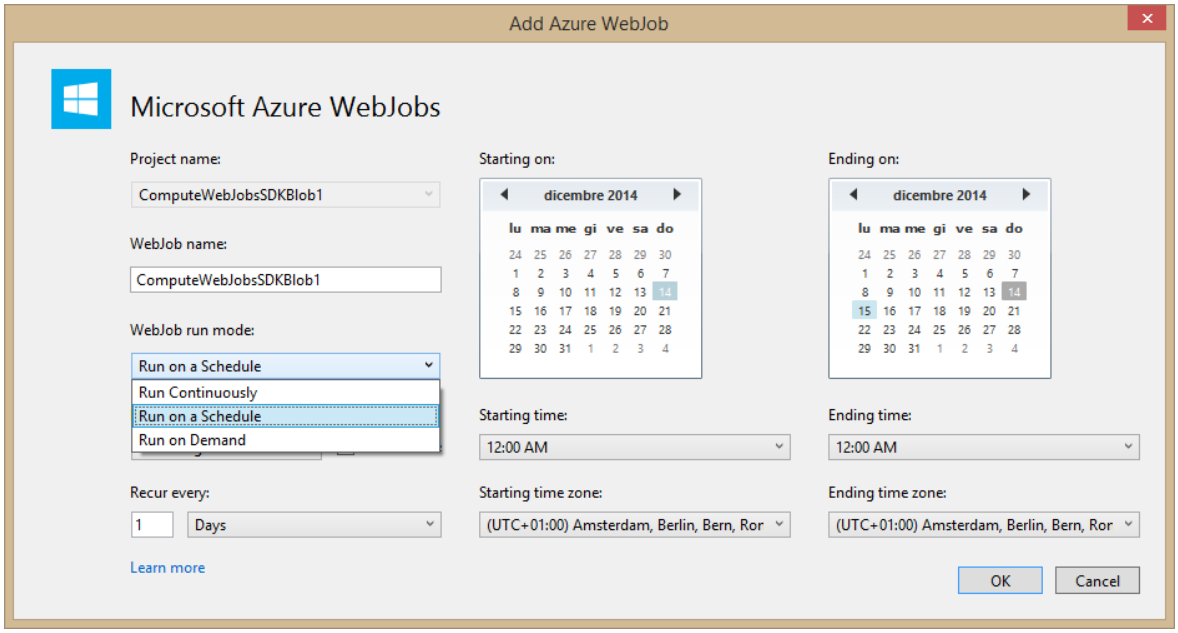

Figure 93: Azure WebJobs Quick Starts from the SDK.

When the WebJob is published to Azure, Server Explorer will allow the following:

- Show a WebJobs node under the related web site name.
- Attach the debugger for remote debugging to the specified WebJob.
- Start and stop WebJobs.

All of these interactions have been made possible due to the WebJobs SDK, which not only simplifies the amount of code you write, but also makes it easier to work with WebJobs via visual tooling.

Chapter Summary

This chapter explained some key features in Visual Studio 2015 for web and cloud development. In the first part, you learned about new project templates that enable developers to target the ASP.NET 5.0 Core runtime. You then learned how templates are organized into a new project system, with new project files, new JSON files for configuration, and an easier package management that allows this kind of applications running cross-platform and using only the needed packages via NuGet. In the second part of the chapter, you became familiar with new tooling for Microsoft Azure, such as Quick Starts project templates and support for multiple Microsoft Accounts to easily connect to cloud resources from Server Explorer. You learned how Visual Studio 2015 allows managing Blob folders and how it simplifies the way you add connected services to your projects. You learned about the Code Analysis for Azure package, which provides real-time code analysis and appropriate suggestions to possible code issues. Finally, you learned how to interact with HDInsight services for Big Data and how to run background programs on your web sites via WebJobs.

Chapter 9 Visual Studio 2015 for Mobile Development

The latest releases of Visual Studio provided optimal support for building mobile applications on top of Microsoft operating systems and platforms, such as Windows Phone 7.x, Windows Phone 8.x, and Windows 8.x. However, these IDEs only support app development for devices running Microsoft operating systems. It's no secret that a huge role in the market of mobile devices, such as tablets and smartphones, is played by Android and iOS operating systems. Microsoft is aware of this and with Visual Studio 2015 they open up to new scenarios, providing tools for building cross-platform mobile apps. This chapter provides you with a background about what possibilities you have in Visual Studio 2015 and explains new opportunities for different platforms.

The Background

One of the biggest goals for developers working on mobile apps is the ability to write an app once and see it run on multiple devices and operating systems. This is actually the meaning of cross-platform mobile development. Without a doubt, Microsoft Visual Studio has been the best development environment for many years. The IDE supports many languages (including HTML5 and JavaScript, the most popular language for building cross-platform apps), offers rich designers for every supported development platform, interactive code editors, debugger integration, code analysis, and productivity features. It also simplifies how developers manage the application development lifecycle with powerful visual tools and services. Before Visual Studio 2015, only developers building apps for Windows Phone and Windows 8.x could take advantage of such a rich development environment. The only exception is Visual Studio 2013, which allows cross-platform development with a third-party product by Xamarin. Visual Studio 2015 and the new .NET ecosystem mark a revolutionary time for Microsoft: the company is open-sourcing many building blocks of the .NET runtime. Similarly, Visual Studio 2015 now allows building cross-platform mobile applications using non-.NET platforms and libraries to target Android and iOS operating systems. This means that Microsoft is trying to attract developers who have been using less-evolved environments like Eclipse and Cocoa/Objective C; developers can finally use Visual Studio to leverage all of its power and productivity tools.

What Mobile Apps Can I Build with VS 2015?

Visual Studio 2015 is the most complete development environment for building mobile apps. With this release, you can build:

- Apps for Windows 10. At the time of publication, Windows 10 is available as a Technical Preview and you can build apps for the pre-release version by installing the developer tools for Visual Studio 2015.
- Apps for Windows 8 and Windows 8.1 using XAML and C#/VB.

- Apps for Windows Phone 8 and Windows Phone 8.1 using XAML and C#/VB.
- Universal Windows Apps using XAML and C#. Visual Basic is also supported but requires some additional, manual steps.
- Apps for Windows 8 and Windows 8.1 using WinJS libraries (HTML5/JavaScript).
- Apps for Windows Phone 8 and Windows Phone 8.1 using WinJS libraries (HTML5/JavaScript).
- Universal Windows Apps using WinJS libraries (HTML5/JavaScript).
- Cross-platform apps for Windows, Windows Phone, Android, and iOS with C#/F# and XAML installing extensions and tools from Xamarin.
- Cross-platform apps for Windows, Windows Phone, Android 2.3.3 and later, and iOS (6, 7, and 8) using Apache Cordova and the integrated Visual Studio Tools for Apache Cordova.

Note: Windows 10 app development is not covered in this book because the Universal Windows Platform application model is a very complex environment; there are also several ways to develop Universal apps, including porting existing iOS and Android code, which would cover an entire book.

Microsoft and Xamarin joined together in a partnership to make better tooling integration in Visual Studio 2015. However, the Xamarin suite does not ship with Visual Studio and is a product that you must purchase separately. For building apps for Windows 8.x and Windows Phone 8.x with either XAML/.NET or HTML5/JavaScript, the MSDN library provides very good documentation. That said, in this chapter we focus on new tools for building apps using the JavaScript APIs from Apache Cordova with Visual Studio 2015, including a brand new Android emulator built by Microsoft.

Note: From now on, I will refer to Apache Cordova with either its full name or simply with Cordova.

Prerequisites

To develop cross-platform mobile apps with Visual Studio 2015 and Apache Cordova, you need to install some prerequisites. These are summarized in Table 3.

Table 3: Apache Cordova Prerequisites

Prerequisite	Description
Apache Ripple	Emulator for Android and iOS.
Google Chrome	Web browser and components required to run the Ripple emulator.

Prerequisite	Description
Apache Ant 1.8.0	Required for the Android build process.
Android SDK	Required for the Android build process and for the Ripple emulator.
Oracle Java SDK 7	Required for the Android build process.
Node.js	Integrates Visual Studio with the Apache Cordova Command Line Interface (CLI) and the Ripple emulator.
Apple iTunes	Required to remotely deploy an app to an iOS device.

The Visual Studio 2015 installer will install all of the prerequisites for you, just be sure you select all the required components in the installer's welcome window.

Note: Hyper-V is another prerequisite, as it is required to run the Visual Studio Emulator for Android and emulators for Windows Phone 8.1.

Building Apps for iOS

iOS apps cannot be built on Windows. This is not a Windows limitation; it is instead the Apple license that requires developers to build apps on a Mac. For this reason, in order to use Visual Studio 2015 and Apache Cordova to build apps for iOS, you need a Mac computer and some additional configuration for remote debugging. The MSDN documentation has a specific document about configuring an environment with iOS. In this book, we do not cover iOS development because we cannot assume readers have a Mac and because we focus on tools that you can use on Windows and Visual Studio. In this chapter, we talk about Android development with Visual Studio 2015.

Building Apps with Apache Cordova

Apache Cordova is an open-source project that provides a rich set of APIs that enable developers to access native device functions from JavaScript code. For instance, with these APIs you can easily access the device's camera or other sensors. When using Apache Cordova, you basically use JavaScript and do not write a single line of native languages like Java, Objective-C (iOS), or C#. Cordova's JavaScript APIs only allow accessing device capabilities, so you need a combination with a UI framework such as jQuery Mobile or Dojo Mobile. The combination of Cordova and a UI framework allows for writing mobile apps with HTML, CSS, and JavaScript. Both Cordova's JavaScript APIs and UI frameworks are consistent across multiple device platforms, so an app that you build with Cordova can run on multiple operating systems and devices. This is where Microsoft Visual Studio 2015 comes in. The IDE integrates the usual, powerful visual environment with all of its tools with Apache Cordova to provide the best development experience for cross-platform mobile applications, and makes it easy to integrate Cordova projects with UI frameworks. After all, Visual Studio has had enhanced support for JavaScript for many years, with continuous improvements made to the code editor. Of course, we assume that you know how to code in JavaScript to take advantage of the features described in this chapter.

Visual Studio Tools for Apache Cordova

Behind the scenes, Visual Studio 2015 is able to support development with Cordova via Visual Studio Tools for Apache Cordova, an extension that is part of the IDE. Among others, these tools offer a project template called **Blank App (Apache Cordova)**, which is located in the **Apache Cordova Apps** node under the **JavaScript** folder, in the **New Project** dialog (see Figure 94).

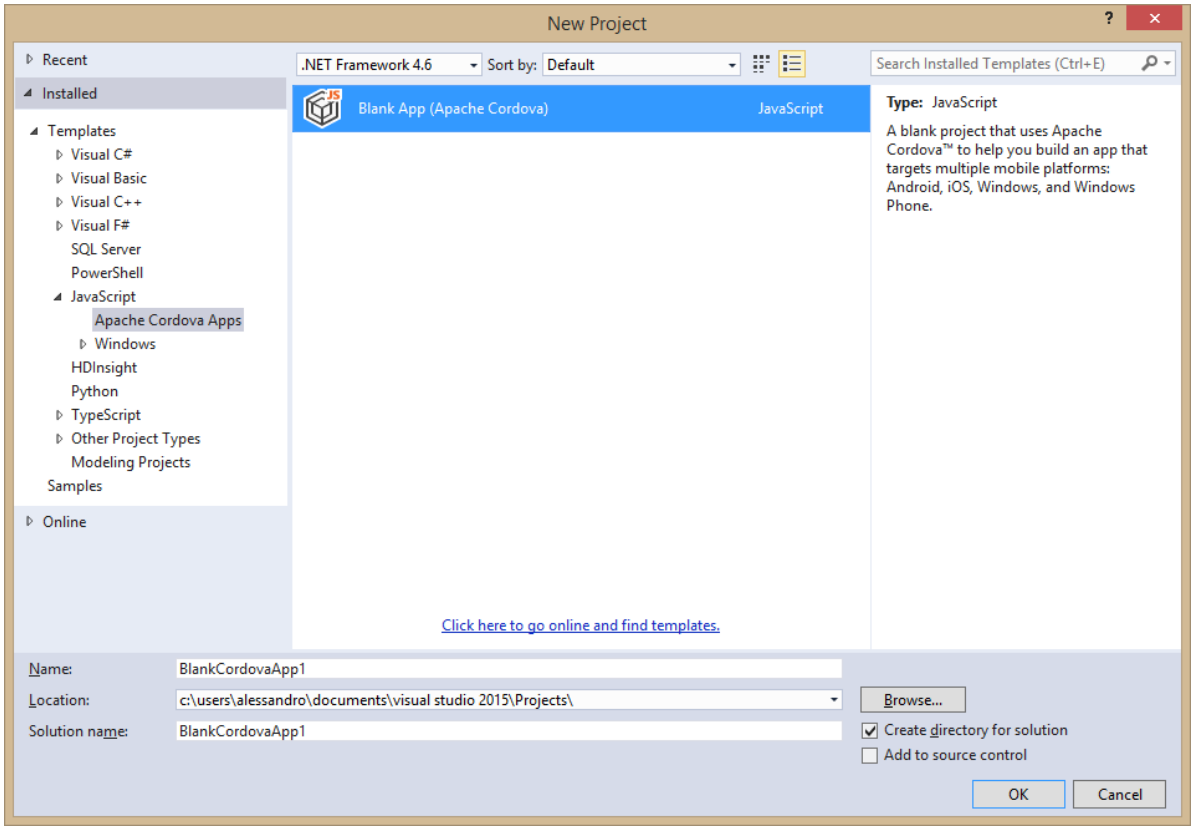

Figure 94: The Apache Cordova project template.

When you create a Cordova project, Visual Studio shows a “Getting started” page with many useful links; you will see many common elements in the Solution Explorer (see Figure 95) like:

- An **index.html** file, which is the home screen for your app.
- A **config.xml** file, which contains configuration information and that can be edited with a visual editor simply double-clicking it.
- Platform-specific folders containing optimized icons, resources, and scripts for Android, iOS, and Windows/Windows Phone.
- A **res** folder, which contains platform-specific JavaScript code. It is organized in subfolders, each targeting a specific platform.
- A **script** folder, the default location for JavaScript or TypeScript files.

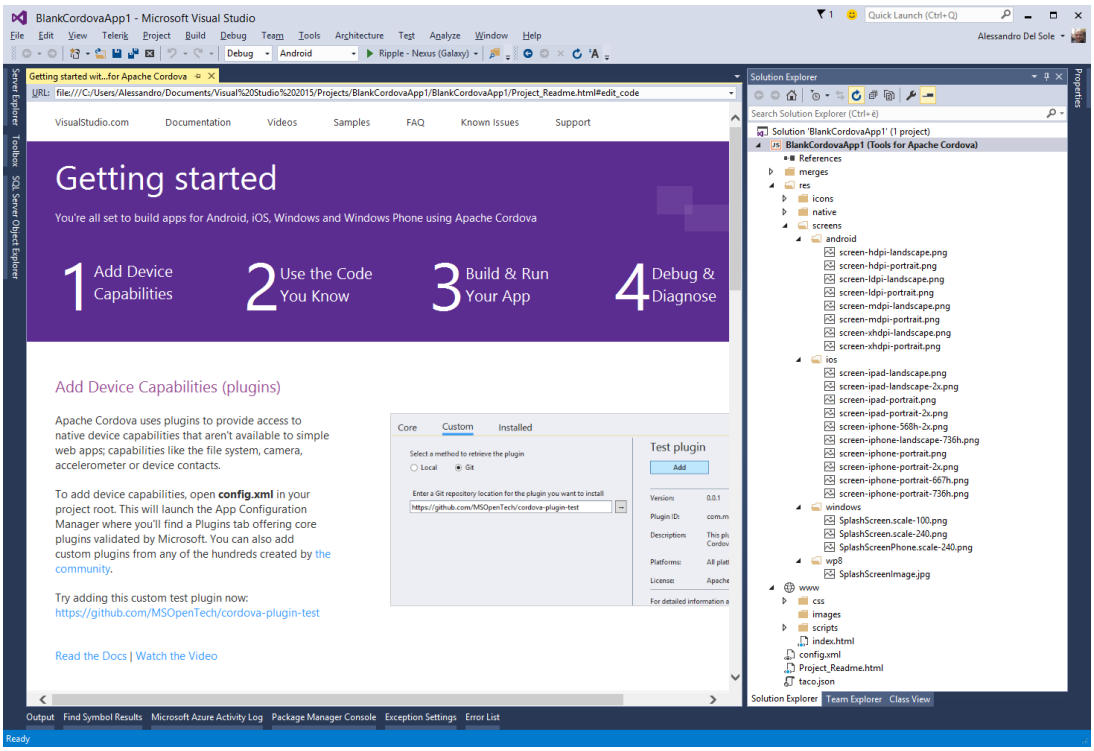

Figure 95: The Apache Cordova project structure.

Configure the project by double-clicking the **config.xml** file. When you do this, Visual Studio 2015 shows a special designer, which is shown in Figure 96.

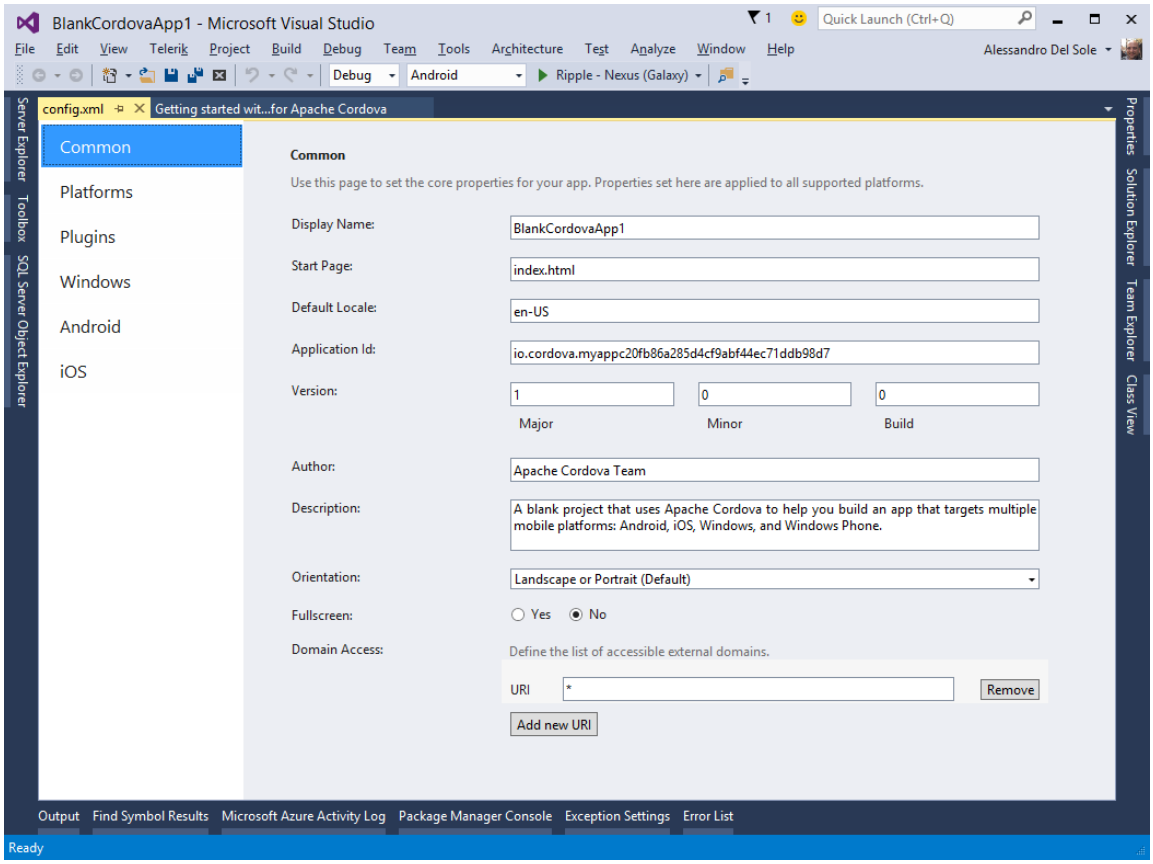

Figure 96: Configuring common app properties.

The configuration designer is made of six tabs. The Common tab is about common properties, which are all self-explanatory. The Platform tab allows you to specify the version for the Cordova CLI (Command-Line Interface), the infrastructure required to build and test Cordova apps. The Plugins tab (see Figure 97) is very important because it allows you to add plugins; each plugin provides an API to access specific device features.

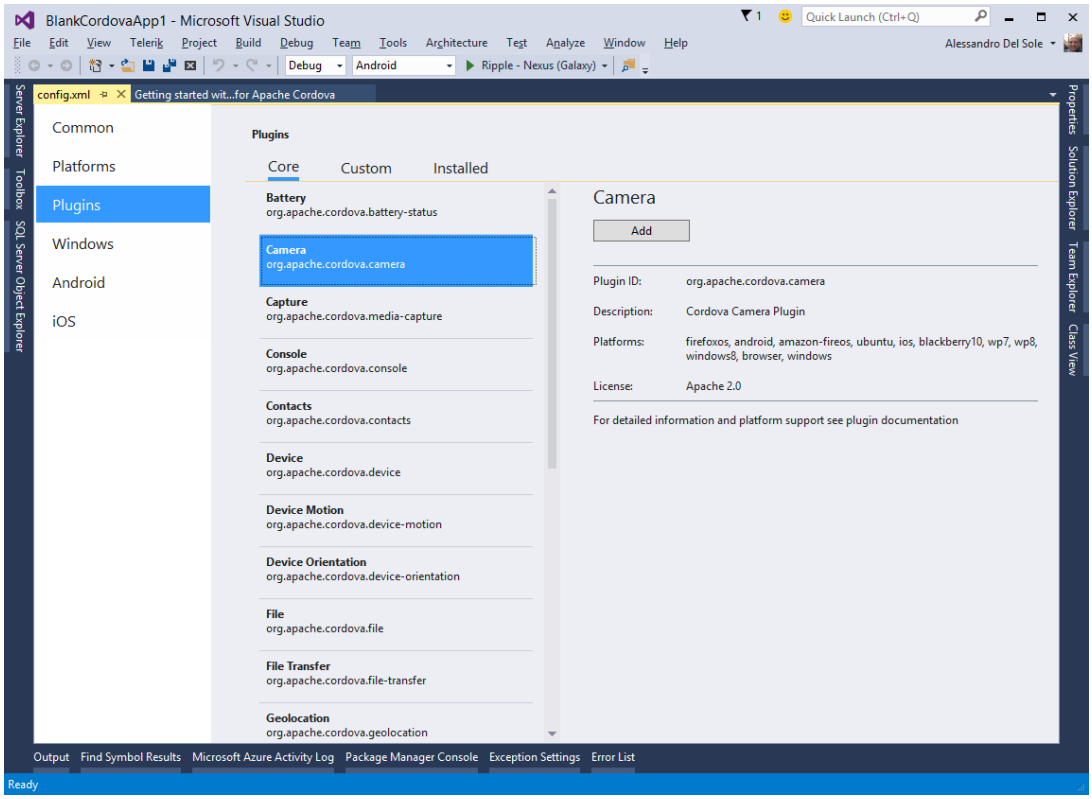

Figure 97: Adding device access functionalities with plugins.

It is worth mentioning that, for every plugin, Visual Studio 2015 shows the supported platforms. When you add a plugin, Visual Studio adds to the project a number of JavaScript files and platform-specific files that you can easily discover in Solution Explorer. The other tabs allow setting platform-specific options, especially information for app stores. Exploring those tabs is left to you as an exercise. To understand how testing apps works, a good idea is to use one of the existing examples for Apache Cordova.

Note: Using an existing example is also a good idea because this book doesn't explain plugins or detail all the Cordova APIs.

The MSDN documentation offers a sample app called ToDoList. This example is available in three different flavors, each using a different UI framework. For the current example, download the AngularJS version and extract the downloaded .zip archive to a local folder on your hard drive. The ToDoList app is very nice because it is also an example of how to use Bing Maps and connected services. Assuming you have opened the sample solution in Visual Studio 2015, at this point you want to try the sample app. Select the target operating system, which can be done via the appropriate combo box on the standard toolbar, as shown in Figure 98.

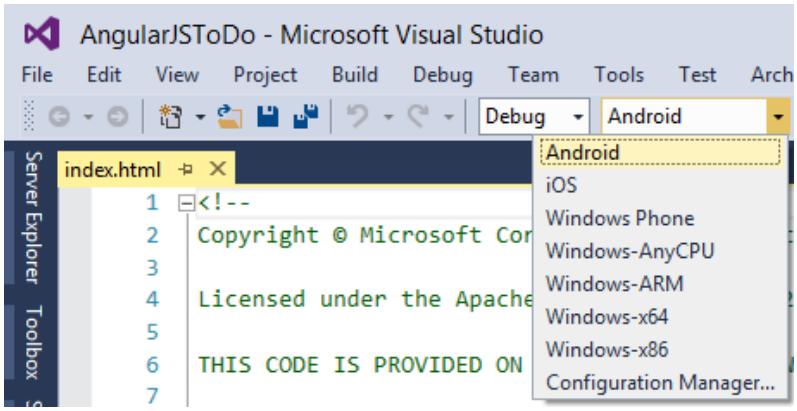

Figure 98: Specifying the target operating system.

Next, you have to select where you want to run your app, which can be a physical device or an emulator, as shown in Figure 99.

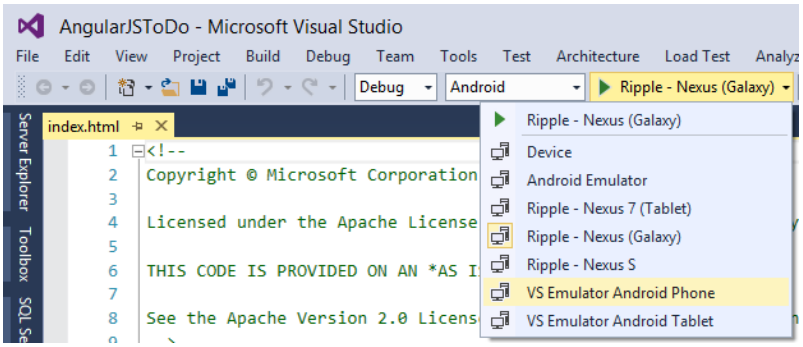

Figure 99: Selecting a physical device or an emulator.

You have several different options, especially for Android. You have the Android Emulator, which is included in the Android SDK, as well as the Ripple emulator, which ships with Cordova; finally, you have the new Visual Studio Emulator for Android. For the current example, select the **VS Emulator Android Phone** and press **F5**.

The Visual Studio Emulator for Android

In addition to the existing Android emulators, Microsoft has developed a new Visual Studio Emulator for Android. Figure 100 shows the sample app running inside this new emulator.

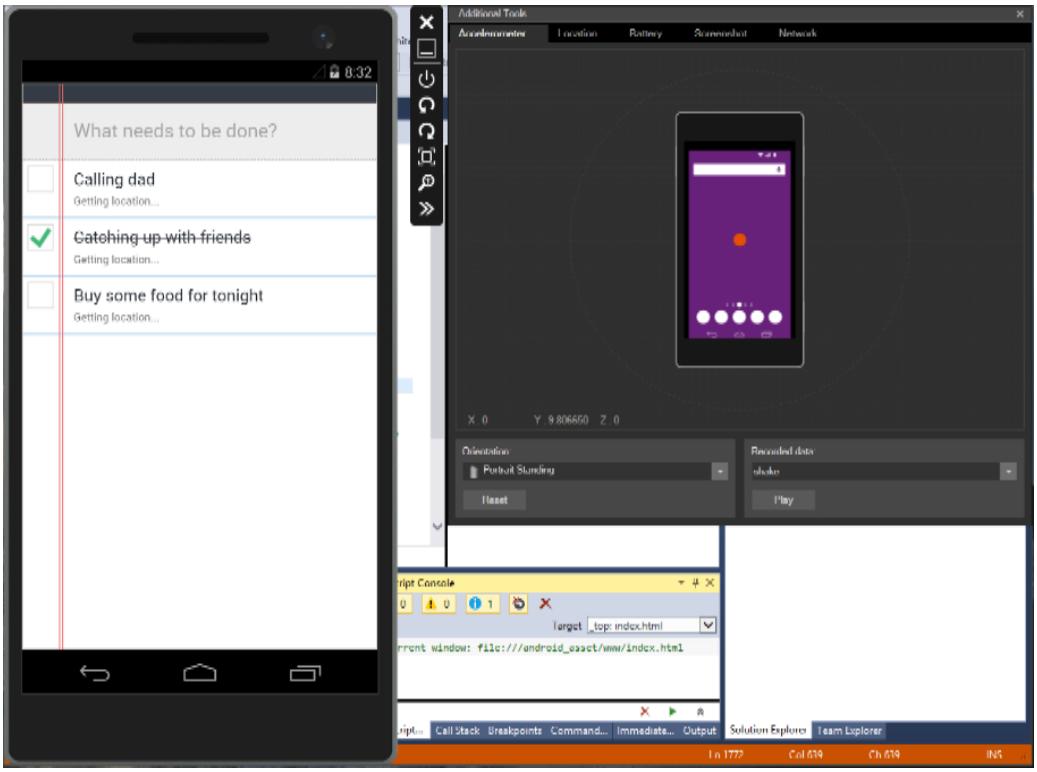

Figure 100: The Visual Studio Emulator for Android.

If you have built apps for Windows Phone, the Android emulator will look familiar as they have similar layouts and tools; both include the ability to simulate sensors like the accelerometer and location, or to take screenshots. This emulator also offers an important feature for testing, which is adjusting the battery level. The emulator also has support for camera, simulating an SD card, audio playback, and keyboard input. On the Visual Studio Blog you can find a detailed blog post about available features in the Android emulator. Behind the scenes, Visual Studio creates a virtual machine hosted in Hyper-V. Additional tutorials and code examples can be found in the MSDN documentation.

Packaging and Deploying Apache Cordova Apps

As a general rule, mobile apps must be packaged into a proper binary file and deployed to the platforms' stores. Android, iOS, and Windows apps have different characteristics and different stores. Fortunately, Visual Studio 2015 makes it easy to create packages for all the supported platforms by editing the config.xml file via a convenient designer. When you build the solution, Visual Studio creates a subfolder called bin where you can find packages for each platform. Once you have built your project, you will need to deploy the app package to the appropriate store, as you would normally do. Additional information on building packages can be found on the specific MSDN page.

Chapter Summary

Cross-platform mobile development is a key topic when talking about Visual Studio 2015. This chapter gave you an overview of the possibilities you currently have to build mobile apps with Visual Studio. You then learned about new features for building mobile apps for Android and iOS, as well as for Windows 8.x and Windows Phone 8.x, with the Visual Studio Tools for Apache Cordova. You have project templates, the complete Visual Studio tooling, and now a brand new Visual Studio Emulator for Android, which simplifies the process of testing your apps by simulating device capabilities.