

This may be out of date. [Read what's new here!](#)



THE DEVELOPER'S GUIDE TO THE NEW .NET

By Michael Crump and Sam Basu



CONTENTS

The Future Is Upon Us

Introduction	3
.NET Goes Open Source	3
Taking a Look at What OSS Means	3
The 10K-Foot View	4
.NET Goodies	5
.NET Foundation	5
Cross-Platform	5
ASP.NET vNext	5
Tooling	6

Windows 10

Introduction	7
A Package Manager Built-In	8
New Console Improvements	8
Modern Mode vs. Desktop Mode?	9
A Better Task Manager	11
Multiple Desktop Support	11
At the End of the Day...	11

Visual Studio 2015

Introduction	12
Custom Window Layouts	13
Better Code Editor	14
Expanded Shared Projects Templates	15
Intellisense for Bower and NPM	16
Debugging Lambdas	18
A Quick Look at Blend for	
Visual Studio 2015	19
Wrap-Up	20

C# 6.0

Introduction	21
Diving in Feet First	22
Static Using Syntax	23
Auto-Property Initializers	23
Dictionary Initializers	24
Exception Filters	25
Async in a Catch and Finally Block	25
Name of Expressions	26
String Interpolation	27
More to Come	27

Roslyn

Introduction	28
Getting Started	29
I See a Compiler!	30
Taking a Look Under the Hood	32
SemanticModels	35
Next Steps for Exploring Roslyn	35

.NET on a Mac

Introduction	36
Windows Finds the Perfect Hosts	37
Visual Studio “Monaco” Editor	38
A Look at ASP.NET	42
OmniSharp	44
Sublime Text	44
Conclusion	46

Wrapping Up

The Future Looks Very Bright for .NET Developers	47
Using the Telerik Stack to Be More Productive	48

THE FUTURE IS UPON US

Introduction

Are you a .NET developer? If so, I'm sure you're hearing a lot of buzz lately, but may feel a little befuddled by all that's going on. Major changes can bring initial uncertainty and hesitancy.

However, allow me to prove to you why this is one of the best times to be a .NET developer. This is no fluff—just a developer-to-developer breakdown of what's in store. The future of .NET is awesome, and I think you'll be glad you're a part of it.

.NET Goes Open Source

November 12, 2014 will be marked as a day of monumental shift in Microsoft development stacks. At the [ConnectO](#) event in NYC, it was [announced](#) that the core of your beloved .NET Framework is now entirely open sourced and usable under a [MIT license](#). This will include everything needed to execute .NET code—including the Common Language Runtime ([CLR](#)), Just-In-Time Compiler ([JIT](#)), Garbage Collector ([GC](#)) and core .NET base class libraries.

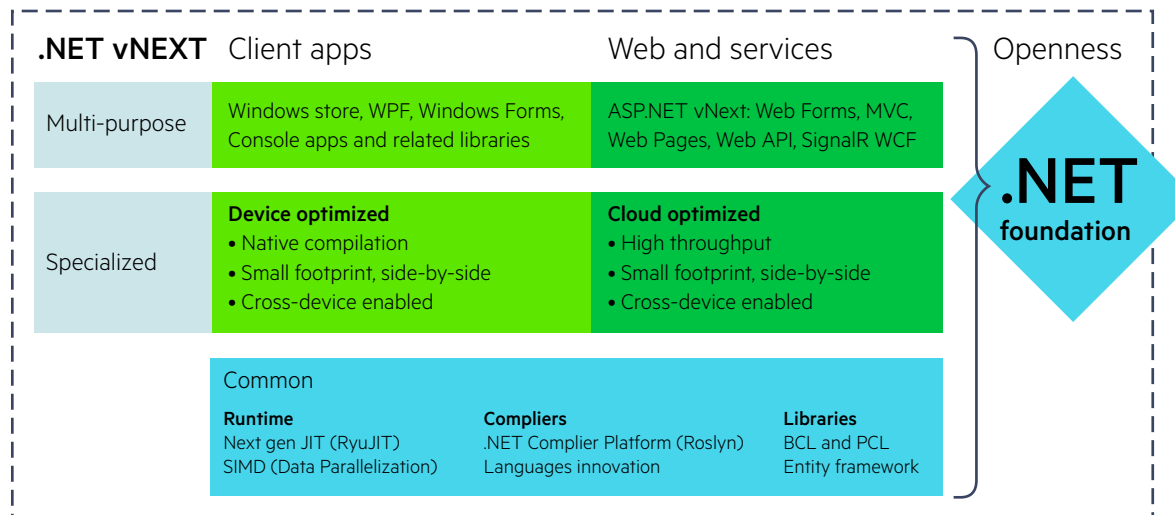
So, let's embrace this openness and [grab the source code](#), use it, build it or fork it. Microsoft is committed to accepting meaningful community contributions, and you can be assured of the quality of the .NET base classes. Every .NET developer is encouraged to check out the keynote and breakout sessions now available on-demand at the [ConnectO](#) website.

Taking a Look at What OSS Means

As a refresher, Open Source Software ([OSS](#)) means that the software's source code is available publicly and usable (to study/change/distribute) under a variety of license constraints. One fundamental trait of OSS is it's often developed in a collaborative manner, thus leveraging continuous feedback. Don't like something? Pull down the source code and make changes to fit your needs. OSS benefits tremendously from developer community involvement: meaningful changes can be accepted back into the primary source code and made available to all users, or projects can be forked, creating entirely new offshoots.

Some of you may believe that managing OSS is tricky or that it's often just a marketing gimmick. Sure, managing open source projects presents new challenges, and some companies have cynically open-sourced projects while not wholeheartedly buying into the spirit of OSS. And, enterprises need to consider the legal implications of OSS in their development stacks. Still, I believe the overall benefits of open source far outweigh anything else.

The 10K-Foot View



The representation of .NET vNext above depicts how the framework is moving forward—it is very familiar and different at the same time. Fundamentally, it is the role and usage of .NET that is changing to offer increased flexibility. .NET used to be a behemoth, serving desktop, web or mobile app development and server installations with equal footing. Moving forward, .NET will be much more specialized, serving cloud,

devices and servers with optimizations. Powering .NET will be a common set of features (runtime, compilers and base libraries), but you get to pick and choose exactly what you want to use. Are you waiting on IT or have an elaborate process before you upgrade the .NET Framework on your server? Now you can roll in the .NET framework self-contained into your apps and have multiple versions side-by-side.

.NET Goodies

The .NET Framework vNext packs a punch when it comes to some new language and compiler-level enhancements. For starters, there is [Roslyn](#)—the .NET compiler platform. Roslyn allows for innovative C# or VB compilations in the cloud and has loads of benefits inside IDEs like Visual Studio, as well as third-party integrations. Windows Store apps benefit from [.NET Native](#) ahead-of-time compilation, resulting in significant performance improvements with quicker app start-up times and smaller memory footprint. Desktop and server apps benefit from next-generation 64-bit [RyuJIT](#) (Just-In-Time) compilers. All this goodness for .NET is spreading to all application types and leading to a convergence of development experiences.

.NET Foundation

Back at [Build 2014](#) conference, [Scott Guthrie](#) announced the .NET Foundation. Quoting the [Home page](#):

“The .NET Foundation was created as an independent forum to foster open development and collaboration around the growing collection of open source technologies for .NET.”

The Foundation has since garnered a lot of support to host an impressive array of [OSS projects](#), both from Microsoft and supporting partners, and serves as the de facto steward for open source .NET repositories.

Cross-Platform

We do not live in a silo and Windows isn't the only thing in the world—this mindset change within Microsoft is evident in what's next for the .NET Framework. Official distributions of .NET will be available for Linux and OS X! Want me to repeat that? Let it sink in, because it is a big deal.

Cross-platform developers are now welcome to use the .NET framework on a platform of their choice. Web, desktop, cloud or mobile development on almost any platform can now be targeted with the .NET framework. You can build native ASP.NET web applications on a Mac. You can build native iOS or Android applications using C#, along with Windows counterparts. And you can build Hybrid single-codebase cross-platform mobile apps using plain HTML/CSS/JS and the [Apache Cordova](#) open source framework. This is a huge and welcome change and can potentially spread the use of .NET to non-Microsoft developers.

ASP.NET vNext

Perhaps the biggest shift in mindset is evident in what's next for [ASP.NET](#). Don't fret though, as it's just offering flexibility. Let me explain with few points:

1. Per the trend, ASP.NET vNext is entirely [open source](#); most of the code is available in the .NET Foundation. You need not have insider access anymore—simply look at the source code and build, change, pull or fork.

2. ASP.NET vNext sees the convergence of several frameworks into a unified programming model for MVC 6. This includes latest MVC, WebPages, Web API, SignalR and Entity Framework. You can now have a single controller that returns both MVC views and formatted Web API responses on the same HTTP verb.
3. ASP.NET is completely modular: you pick the pieces you need and get them through NuGet. The future of [.NET on the server](#) looks interesting, to say the least.
4. There is a brand-new HTTP processing pipeline with terrific throughput. The new [KRuntime](#) is the core of ASP.NET vNext—it comes with built-in [Version](#) and [Package](#) managers along with loads of other tooling.
5. ASP.NET applications can be hosted in IIS or outside in its own processes. Yes, they run on Mono in Mac and Linux.
6. There are a lot of [tooling improvements](#) for ASP.NET in Visual Studio. But you can also [build ASP.NET applications natively on a Mac](#). Simply use the Command Line Interfaces (CLI) and tools like [Sublime Text](#).
7. Web Forms are alive and healthy. Instead of using the modular .NET framework, simply use the whole, just like old times. All your tools and extensions for web forms carry forward.
8. ASP.NET vNext is cloud ready. Features like Session State and Caching provide consistent APIs and adjust themselves with behavior well-suited for cloud hosting versus traditional servers. Dependency injection is rampant within ASP.NET: plug-and-play to your heart's desire.

9. Skip the Build step: simply make code changes and refresh your browser to see them, courtesy of JIT compilation from Roslyn, the new .NET compiler platform.
10. ASP.NET MVC syntax gets those little magical [Tag Helpers](#). Take time to learn about them, since they're quite powerful.

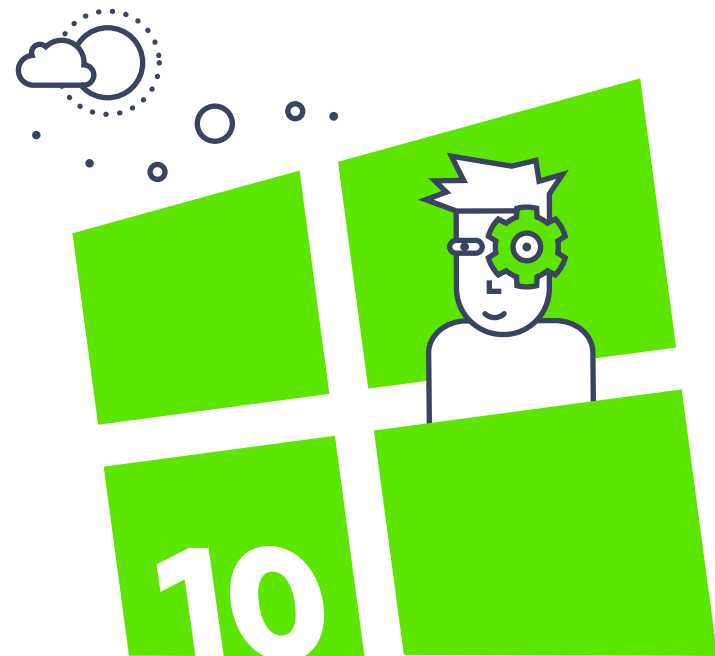
Tooling

There are lots of tooling improvements for .NET developers inside Visual Studio, but the big news about a brand-new Visual Studio SKU trumps everything else. Remember the VS Express editions? They were free, for sure, but you could only develop one type of application with each specific VS Express edition. Now, there is the new [Visual Studio Community Edition](#)—the one SKU to rule them all! Yes, it's free, and you can develop any type of .NET application with it: web, cloud, desktop or mobile. It is a full-featured IDE and supports advanced features like plugins and extensions.

WINDOWS 10

Introduction

Windows 10 was unveiled on September 30, 2014 as a [technical preview](#) for individuals that signed up to become a Windows Insider. Since the release, we've seen a lot of coverage from media outlets regarding new features that end users will be excited about, but very little coverage for developers. In this ebook, I'm going to point out several things that caught my eye as a developer working daily with the Microsoft stack.



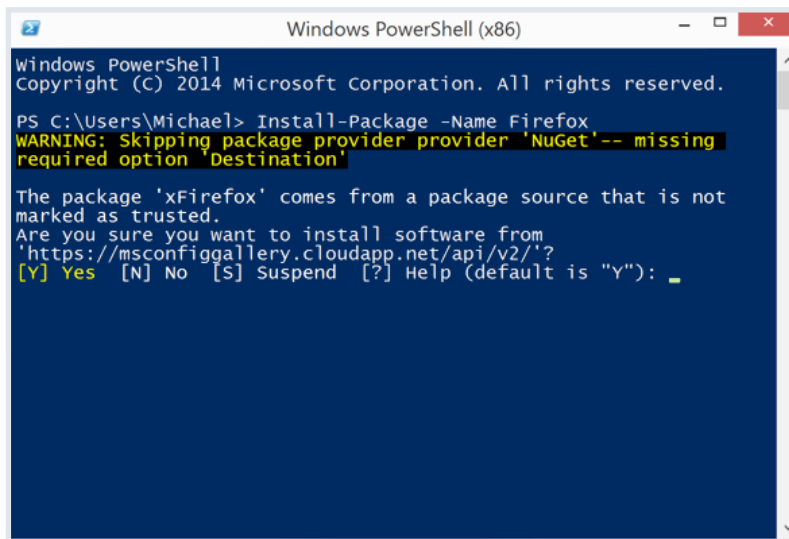
A Package Manager Built-In

Developers have grown to love package managers inside IDEs to install frameworks, libraries and so on. They also enjoy OS package managers to quickly find and install third-party applications. With Windows 10, developers finally get one.

OneGet is a package manager included in Windows 10 that allows you to install additional software from the PowerShell command line. Simply open a PowerShell window and type the following command:

```
Install-Package -Name Firefox
```

In Windows 10, you will have access to this, as shown below. You will want to read this [blog post](#) for more details on what commands [OneGet](#) offers.



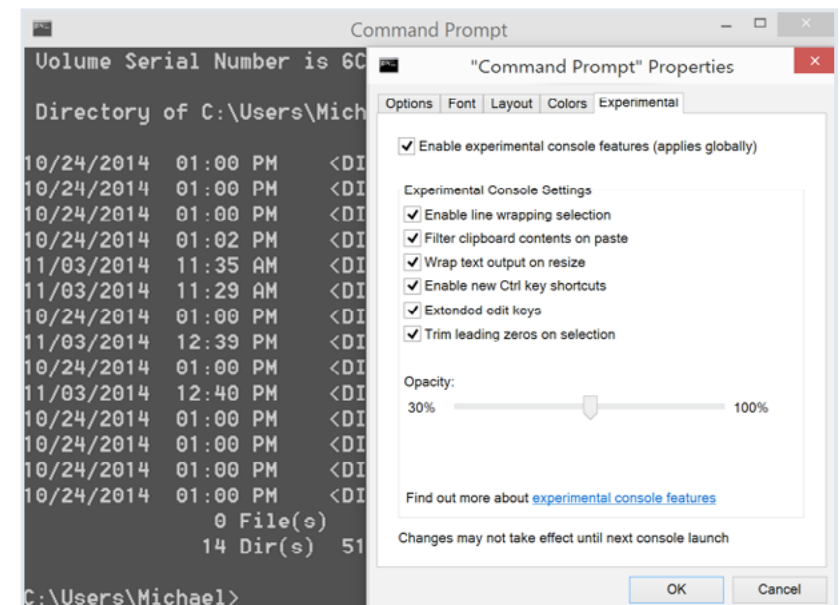
```
Windows PowerShell (x86)
windows PowerShell
Copyright (C) 2014 Microsoft Corporation. All rights reserved.

PS C:\Users\Michael> Install-Package -Name Firefox
WARNING: Skipping package provider provider 'NuGet'-- missing
required option 'Destination'

The package 'xFirefox' comes from a package source that is not
marked as trusted.
Are you sure you want to install software from
'https://msconfiggallery.cloudapp.net/api/v2/?'
[Y] Yes [N] No [S] Suspend [?] Help (default is "Y"): _
```

New Console Improvements

Every developer is looking for a way to enhance productivity, especially using the Windows Console. In the Windows 10 Technical Preview, you can turn on “Enable experimental console features” by right-clicking on the command prompt and going to “Properties.” There you will find an “Experimental” tab that you can turn on, as shown below.



Aside from the ability to use the control key to copy and paste text in the console, it comes with several other line features that are self-explanatory. In this example, I changed the opacity, and you can see the background coming through.

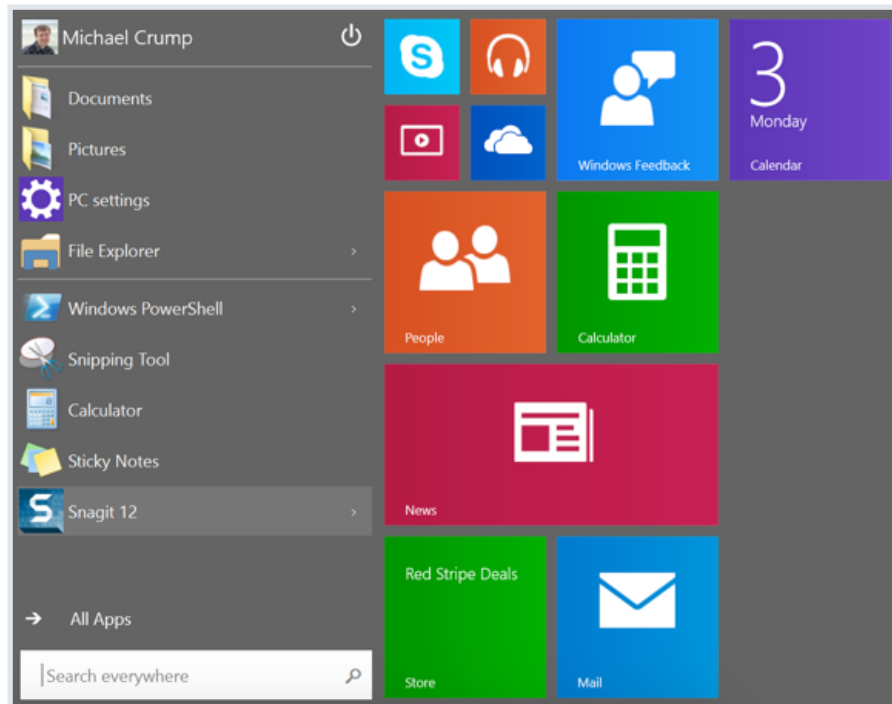
If you click on the link at the bottom of the Property page, you'll be taken to this [site](#), which enables you to provide feedback and has a comprehensive list of [console improvements](#) in Windows 10.

Modern Mode vs. Desktop Mode?

If you are a developer, one thing you are used to is switching between “Desktop” and “Modern Apps” modes. I think it is safe to assume that most developers who use Windows 8.1 stay in Desktop mode for most of the day. This is where they are the most productive and where Visual Studio, Expression Blend and Microsoft Office live. Although you may switch back to the Modern App mode to run Windows 8.1 apps, search or shutdown the PC occasionally, it’s rare (at least it is for me).

With Windows 10, you can change the signed-in user, turn the PC off, pin Modern Apps to the Start menu and much more, without ever leaving Desktop mode.

Windows 10 also has a Continuum mode that’s smart enough to determine if you are using a Surface or a laptop, and launch either the Desktop or Modern App mode, automatically. If you are using a Surface and attach a keyboard, it will automatically go into Desktop mode—remove the keyboard and it switches back. The graphic below shows the new Start button in Windows 10.



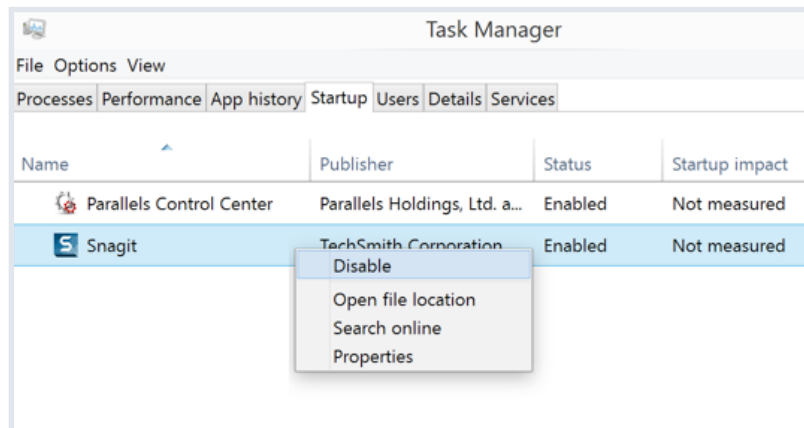
You also have the ability to search everywhere on the local machine and even the Internet. This plays a major role if you created a modern app, as it can be discovered from the Start button. Imagine that you need a tip calculator; if you type “tip calculator” in the box, it will search your local machine for one. If it can’t find one, it gives you Bing search results along with a suggested app in the Windows Store.

Once you launch a modern app, you are given several options previously only available via the share charm, as shown below.



A Better Task Manager

While not exactly new to Windows 10, a better task manager has evolved. How many times have you wanted an easy way to disable applications that run automatically, or learn more detailed information about a program or the impact of it starting up? As a developer, I've invested in the latest hardware and don't want an app to slow my system down or a suspicious program that might be a virus. Thankfully, you can take care of both of those issues with the Task Manager, as shown below.

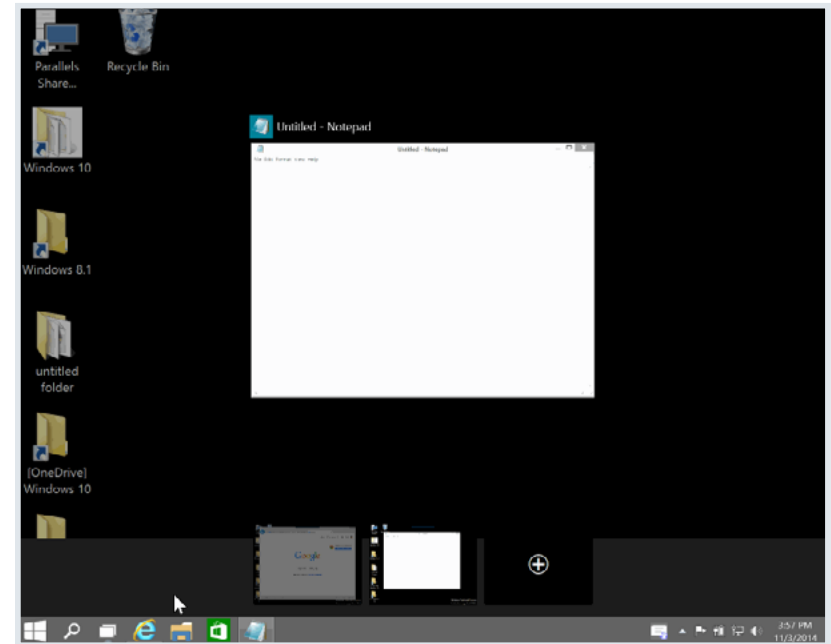


Multiple Desktop Support

Another feature I found extremely helpful is multiple desktop support. Imagine you want to have multiple desktops configured with certain apps and be able to toggle through them as needed. With the Windows 10 Technical Preview, it's very easy to do.

Simply select Task View, then Create Desktop and place the applications in it, as needed. You can use the keyboard shortcut: WINKEY + Ctrl +

Left Arrow or WINKEY + Ctrl + Right Arrow to toggle between desktops. You can even move a window to another desktop: right-click then select Move to and the desktop of your choice.



At the End of the Day...

Even though Windows 10 is brand-new, .NET developers will continue to write the code they know and love. Thankfully, Telerik® has solutions for [Windows Universal Apps](#) that span phone, tablet and desktop to [WPF](#) and [Web Apps](#), and they're ready to implement today. Now, let's take a look at some of the improvements in Visual Studio 2015.

VISUAL STUDIO 2015

Introduction

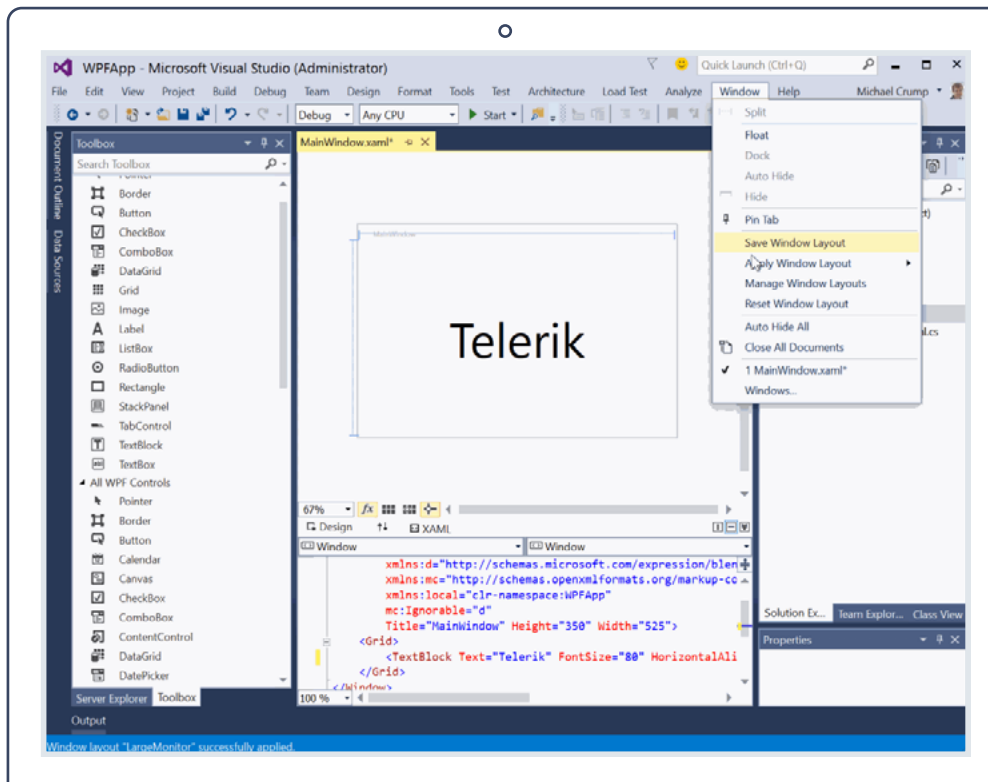
Visual Studio 2015 includes many new features that enhance the way developers work with everything, from the web and desktop to mobile apps. Several features have had the spotlight, such as gesture support in the editor, Cordova tooling, C++ enhancements and the new Android emulator. But there are several other, less talked about features that every developer using Visual Studio 2015 will love. With that said, let's jump straight in!



Custom Window Layouts

This feature comes in handy if you develop on multiple devices. Say, for example, you use a 23-inch monitor during the day and a Surface Pro to develop on your train ride home. You can quickly switch between devices by going to Window -> Apply Window Layout and selecting one you created earlier. Support for keyboard shortcuts is included, enabling fast navigation to your favorite layout. Additionally, the profile roams with you, as long as you are signed into Visual Studio 2015.

Below is an example of switching between my Surface device and a desktop monitor. Notice that with the Surface device, I show only the XAML file, whereas on the larger monitor I can see everything.

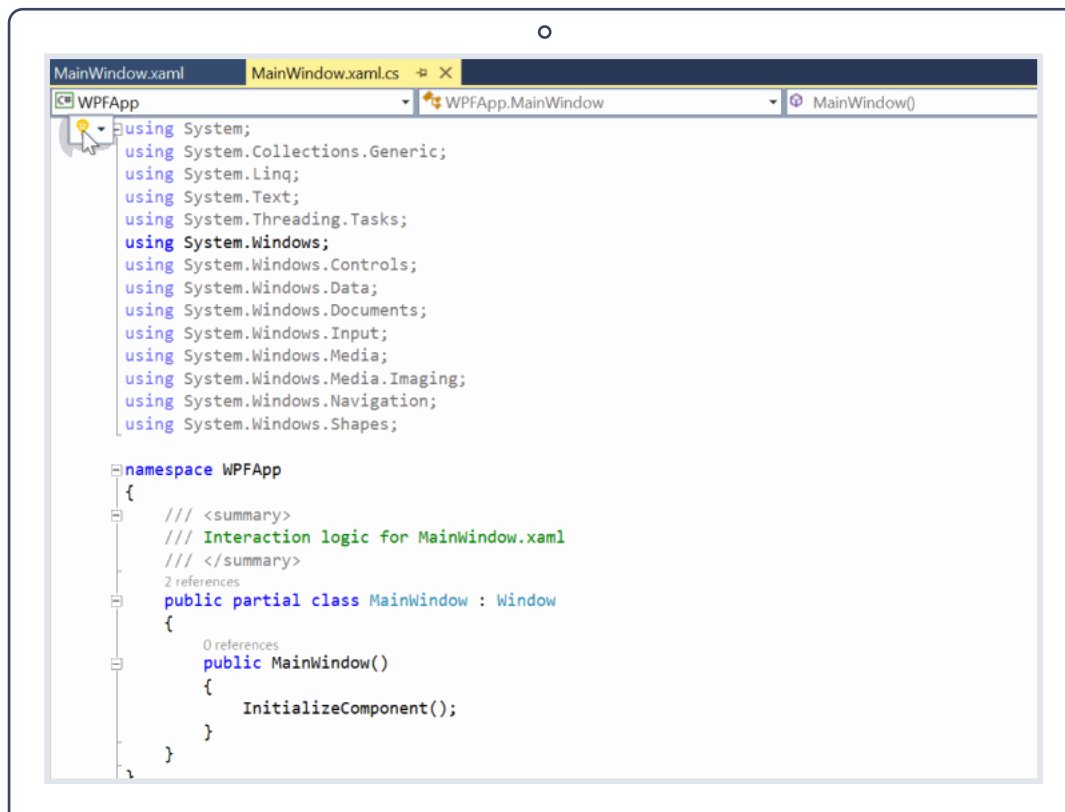


Better Code Editor

The code editor has been replaced with “Roslyn” to give you a new and improved code editing experience. Light bulbs appear when you need to include fixes to your code or refactor it. When you see a light bulb, click it for suggestions based on the code it has analyzed.

In this example, the code editor has determined that we included unnecessary “using” statements and helps remove them. You can first

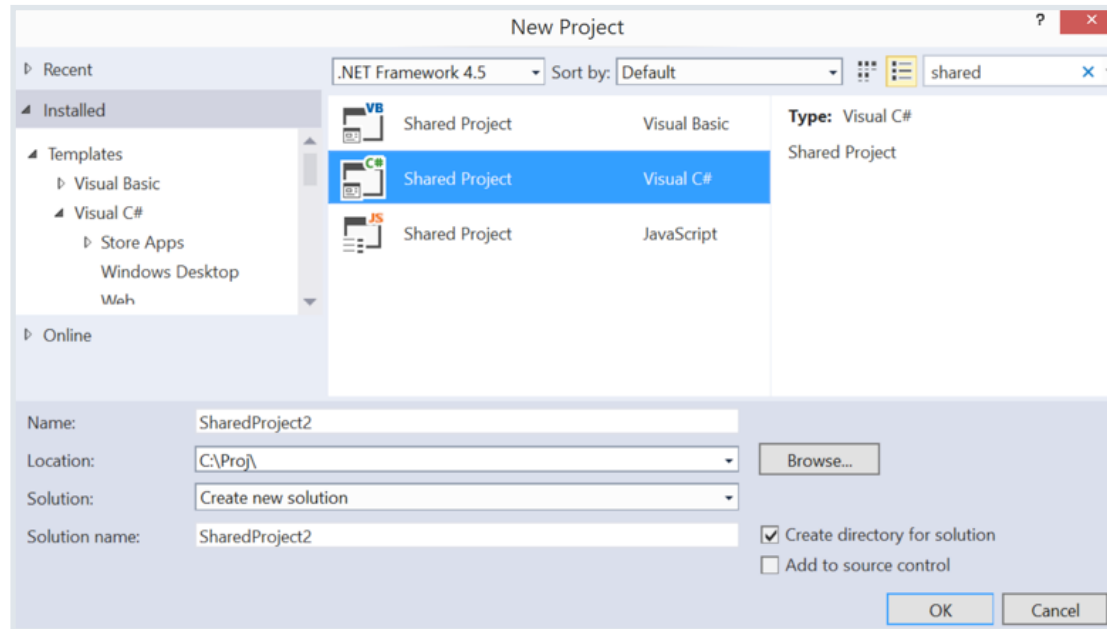
generate a preview and have the changes affect the whole document, project or solution. While those features have appeared in [JustCode](#) and earlier versions of Visual Studio for years, we will be releasing a new version of [JustCode for Visual Studio 2015](#) that will take advantage of Roslyn for enhanced productivity tools.



Expanded Shared Projects Templates

How many times have you wanted to use a shared project outside of a Windows Universal App? Now you can!

After you open Visual Studio 2015 and search for “shared,” you will see the following:



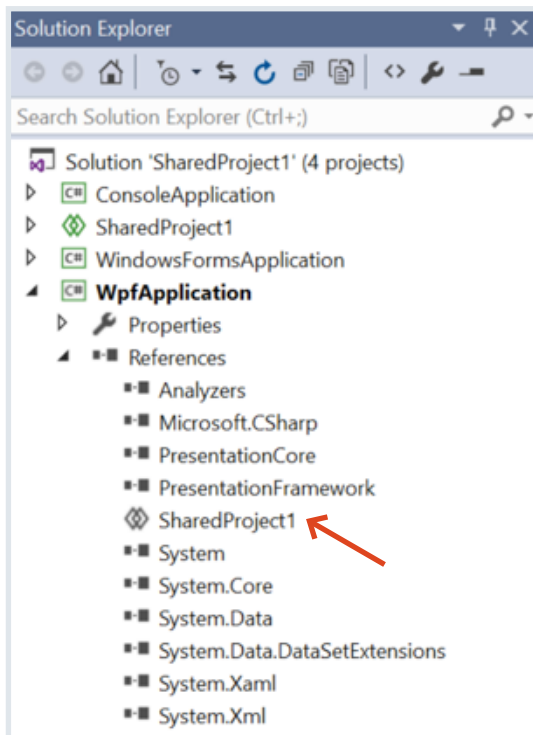
Select the Visual C# Shared Project and create a class named Person.cs and add the following code:

```
class Person
{
    public string FirstName { get; set; }
    public Person()
    {
        FirstName = "Michael";
    }
}
```

Create a new Console application and reference the shared project we just created. Now you can write code such as:

```
var person = new Person();  
Console.WriteLine(person.FirstName);  
Console.ReadLine();
```

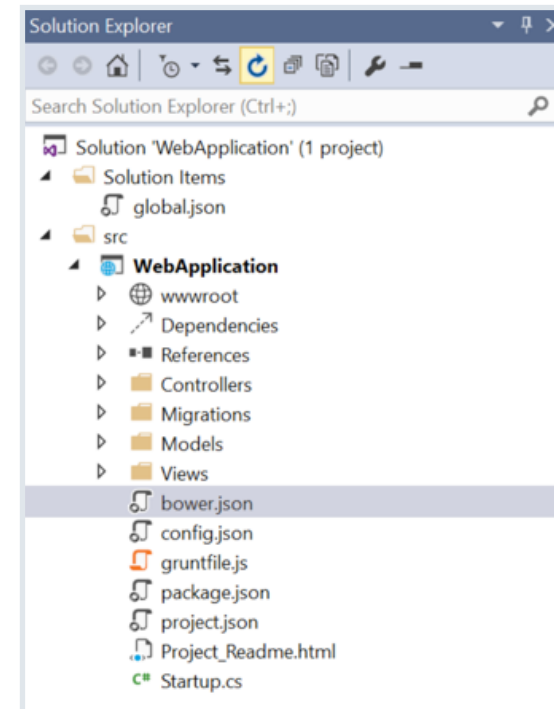
If you run the console app, it retrieves the FirstName from our shared project. Add a WPF or Windows Form application and access the Person class, as you normally would. This also works for class libraries. After you add several projects, the solution explorer looks like this:



Notice the only thing you have to do is reference the shared project.

Intellisense for Bower and NPM

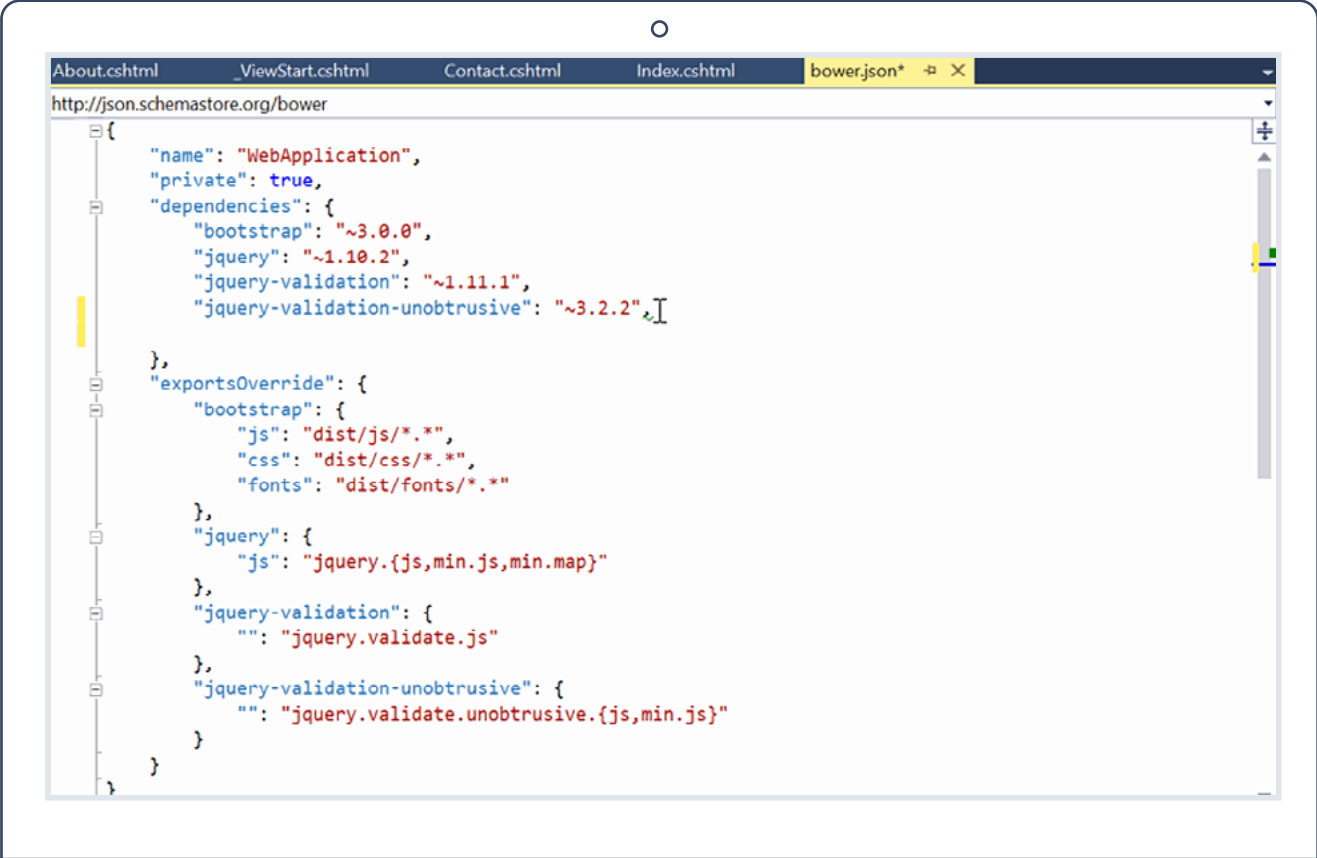
If you create a new ASP.NET 5 web project, you'll notice several items load into the template automatically:



Aside from an updated file structure, you now have a folder called **Dependencies** that contains Bower and NPM. Generally speaking, think of Bower for client-side packages (such as jQuery and Angular) and NPM for developer tools (such as Grunt and Gulp). Both of the package managers are controlled by JSON files found in the solution.

- **bower.json** for Bower
- **config.json** for NPM

To add a library using Bower, simply open the bower.json file and add the desired package. Here, we added the latest version of Angular, without going to the Angular site to download and add it manually to the project.



The screenshot shows a web browser window with the address bar displaying `http://json.schemastore.org/bower`. The browser has several tabs open: `About.cshtml`, `_ViewStart.cshtml`, `Contact.cshtml`, `Index.cshtml`, and `bower.json*`. The main content area displays the JSON configuration for a Bower project. The code is as follows:

```
{
  "name": "WebApplication",
  "private": true,
  "dependencies": {
    "bootstrap": "~3.0.0",
    "jquery": "~1.10.2",
    "jquery-validation": "~1.11.1",
    "jquery-validation-unobtrusive": "~3.2.2"
  },
  "exportsOverride": {
    "bootstrap": {
      "js": "dist/js/*.js",
      "css": "dist/css/*.css",
      "fonts": "dist/fonts/*.woff"
    },
    "jquery": {
      "js": "jquery.{js,min.js,min.map}"
    },
    "jquery-validation": {
      "": "jquery.validate.js"
    },
    "jquery-validation-unobtrusive": {
      "": "jquery.validate.unobtrusive.{js,min.js}"
    }
  }
}
```

Debugging Lambdas

Yes, the time has finally come where we can debug lambda expressions. Let's take a look at the following code:

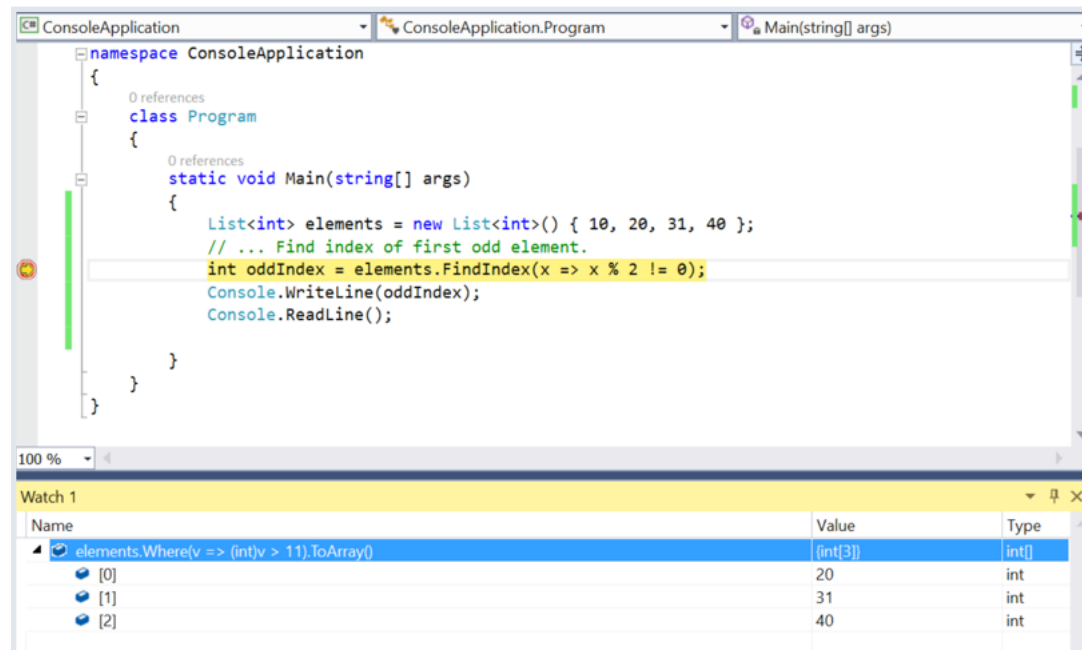
```
List<int> elements = new List<int>() { 10, 20, 31, 40 };  
// ... Find index of first odd element.  
int oddIndex = elements.FindIndex(x => x % 2 != 0);  
Console.WriteLine(oddIndex);
```

The console will return the value of 2. But what if we wanted to add a watch and perform additional analysis of the expression?

In this sample, we added a watch on the breakpoint and the following code:

```
elements.Where(v => (int)v > 11).ToArray()
```

As expected, it returned three items with a value greater than 11. This information is useful in the current and other debugger windows. The capability is supported in C# and Visual Basic.



The screenshot shows the Visual Studio IDE with a debugger breakpoint set on the lambda expression `x => x % 2 != 0` in the `FindIndex` method. The watch window displays the result of the lambda expression, which is an array of integers: `{ 20, 31, 40 }`.

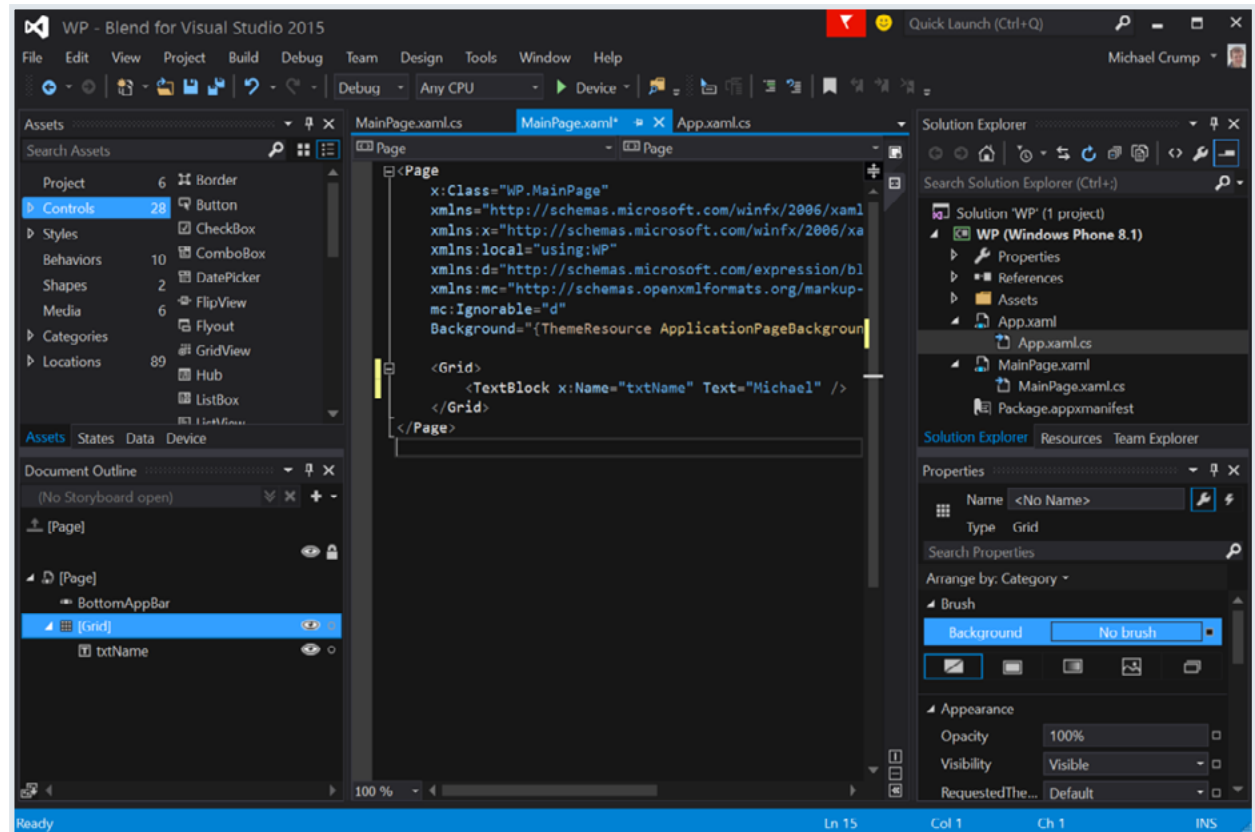
Name	Value	Type
<code>elements.Where(v => (int)v > 11).ToArray()</code>	<code>{int(3)}</code>	<code>int[]</code>
<code>[0]</code>	<code>20</code>	<code>int</code>
<code>[1]</code>	<code>31</code>	<code>int</code>
<code>[2]</code>	<code>40</code>	<code>int</code>

A Quick Look at Blend for Visual Studio 2015

Blend comes with several enhancements but, by far, the one that was most-needed was the UI overhaul. Blend includes most of the functionality that we have grown to love in Visual Studio.

Some notable features are:

- Basic debugging support
- Peek in XAML
- Custom Windows layouts (see feature #1)
- Source control
- NuGet
- XAML IntelliSense



```
MainPage.xaml* App.xaml.cs
Page
<Page
  x:Class="WP.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="using:WP"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  mc:Ignorable="d"
  Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
  <Grid>
  </Grid>
</Page>
```

After just a few minutes of playing with the new Blend, I can tell Microsoft is committed to making the experience similar that of Visual Studio.

Wrap-Up

By far, this is the best Visual Studio to date! We've only had a taste of what Visual Studio 2015 has to offer. I encourage you to check the documentation to learn about other features not included in this ebook.

Let's switch gears and take a look at some of the language features included in C# 6.0.

C# 6.0

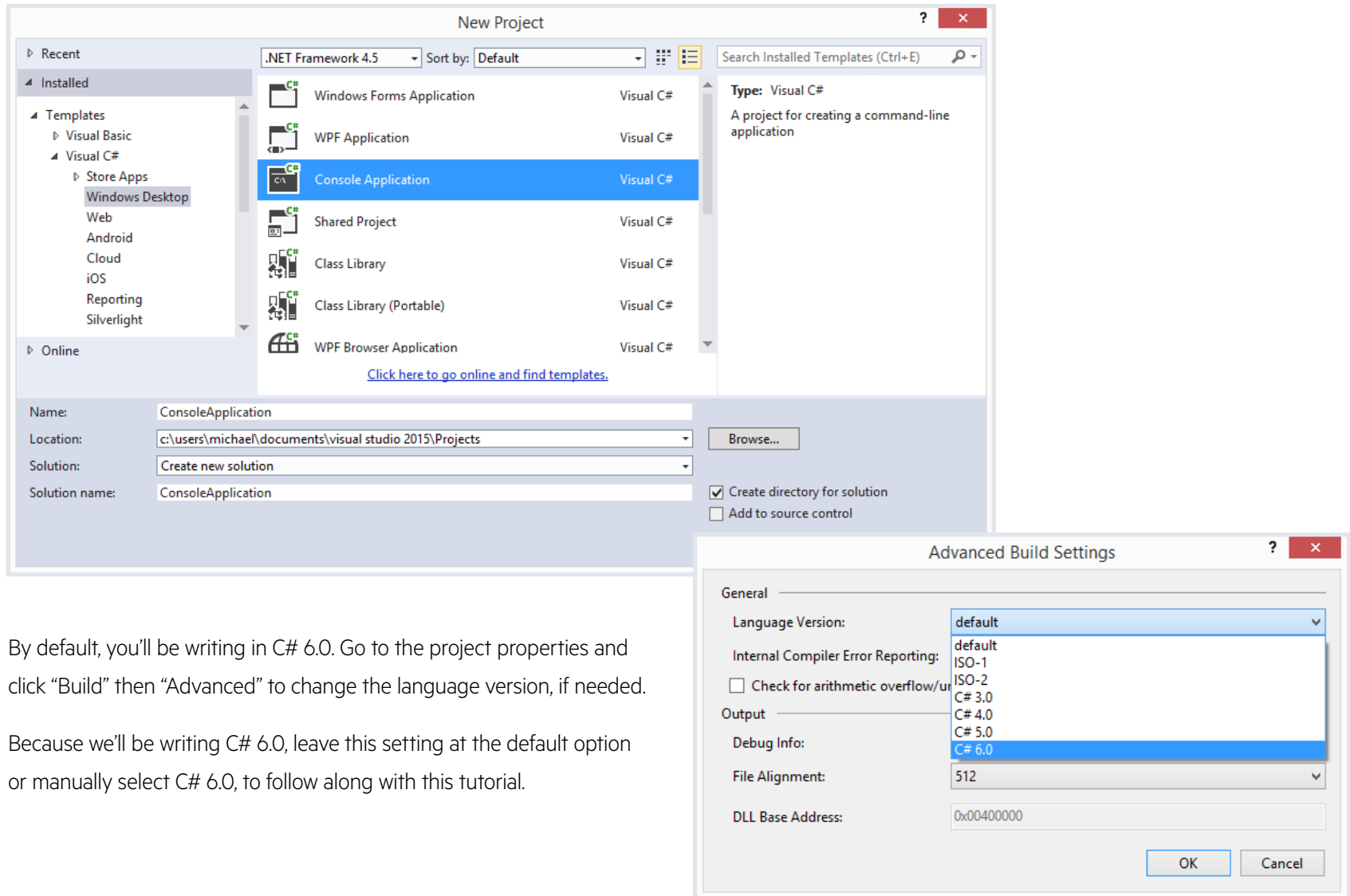
Introduction

After you install Visual Studio 2015, you can begin exploring the new language features found in C# 6.0. Although there haven't been as many changes to the language as in previous versions of C#, several features are worth noting. In this section, I'll cover some of the language enhancements in C# 6.0.



Diving in Feet First

Open Visual Studio 2015 and create a new C# Console Application, as shown below. Don't worry about changing the .NET Framework version number.



By default, you'll be writing in C# 6.0. Go to the project properties and click "Build" then "Advanced" to change the language version, if needed.

Because we'll be writing C# 6.0, leave this setting at the default option or manually select C# 6.0, to follow along with this tutorial.

Static Using Syntax

In previous versions of C#, we would have to add the proper using statement, such as System.Console, then write:

```
Console.WriteLine("Hello TDN!");
```

With C# 6, you can add the using statement and reference the WriteLine method by itself, as shown below:

```
using System.Console;

namespace TDNCSharpSix
{
    class Program
    {
        static void Main(string[] args)
        {
            //Sample One
            WriteLine("Hello TDN!");
        }
    }
}
```

Auto-Property Initializers

In the past, we may have created our properties with a get-and-set, then initialized our constructor with the value that we wanted:

```
public class Customer
{
    public Customer()
    {
        customerID = Guid.NewGuid();
    }

    public Guid customerID { get; set; }
}
```

Now, we can reduce this code block to one line, as shown below. No longer do we need to create a setter or constructor.

```
public class Customer
{
    public Guid customerID { get; set; } = Guid.NewGuid();
}
```

Dictionary Initializers

Many C# developers felt the format shown below was confusing for creating dictionaries, mainly because of the use of brackets and quotation marks for the data.

```
Dictionary<string, string> customerNames = new Dictionary<string, string>()
{
    { "Michael Jordan", "Basketball" },
    { "Peyton Manning", "Football" },
    { "Babe Ruth", "Baseball" }
};
```

The compiler team decided to change this and make it more readable with the following format. This new format will only save you a few keystrokes, but it is much easier to read.

```
public Dictionary<string, string> customerNames = new Dictionary<string, string>()
{
    ["Michael Jordan"] = "Basketball",
    ["Peyton Manning"] = "Football",
    ["Babe Ruth"] = "Baseball"
};
```


Exception Filters

Exception filters have been supported in Visual Basic, but are new to the C# compiler. They allow you to specify a condition for a catch block. As shown in the following sample, the last catch statement will fire:

```
try
{
    throw new Exception("Error");
}
catch (Exception ex) if (ex.Message == "ReallyBadError")
{
    // this one will not execute.
}
catch (Exception ex) if (ex.Message == "Error")
{
    // this one will execute
    WriteLine("This one will execute");
}
```

Async in a Catch and Finally Block

Many developers will love this feature, because they often need to log exceptions to a file or database without blocking the current thread. Here is an example of how one would work:

```
public static async void DownloadAsync()
{
    try
    {
        throw new Exception("Error");
    }
    catch
    {
        await Task.Delay(2000);
        WriteLine("Waiting 2 seconds");
    }
    finally
    {
        await Task.Delay(2000);
        WriteLine("Waiting 2 seconds");
    }
}
```

Name of Expressions

How many times in your code do you retrieve the name of a member of your class? Typically, you would wrap the field in quotation marks, as shown below.

```
public static void DoSomething(string name)
{
    if (name != null) throw new Exception("name");
}
```

However, wrapping the field in quotation marks forces you to put strings in your code to represent the name of a member. In C# 6.0, there is a new operator called `nameof` that enables you to refactor code to remove those string literals. The sample shown below demonstrates how to refactor that same method.

```
public static void DoSomething(string name)
{
    if (name != null) throw new Exception(nameof(name));
}
```

The result is cleaner code and safety when retrieving member names.

String Interpolation

Prior to C# 6.0, you could concatenate two or more strings together in one of the following ways:

```
string firstName = "Michael";
string lastName = "Crump";

WriteLine("Name : " + firstName + " " + lastName);
WriteLine(string.Format("Name : {0} {1}", firstName, lastName));
```

In C# 6.0, there is a cleaner format, as shown in the first WriteLine call. Also, you can place expressions directly in the string literal to evaluate an expression:

```
string firstName = "Michael";
string lastName = "Crump";
int orderNumber = 250000;

WriteLine("Name : {firstName} {lastName}");
WriteLine("Name : {firstName} {lastName}\nDiscount : {orderNumber == 250000 ? " You
get 25% off your order!" : ""}");
```

In this sample, the console returns the following information, as it evaluates the orderNumber is equal to 250,000; otherwise won't print anything:

Michael Crump

Discount : You get 25% off your order!

More to Come

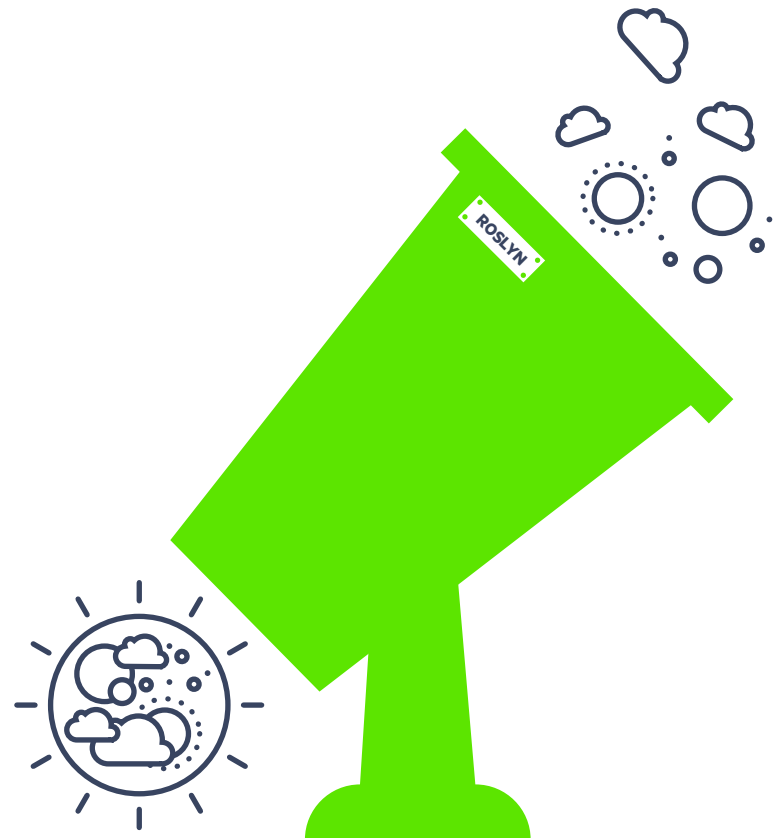
It looks like C# 6 will be the starting point for many new projects in 2015. While I touched on several of my favorite language features, there are many more. I encourage you to keep watch on <http://msdn.microsoft.com/> for updated information.

Speaking of languages, let's take a look at Roslyn.

ROSLYN

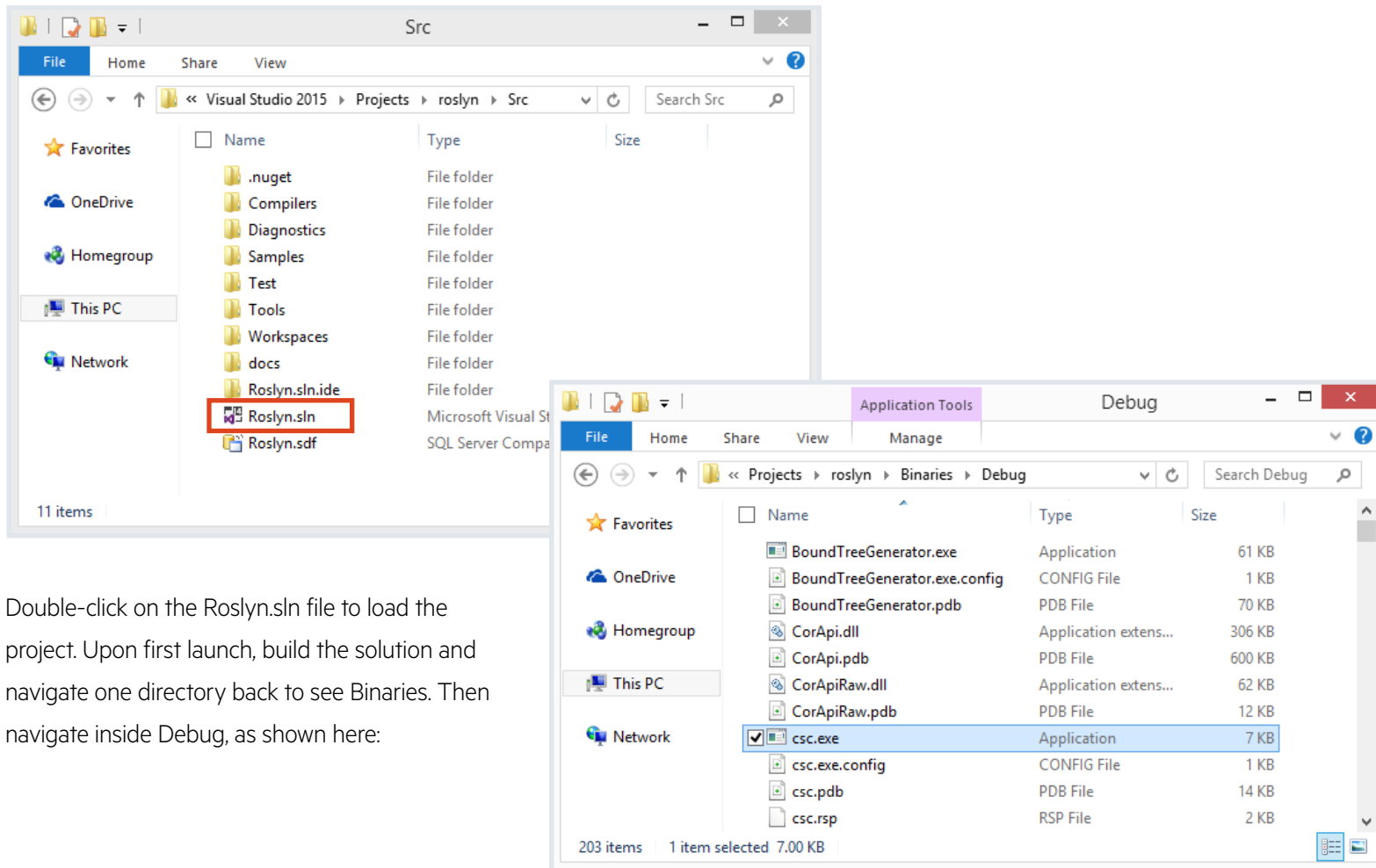
Introduction

Roslyn is an open-source C# and Visual Basic compiler with code analysis APIs for Microsoft's development stack. Very few people realize that Roslyn came out in October, 2011 as a preview that worked with Visual Studio 2010 SP1. While there have been several changes since 2011, Roslyn took the spotlight at the Build conference in 2014, when Microsoft open-sourced it and made it available for Visual Studio 2013. Today, as you can see on the [landing page](#), Visual Studio 2015 is front and center for Roslyn. Here, I'll walk you through a few samples that I found helpful for wrapping my head around Roslyn.



Getting Started

One of the easiest ways to get started with Roslyn is to download and install [Visual Studio 2015](#). Then, navigate to the Roslyn Project and [download the source code](#). You should see a folder containing several files and folders. We are only concerned with the one named **Src**, so navigate to it, as shown below:



Double-click on the Roslyn.sln file to load the project. Upon first launch, build the solution and navigate one directory back to see Binaries. Then navigate inside Debug, as shown here:

I See a Compiler!

Anyone used to C# will be familiar with **csc.exe**; we just compiled our own version of it. How does that help you?

Run csc.exe from the command prompt:

```
C:\Users\Michael\Documents\Visual Studio 2015\Projects\roslyn\Binaries\Debug>csc

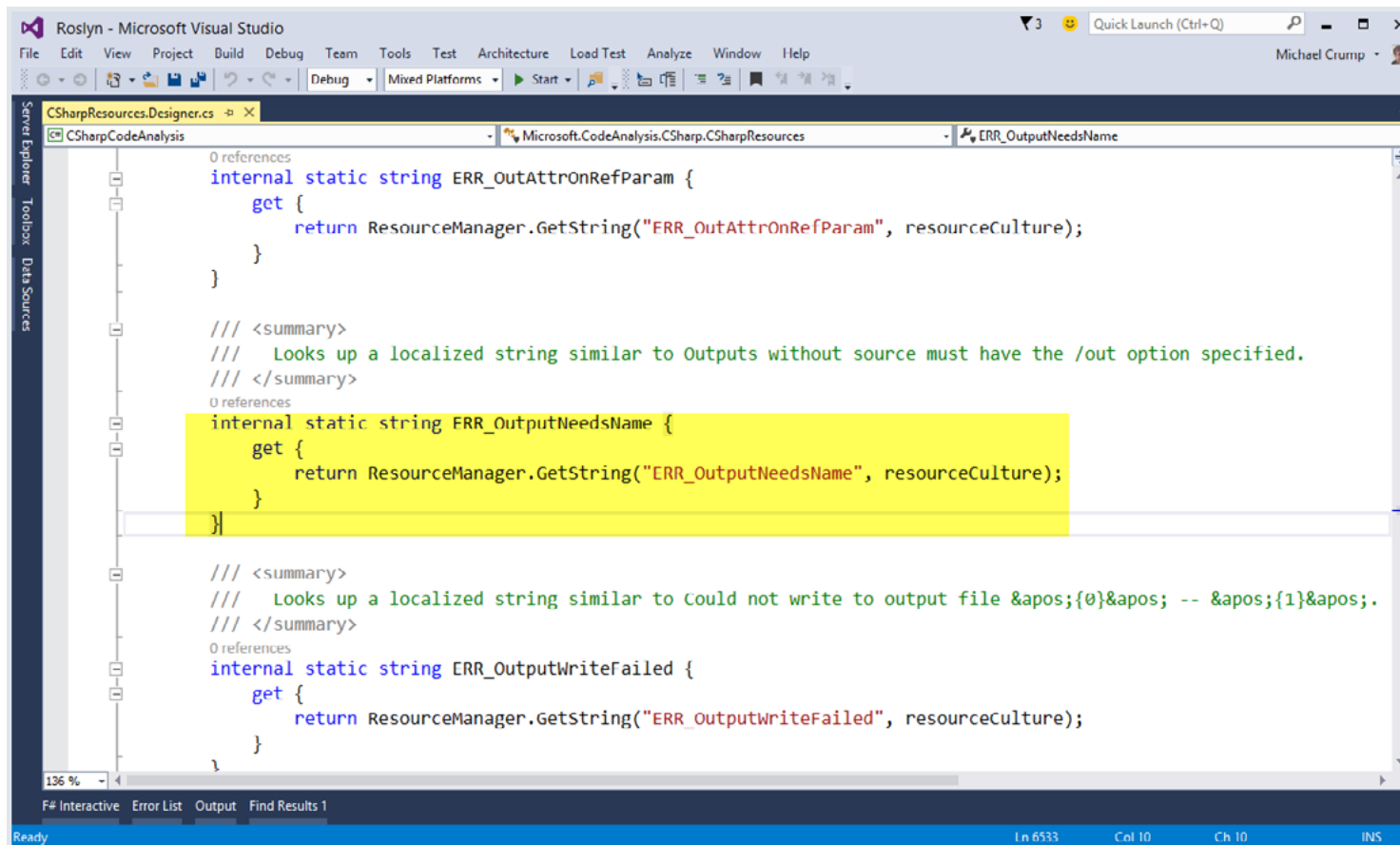
Microsoft (R) Visual C# Compiler version 1.0.0.0
Copyright (C) Microsoft Corporation. All rights reserved.

warning CS2008: No source files specified.
error CS1562: Outputs without source must have the /out option specified

C:\Users\Michael\Documents\Visual Studio 2015\Projects\roslyn\Binaries\Debug>
```

There aren't any surprises here, because you didn't specify a source file to be compiled, but what is really interesting is the ability to look into the source code and understand the warning and error messages generated.

If we search for “Outputs without source must have the/out option specified,” we find the place in the code from which it pulls that error message. In this case, it’s coming from the ResourceManager class.



The screenshot shows the Visual Studio IDE with the Roslyn - Microsoft Visual Studio window. The main editor displays the C# code for the ResourceManager class in CSharpResources.Designer.cs. The code includes several static string properties with getters that call ResourceManager.GetString. The property ERR_OutputNeedsName is highlighted in yellow, indicating it is the search result for the error message "Outputs without source must have the/out option specified". The code also includes XML comments for each property.

```
0 references
internal static string ERR_OutAttrOnRefParam {
    get {
        return ResourceManager.GetString("ERR_OutAttrOnRefParam", resourceCulture);
    }
}

/// <summary>
/// Looks up a localized string similar to Outputs without source must have the /out option specified.
/// </summary>
0 references
internal static string FRR_OutputNeedsName {
    get {
        return ResourceManager.GetString("ERR_OutputNeedsName", resourceCulture);
    }
}

/// <summary>
/// Looks up a localized string similar to Could not write to output file &apos;{0}&apos;; -- &apos;{1}&apos;.
/// </summary>
0 references
internal static string ERR_OutputWriteFailed {
    get {
        return ResourceManager.GetString("ERR_OutputWriteFailed", resourceCulture);
    }
}
```

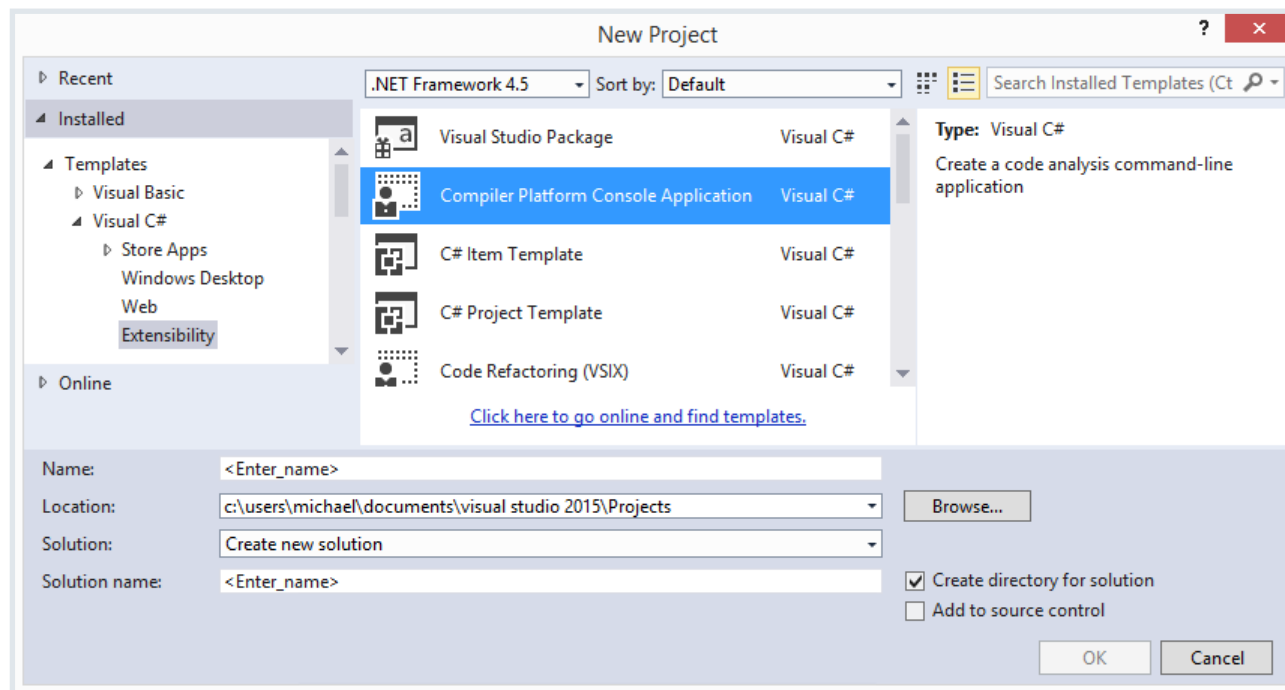
To change the way the compiler behaves as it reads through a C# program, read this [blog post](#), which describes how Anders Hejlsberg added support for French quotes in the C# compiler by adding just a few lines of code.

Taking a Look Under the Hood

Download the following files:

- [Visual Studio 2015 SDK](#)
- [Visual Studio Project Templates for Roslyn](#)
- [Syntax Visualizer for Roslyn](#)

Once those are installed, open Visual Studio 2015 and navigate to the "Compiler Platform Console Application" template:



Once the project spins up, you'll notice several references have been added; primarily, you'll be using the Microsoft.CodeAnalysis namespaces.

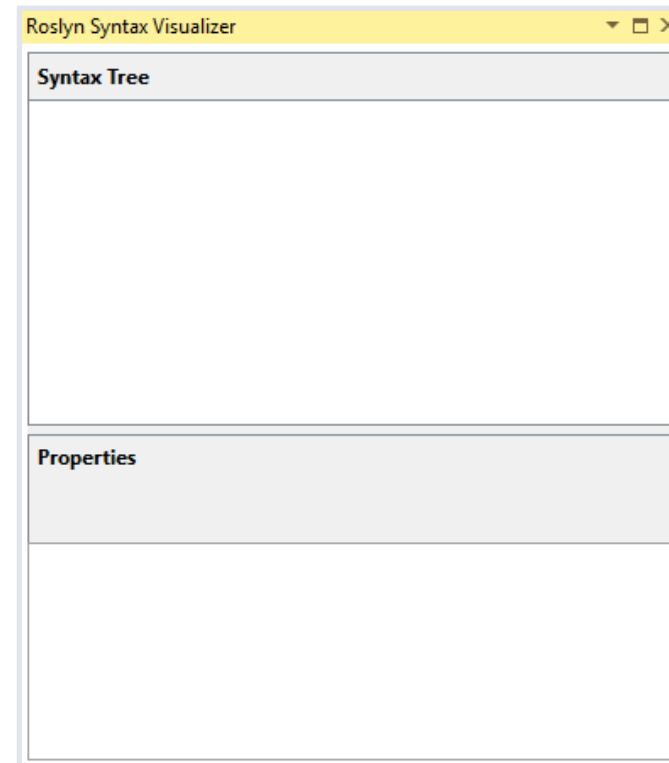
Add the following code to your Main method:

```
public static void Main(string[] args)
{
    SyntaxTree tree = CSharpSyntaxTree.ParseText(
        @"using System;
using System.Collections.Generic;
using System.Text;

namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("""Hello, TDN!""");
        }
    }
}");

    var root = (CompilationUnitSyntax)tree.GetRoot();
    var compilation = CSharpCompilation.Create("HelloTDN")
        .AddReferences(references: new[]
    { MetadataReference.CreateFromAssembly(typeof(object).Assembly) })
        .AddSyntaxTrees(tree);
}
```

Set a breakpoint on the last curly brace, and turn on the Roslyn Syntax Visualizer by going to View -> Other Windows -> Roslyn Syntax Visualizer. Once you've done this, you'll see the following (which should be blank at the moment):



Run the application and scroll to the top of the document. Click on SyntaxTree as declared in code, and you'll be taken to the proper node. From there, you can view a vast variety of information. Here, I'm looking at the Leading and Trailing WhiteSpace. Of course, you can dig deeper and learn exactly what is going on underneath the hood of your app.

The image shows a code editor on the left and the Roslyn Syntax Visualizer on the right. The code editor displays the following C# code:

```

0 references | Michael Crump, 1 day ago | 1 change
public static void Main(string[] args)
{
    SyntaxTree tree = CSharpSyntaxTree.ParseText(
@"using System;
using System.Collections.Generic;
using System.Text;

namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello, TDN!");
        }
    }
}");

```

The Roslyn Syntax Visualizer window shows the following syntax tree structure:

- IdentifierName [412..422]
 - IdentifierToken [412..422]
 - Lead: WhitespaceTrivia [400..412]
 - Trail: WhitespaceTrivia [422..423]
 - VariableDeclarator [423..863]
 - SemicolonToken [863..864]
 - LocalDeclarationStatement [880..929]
 - LocalDeclarationStatement [943..1216]
 - CloseBraceToken [1230..1231]
 - CloseBraceToken [1237..1238]
 - CloseBraceToken [1240..1241]

The Properties window shows the following information for the selected IdentifierToken:

Type	SyntaxToken
Kind	IdentifierToken
ContainsDiagnostics	False
ContainsDirectives	False
FullSpan	[400..423)
HasLeadingTrivia	True
HasStructuredTrivia	False
HasTrailingTrivia	True
IsMissing	False

You have just seen an example of compilation. A Compilation is an abstract class with language-specific derivatives.

SemanticModels

The next step is to ask the compilation for a SemanticModel for any SyntaxTree contained in that compilation.

SemanticModels as can be queried to answer questions such as:

- “What names are in scope at this location?”
- “What members are accessible from this method?”
- “What variables are used in this block of text?”
- “What does this name/expression refer to?”

Such queries can be achieved with the following code:

```
var model = compilation.GetSemanticModel(tree);
var nameInfo = model.GetSymbolInfo(root.Usings[0].Name);
var systemSymbol = (INamespaceSymbol)nameInfo.Symbol;
foreach (var ns in systemSymbol.GetNamespaceMembers())
{
    Console.WriteLine(ns.Name);
}
```

In this sample, we create a model using the GetSemanticModel method passing in our tree we defined earlier. We declare another variable, nameInfo, which will bind to the "using System;" namespace and pull all the systems for the member into a console window.

Congratulations! You just successfully bound a name to find a symbol.

Next Steps for Exploring Roslyn

I hope this overview gives you an idea of how powerful Roslyn is. While I was writing this article, I came across a lot of documentation that was outdated, but it looks like the community is quickly fixing it and adding more features. We haven't dug into binding expressions, syntax transformation and diagnostics, but, thankfully, those features have been [documented](#).

At Telerik, we've already been working with Roslyn. Look for a new version of [JustCode](#) for Visual Studio 2015, which will leverage Roslyn to help enhance developer productivity.

.NET ON A MAC

Introduction

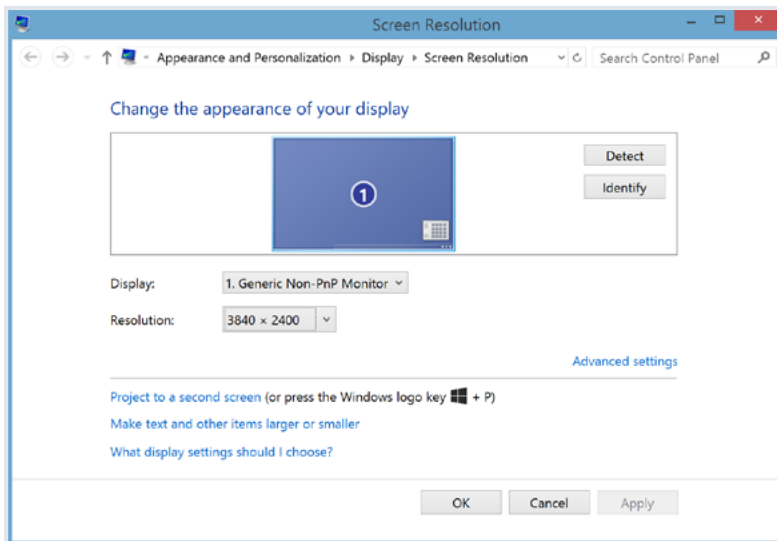
Let's face it—Apple makes some of the most desired notebooks in the industry. The MacBook Pro Retina is loved by developers for its screen real estate and silent high performance. The MacBook Airs, on the other hand, offer portability and long battery life for productivity. You're not alone in your love for the notebooks with glowing fruit.

If you're a Windows developer, you may be wondering if it's possible to use your new Apple device as a C#-friendly development machine. Or, you may be a long-time Mac developer, but stoked about .NET going Open Source announcements and want to try out some native ASP.NET on OSX. Either way, your goal is to marry two of the best things for developers: quality Mac hardware with the ease and popularity of C#. The good news is they coexist happily!



Windows Finds the Perfect Hosts

It's always good to see the occasional innovation from PC manufacturers, like the recent beautifully thin [Lenovo 2-in-1 Yoga Pro 3](#). Not to mention Microsoft's own tablet convertible [Surface Pro 3](#) is selling like hot cakes. But it's also no secret that MacBooks make wonderful laptops for running full Windows as an OS. As an added bonus, if you have a MacBook Pro Retina laptop, your Windows installation enjoys excellent high resolution. Developers love screen real estate, even at the expense of squinting eyes, right?



Turns out, there are two ways you can run Windows on a Mac:

- 1. Bootcamp:** This is Apple's way of allowing you to [run Windows on an Intel-based Mac](#). Simply use the built-in [Bootcamp Assistant](#), make a partition and install Windows. Then you can easily boot into Windows instead of OSX and reuse all of the I/O (input-output) drivers for peripherals. In this mode, Windows is running natively "on the metal," and you get full performance benefits. Of course, this means you can install Visual Studio and write C# all day, just as in a Windows machine.
- 2. Virtual Machines:** If choosing an OS to boot into isn't your cup of tea, your other option is to run Windows in a virtual machine (VM) with OSX acting as the host OS. There exists dedicated software that will do the heavy lifting for your VM, such as managing virtualization, memory and peripherals. [Parallels](#) and [VMWare Fusion](#) are two excellent options for running Windows VMs on your Mac. With customizable virtualization, resource fine-tuning options and easy switchability between Windows/OSX, you should be rocking Windows running as a VM in no time. Just as easily, you can install Visual Studio inside your Windows VM and write C# to your heart's content.

Now, whether you go BootCamp or VM, you are running full Windows. And that means, you get to rock out Telerik [DevCraft](#) suite to supercharge your .NET productivity for any type of application you are building!

Visual Studio “Monaco” Editor

Although not yet fully baked, C# is about to get a new ubiquitous code editor—the browser. C# in the browser comes courtesy of a special light-weight editor codenamed “Monaco,” which was launched with Visual Studio 2013. To quote Microsoft:

“

With Monaco, we want to provide developers with a lightweight, friction-free companion to the Visual Studio desktop IDE that is accessible from any device on any platform. Monaco is a rich, browser-based, code-focused development environment optimized for the Windows Azure platform, making it easy to start building and maintaining applications for the cloud.”

Check out this [Channel 9 video series](#) on how to get started with Visual Studio Online Monaco editor, and to keep up with the latest enhancements.

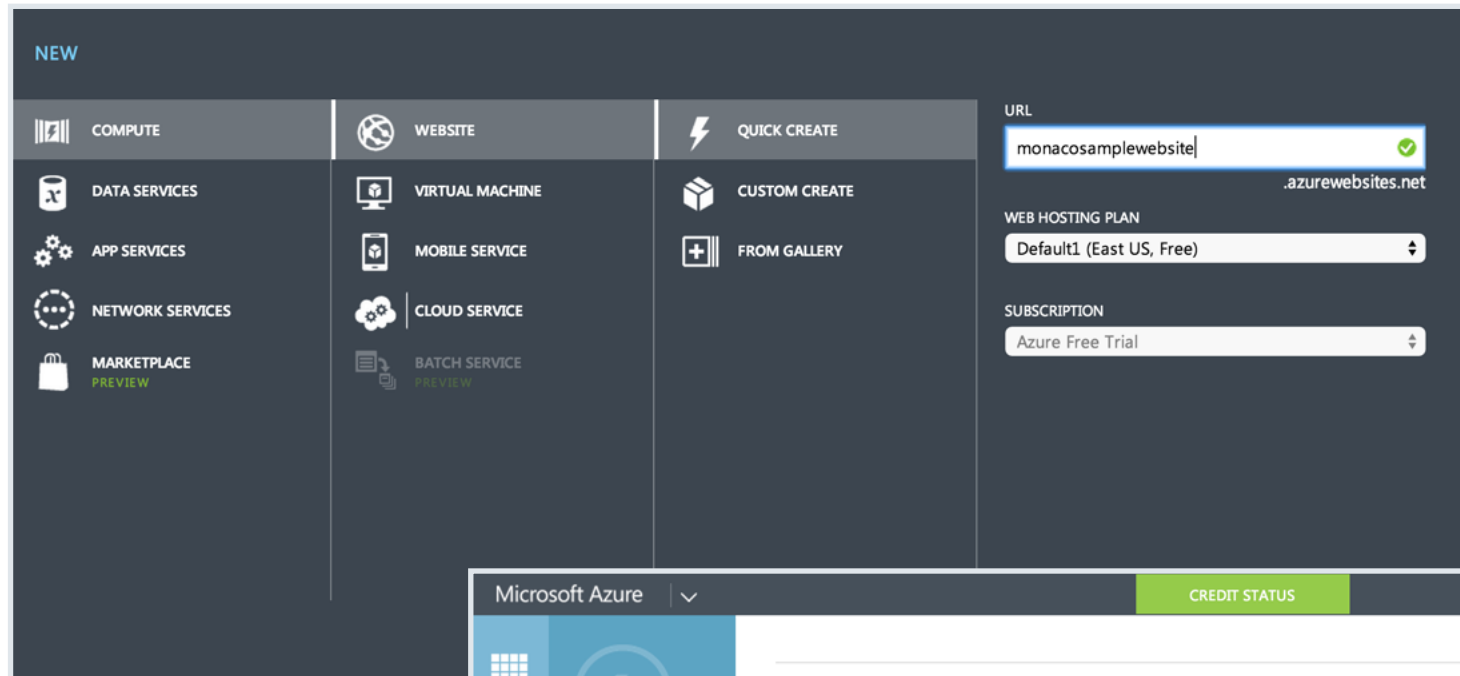
Want to try out the Monaco editor today?

It’s geared to work with Windows Azure websites hosted in Windows Azure, for now, but you can absolutely write the C# code-behind code for the server-side, hook up your code to a source control and perform builds/deployments.

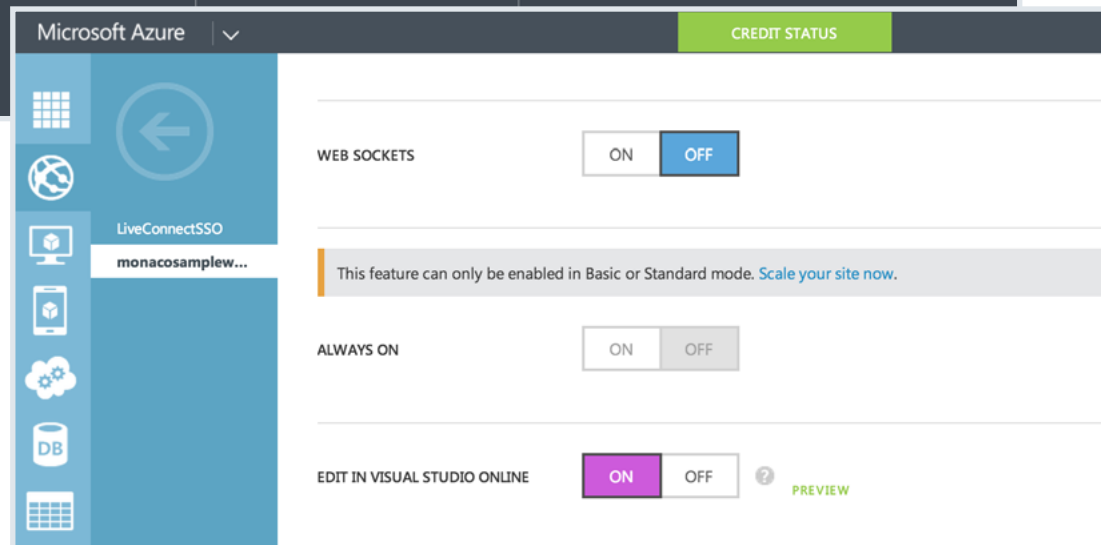
Here’s how to get started:

1. First, create a Windows Azure account [Sign up for free](#).
2. Login to the [Windows Azure Management Portal](#)
3. Create a new Azure Website from the big ‘+’ sign on bottom left
4. Open up Website Configuration and turn on the option for “Edit in Visual Studio Online”
5. From the website dashboard, click on “Edit in Visual Studio Online”
6. Enjoy C# edits in your browser

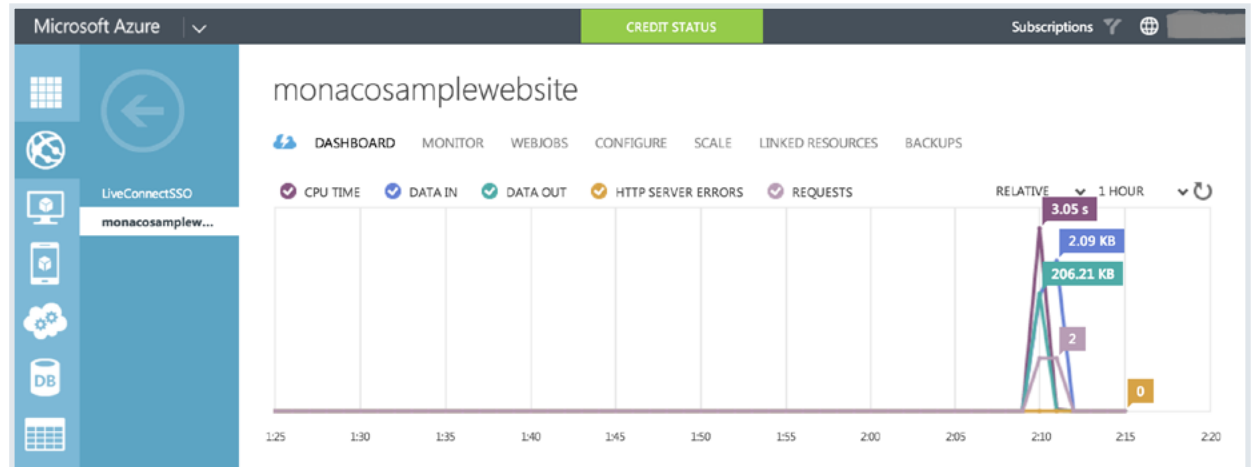
Pictures are worth a thousand words, right? Here are the steps visually, starting with creating the Azure website (be sure to choose your Azure Subscription and hosting appropriately):



Next, navigate to your newly created Website Dashboard and click on the “Configure” tab. Scroll down a little with your configurations and you’ll see the “Edit in Visual Studio Online” setting. Turn it on to see the magic:



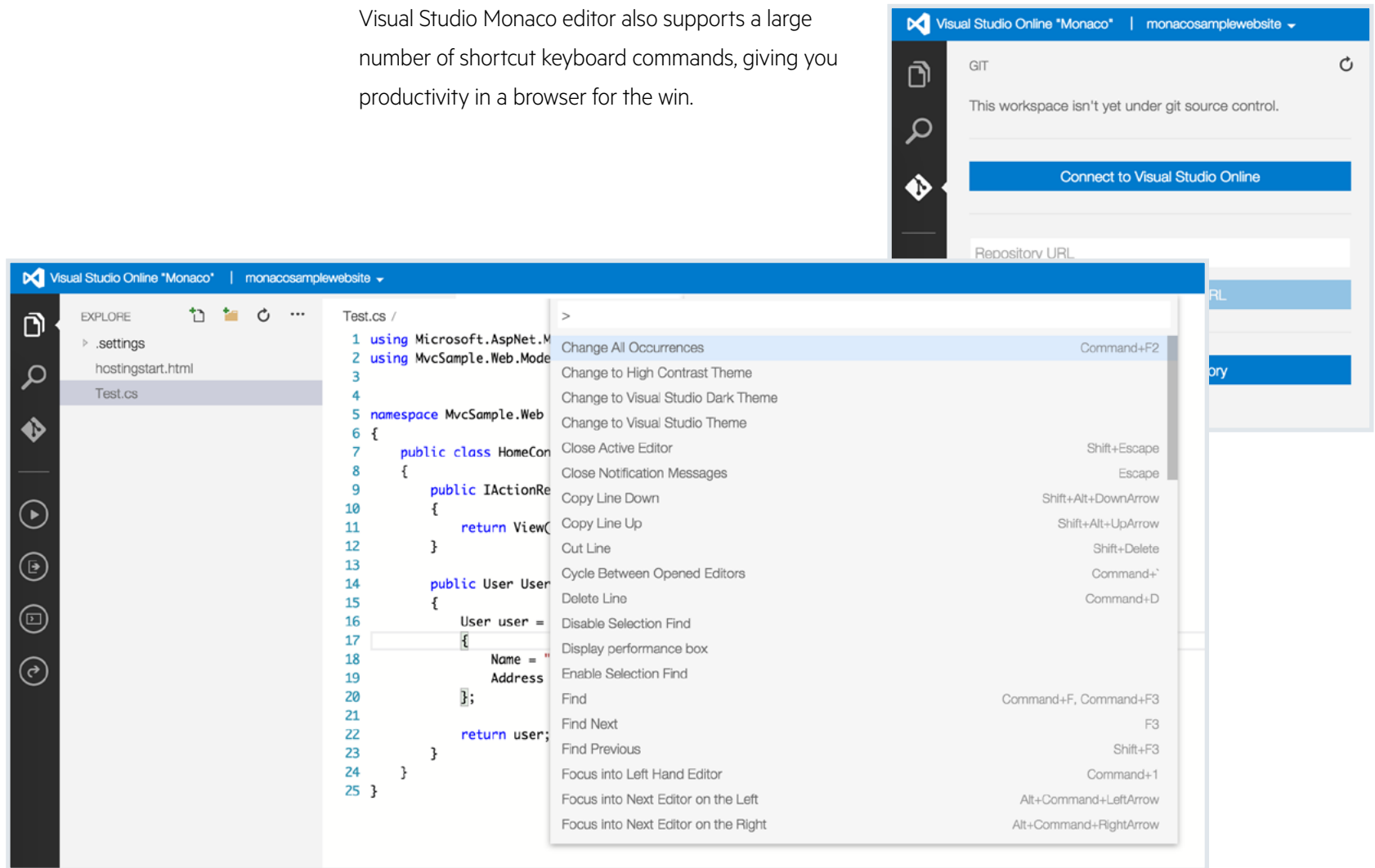
Back on your Website Dashboard, the “Edit in Visual Studio Online” link will light up:



```
1 using Microsoft.AspNet.Mvc;
2 using MvcSample.Web.Models;
3
4
5 namespace MvcSample.Web
6 {
7     public class HomeController : Controller
8     {
9         public IActionResult Index()
10        {
11            return View(User());
12        }
13
14        public User User()
15        {
16            User user = new User()
17            {
18                Name = "My name",
19                Address = "My address"
20            };
21
22            return user;
23        }
24    }
25 }
```

Go ahead, and click the link. Voila—a new page opens up with light-weight code edits for your Windows Azure Website. You can see here that I added a new Test.cs file and I’m able to write C# code in Chrome browser on my Mac—how cool is that!

Visual Studio Monaco editor also supports a large number of shortcut keyboard commands, giving you productivity in a browser for the win.



If editing HTML/CSS/JS, you'll find the Visual Studio Monaco editor rather smart with Intellisense; the C# experience keeps improving every release. You'll also find solid support for online editing of LESS, PHP, Node.js and TypeScript. When done with your edits in the browser, simply fire up a build and see output in a console window, or even run

your Azure website to pick up the changes. In effect, you can build out a full ASP.NET website in your browser, including writing C# on any browser running on your Mac OSX.

A Look at ASP.NET

The best has been reserved for last, because you're going to have the most fun writing C# on a Mac when building modern web/mobile applications with ASP.NET. You may have already heard about [.NET Framework core being open sourced](#); the [future of .NET](#) is modular, cross-platform and rather exciting.

[ASP.NET vNext](#) leads the way, with [improved tooling](#) and flexible hosting outside of IIS using a new [KRuntime](#). If you're using the core cross-platform .NET Framework, your ASP.NET web applications will run everywhere. Yes, that means natively in OSX on your Mac!

Getting Ready

Before you start rocking C# on your Mac to build your next ASP.NET vNext application, there are few things to set up in your environment. Let's walk through the steps:

1. First, visit the [ASP.NET vNext home on GitHub](#) to make sure you understand the moving pieces and check minimum system requirements.
2. ASP.NET vNext architecture is modular, and you'll be using several packages or components. Let's get some prerequisites out of the way first.
3. Install the latest version of [Node](#) and grab the latest ubiquitous package manager [NPM](#).
4. Homebrew is another excellent open source package manager that allows you to install/manage software that can't be installed from the

OSX terminal. So, install Homebrew as well—simply fire up the following Ruby code in your terminal:

```
ruby -e "$(curl -fsSL  
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

5. Tap the ASP.NET vNext related Git repositories using Homebrew, like so:

```
brew tap aspnet/k
```

6. Next, install the [K Version Manager \(KVM\)](#). You can use KVM to install and switch between different ASP.NET runtimes. Simply fire up the brew command (note this step will install [Mono](#) on your OSX if not already present).

```
brew install kvm
```

Alternatively, you can always manually pull down [Mono from GitHub](#) and build it yourself; ASP.NET runtime on a Mac depends on Mono for now.

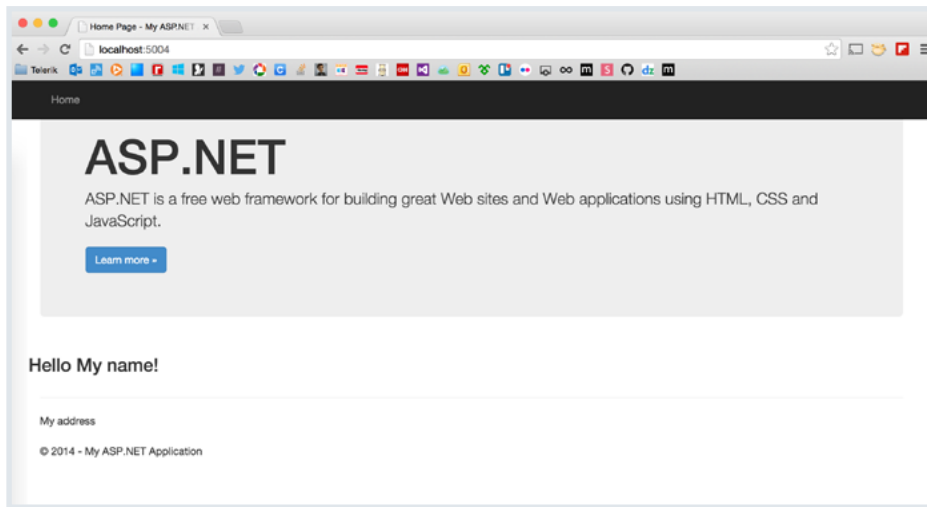
7. After you set up KVM, install the latest [K RunTime Environment \(KRE\)](#), like so:

```
kvm upgrade
```

8. Now, you are ready to run ASP.NET vNext natively on your Mac, technically speaking. But a few more handy tools will help down the line, starting with [Yeoman](#). Yeoman is a sleek and open source scaffolding tool that can work for your ASP.NET projects. Grab Yeoman and the ASP.NET Yeoman generators:

```
npm install -global yo  
npm install -g generator-aspnet
```


Your terminal console should show a message indicating the site has been started. Now, simply pull up any browser and navigate to <http://localhost:5004> (ASP.NET running on LocalHost in Chrome on a Mac):



Accept it: the first time you see a native ASP.NET web application running on a Mac, you'll have a "WHOA" moment!

OmniSharp

At this point, you have a fully scaffolded ASP.NET web application running natively on your Mac. But you need a little more help as you get to working on the website's code, especially if you're writing a lot of C# server-side code. Enter the marvelous [OmniSharp.NET](#).

It may be adventurous to run .NET applications on OSX or Linux, but is it practical to write C# code outside of the Visual Studio comfort?

OmniSharp is here to help. To quote the website [omnisharp.net](#), "OmniSharp is a family of Open Source projects, each with one goal: to enable great .NET development in YOUR editor of choice." This goal extends to non-Windows editors, the likes of Sublime Text, Atom, Emacs, Brackets or Vim. Yep, you can write C# in any of those editors!

Sublime Text

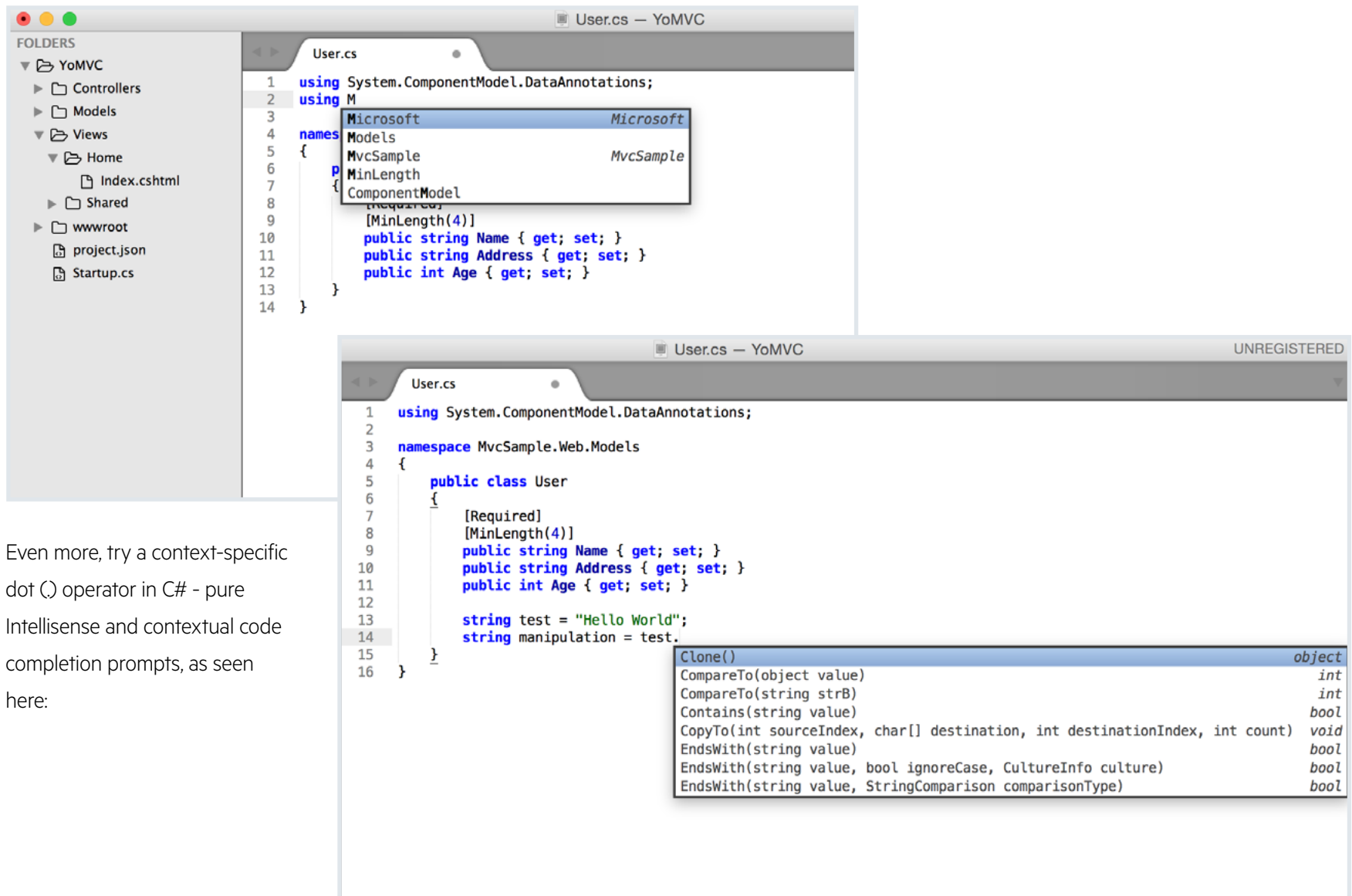
One of the most popular editors is [Sublime Text](#). Want to write C# in Sublime Text and get some of the benefits you expect from Visual Studio? Turns out, efforts are underway to [make C# a first-class citizen](#) in editors such as Sublime Text.

Here are the few simple steps to set up Sublime Text for writing C# in ASP.NET applications:

1. Download the [beta of Sublime Text 3](#)
2. Make sure to have the latest [Sublime Text Package Manager plugin installed](#)
3. Install the [Kulture plugin](#) through Package Control
4. Install the [Omnisharp plugin](#) through Package Control
5. Make some [C# language specific settings](#) to provide appropriate Intellisense event triggers

That's all the magic that's needed. Now, open up the Yeoman scaffolded ASP.NET vNext project from your OSX directory in Sublime Text (point

to root folder). Sublime shows all your files in a Project tree, each of which is perfectly editable. Open up or create a new C# file and start typing. Boom! Visual Studio like Intellisense as you type words, all inside Sublime Text:



Even more, try a context-specific dot (.) operator in C# - pure Intellisense and contextual code completion prompts, as seen here:

How cool is that? Wonder how all this is working in Sublime Text or other editors? This is courtesy of the [OmniSharpServer](#). According to [Jason Imison](#), “OmniSharpServer is a local web server (written in [Nancy](#)) that accepts requests to various different endpoints, which returns results about the code you send to it. For example, in Sublime Text, when you have a string variable and you type (.) after the variable name, a request is sent to OmniSharpServer with a specific payload, and the response contains all the possible completions for that variable.

NRefactory is the C# analysis library used in the OmniSharpServer. It allows applications to easily analyze both syntax and semantics of C# programs. It is quite similar to Microsoft's Roslyn project, except that it is not a full compiler—NRefactory only analyzes C# code. It does not generate IL code.”¹

Sounds complicated, but you get to reap the benefits as the open source community works on OmniSharp.NET with Microsoft's endorsement. Essentially, as you type your C# code, the locally hosted Omnisharp Server is doing all the heavy lifting, trying to provide you with contextual Intellisense, while stopping short of actually compiling your code. You get to write C# on your editor of choice, complete with Visual Studio-like code editor features. As for Sublime Text, you get to enjoy features like Intellisense, Go To Definition, Rename, Find Usages, Go To Implementation, Format Document, Override, Add Reference, Syntax/Semantic Errors, Code Refactoring, Build Solution and many more. Check out this [wonderful post](#) on what to expect when writing C# in Sublime Text, complete with GIFs for each interaction.

¹Source: <http://blog.jonathanchannon.com/2014/11/12/csharp-first-class-citizen-sublime-text/>

Conclusion

Choice is a good thing for developers, and being able to choose the most popular .NET language on one of the best possible laptop seems like a solid match. Pick what works best for you: VM, browser or native editors with OmniSharp. The bottom line is you can write C# on a Mac, like a champ.

WRAPPING UP

The Future Looks Very Bright for .NET Developers

Microsoft is focusing on the convergence of the Operating Systems (OS) and developer platforms, and .NET is your best bet. Windows 10 represents the next-generation OS for most devices you're used to running Windows on and then some: desktops, laptops, phones, "Internet of Things (IoT)" devices and embedded devices. Other application stacks or technologies you have used with .NET in the past all move forward—simply choose the modular or complete .NET framework based on your needs. Check out the recently published [roadmap and tooling improvements](#) for WPF, which were met with much fanfare from the .NET development community. Let's embrace the changes and gear up for a flexible, better tomorrow. .NET is a best-in-class tooling and open source cross-platform approach to building the next generation of amazing apps for any platform.



Using the Telerik Stack to Be More Productive

Now that we've explored the new goodies coming in 2015 from Microsoft, let's take a step back and see how Telerik can help make development even easier.

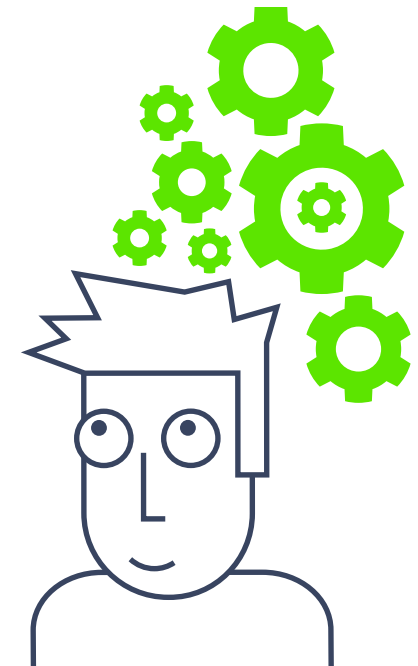
For all .NET developers, [Telerik DevCraft™ suite](#) is a complete .NET toolbox for web, mobile and desktop development. The DevCraft bundle includes UI controls for all .NET platforms, reporting, productivity, code quality and data tools:

- For the web developer, we have UI components for [ASP.NET AJAX](#), [MVC](#) and [Silverlight](#) to jumpstart your next app. We also have a HTML5/JS Framework called [Kendo UI](#).
- For the desktop developer, we are still making advances in [WinForms](#) and [WPF](#).
- For the mobile developer, we have UI for [Windows Universal](#), [Windows Phone](#) and even [Xamarin](#) components.
- We haven't forgotten about productivity and quality, as we have [JustCode](#) (the new version is based off of Roslyn), [JustMock](#) and a completely free [Testing Framework](#). For debugging, we have two popular and free tools, [Fiddler](#) and [JustDecompile](#). Looking for that memory leak? Check out [JustTrace](#).
- For those of you into data, we have [Reporting](#) and [Data Access](#). Both of which are crucial in enterprise applications.

Download a free trial of the DevCraft suite
with free support for 30 days



Our [Telerik Platform™ solution](#), which allows you to develop [cross-platform and mobile applications](#), has several key features that .NET developers can take advantage of, such as [Analytics](#) and [Backend Services](#). Our Visual Studio extension that enables you to create hybrid apps in your favorite IDE, without owning a Mac.



Thanks for reading, and we hope this ebook helped supercharge your .NET knowledge.

ABOUT THE AUTHORS



Michael Crump

Michael Crump works at Telerik and is a MS MVP, coder, blogger and speaker of various software development topics. He has a passion for a wide range of technology stacks that involve web and mobile. In his free time, he likes to experiment with wearables and is a big fan of IoT. Michael can be found on twitter at [@mbcrump](#) or by visiting michaelcrump.net.



Sam Basu

Sam Basu ([@samidip](#)) is a technologist, Apress/Pluralsight author, speaker, Microsoft MVP, believer in software craftsmanship, gadget-lover and Developer Advocate for Telerik. With a long developer background, he now spends much of his time advocating modern web/mobile/cloud development platforms on Microsoft/Telerik stacks.

He passionately helps run [The Windows Developer User Group](#), labors in [M3 Conf](#) organization, serves as [INETA Secretary](#) and can be found with at-least a couple of hobbyist projects at any time. His spare times call for travel and culinary adventures with the wife.

Find out more at <http://samidipbasu.com>.