THE EXPERT'S VOICE® IN WINDOWS

Pro Windows 8.1 Development with XAML and C#

DISCOVER THE FUTURE OF PROGRAMMING ON THE MICROSOFT PLATFORM USING XAML AND C#

Jesse Liberty, Philip Japikse, and Jon Galloway

Apress®

For your convenience Apress has placed some of the front matter material after the index. Please use the Bookmarks and Contents at a Glance links to access them.

Apress®

Contents at a Glance

About the Authors xvii

About the Technical Reviewers xix

Acknowledgments xxi

■ Chapter 1: Getting Started 1

■ Chapter 2: Building Your First Windows 8 App 25

■ Chapter 3: Themes, Panels, and Controls 45

■ Chapter 4: Binding 79

■ Chapter 5: Views 99

■ Chapter 6: Local Data 131

■ Chapter 7: Remote Data and Services 159

■ Chapter 8: Search and Share Contracts 187

■ Chapter 9: Notifications 209

■ Chapter 10: Application Life Cycle 261

■ Chapter 11: Making Money 277

■ Chapter 12: Publishing Your App 295

Index 319

CHAPTER 1

Getting Started

Windows 8.1 development with C# and XAML carries a lot of similarities with developing Windows Presentation Foundation (WPF) applications. Well, they both use XAML and C#. And many of your existing skills with user interfaces (UIs) and program code can be leveraged for Windows 8.1 apps. But there are a lot of differences, as well. Windows 8.1 apps are touch-centric (but also support mouse and keyboard interactions). There are specific design guidelines, quite contrary to WPF development. The isolation level for Windows 8.1 apps is much higher than WPF, presenting a new level of safety and security, as well as unique challenges when working with data. Apps are deployed through a central store (as opposed to click-once deployment or Microsoft Installer packages).

Not a WPF developer? No worries! This book will take you through everything you need to know to build Windows 8.1 apps.

Background

Microsoft released the latest revision of its Windows operating system, Windows 8.0, on October 26, 2012. This release was groundbreaking for several reasons, but at the top of the list were the dual interfaces of Windows 8 and support for ARM devices.

Dual User Interfaces

The original release of Windows 8.0 introduced two UIs—the desktop for Intel-based hardware and the tiled interface (formerly called Metro) not only for Intel-based hardware, but also ARM-based hardware, a first for Windows.

The desktop UI, simply put, is a better version of Windows 7. The install is much smaller (I regained 6 GB of hard disk space when I upgraded), it's even more secure, and it runs faster on the same hardware that was supported by Windows 7.

The tiled UI was completely reimagined for Windows 8.0. Although the same type of tiled interface had previously been used in the Windows Phone operating system, this was the first time that a non-phone-based version of Windows had a major change in many, many years.

ARM Support

ARM-based processors (created by ARM Holdings, a British Company) have dominated the tablet market. What was very clearly missing from that market was an offering from Microsoft—at least from Microsoft's point of view. But also for Windows users. The introduction of the Microsoft Surface RT brought a Windows-based offering in the tablet space, allowing users to sync their desktop/laptop, phone, and tablet. And Microsoft Office runs on the Surface RT!

Acceptance

How did this dramatic change go over in the first year? If you listen to the press and the analysts, terrible. Not with the operating system itself, but with the response from the masses. From "Where is my Start menu?" to "How do I print?," there were outcries about moved cheese and change. Were they legitimate complaints? It depends on your point of view, but the quick release of Windows 8.1 bringing back the Start button and addressing some of the most common complaints certainly adds credence to those people who weren't happy with the original version.

Interestingly enough, in looking at the numbers from several different sites, Windows 8 adoption in the first year rivals that of XP adoption in its first year. And if you are old enough to remember the Windows 95 revolution and the outcry at that release, this is same old same old. Things change, people freak out, and then after the initial shock, they settle down and start using the new software.

Fast-Release Cycle

Less than one year after the release of Windows 8, Microsoft released Windows 8.1 on October 17, 2013. Much more than a service pack, this release addressed many of the issues that people were complaining about, such as the return of the Start button and the ability to boot straight to desktop mode. Made freely available through the Microsoft Store, the install rate for Windows 8.1 has been extremely high.

The Microsoft Store

How many times have you had to do tech support for a family member because he clicked on some random pop-up on the Internet, or installed some software that a friend told him about? The main mechanism for getting apps is from the Microsoft Store. Having that one central place to get apps for Windows 8/8.1 helps prevent rogue software from getting installed, increasing the security and reliability of the device. It also provides a centralized location for developers to place their app for others to find. For more information about submitting your app to the Microsoft Store, please see Chapter 12.

What's New in Windows 8.1

There are a lot of changes between Windows 8.0 and Windows 8.1. At the top of each chapter in this book, look for the "What's New in Windows 8.1" sidebar to get a high-level overview of the changes in Windows 8.1 concerning the chapter's topic. The chapters themselves are dedicated to using Visual Studio 2013 and Windows 8.1, with detailed information is included in the body of each chapter.

Windows Design Guidelines

In order to get your apps accepted into the store, you must make sure they meet the seven traits of a great app and also follow the five Microsoft design principles. There are additional technical requirements that will be discussed in Chapter 11.

Let's look at the seven traits of a great app first. To achieve greatness, it must:

- Be fast and fluid
- Size beautifully
- Use the right contracts
- Invest in a great tile

- Feel like it is connected and alive
- Roam to the cloud
- Embrace modern app design principles

Being Fast and Fluid

Modern apps can be run on a variety of devices with a wide range of capabilities. While Microsoft has set minimum standards for all hardware that carries the Windows 8 logo, it's important for the success of your app (as well as the success of Windows 8) that your app doesn't perform poorly or cause the hardware to perform poorly. You will see as you work your way through this book that in order to develop Windows 8.1 applications, you must use asynchronous programming to ensure a responsive UI. Additionally, the very design of the Windows 8.1 process lifetime management cycle ensures that background apps don't drain the battery or use up precious system resources.

Use the async pattern liberally. If your app is taking a long time to load or to run, people will uninstall it. Or, worse yet, they will write a scathing review and *then* uninstall it.

Sizing Beautifully

Windows 8 devices come in a variety of sizes and screen resolutions. Apps can be run in a landscape or portrait view as well as resized to share the screen with other apps. Your app needs to able to adjust to different layouts and sizes, not only in appearance but also in usability. For example, if you have a screen showing a lot of data in a grid, when your app gets pinned to one side or the other, that grid should turn into a list.

Using the Right Contracts

Windows 8 introduces a completely new way to interact with the operating system and other applications. Contracts include Search, Share, Settings. By leveraging these contracts, you expose additional capabilities into your app in a manner that is very familiar to your users.

Investing in a Great Tile

Tiles are the entry point into your applications. A live tile can draw users into an app and increase the interest and time spent using it. Too-many updates can lead them to turn off updates, or worse yet, uninstall your app.

Secondary tiles are a great way for users to pin specific information to their Start screen to enable quick access to items of their interest.

Feeling like It Is Connected and Alive

Users are a vital component to Windows 8 apps. It is important to make sure that your app is connected to the world so that it can receive real-time information. Whether that information is the latest stock prices or information on sales figures for your company, stale data doesn't compel users to keep using your app. They already know what yesterday's weather was. The current forecast is much more interesting.

Roaming to the Cloud

Windows 8 allows the user the capability to share data between devices. Not only can application settings be synced but so can the application data. Imagine the surprise for a user who enters some data into your app at work and then picks up another Windows 8 device at home, starts your app, and the data is right there.

It is important to leverage the cloud whenever possible to make transitioning from one device to another as seamless as possible.

Embracing the Modern App Design Principles

In addition to the traits just mentioned, your app must meet the modern app design principles. Microsoft's full list can be found here: http://go.microsoft.com/fwlink/p/?linkid=258743. A brief summary of the five principles follows:

- *Show pride in craftsmanship*: This is fairly simple. Pardon the vernacular, but just don't build crappy software. Build software that you would be willing to list on your resume. If you're making an app in order to make money, you will want people to use it, to "like" it on social media, to show it off to their friends. These things are important unless you have a huge marketing budget, but even then that will only get first-time users. Once word gets out that the app isn't very good (and remember there is a rating system in the Microsoft Store), your app is done.

  So, make sure it works. Consider hiring a graphic designer if you're not design inclined. Let users test it (consider friends and family). Get feedback. Fix the problems—whether the problems are bugs or user-experience issues.

- *Be fast and fluid*: Microsoft is serious about this one, having listed it here and in the traits for a great app. In addition to the traits previously listed, design for touch and intuitive interaction. Be responsive to user interaction, and make a UI that is immersive and compelling.

- *Be authentically digital*: Take full advantage the capabilities of the device. Use bold colors, beautiful typography, and animations. Of course, use all of the effects (especially motion) with purpose. Just because you can add flaming arrows shooting across the screen doesn't mean you should!

- *Do more with less*: Chrome may look great on a motorcycle, but in software, it just distracts users from what they care about, which is the data. Instead of buttons and tabs, leverage the app bar and nav bar. Take advantage of the charms and contracts to reduce even more chrome.

  Focus the functionality of your app. Don't try to solve all problems. Just solve one or two really well. Be focused on that solution, and do your best to immerse your users in their data, not the app.

- *Win as one*: Work with the Windows 8 paradigm. Leverage common touch gestures and familiar mouse and keyboard themes so your users can leverage what they already know. Work with other apps through contracts so that your app can become bigger than just the sum of its parts.

UX Guidelines

There are many more guidelines suggested by Microsoft. For the full guidelines, see http://msdn.microsoft.com/en-us/library/windows/apps/hh465424.aspx. If you would like a downloadable PDF of the same guidelines, you can download it here: http://go.microsoft.com/fwlink/p/?linkid=258743.

Tooling

While you can certainly remain in Visual Studio the entire time you are developing your app, leveraging a combination of the available tooling provides the best experience. For developing Windows 8 applications, the two main tools you will use are Visual Studio 2013 and Blend for Visual Studio 2013.

Visual Studio 2013

In November of 2013, Microsoft released Visual Studio 2013. Among the changes in the latest version of Visual Studio is support for creating Windows 8.1 apps. This is a mandatory upgrade, as Visual Studio 2012 will not support building apps for Windows 8.1.

Versions

If you are reading this book, then you are probably very familiar with Visual Studio. In this section, I'll talk about the different versions of Visual Studio 2013 available to you and some of the differences between them. If you aren't very familiar with Visual Studio, don't worry. As we move through the chapters of this book, the relevant features will be discussed in greater detail.

Visual Studio Express

Visual Studio Express 2013 is now separated by the target platform. Although I haven't seen any official communication as to why, I think it was just too big to keep it in one free download, and people who were new to Visual Studio were getting lost in all of the features.

The available versions are:

- *Visual Studio 2013 Express for Web*: for ASP.NET developers
- *Visual Studio 2013 Express for Windows*: for Windows 8.1 app developers
- *Visual Studio 2013 Express for Windows Desktop*: for Windows Client application developers using Windows Presentation Foundation (WPF) or WinForms

For the purposes of this book (and creating Windows 8.1 apps), you will need Visual Studio Express 2013 for Windows.

Visual Studio with MSDN

There are essentially three paid versions of Visual Studio 2013 for developers: Professional, Premium, and Ultimate. All of them are part of MSDN and provide everything you need for developing Windows 8.1 apps plus a whole lot more. Note that there is also a fourth version, Visual Studio for Test Professional, but it doesn't apply to building Windows 8.1 apps, so we don't discuss it here. For all of the nitty-gritty details of what's in each version, see the documentation on the Visual Studio site here: `www.visualstudio.com/products/compare-visual-studio-products-vs`.

The Windows 8.1 Simulator

All versions of Visual Studio come with the ability to run your Windows 8.1 app in a simulator. This is essentially a remote desktop session to your PC with the added ability to change orientation, form factor, gesture support and to simulate many factors of a tablet (even if you are developing on a nontouch device). This is one of the reasons that you must be working in a Windows 8.1 environment to develop Windows 8.1 apps.

Creating Your First Windows 8.1 App

To create a Windows 8.1 app, create a new project in Visual Studio 2013 by selecting File ➤ New ➤ Project. In the left rail, you will see all of the installed templates for your Visual Studio installation (your mileage may vary based on version you installed and what third-part products you use). Select Installed ➤ Templates ➤ Visual C#➤ Windows Store, and you will be presented with the dialog shown in Figure 1-1.

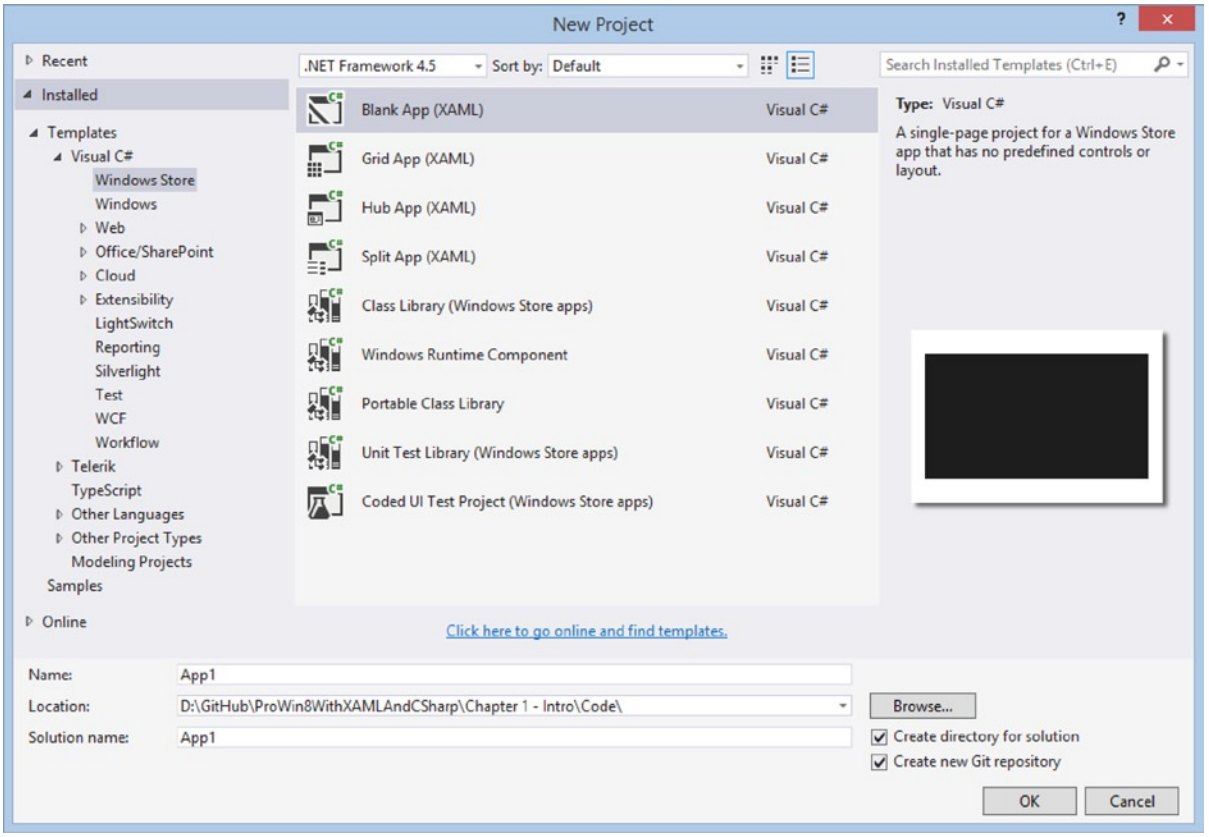


Figure 1-1. *New Project templates for Windows 8.1 apps*

In Chapter 4, we will go into great detail for all of the project templates, so for now, just select Blank App (XAML). In fact, this will be the starting template for *most* of our projects in this book, and is the template I typically start with when I develop Windows 8.1 apps. You can leave the project name as the default App1.

After you create your project, take a look at the Default Solution folder (shown in Figure 1-2). The Blank App template actually does a lot for us. In addition to creating the project and bringing in the appropriate references, it supplies us with several assets, the `App.xaml` file, and `MainPage.xaml`. The Assets folder contains the images for the splash screen and the default tiles (more on that later in this book), and if you are familiar with WPF, the `App.xaml` and `MainPage.xaml` files should be very familiar. Again, we will spend a lot of time in the book on those files.

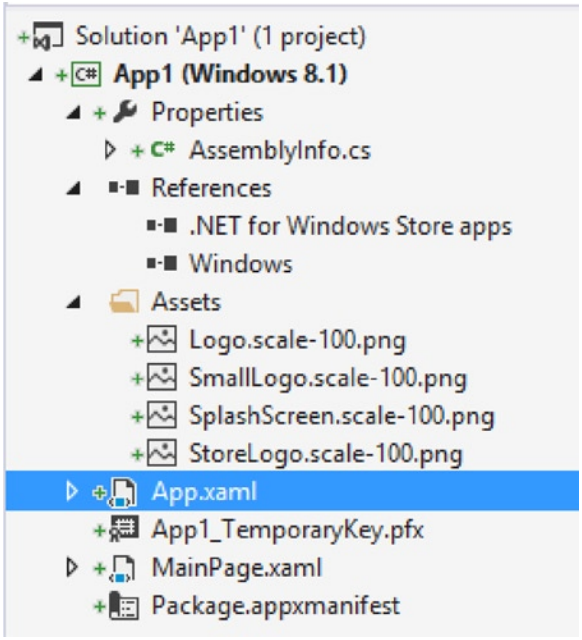


Figure 1-2. *Default Solution Explorer files*

To run the app, you can press F5 (to start with debugging), Ctl-F5 (to start without debugging), click on Debug in the menu (to be presented with the same options), or click the toolbar item with the green arrow (as shown in Figure 1-3).

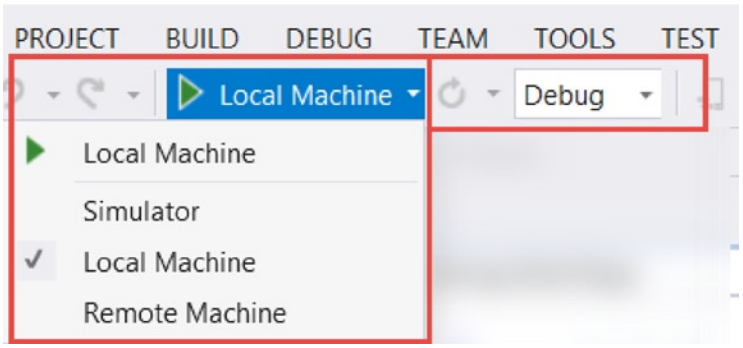


Figure 1-3. *Run toolbar utility*

By default, Visual Studio will run your app on the local machine in Debug configuration. Go ahead and click on the green arrow (or press F5) to run the app. We would expect to see a completely blank screen, but instead we are presented with some changing numbers (they change as you move the mouse around the screen) in the top corners of the screen, as shown in Figure 1-4. The frame rate counters show you, from left to right, the UI frame rate (frames per second), the App CPU usage of UI thread, the system composition frames per second, and system UI thread CPU usage. If you run the app without debugging, you will not see the numbers in the corner. This is because all of the Visual Studio–supplied templates enable the frame rate counter display while running in debug mode.

Figure 1-4. *Debugging with FrameRateCounter*

Turning this off is very simple—you just open `App.xaml.cs`, and in the `OnLaunched` event handler, comment out this line of code:

```
this.DebugSettings.EnableFrameRateCounter = true;
```

so that it looks like this:

```
//this.DebugSettings.EnableFrameRateCounter = true;
```

Now, when you run you app in debug mode, the numbers are no longer displayed.

Adding a Basic Page

Even though I typically start with the Blank App template, I rarely keep the supplied `MainPage.xaml` (and its code behind file `MainPage.xaml.cs`). Visual Studio provides a Basic Page file template that provides a lot of necessary functionality. Delete the `MainPage.xaml` (we will be replacing this), and right-click your project and select Add ➤ New Item. From the Add New Item—App 1 dialog, select the Basic Page and name the page `MainPage.xaml`, as shown in Figure 1-5.

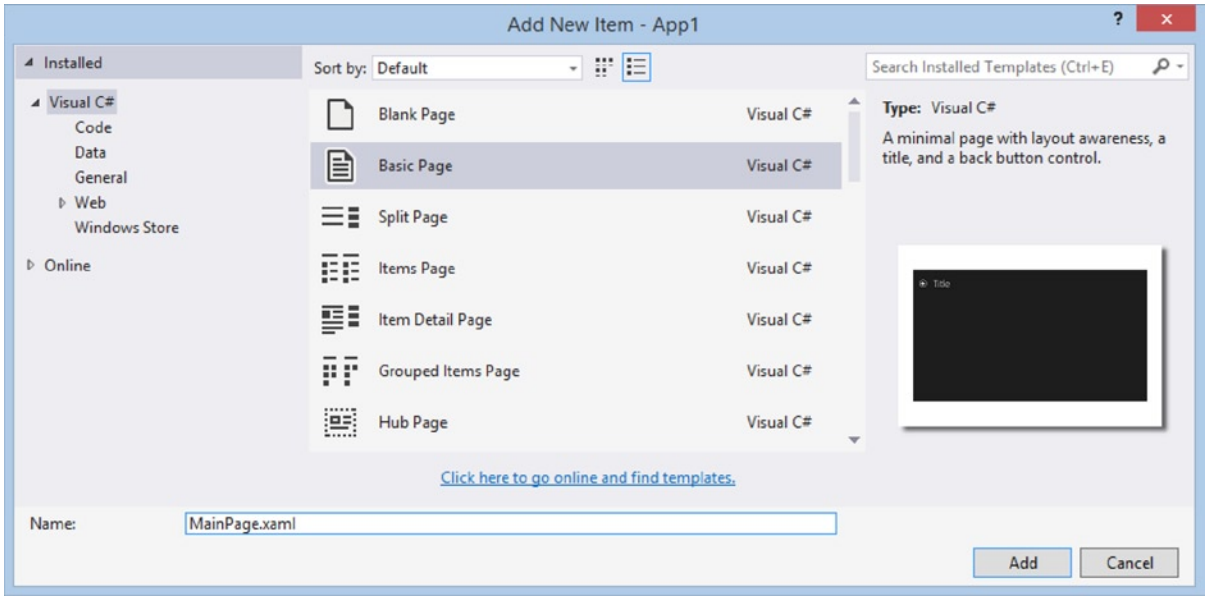


Figure 1-5. *Adding a new Basic Page*

■ **Note** We call it `MainPage.xaml` so we don't have to change `App.xaml.cs`. If you want to call the files something else (or change the page that gets loaded when an app first starts, open `App.xaml.cs`, navigate to the end of the OnLaunched event handler, and change the following line to the name of the page you added:

```
rootFrame.Navigate(typeof(MainPage), e.Arguments);
```

When you add a new Basic Page, Visual Studio prompts you that it will add several files into your project. Say Yes! These are extremely helpful files and will be used extensively though the course of this book. However, for now, we just want to have some text to display. Change the option on the debug location toolbar to run in the simulator, and then press F5 (or click the green arrow to start debugging). You'll now see a title for the app (running in a window that resembles a tablet) and a series of controls on the right rail of the simulator, as shown in Figure 1-6.

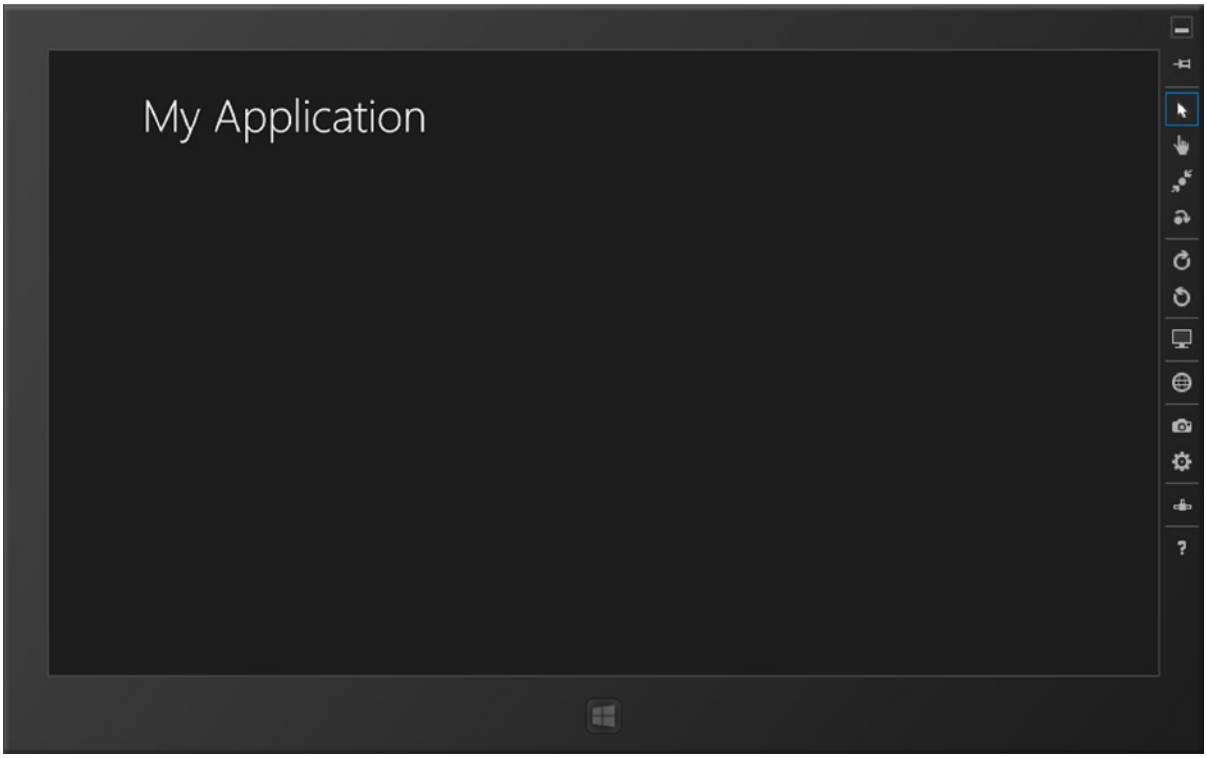


Figure 1-6. *The Simulator*

The Simulator Controls

Most of the simulator controls are very self-explanatory, but I struggled in my early days of Windows 8 apps to remember what each icon stood for, so I've listed the explanations here to help you out.

 Minimize the simulator

 Always keep the simulator on top

The touch modes in the simulator are important to be able to test your app's responsiveness to touch if you don't own (or develop on) a touch device. The mouse mode button takes you back out of touch mode to keyboard and mouse mode.

 Mouse mode


Basic touch mode, pinch/zoom touch mode, rotation touch mode

The rotation and resolution controls help testing by responding to different orientations and form factors.


Rotate clockwise (90 degrees)/rotate counterclockwise (90 degrees)


Change the resolution

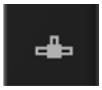
If you are building a location-aware application, you can test that by setting the location that is sent to the app from the hardware.


Set location

The screenshot commands are invaluable for the submission process, as you will see in Chapter 12. They are also useful to create screenshots for building documentation, advertising your app on your website, and so on.


Copy screenshot/screenshot settings

The network control allows for testing occasionally connected scenarios, bandwidth usage, and making other networks variables.


Change network properties


Help

Blend for Visual Studio 2013

Expression Blend has long been a staple of the WPF developer. Long sold as a separate product from Visual Studio, it was part of the Expression suite. Starting with Visual Studio 2012, Blend for Visual Studio was released as a free companion application for Visual Studio. Unfortunately, the first iteration left the XAML developer behind in the dust and completely focused on the HTML/JavaScript developers for Windows 8 apps.

That has been fixed, and Blend for Visual Studio 2013 is now back with a vengeance to help XAML developers. To open your project in Blend, you can right-click on any XAML file in Visual Studio 2013 and select Open in Blend. This will open not just the file that you selected but also the entire project/solution.

Many of the features of Blend are covered in subsequent chapters, but some of the biggest benefits of using Blend are:

- Full control of your UI in a compact layout—the Visual Studio XAML designer pales in comparison to what can be accomplished in Blend. While I am not a designer (and don't make any claims to having design skills), Blend has enabled me to make much-better-looking UIs as well as to make changes much faster than in Visual Studio (regardless of being in design or XAML mode in Visual Studio).
- The ability to easily add animations, gradients, and styles to your app/page
- The ability to quickly add states to your page (for layout updates) and state recording
- The ability to view your page in many layouts and form factors (much like the Simulator, but without the benefit of the page running—WinJS/HTML developers still have the advantage here)

Additionally, Visual Studio and Blend for Visual Studio keep your files in sync. If you have your project open in both, when you make changes (and save them) to your app/pages in one program, switching to the other program will prompt you to reload. Make sure that you actually save the changes, as making changes in both without saving will result in concurrency problems.

Opening Your Project in Blend for Visual Studio

Visual Studio and Blend work extremely well together. To open your project in Blend, right-click on the `MainPage.xaml` in your project and select Open in Blend (see Figure 1-7).

Figure 1-7. *Opening a file in Blend*

Visual Studio invokes Blend, opening your entire project (not just the file you clicked on). Once the file is opened, you will see a screen similar to Figure 1-8. Blend will open the file you right-clicked on in Visual Studio.

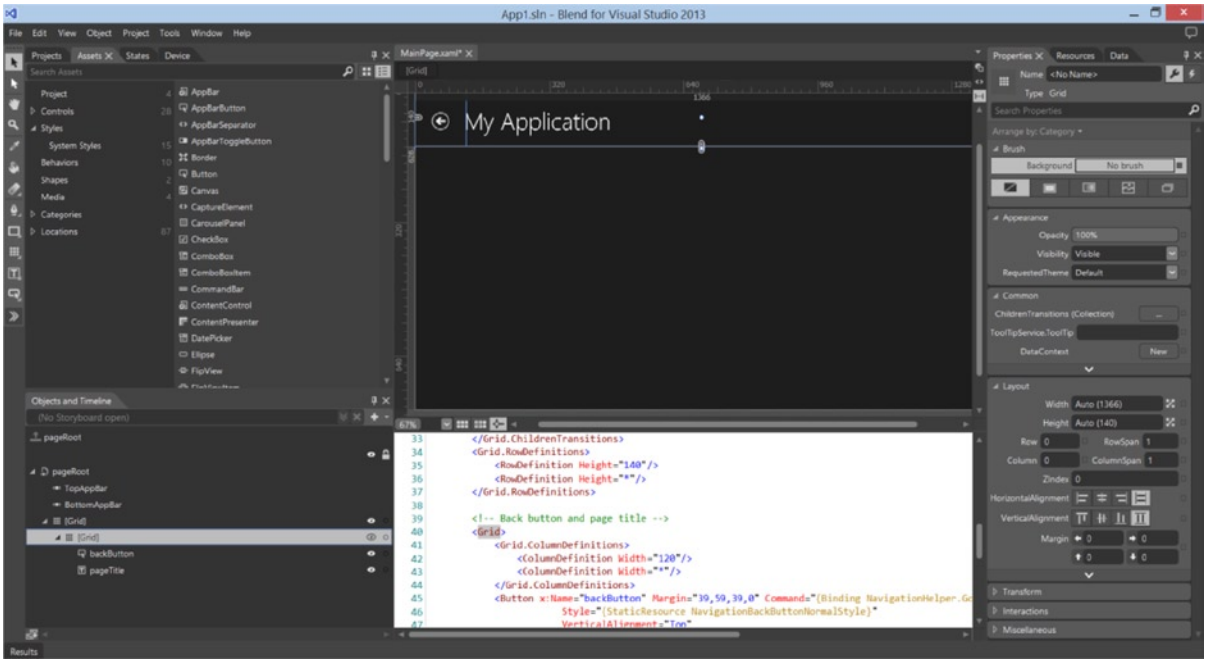


Figure 1-8. *MainPage.xaml opened in Blend*

That's a lot of windows, but at least in the default layout (much like Visual Studio, you can change the layout to suit your needs). Let's look at them in a little more detail.

Projects, Assets, States, and Device Tabs

The top-left corner of the window contains the Projects, Assets, States, and Device tabs, which allow you to do the following processes:

- The Projects tab shows all of the files in your solution (much like Solution Explorer in Visual Studio). Nothing too exciting to report here.
- The Assets tab lists all of the assets available to add to your page. Think of this as a turbo-charged Visual Studio Toolbox. In addition to controls and panels that you can add to your page, you can also add (and modify) styles, behaviors, and media.
- The States tab allows you to add the Visual State Manager XAML as well as Visual State groups to your page. It also allows for easy addition of transitions for your visual states.
- The Device tab allows you to change the resolution and orientation as well as connected edges (more on this in subsequent chapters). You can also change the theme (between light and dark) as well as the minimum width.

Objects and Timeline

The Objects and Timeline panel (lower left) provides the document outline as well as the ability to add and modify storyboards (to be used in conjunction with the Visual State Manager).

Page Designer, Markup, and Code

The center of the workspace is the designer and code editor. Just like in Visual Studio, you can have a split view, all design, or all markup. You can also load code files into the center pane. While you get features like Intellisense, the development experience doesn't contain all of the great features of Visual Studio like navigation and refactoring. Plus, you lose any productivity plug-ins like Telerik's JustCode that you might have installed in Visual Studio.

Properties, Resources, and Data Tabs

The right rail of the workspace contains the Properties, Resources, and Data tabs, which can be described as follows:

- The Properties tab is where I spend a significant portion of my time in Blend. In addition to the simple items like Name and Layout and properties like `Width` and `Height`, there are a host of properties that are difficult to set by hand in markup. Brushes, Transforms, and Interactions can all be set using the Properties panel.
- The Resources tab contains all of the application and page-level resources as well the option to edit and add more resources.
- The Data tab allows you to set the data context for your page, create sample data, and create different data sources. This is helpful to see what the page will look like with data at design time instead of always having to run the app.

Blend for Visual Studio is an extremely powerful tool and it would take an entire book to discuss all of the features. My development workflow involves keeping both Visual Studio and Blend open at the same time, and I switch back and forth depending on what I am trying to accomplish. Explore Blend, and see what works best for you.

Git

Software version control has been around for a long time. If you have been in the Microsoft space for a significant length of time, you might remember Visual Source Safe. In the .NET world, the MS developer was left with Team Foundation Server (TFS) as the only integrated source-code-management (SCM) system.

TFS is a powerful application lifecycle management (ALM) tool (including project management, bug tracking, SCM, and other components). That is a lot of tooling when you are only looking for SCM. The SCM portion of TFS is Team Foundation Version Control (TFVC) and is a centralized SCM system. This means that a single repository is the source of record, and all developers check their code in and out of this single repository. Later versions of TFVC have included the capability to shelve work and create branches, providing some isolation for work in progress.

Git, developed by Linus Torvalds in 2005 for the Linux kernel, is a distributed version control system (DVCS). This means that every developer using Git has a full-fledged repository on his local machine with complete history and tracking capabilities. Many Git users (especially in a team environment) have a central repository in addition to their local repository. This frees the developer to spike different ideas, work on features independent of the rest of the team, and check in rapidly as often as they like without worrying about network latency or affecting other team members.

Which SCM system you choose to use is completely up to you. They both have their merits (and there are many other SCM systems available to you as well that are very effective in what they do). It's more how you work and whom you work with that usually determines which system to use. So why do I bring up Git specifically in this book? Because if you are a single developer creating a Windows 8 app, Git is custom tailored to you, and with Visual Studio 2013 (and updated to Visual Studio 2012), Git support is now included.

There are entire books written about effectively using Git, so this is just a quick look into the Visual Studio integration, and not a treatise on DVCS.

Using Git in Visual Studio

One of the advantages of using Git is its simplicity. A Git repository can be created anywhere—on a local disk, network share, or web site (like GitHub).

GitHub for Windows

The easiest way to start working with Git if you are new to the system is to install GitHub for Windows, which is available from `https://windows.github.com/`. Creating a new repository is as easy as clicking on the Create button in GitHub for Windows. Once Visual Studio is configured to use Git, any projects created inside an existing repo will automatically tie into the Git repo.

Enabling Git in Visual Studio 2013

The first step to using Git with your project is to enable the Microsoft Git Provider. Do this by selecting Tools ➤ Options ➤ Source Control ➤ Plug-in Selection, and then select the Microsoft Git Provider for the Current source control plug-in, as in Figure 1-9.

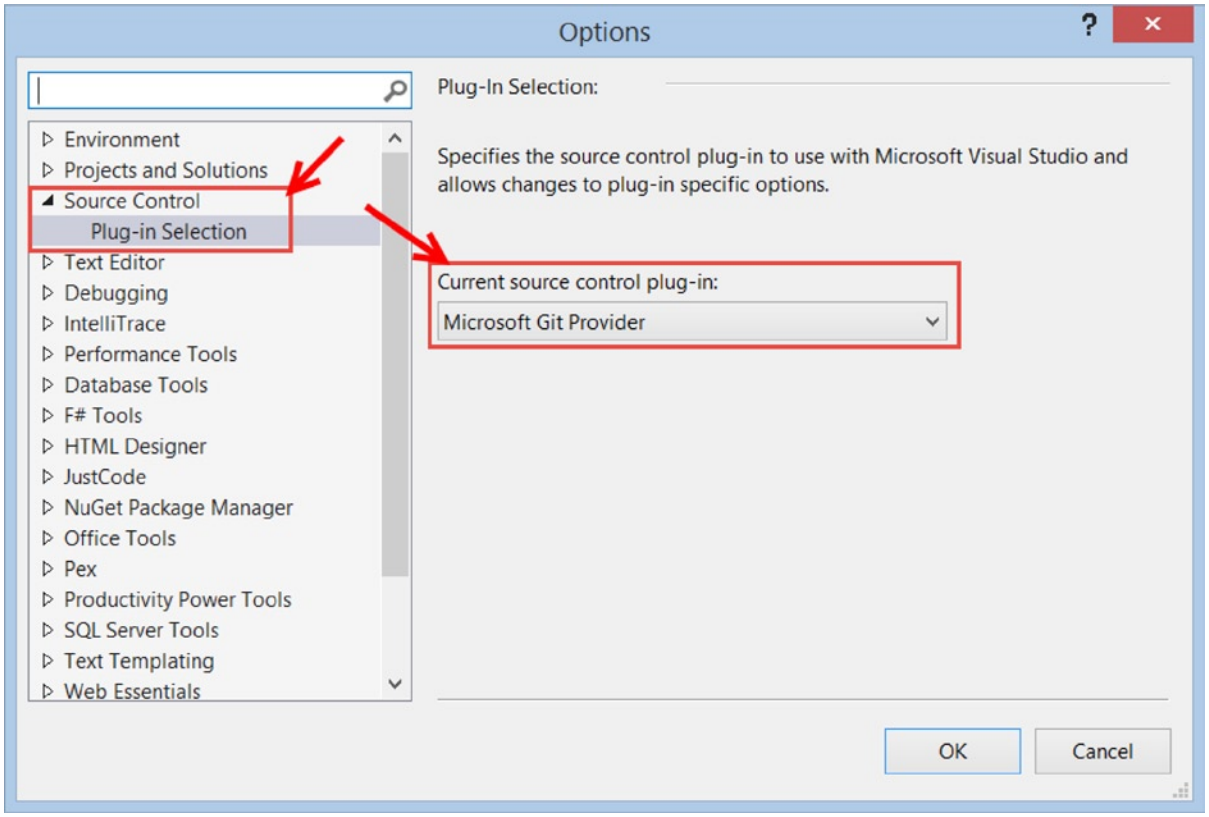


Figure 1-9. *Selecting the Microsoft Git Provider*

Selecting Team Explorer (View ➤ Team Explorer) in the right rail of Visual Studio (the default location) allows you to manage your local Git repository. By default, VS 2013 creates the appropriate Git ignore files so local files such as `/bin` and `/obj` files, temp files, user files, and so forth don't appear in the repository. There are also attributes on how Git should handle conflicts in project files. To view both of these files, select Git Settings, as shown in Figure 1-10.

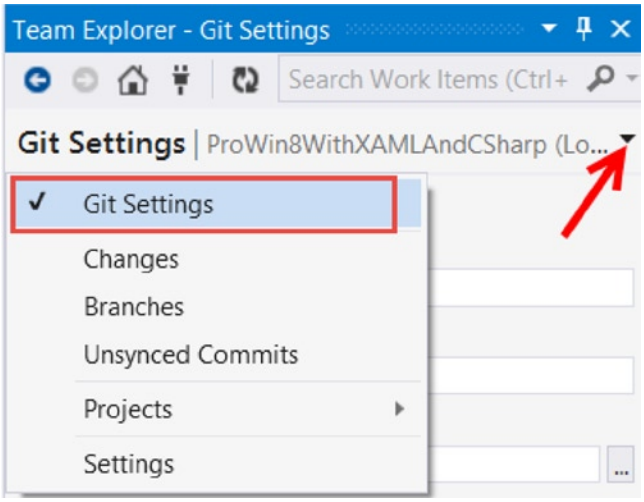


Figure 1-10. *Accessing the Git repository settings*

This is also where you enter your username and e-mail address as well as the default Git directory, as shown in Figure 1-11.

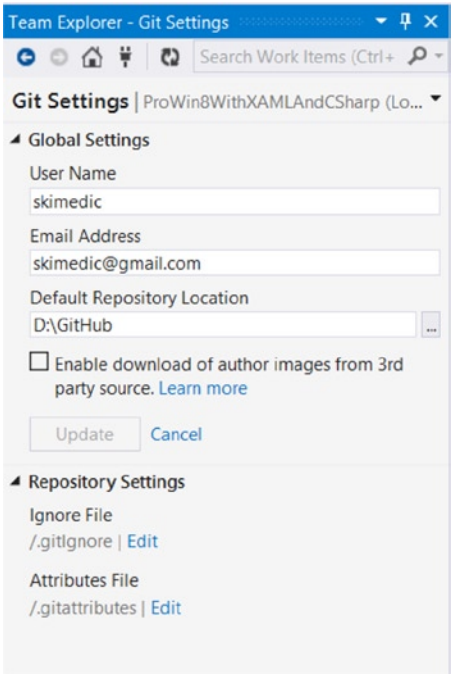


Figure 1-11. *Git Settings*

Checking in Changes

To check in changes, select Changes from the same menu, as shown in Figure 1-10. You will see changes that will be included in this check-in and excluded changes as well as untracked files. To commit the changes, enter a comment in the text box with the watermark "Enter a commit message <Required>" and click on Commit. You can also Commit and Push to a remote repository to share your changes, or Commit and Sync with a remote repository to share your changes and get the latest version from the remote repository as shown in Figure 1-12.

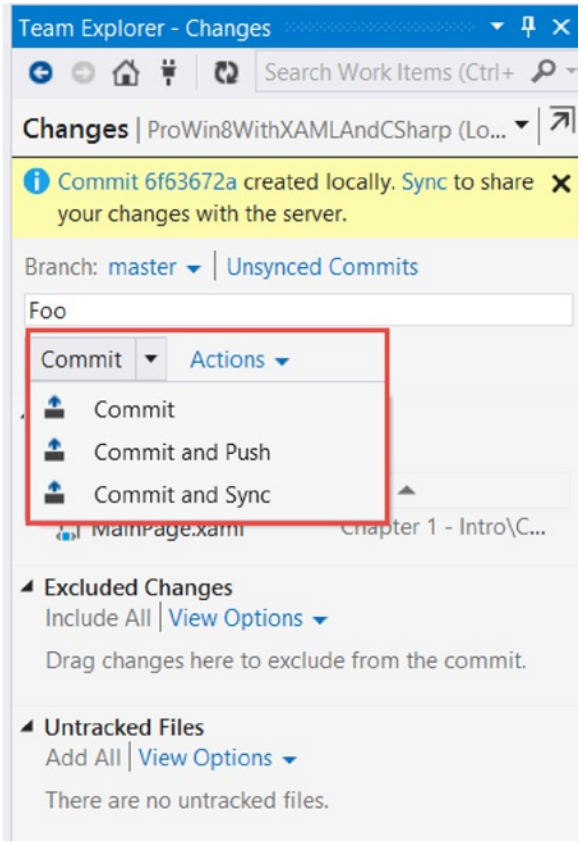


Figure 1-12. *Committing changes to the local repository*

Remote Repositories

There are many places where you can host remote Git repositories, with the most popular being GitHub (`https://github.com`). Once you set up a remote repository, you can point your project to it by entering its URL, as in Figure 1-13.

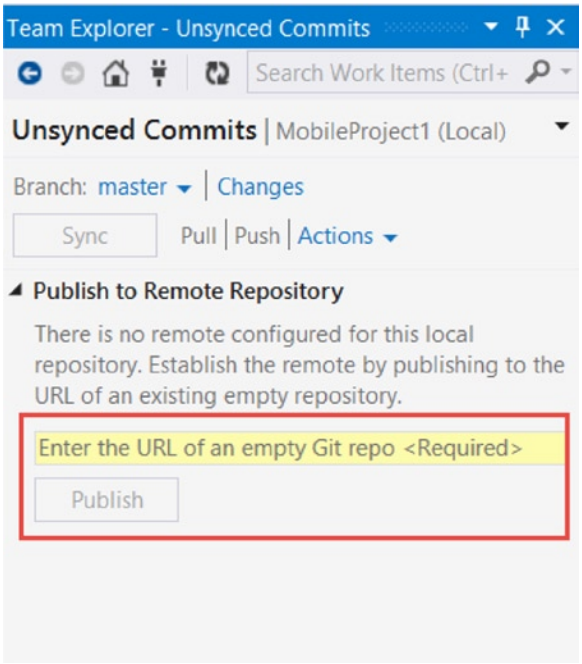


Figure 1-13. *Publishing to a remote repository*

Reverting Changes

If you totally mess up while developing, Git makes it very easy to restore from the repository. Right-click on your file in Solution Explorer and you will see the Git features exposed: Undo, View History, Compare with Unmodified, and Commit (see Figure 1-14). Undo does just what it says—it throws away your changes and restores the file from the repository. It's like your own personal security blanket!

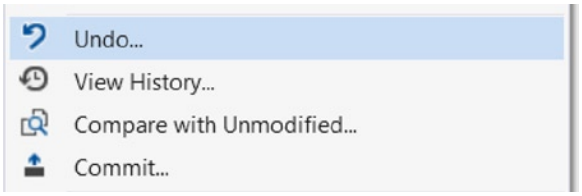


Figure 1-14. *Git functions exposed through Solution Explorer*

Again, this isn't a full explanation of how Git works but a quick overview of the Visual Studio features that support Git. If you've never used source code control systems, Git is an easy first one to use. You'll thank yourself in the end.

NuGet

From the official NuGet site (www.nuget.org): "NuGet is the package manager for the Microsoft development platform including .NET. The NuGet client tools provide the ability to produce and consume packages. The NuGet Gallery is the central package repository used by all package authors and consumers."

Instead of scouring the web for tools to add into Visual Studio, you can use NuGet as a single source to get a wide variety of add-ins for your solution. Rather than installing the tools on your development machine, the packages are installed at the *solution* level. This permits different versions to coexist on the same developer machine.

Another very large advantage to NuGet is that each package lists its dependencies within its package manifest. When a package is installed through NuGet, all of its dependencies get installed as well.

Yet another benefit of NuGet is the ability to create private NuGet package sources. To change the source, select Tools ➤ Options ➤ NuGet Package Manager ➤ Package Sources, as in Figure 1-15.

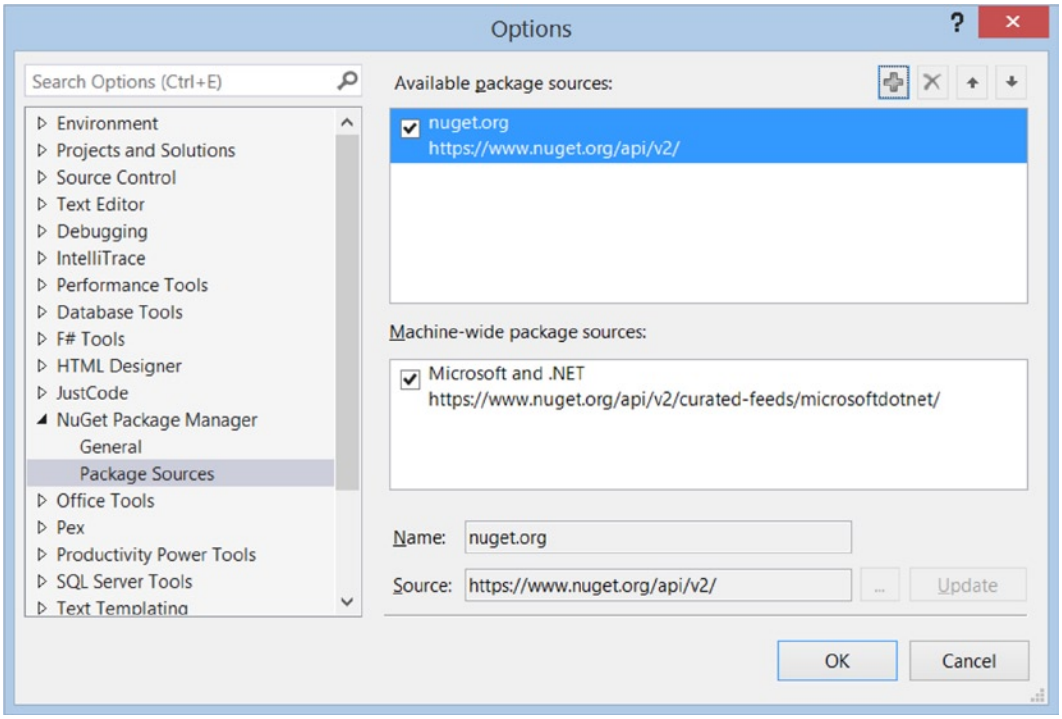


Figure 1-15. *NuGet Package Source dialog*

Installing NuGet

In the off chance that NuGet wasn't preinstalled with Visual Studio, installation is easy. It's available in the Visual Studio Extension Gallery (accessible from Tools ➤ Extensions and Updates).

Once the Extensions and Updates window is open, select Online ➤ Visual Studio Gallery in the left rail. In the search box, enter NuGet, and look for the NuGet Package Manager for Visual Studio 2013. In Figure 1-16, there is a green check mark by the extension since I already have NuGet installed.

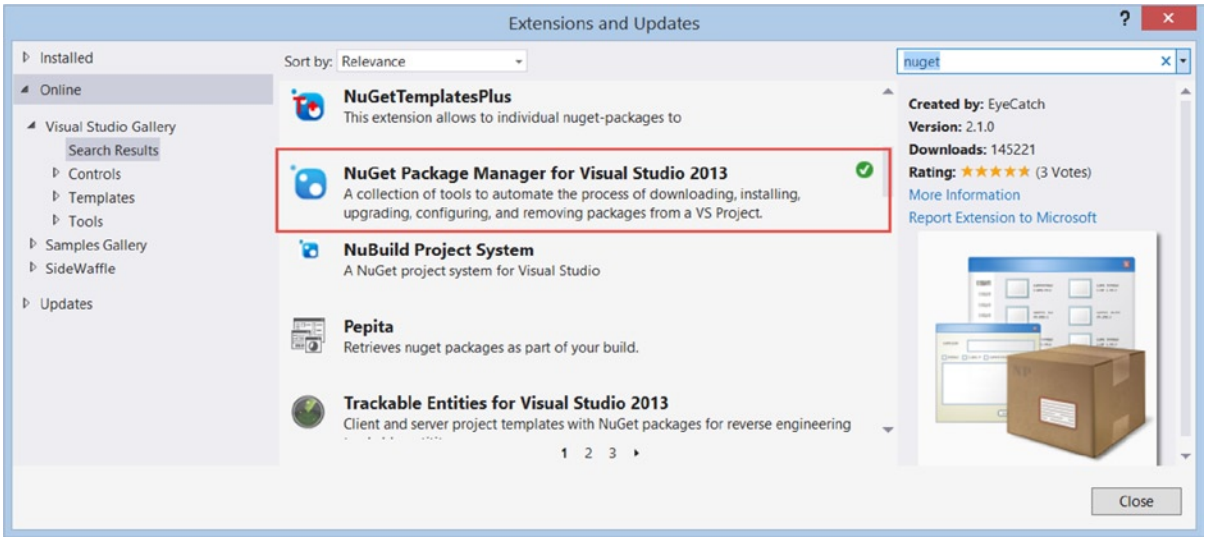


Figure 1-16. *Installing NuGet Package Manager*

Enabling Package Restore

Package Restore is a feature that can significantly decrease the size of your project when shipping source code (note that this doesn't affect checking in/out of your SCM system). All of the NuGet packages are contained in a folder in your solution aptly named "Packages." By default, Windows 8.1 projects don't have many packages installed, but if you create an ASP.NET project, you will see a lot of packages, only some of which are used by default.

To enable Package Restore, right-click on your solution (note that it is not the project file) and select Enable NuGet Package Restore. You will be prompted with a series of dialogs. as shown in Figures 1-17 through 1-19.

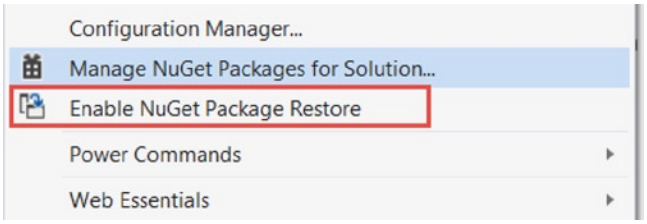


Figure 1-17. *Enabling Package Restore*

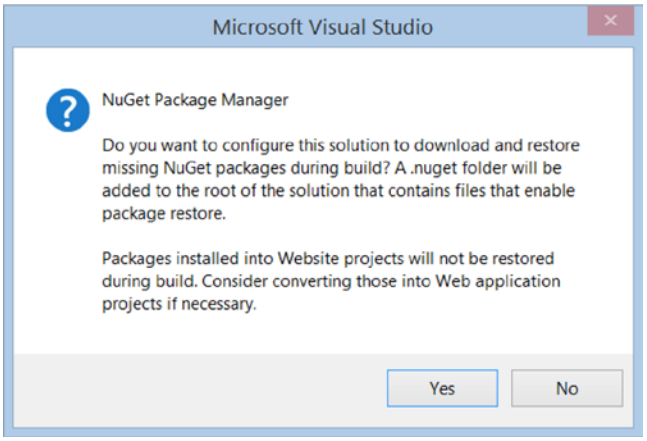


Figure 1-18. *Confirmation dialog for Package Restore*

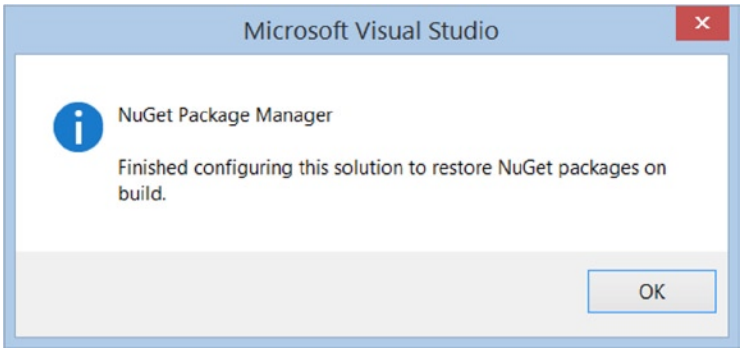


Figure 1-19. *Confirmation dialog*

Once you have enabled Package Restore, you will see the changes to your project as shown in Figure 1-20.

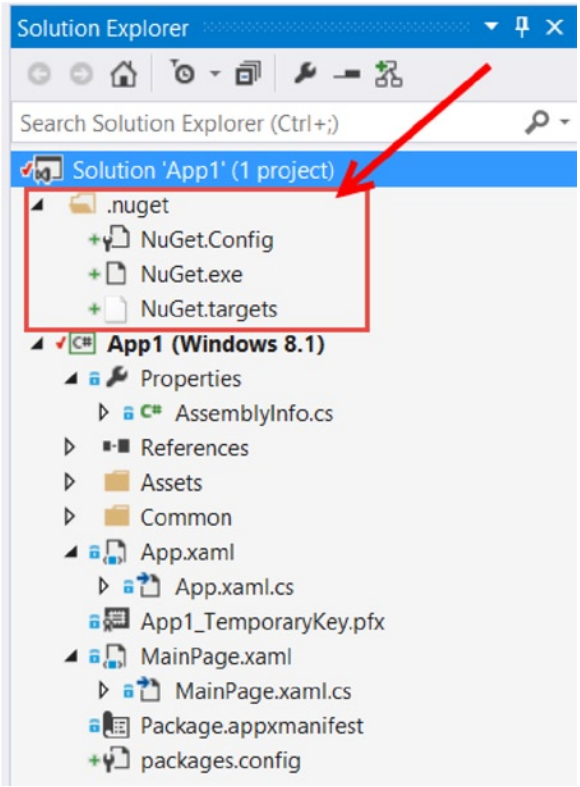


Figure 1-20. *Changes to the solution*

Installing Your First Package

One of the "Can't live without" packages for developing Windows 8.1 apps is Newtonsoft's Json.NET. We'll use Json.NET later in this book, but for now, let's just get it installed. There are two ways to install packages—by using the Package Manager Console command line or by using the Package Manager GUI.

Installing from the Command Line

Access the Package Manager Console by selecting View Ȩ Other Windows Ȩ Package Manager Console if it isn't currently visible in the bottom rail of Visual Studio.

Type "install-package newtonsoft.json" and you'll see the dialog shown in Figure 1-21. At the time of this writing, 6.0.1 is the current version. NuGet will install the current version unless you specify a version. Another benefit of using NuGet.

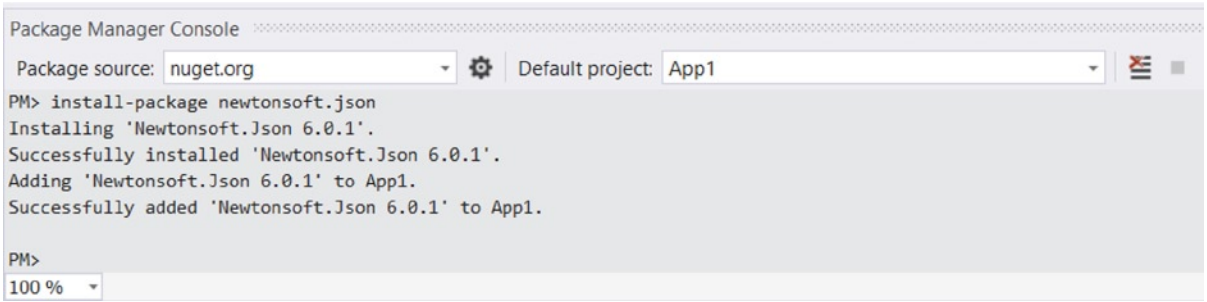


Figure 1-21. *Command line installation of Json.NET*

Installing from the Graphical User Interface

Installing from the graphical user interface (GUI) is very simple, and provides a search mechanism if you don't know the exact name of the package that you are looking for. For example, everyone refers to the package as "Json.NET." The actual package name in NuGet is "newtonsoft.json." This is a great example of where the search in the NuGet GUI is very helpful.

To access the GUI, right-click on your solution and select Manage NuGet Packages for Solution, as in Figure 1-22.

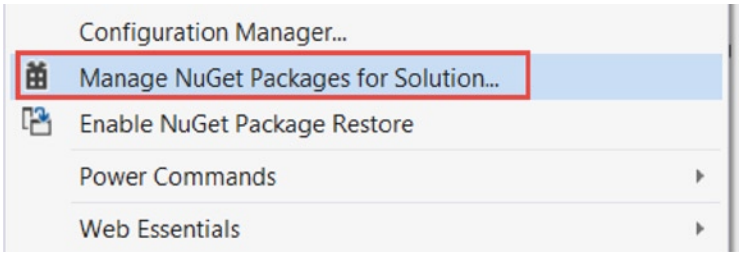


Figure 1-22. *Launching the NuGet GUI*

Select Online in the left rail and enter Json.NET in the search dialog. You will see results similar to Figure 1-23. Merely click install to install Json.NET.

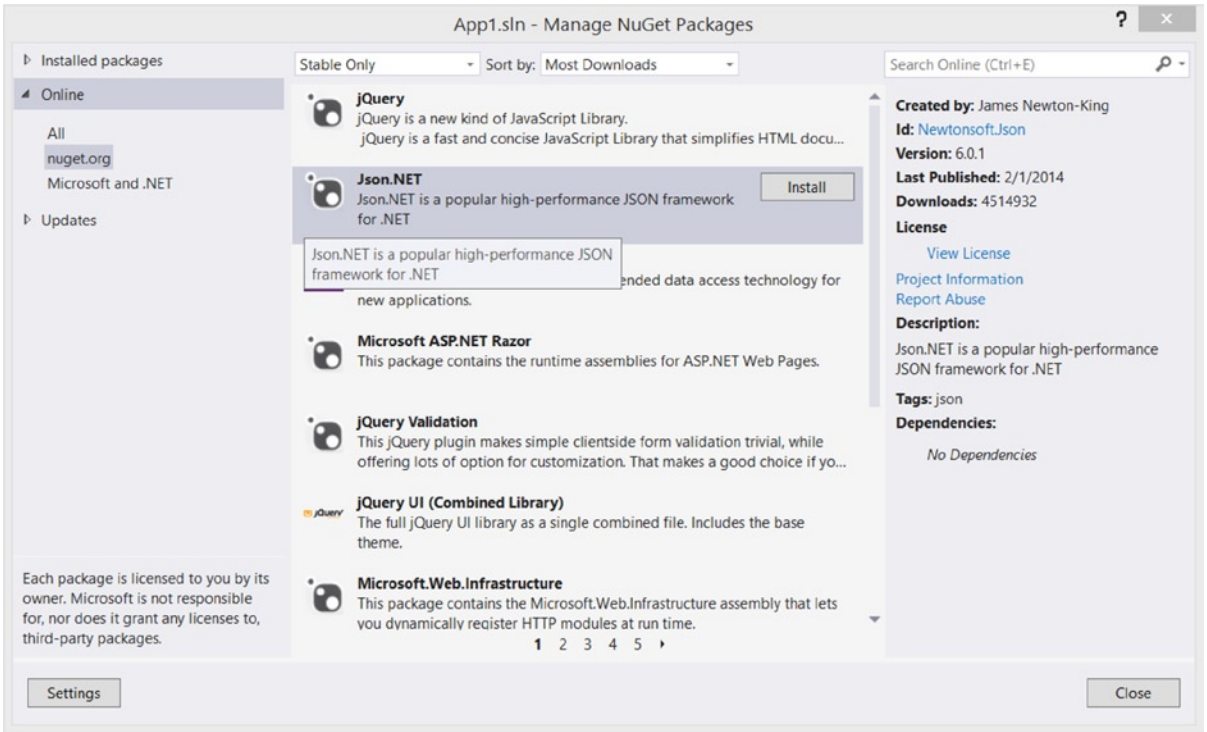


Figure 1-23. *Installing Json.NET with the Package Manager GUI*

Summary

Windows 8 apps represent a very large paradigm shift from traditional Windows desktop applications (such as WPF or WinForm) or web apps (such as ASP.NET Webforms or MVC). Whereas traditional applications were developed with a wide range of tools but no real design guidelines and no expectations of performance, Windows 8.1 apps must meet a series of expectations, both in terms of UI design and app performance. They are distributed through the Microsoft Store after a stringent certification process.

Developing Windows 8 apps involves a lot more than just Visual Studio. Blend for Visual Studio helps build compelling UIs, Git provides security for your source code, and NuGet enables easy addition of packages and add-ons to Visual Studio.

Now that you know the tools to use, let's build that first app!

CHAPTER 2

Building Your First Windows 8 App

Chapter 1 covered the design guidelines as well as the tooling commonly used to build Windows 8.1 apps. In this chapter, we will cover some of the core principles of Windows 8.1, including its architecture, all of the many parts of its apps in Visual Studio, the Model-View-ViewModel pattern, and navigation. All in the context of building your first Windows 8.1 app.

Windows Architecture (For Developers)

There are many options when choosing how to develop apps that can run on Windows 8.1 machines as you can see in Figure 2-1, and even more options when those Windows 8.1 machines are based on the x86 or x64 chipset.

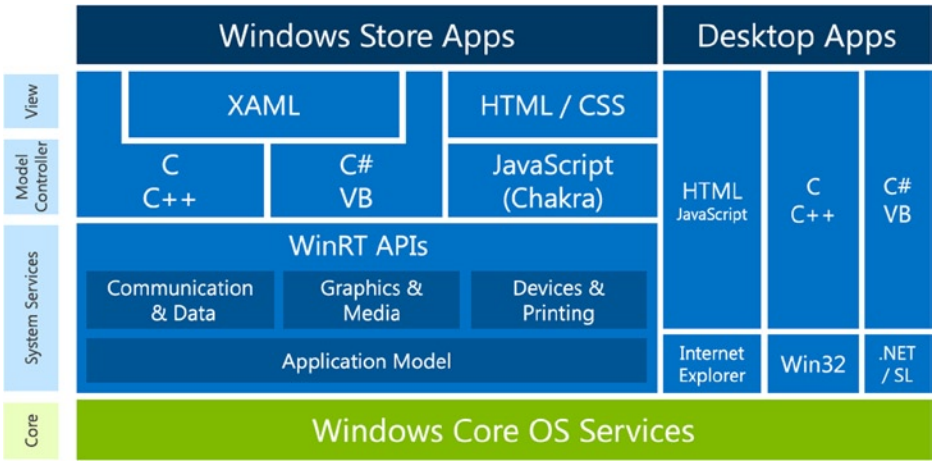


Figure 2-1. *Windows architecture*

Windows 8.1 apps that run in the modern interface (formerly called Metro; sometimes also called the Windows Store interface) can be developed with an XAML- or HTML-based UI. XAML-based apps can be written with C++, C#, or VB.NET. HTML-based apps are developed using JavaScript, leveraging the WinJS library. This book is about writing Windows 8.1 apps using XAML and C#. If you are interested in C++, VB.NET, or WinJS/HTML, Apress has an extensive library of books on those topics. Throughout this book, we'll dig deeper into the system services and the Windows 8.1 Core.

You can also develop applications for the desktop mode of Windows 8 on non-ARM-based devices (such as the Surface Pro). In desktop mode, you can still develop "traditional" applications such as smart client and browser-based ones, using all of the tools that you are familiar with such as Windows Presentation Foundation, ASP.NET, and even Silverlight and Winforms. For all of these topics, Apress has many books that can help you become even better with many great books on these subjects. (See `www.apress.com` for the full Apress catalog.)

Creating Your First App

To create our first app, let's start Visual Studio and select File ➤ New ➤ New Project, and then select Templates ➤ Visual C# ➤ Windows Store. We'll talk about the different templates in later chapters, but for now just select the Blank App and leave the default name as App1, as in Figure 2-2.

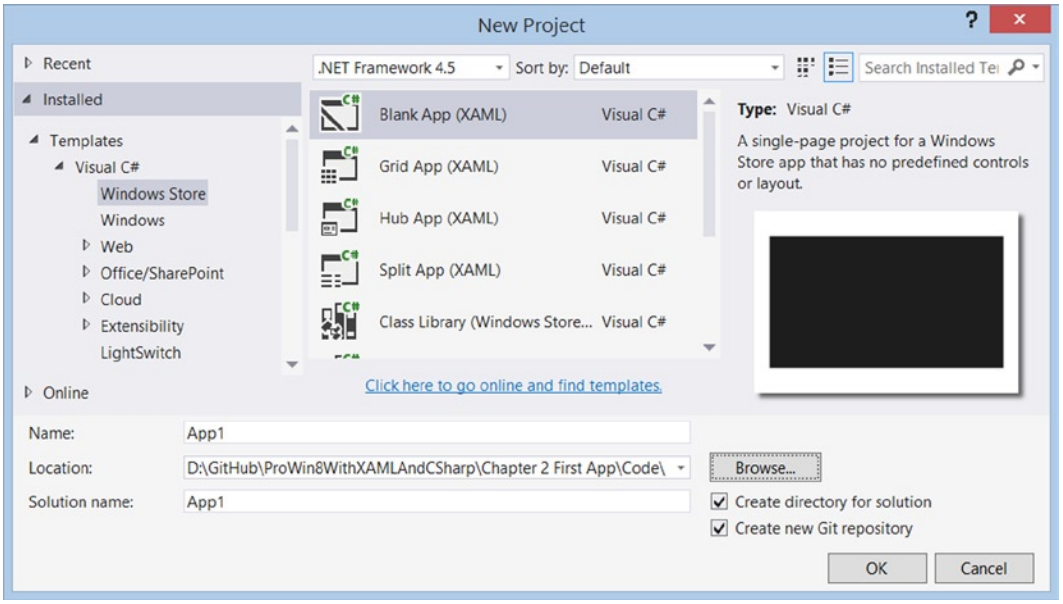


Figure 2-2. *Creating a new Windows 8.1 app*

Let's look at the nodes and files that are created as part of the template. Much of the project should be familiar to you.

App Project Overview

The New Project template introduces a significant number of folders and files, as shown in Figure 2-3.

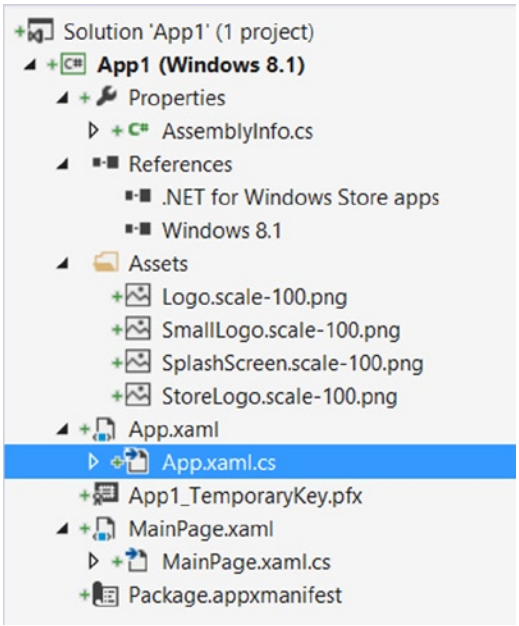


Figure 2-3. *Folders and files in the Blank App template*

Properties

Under the Properties node in the New Project template is the `AssemblyInfo.cs` file, the standard metainformation container for C# projects. Feel free to update the information such as description, copyright, and so on. Most of this information isn't necessary for modern apps, but I tend to update the information anyway out of habit.

References

The template also includes the standard References node, which is prepopulated with two references: .NET for Windows Store apps and Windows. These references provide that vast majority of functionality, as diagrammed in Figure 2-1, and must be included. Throughout this book, we will add additional references to supplement the default features available to us.

Assets

There is also the Assets folder, which contains all of the images that are part of your application. The tile images and splash screen graphics go in this folder as well as any other images or assets that need to be packaged with your app when it gets deployed. Click on one of the images in Solution Explorer (such as `Logo.scale-100.png`) and press F4 to view the properties. The Build Action for the images is set to Content and set not to copy to the Output directory, as in Figure 2-4. Alternatively, you can have the content copied to the Output directory or run a custom tool, although you will not want to do that for the images.