TOP TEN Xamarin TIPS AND TRAPS

Published By:

INFRAGISTICS®

By Charles Pluta

A Note from Infragistics

THANKS FOR CHECKING OUT *Top 10 Xamarin Tips and Traps*. Whether you are an experienced or new mobile app developer, the ten topics presented in this eBook provide insights into Xamarin's capabilities and resources to help you get started with it. Among the topics you'll learn about:

- accessing Xamarin and GitHub resources
- versioning with Xamarin
- integrating NuGet packages
- increasing app performance
- rendering data natively

If you're considering writing apps in Xamarin, you probably already know that it enables you to create cross-platform native apps that run on Apple iOS, Android, and Universal Windows Platform (UWP). You also probably know that the Xamarin SDK and its .NET core are open source development platforms for iOS, Android, Universal Windows Platform (UWP), and Mac.

You might also *think* you know about Xamarin's design-time trade-offs for its cross-platform code sharing. Why the italics? Because a new tool from Infragistics—Ultimate UI for Xamarin—eliminates those trade-offs and makes Xamarin app development a no-compromise solution.

Meet—and Try—Ultimate UI for Xamarin

Xamarin is a versatile development framework, and this book will give you valuable tips on how to use it. We think you'll go even farther, faster, if you also check out Infragistics *Ultimate UI for Xamarin* before you start coding. Ultimate UI for Xamarin combines lightning-fast controls with a RAD WYSIWYG design-time experience, empowering you to build beautiful, quality, high-performing applications with a rich UX and a robust feature set.

Ultimate UI for Xamarin includes flexible iOS, Android, and Xamarin.Forms controls designed for the most demanding apps, *outperforming all other Xamarin controls on the market*. Ultimate UI for Xamarin also includes the ground-breaking Xamarin.Forms Productivity Pack, delivering unparalleled time-savings and productivity for Xamarin. Forms developers. The Productivity Pack includes "AppMap," a visual page-creation tool call, and control configurators which eliminate the need for manual XAML coding, greatly speeding up app control layout. You'll also get page templates and code snippets.

- **It's Fast:** Even with real-time and large data sets, no other Xamarin grids and data chart controls outperform Infragistics Ultimate UI for Xamarin.
- **It's Versatile:** Ultimate UI for Xamarin includes iOS and Android native controls, along with Xamarin.Forms controls for maximum code-sharing.
- **It's Easy to Use:** AppMap, control configurators, and templates enable rapid application view and view model creation and visual control configuration.
- **It's Easy to Learn:** Visual control configurators enable you to easily and rapidly learn rich Xamarin Forms controls.

In short, Ultimate UI for Xamarin is the first no-compromise library of controls and productivity tools for Xamarin. And you can *try it right now for free*.

Learn More

Once you install Ultimate UI for Xamarin, get productive in just a few minutes with six *write fast, run fast* lessons for you to watch, read, and try.

When you're ready to dig in a little more deeply, check out *Moo2U*—an end-to-end, best practices Xamarin.Forms reference application, built with Ultimate UI for Xamarin's Productivity Pack and UI widgets. Moo2U not only demonstrates how to build a stylish, scalable Xamarin.Forms app.

We look forward to hearing how you like this book…and how we expect and hope you'll *love* Ultimate UI for Xamarin.

Best,

Ken Rosen (krosen@infragistics.com)
Sr. Director, Product Management
Infragistics

TOP TEN
Xamarin
TIPS AND TRAPS

Contents

#1 TIP

Accessing Xamarin resources

LEARNING HOW TO USE Xamarin and its components to develop native applications is like learning how to ride a bicycle: It might seem difficult or complicated at first, but becomes simple with practice. As with anything that you learn to do the first time, preparation is a crucial element of success. There are common pitfalls, mistakes, or oversights that can occur for even the most experienced developer. Fortunately, there are tons (literally, if you printed them!) of resources available to help ensure your success. These include resources from the following providers: Xamarin SDKs

- Xamarin University
- Xamarin Developer Center
- Xamarin on edX
- Microsoft Virtual Academy
- Third-party resources

Xamarin SDKs

The source code for Xamarin.iOS, Xamarin.Mac, Xamarin.Android, and Xamarin.Forms is available from the Xamarin website at *http://open.xamarin.com as well as on the Xamarin GitHub*. The open source nature of Xamarin enables you to create native apps for any device in C# or F#. You can also take advantage of open source bindings and native libraries for Facebook and Google Play, as well as useful functions such as messaging and GPS.

The source code, bindings, and libraries are available through the Xamarin GitHub. (For more information, see *TIP: Using GitHub resources*.)

Xamarin University

Xamarin University is an online platform that provides live and interactive training, including five free, on-demand classes in the Visual Studio Dev Essentials program. Xamarin University also offers self-guided classes, such as those shown in Figure 1, that you can work through at your own pace to learn Xamarin.

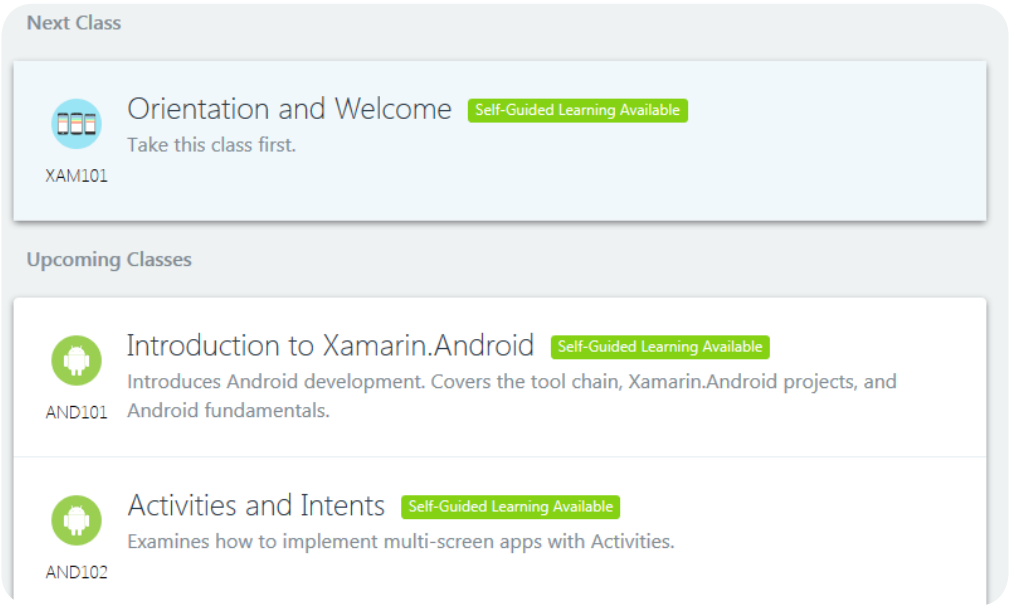

Figure 1: *The first few classes in the Xamarin University portal.*

Xamarin University also offers a Xamarin certification. You can combine the instruction from the self-guided classes, videos, and other available resources to complete the certification. You can access Xamarin University at *https://university.xamarin.com*.

Xamarin Developer Center

The Xamarin Developer Center provides a collection of online resources that help you successfully develop mobile apps. The Developer Center includes Xamarin documentation, guides, workbooks, recipes, samples, and more for the various platforms. Dozens of guides are available for cross-platform development and for individual platforms. Figure 2 shows the current categories of Xamarin Developer Center content.

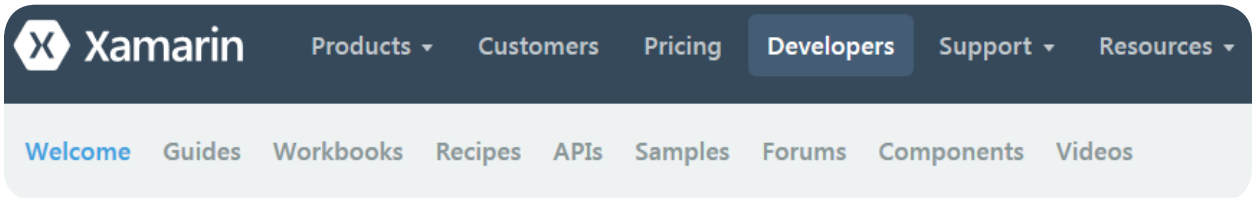

Figure 2: *Xamarin Developer Center category links.*

Some of the resources available from the Developer Center include:

- Interactive workbooks that enable you to experiment and explore within real projects. You can edit code and text to customize the project and practice using different

components of Xamarin. Workbooks are available for Azure, Mac, mobile and other platforms.
- Recipes that document a specific Xamarin component or resource. Recipes are available for the GPS, accelerometer, geocoder, and other device resources.
- APIs and sample projects that use common Xamarin features. Samples of 3-D scenes, physics manipulation, animation, rendering, lighting, and other features are available for various platforms.
- Add-in components for Xamarin and Visual Studio that simplify or add configuration utilities to the development process. Components are available for cloud services, plug-ins, themes, development resources, and more.

You can review the available documentation and resources in the Xamarin Developer Center at *http://developer.xamarin.com*.

Xamarin on edX

Microsoft also offers courses on the edX online training platform to provide self-paced developer-related courses for subjects including Microsoft Azure, Microsoft Exchange Server, and Xamarin. Currently available courses include a two-week introduction course to Xamarin.Forms, and a six-week *Programming with C#* course that includes useful information for new application developers. To locate the Xamarin.Forms course, search for Xamarin on *http://www.edx.org*.

Microsoft Virtual Academy

The Microsoft Virtual Academy (MVA), provides free video-based training for a variety of Microsoft and fundamental technologies. Several beginning, intermediate, and advanced Xamarin courses are available, including *Xamarin for Absolute Beginners*, designed for experienced .NET developers who want to use Xamarin. You can find the MVA at *http://mva.microsoft.com*.

Third-party resources

Many websites, including those described here, provide learning materials related to Xamarin.

- Pluralsight is a subscription-based learning portal that offers courses for a wide range of technologies. As of this writing, Pluralsight offers over 125 Xamarin-related

courses, ranging in length from 45 minutes to eight hours. Pluralsight is located at *http://www.pluralsight.com*.

- Lynda.com offers subscription-based learning that includes Xamarin-related resources. You can learn more at *http://www.lynda.com*.
- Tuts+ is a free online resource that provides how-to tutorials for various technologies. Several Xamarin-related tutorials are available. You can learn more at *http://www.tutsplus.com*.

#2 TIP

Using GitHub resources

XAMARIN AND INFRAGISTICS PROVIDE a vast amount of resources through their respective GitHub sites. Many open license projects are available on GitHub and can be used immediately in your next app. If the foundation for a module is available on GitHub, it isn't necessary for you to create it from scratch.

Xamarin GitHub repositories

The Xamarin GitHub has dozens of repositories for samples, components, workbooks, and more. The most popular Xamarin repositories include:

- Xamarin.Forms
- Xamarin.MaciOS
- Xamarin.Android

Each of these repositories has several available branches and builds. Each build includes a readme file that contains the build requirements and configuration steps for using the build. For example, the build requirements for a recent version of Xamarin. MaciOS include:

- Autoconf, automake, and libtool
- CMake
- Xcode
- Mono SDK
- Xamarin Studio

The readme file describes how to install the dependencies that are required to use the module. The primary Xamarin repositories can be found at *https://github.com/xamarin*.

Reporting a bug

Xamarin handles bug reporting through an instance of Bugzilla that is available at *https://bugzilla.xamarin.com/newbug*. If you encounter a problem with any of the

modules in the Xamarin repositories, provide the following information in the bug report:

- The steps to reproduce the bug.
- The expected behavior or result.
- The actual behavior or result.
- Any available supporting information, such as log files, images, or videos.
- The development or testing environment in which the bug occurs.

Samples

Another GitHub repository and resource for developing apps is Mono for Android and MonoTouch for iOS. Both iterations of Mono provide sample projects and use the native API for each platform to provide rich features and capabilities for the given platform. A typical sample project is the tip calculator shown in Figure 3, which was recently updated for Xamarin.Forms. This sample is available at *https://developer.xamarin.com/samples/xamarin-forms/TipCalc*.

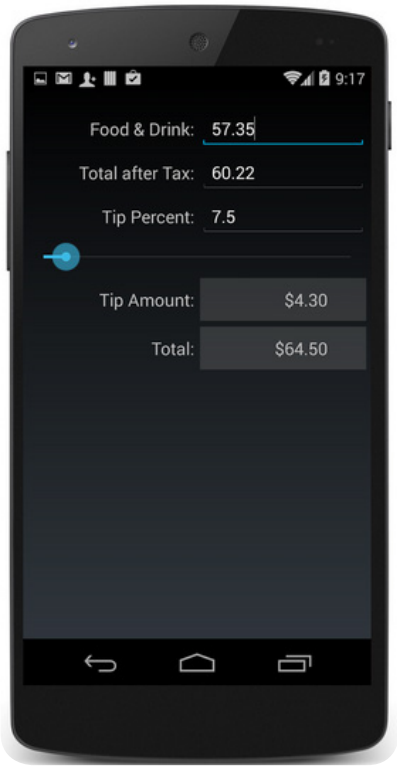

Figure 3: *Tip calculator sample.*

As previously mentioned, the Xamarin Developer Center provides samples for multiple platforms. Samples can be organized by platform, including:

- Cloud
- Data
- Games
- Operating system version
- Navigation
- Notifications
- Physics

You can download a compressed folder of these samples that contain the solution files, or you can browse the code for each sample on GitHub.

Infragistics GitHub repositories

Infragistics also maintains GitHub repositories for Ignite UI, the ReportPlus component, and other components. The Infragistics repositories are available from *https://github.com/Infragistics*. The IgniteUI repositories are available from *https://github.com/IgniteUI*.

MONO

Mono is the underlying foundation that enables cross-platform development using Xamarin. Mono uses an open source version of .NET and is licensed under the MIT license. The Xamarin SDKs for iOS and Android and the source code for these projects have been contributed to the .NET Foundation.

DOCKER

Docker enables the use of containers and images to run applications in a continuous integration/continuous deployment model. Docker is open source, and is available for the Windows Server 2016, Linux, and Mac OS X platforms. You can configure applications to run on specific versions of an operating system that is used as the container. For example, you can combine Mono and Docker to create containerized versions of applications that run without any additional operating system configuration.

#3 TRAP

Deploying and testing with emulators

CHANCES ARE THAT YOU don't have dozens of mobile devices running all the versions of the mobile operating systems you want to deploy and test your mobile app on. You can avoid the expense and trouble of maintaining a full array of test devices by using virtual emulators such as the one shown in Figure 4. However, this does introduce another layer of complexity.

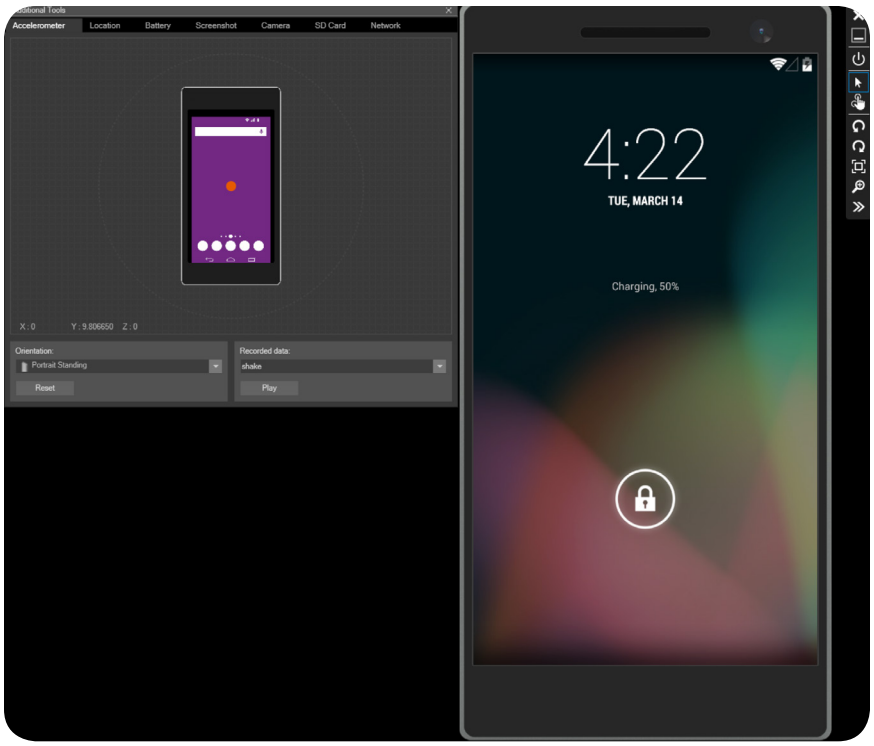

Figure 4: *An Android emulator on Hyper-V.*

Visual Studio uses the Client Hyper-V functionality that was introduced with Windows 8. This is the same enterprise virtualization technology capability that is available in the Windows Server OS, but at the client level. Instead of needing dozens of devices that run different OS versions, you can use virtual machines (VMs) that emulate each

version of the software. Visual Studio will communicate with the VMs and automatically deploy your app on the VM when testing. The Android emulators through Visual Studio are typically must faster than the emulators provided by Google.

If you are running Visual Studio in a VM already, then you must configure additional virtualization extensions to run Hyper-V nested within a VM. Nested virtualization is only supported on Windows Server 2016 and Windows 10 Anniversary Update. For more information, visit *https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/user-guide/nested-virtualization*.

Four Hyper-V components that you need to be aware of when testing mobile apps on VMs are:

- Dynamic memory
- File storage locations
- Virtual switches
- Code change deployment

Dynamic memory

To effectively test your apps on VMs, your development machine must have the same storage and memory specifications that the mobile device will have. Most mobile devices have only a small amount of RAM and storage, so one mobile OS or app doesn't require significant resources. However, if you plan to test several devices or OS versions, you might need more RAM than is available off the shelf. Figure 5 shows the memory settings of the emulator VM.

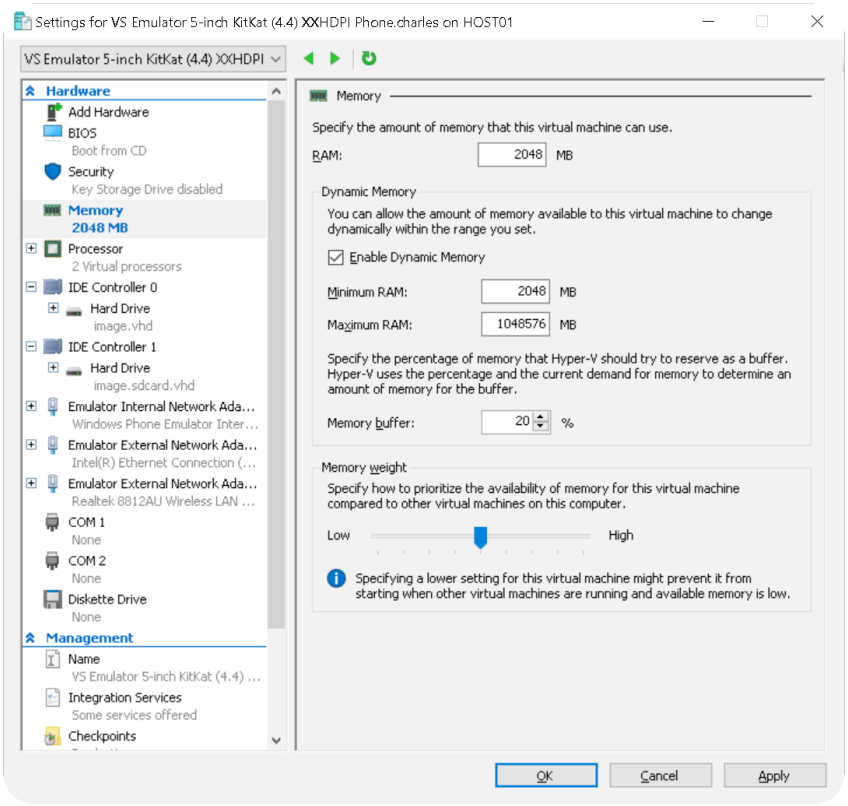

Figure 5: *Memory settings of the Windows Phone OS emulator.*

Hyper-V VMs for all operating systems have dynamic memory that expands when the VM requests it, from a minimum amount to a maximum that you specify. In the example shown in Figure 6, the emulator will start with 512 MB of RAM. If you are testing only the functionality of your app, you can enable dynamic memory to provide the emulator with as much RAM as it requests. However, this can provide you with some false hope as you test your application, because you might not be aware of it when the emulator exceeds the RAM that would typically be found on a mobile device. You should disable dynamic memory, and set a hard limit on the maximum amount of RAM the emulator can use to more accurately test how an actual mobile device will respond.

File storage locations

All Hyper-V VM images and files are stored in the user directory, which typically resides on the local OS drive. Therefore, if you are running Visual Studio on a laptop or other computer that has only one drive, you might see slower storage performance than the actual mobile device would have. Figure 6 shows the storage settings for an Android-based emulator running KitKat.

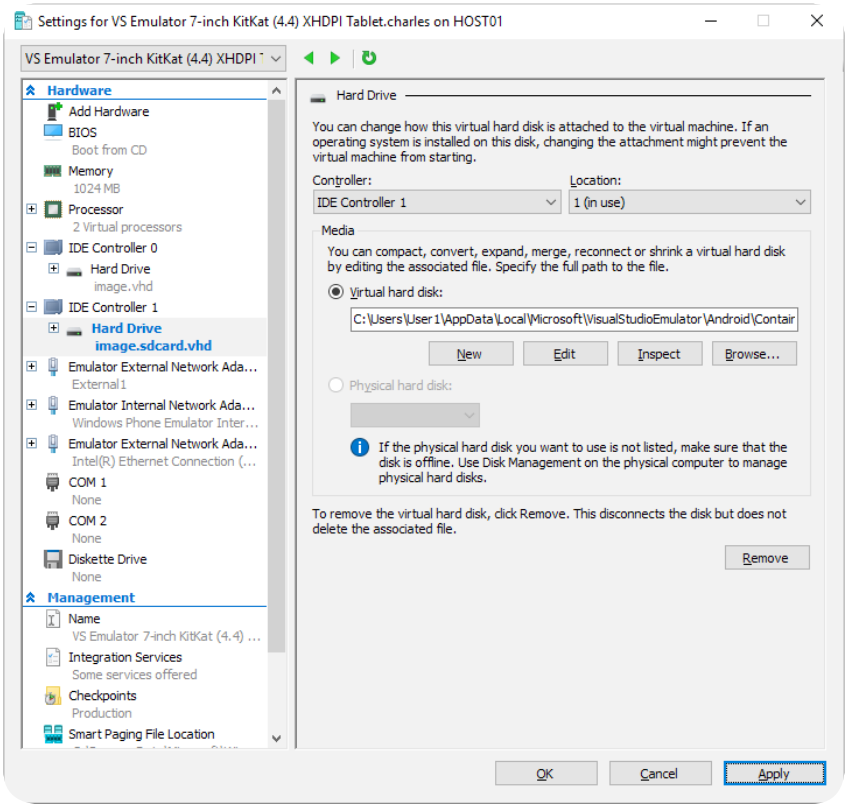

Figure 6: *An Android emulator stored on the OS drive.*

To more accurately emulate the dedicated storage space of a mobile device, run your emulators on a separate data drive.

Virtual network switches

Hyper-V uses virtual switches to bridge the network connection of a VM or emulator with the actual network adapter, whether physical or wireless, of the host computer. Figure 7 shows the Hyper-V Virtual Switch Manager for a host computer that has five switches configured.

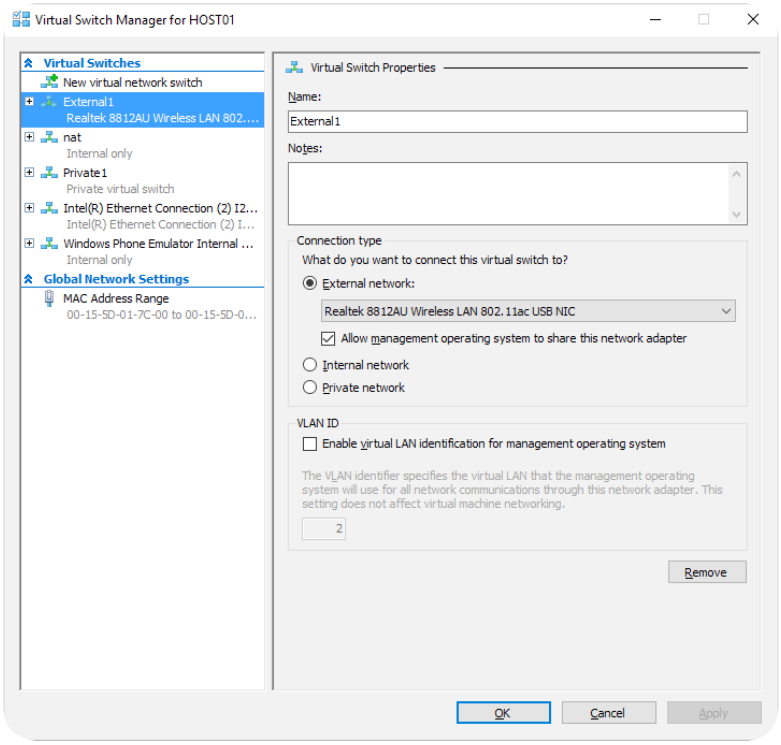

Figure 7: *The Virtual Switch Manager for Hyper-V.*

Hyper-V uses three virtual switch connection types :

- External connections create a bridge from the VM to the adapter on the host.
- Internal connections provide a separate network, similar to a virtual LAN (VLAN), that enables communication between the host and VMs but does not provide Internet access.

Private connections provide a segmented network for internal VM ca communications without access to the host or the Internet.

In most cases, you would want to use an external switch for your emulators or VMs so that they have the same network access as the host computer. If you have multiple network adapters, you can assign a dedicated connection for the VMs from the drop-down list. If you have only one network adapter, the Allow management operating system to share this network adapter check box must remain selected to allow the host OS to share the adapter. Otherwise, Hyper-V will completely control the adapter and the host OS will lose network connectivity.

Code change deployment

When using emulators to test an app, Xamarin and Visual Studio can sometimes try to be overly helpful. Small changes to your app code might not always be deployed to the emulator. If you think that code changes haven't been deployed, delete the app from within the emulator, clean the solution in Visual Studio, and then retry the deployment. Cleaning the solution should include deleting the bin and object folders. This will ensure that the latest build is pushed to the emulator. It is also important to note that emulators can perform very differently from actual hardware devices. Before you release an app, you should test the app on as many physical devices that you realistically can.

You can find a walkthrough of setting up Visual Studio for use with Xamarin at *https://msdn.microsoft.com/en-us/library/dn879698.aspx*, and the steps for verifying that your environment is ready for mobile app development at *https://msdn.microsoft.com/en-us/library/mt488769.aspx*.

#4 TIP

Using NuGet

IF YOU HAVE ANY previous experience with .NET, you are probably familiar with using NuGet. NuGet is the package manager for developing .NET applications. Packages can be integrated into an app by using NuGet. Xamarin offers several packages that can be combined with existing .NET packages to enable cross-platform development. To navigate to the NuGet package manager in Visual Studio, right-click a solution, and then click Manage NuGet Packages for Solution, as shown in Figure 8.

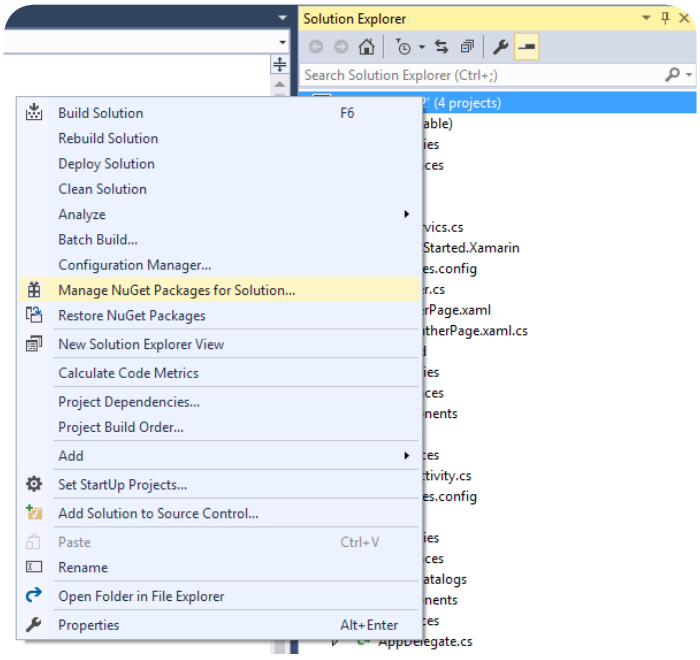

Figure 8: *Managing NuGet packages from within Visual Studio.*

Adding a package to a solution places it in the References tree of the solution. After a package is added to a solution, you can use the APIs while developing your app. Even a simple cross-platform application can have several packages integrated with it. For example, the solution for the WeatherApp tutorial on MSDN includes over a dozen NuGet packages. (The WeatherApp tutorial is available at *https://msdn.microsoft.com/en-us/library/dn879698.aspx*.)

The NuGet package manager, shown in Figure 9, makes it easy to install packages from within Visual Studio and update the packages you have integrated.

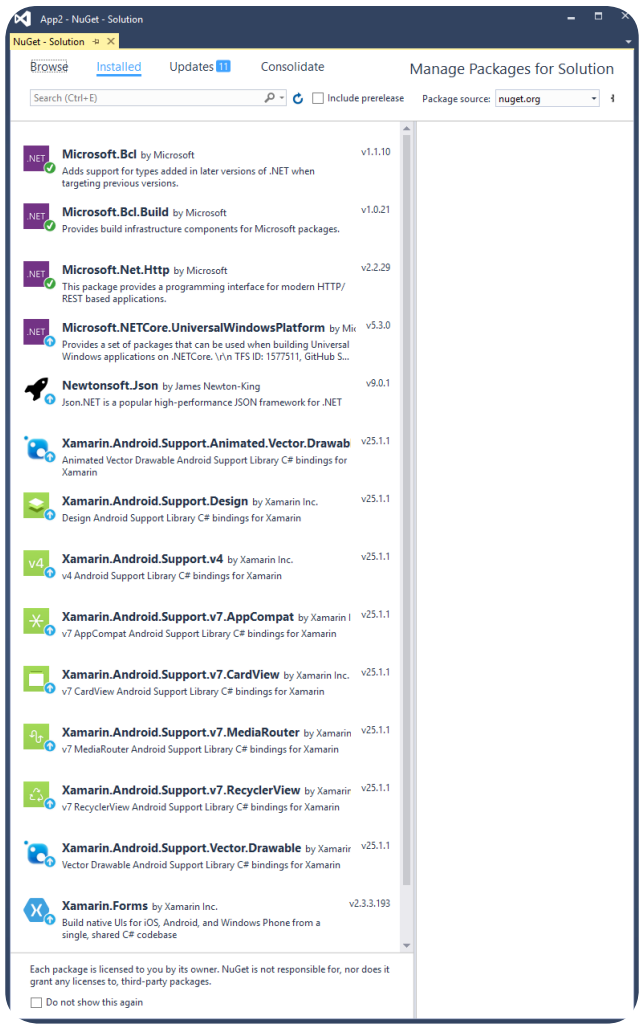

Figure 9: *NuGet packages integrated with a sample app.*

The Updates tab of the package manager lists the packages that have available updates. To obtaining an update, simply select its checkbox and then click **Update**.

By default, the NuGet package manager displays only the packages that are available from NuGet.org. If you plan to create or use a custom or licensed package, you can add another package source to the package manager from the Options dialog box shown in Figure 10. To open the Options dialog box, click the gear icon within the package manager. In the Options dialog box, click the **Add** button (the plus sign) to define a new source, and then configure the name and URL or disk location of the source. After adding the source, you can add packages from the source by selecting it from the drop-down menu in the package manager.

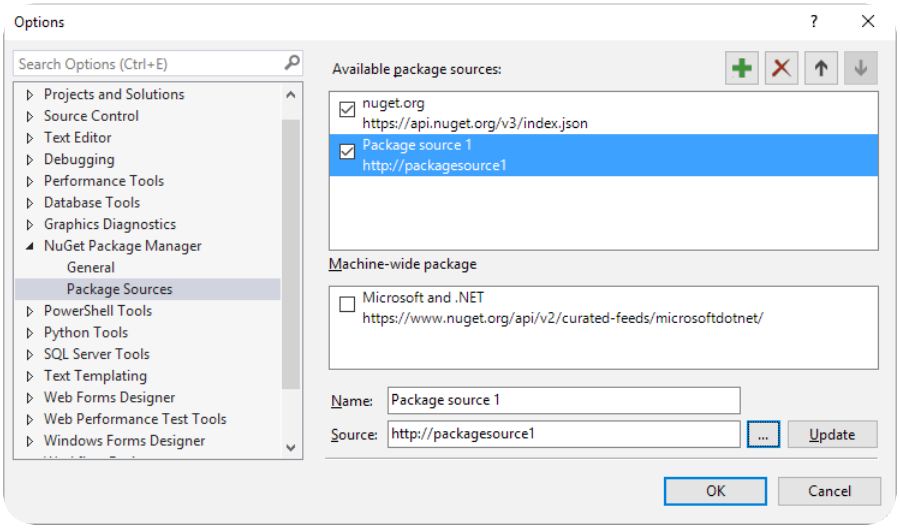

Figure 10: *Adding a package source.*

A common scenario is to use NuGet packages without committing the packages that you are using into source control. You can configure Visual Studio to automatically restore packages from the package source server so that the package binaries are not included in source control. You configure this setting on the General tab of the Options dialog box, as shown in Figure 11.

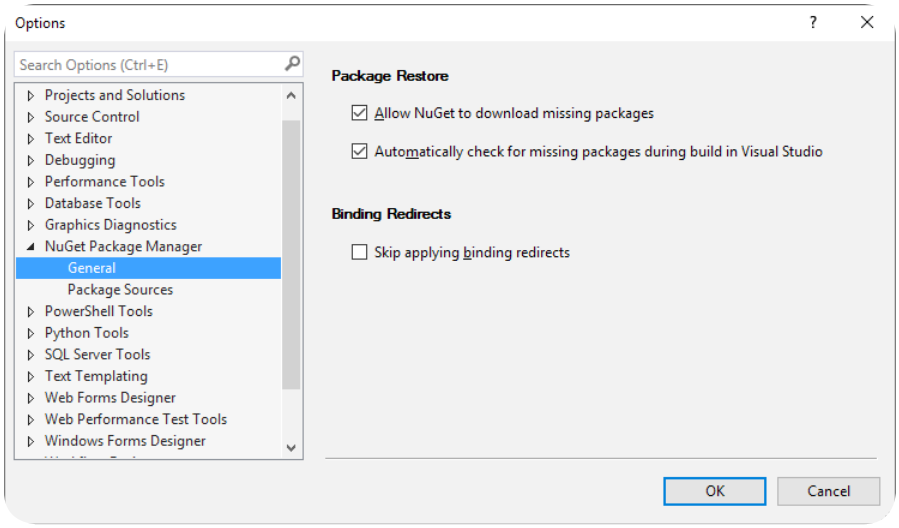

Figure 11: *Configuring general package options.*

For more-advanced solutions, you can create a custom .nuspec file that defines the framework names. The .nuspec file content would be similar to the following code segment:

```
<files>
    <file src="Mac\bin\Release\*.dll" target="lib\Xamarin.
Mac20" />
    <file src="iOS\bin\Release\*.dll" target="lib\Xamarin.
iOS10" />
   <file src="Android\bin\Release\*.dll" target="lib\Mono-
Android10" />
    <file src="iOSClassic\bin\Release\*.dll" target="lib\
MonoTouch10" />
</files>
```

You can specify package versions in the .nuspec file. This would be necessary if the solution targets a certain minimum version that must be used. For example, in the target fields of preceding code segment, Xamarin.Mac20 specifies version 2.0 and Xamarin.iOS10 specifies version 1.0.

Implementing grid layouts with Xamarin

RECENT MOBILE CONTENT USES square layouts to modernize the user interface. Xamarin has a built-in layout named Grid that arranges content into rows and columns. The Grid layout is useful if you want to arrange buttons or content into rows and columns. Some examples include the numbers in a calculator app, the iOS and Android home screens, or toolbars that have data in equally-sized squares.

The Grid layout is unlike a table in that the content does not determine the number of cells; instead, you specify the number of rows and columns.

The Grid layout includes the following components:

- Rows and columns
- Data placement
- Cell spacing
- Cell spanning

Rows and columns

The data that defines the number of rows and columns of a Grid layout is stored in the *RowDefinitions* and *ColumnDefinitions* collections. Each collection has only one property to configure: rows are defined by the **Height** property, and columns are defined by the **Width** property. The **Height** and **Width** values can be defined in one of three ways:

- Auto automatically fits the column or row to its content.
- Proportional (defined by an asterisk) sizes the column or row to fill the available space.
- Absolute defines a specific value for the column height or row width.

For example, imagine that you need to define two rows and two columns of data as follows:

- The first row should be 100 pixels high.
- The second row should fill the remaining vertical space.
- The first column should adjust automatically to the width of its content.
- The second column should fill the remaining horizontal space.

The XAML code for the layout would be as follows:

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition Height="100" />
    <RowDefinition Height="*" />
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="Auto" />
    <ColumnDefinition Width="*" />
  </Grid.ColumnDefinitions>
</Grid>
```

By default, Microsoft platforms automatically use proportional widths when using XAML. However, with Xamarin.Forms, the default column width is set to Auto if it is not defined in XAML.

Data placement

After you define the layout of a Grid, you can add children and specify the cell that you want to display the content in. The specific position of the data is defined in the **Grid.Row** and **Grid.Column** attached properties for an object. When defining the values for the properties, each row and column starts at zero, so the top left cell of the Grid layout has the position 0,0. Building on the previous example, the following XAML code segment adds label objects at each Grid location:

```
<Label Text="Top Left" Grid.Row="0" Grid.Column="0" />
<Label Text="Top Right" Grid.Row="0" Grid.Column="1" />
<Label Text="Bottom Left" Grid.Row="1" Grid.Column="0" />
<Label Text="Bottom Right" Grid.Row="1" Grid.Column="1" />
```

Cell spacing

You define the cell spacing of a Grid layout by defining the spacing between columns and between rows in the initial **Grid** object, as shown in the following XAML code segment:

```
<Grid ColumnSpacing="10" RowSpacing="10">
  <Grid.RowDefinitions>
    <RowDefinition Height="*" />
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="*" />
  </Grid.ColumnDefinitions>
</Grid>
```

Cell spanning

It's common to have labels or buttons that span multiple columns or rows in a grid. For example, the zero button of a calculator or numeric keypad typically spans two columns at the bottom of the interface. The **ColumnSpan** property of the Grid layout specifies the number of columns the object will span, and the **RowSpan** property defines the number of rows the object will span. When using XAML, you define the **ColumnSpan** or **RowSpan** property with the object, as shown in the following code segment:

```
  <Label Text="Bottom Right" Grid.Row="1" Grid.Column="0"
Grid.ColumnSpan="2"/>
```

#6 TIP

Optimizing data rendering across platforms

THERE ARE A FEW different ways of rendering and using data across multiple platforms. Some of these options include:

- AppCompat for Android
- TableView
- WebView
- ListView

AppCompat for Android

AppCompat for Android provides new themes that can be used with Material Design on the Android 6.0 (Marshmallow) framework. Material Design requires the creation of several XML files that define the color and theme to use on the device. These files include:

- Resources/values/colors.xml
- Resources/values/style.xml
- Resources/values-v21/style.xml (For Lollipop and newer frameworks)
- Properties/AndroidManifest.xml
- Resources/layout/tabs.axml
- Resources/layout/toolbar.axml

Google provides a color palette generator at *http://www.materialpalette.com* for color schemes that can be used with Material Design.

In existing Xamarin.Forms apps, the MainActivity.cs class inherits the data from *FormsApplicationActivity*, which must be replaced with *FormsAppCompatActivity* to enable the latest functionality.

TableView

TableView is a common data rendering method when a list of information is provided. Samples include the list of settings that are found for a device, collecting data that is being input into a form, or displaying data row by row. The TableView component has an equivalent view on each platform's API, so that data can be rendered natively.

TableView has two default properties:

- **SwitchCell** is a Boolean switch for on/off or true/false settings.
- **EntryCell** is a field for capturing user input.

The following XAML code segment provides an example of using a **SwitchCell** toggle to enable notifications when a receiving a new message:

```
<TableView Intent="Settings">
    <TableRoot>
        <TableSection Title="Notification">
            <SwitchCell Text="New Message" />
      <SwitchCell Text="New Message" On="true" />
        </TableSection>
    </TableRoot>
</TableView>
```

The **EntryCell** property is used when capturing user data, and can also define options such as the type of keyboard to use (whether the full alphabetical, or the number keys for a phone number) and the text label, color, and alignment.

WebView

As the name suggests, WebView displays web content including CSS, HTML, documents, and local files by downloading it and then rendering it within the application. (This differs from OpenUri, which opens the native web browser on the device.) Note that with documents, the native components are used to render the data. So for iOS and Android, you could display a PDF file because it can be rendered natively. However, Windows Phone does not natively render PDF files, so they can't be opened through WebView.

When defining a WebView, the URL string must also include the appropriate web protocol to use—either HTTP or HTTPS. When using local HTML files, you must define the CSS for each platform that you intend to support. You must also configure the **BaseUrl** property if you plan to link from one local HTML file to another. Otherwise, WebView will not know where the file is located.

ListView

The most commonly used view layout is ListView, which also has the most complexity and the most customization options. ListView is available in Xamarin.Forms. It communicates with the native rendering function through the platform's API. This is accomplished by following these steps:

Create a Xamarin.Forms custom control.

Consume the custom control from Xamarin.Forms.

Create the custom render control for each platform.

The actual steps and code vary depending on the platforms that you are rendering for. A walkthrough that includes code samples of customizing the rendering for each platform can be found in the Xamarin Developer Center at *https://developer.xamarin.com/guides/xamarin-forms/application-fundamentals/custom-renderer/listview*.

When a list contains a large amount of data, it can suffer from slow scrolling performance or a lag in configuring or entering data. You can improve the performance of ListView across platforms by using a caching strategy.

The **RetainElement** and **RecycleElement** properties can be used with ListView to define initialization and cell recycling when scrolling. For a walkthrough of configuring these properties, including code samples, visit *https://developer.xamarin.com/guides/xamarin-forms/user-interface/listview/performance*.

Android optimizations

With Android as the mobile OS, you can create an Adapter that uses the GetView method. To optimize the view, you can use the **convertView** parameter for the method. The convertView parameter attempts to reuse an old view. You should verify that the view is non-null and an appropriate type. If the view cannot be converted, then a new view will be created. This inherently increases the performance of the view.

A common performance hit happens because of using FindViewById with GetView. You can create a different design pattern by creating a new class to store the controls. If you inherit the property from Java.Lang.Object, each view will contain a Tag property that can store data. However, the Tag property belongs to Java.Lang.Object so it must be inherited from that. By calling the ViewHolder that is populated with data will result in better performance than using the native FindViewById property.

Boosting cross-platform performance in Xamarin

MOBILE APP PERFORMANCE CAN be measured in a few different ways, whether it's the perceived performance by the user, the interface response time, or the battery management. There are a few Xamarin techniques that you can use to boost the overall performance of your mobile app, including the following:

- Use the Xamarin Profiler
- Release disposable resources
- Unsubscribe from events
- Use weak references
- Delay object initialization
- Use asynchronous APIs
- Manage garbage collection
- Minimize application size
- Compile XAML

Xamarin University has a free video that discusses how to avoid common pitfalls with Xamarin Apps. The video, slides, and resources are available at *https://university.xamarin.com/guestlectures/avoiding-common-pitfalls-in-xamarin-apps*.

Use the Xamarin Profiler

The Xamarin Profiler is a GUI for the Mono log provider. It identifies code that can be optimized to boost performance, by tracking usage statistics such as memory usage and method run time while the application is running. The Xamarin Profiler is available as a separate download for both macOS and Windows. A Visual Studio Enterprise license is required to enable profiling.

You can enable profiling in the Debugging section of a build. When building an iOS application, the check box is labeled **Profiling**. When building an Android application, the check box is labeled **Developer instrumentation**.

When debugging an application, the profiler collects data about how objects are being created, how garbage collection works, and the time spent processing methods within the code. After you finished debugging the application, the profiler displays the data and charts any identified methods or segments of an application that can be optimized. Running the profiler against an emulator doesn't always yield the best results, especially if the XCode isn't up to date. It is recommended that you use the profiler on physical devices instead of emulators.

Release disposable resources

When a resource is no longer required, it is important to properly release it. This is managed by the *IDisposable* interface, and can be performed by wrapping the object in a **using** statement or in a **try/finally** block.

The *IDisposable* interface should be used only when a class owns an unmanaged resource. Unmanaged resources are typically files, network connections, and streams. You can also use the *IDisposable* interface when a class owns managed *IDisposable* resources. However, Xamarin.Forms does not consistently call *IDisposable*.

One of the reasons that it is import to dispose of these resources is because they are inherited from NSObject or Java.Lang.Object. This proactively tells the system that the it can release the in-memory objects rather than waiting until the object is collected..

The API documentation for Xamarin includes examples of releasing unused resources, and can be found at *https://developer.xamarin.com/api/type/System.IDisposable*.

Unsubscribe from events

Before a subscription object is disposed, events should be unsubscribed to prevent memory leaks in the application. If an object is still referenced, the garbage collection service will not dispose of the object from memory. The following code segment provides an example of unsubscribing from an event:

```
public class Publisher
{
  public event EventHandler MyEvent;
  public void OnMyEventFires ()
  {
    if (MyEvent != null) {
      MyEvent (this, EventArgs.Empty);
    }
  }
}
```

```
public class Subscriber : IDisposable
{
  readonly Publisher;
  public Subscriber (Publisher publish)
  {
    publisher = publish;
    publisher.MyEvent += OnMyEventFires;
  }
  void OnMyEventFires (object sender, EventArgs e)
  {
    Debug.WriteLine ("The subscriber has been notified of
an event");
  }
  public void Dispose ()
  {
    publisher.MyEvent -= OnMyEventFires;
  }
}
```

Use weak references

When the garbage collection service runs, it examines the objects to determine what can be removed from memory. If an object maintains a reference in the application, then the garbage collection cannot dispose of the object. This object relationship is referred to as a *strong reference*. Strong references are common in a parent-child relationship.

A *weak reference* can be useful for objects that use a large amount of memory, but can be easily re-created if they are removed by garbage collection. A weak reference defines the relationship between two objects. A strong reference has the same relationship in both directions. A weak reference changes the relationship from the child object to the parent.

Weak references are important for Xamarin because of how the iOS memory management system operates. The more native references cycles that exist for iOS apps, the worse that they will perform. Reference cycles are perfectly acceptable within the .NET code, but any objects inherited from NSObject will create memory leaks. Using weak references enable you to minimize reference cycles.

You can learn more about weak references at *https://msdn.microsoft.com/en-us/library/ms404247(v=vs.110).aspx*.

Delay object initialization

A common occurrence of wasting processing power and battery life in mobile apps is unnecessarily initializing objects. However, if you delay the initialization of an object until it is actually needed, you can increase the overall performance of the app. If an application might not use the defined object, or if other methods or processes must complete before an object is needed, then wait to define the object until later in the logic. This is referred to as lazy initialization, and is defined by the lazy constructor. You can learn more about lazy initialization at *https://msdn.microsoft.com/en-us/library/dd997286(v=vs.110).aspx*.

Use asynchronous APIs

Many APIs use synchronous processing, which can block threads from processing. Asynchronous APIs never block thread execution for a significant amount of time. When calling an API, ensure that you use asynchronous when available, especially for user interface threads. Blocking UI threads will make the interface appear unresponsive for the user.

Additionally, any operation that runs for a longer period should be processed as a background thread. This limits the possibility of a UI thread being blocked because of background processing.

Manage garbage collection

Xamarin uses two garbage collectors: SGen and Boehm. SGen is the default generational garbage collector in Xamarin. Boehm is a non-generational garbage collector, and is the default collector only for Xamarin.iOS applications that use the Classic API.

SGen uses three heaps to manage space allocation:

- **Nursery.** Small objects are created until the nursery runs out of space. Garbage collection moves live objects to the major heap.
- **Major heap.** Long-running objects are stored in the major heap until there is not enough memory. If garbage collection does not free up enough space at this level, the system will be asked for more memory.
- **Large Object Space.** Objects that require more than 8000 bytes are stored here. If the object is initialized at that size, it will not begin in the nursery.

SGen pauses all threads in an application while performing garbage collection. Therefore, it is important to avoid garbage collection in tight loops. You should also release resources and unregister events as mentioned in previous sections.

Minimize application size

The compilation process differs depending on the platform. For example, iOS uses ahead-of-time compilation to ARM assembly language. Android and Windows Phone applications are compiled to Intermediate Language (IL). The .NET framework is included with both compilations, and by default includes classes that are not being used.

Xamarin includes a process to link classes to builds, but that process is disabled by default. When the solution is being built, the types and methods will be analyzed to determine if they are used for an application. If they are not, then they are removed from the application. The option to enable the linker is in the build settings for a project, as shown in Figure 12.

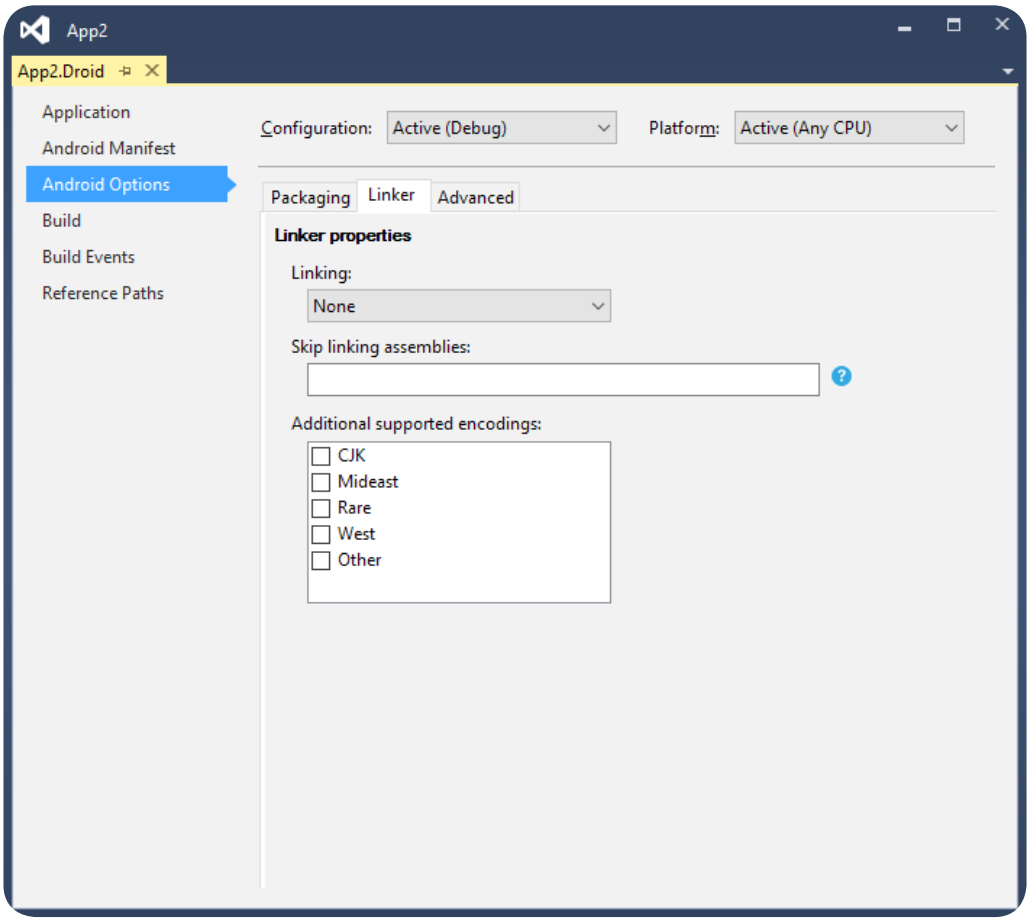

Figure 12: *The Linker settings for an Android project.*

Compile XAML

XAML can also be compiled to .NET, or directly to IL with the XAML compiler. This boosts the performance of an application by reducing load and instantiation time for XAML objects. It also reduces the file size by eliminating .xaml files from the build. The XAML compiler is disabled by default to ensure backward compatibility, but can be enabled by defining the **XamlCompilation** attribute. You can detect errors in XAML when compling that might not be otherwise caught with this attribute. The following code segment provides an example of enabling the XAML compiler:

```
using Xamarin.Forms.Xaml;
...
[assembly: XamlCompilation (XamlCompilationOptions.Com-
pile)]
namespace SampleApp
{
...
}
```

#8 TRAP

Using control and data templates

XAMARIN.FORMS OFFERS TWO TYPES of template for rendering data in a theme: control templates and data templates. Control templates define the appearance of control, including its background and text colors. Data templates define how data is presented when using a control.

Control templates

Control templates separate the theme of a page (its colors and overall appearance) from the page content. You can specify an overarching control template that defines the theme without impacting the actual content that will be displayed on the page.

You can create a control template by using XAML if you replace the default App class with a XAML App class. You create the control template as an object in *ResourceDictionary*, and must call the *MainPage* and *InitializeComponent* methods to load and parse the XAML. For example:

```
public partial class App : Application
{
   public App ()
   {
      InitializeComponent ();
      MainPage = new HomePage ();
   }
}
```

The **ContentView.ControlTemplate** property uses the StaticResource markup extension to assign templates. The **ContentView.Content** property defines how the content is displayed in the layout on the page. The following example uses the TealTemplate template with the Control Template:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="SimpleTheme.HomePage">
    <ContentView x:Name="contentView" Padding="0,20,0,0"
                 ControlTemplate="{StaticResource Teal-
Template}">
        <StackLayout VerticalOptions="CenterAndExpand">
             <Label Text="This demonstrates a template!"
HorizontalOptions="Center" />
            <Button Text="Change the theme!" Clicked="On-
ButtonClicked" />
        </StackLayout>
    </ContentView>
</ContentPage>
```

The preceding code segment defines the display of a simple button in the TealTemplate template. If the corresponding button contains the following code, it will cycle through the available templates:

```
void OnButtonClicked (object sender, EventArgs e)
{
  originalTemplate = !originalTemplate;
  contentView.ControlTemplate = (originalTemplate) ? teal-
Template : aquaTemplate;
}
```

After clicking the button, the template colors would change from teal to aqua, without changing the page content. You can set and define a Control Template by using a style, or for an individual page.

Data templates

Data templates define data for supported controls. For example, imagine a ListView control that displays a collection of names, birthdays, and postal codes.

```
public class Demographics
{
  public string Name { get; private set; }
  public int DoB { get; private set; }
  public string postCode { get; private set; }
  public Demographics (string name, int dob, string post-
code)
```

```
    {
      Name = name;
      Birthdate = dob;
      PostCode = postcode;
    }
}
```

You can then assign the actual data from the **listView.ItemsSource** property by using the following code:

```
public partial class WithoutDataTemplatesPage : ContentPage
{
  public WithoutDataTemplatesPage ()
  {
    InitializeComponent ();
    var people = new List<Person> {
      new Person (“Jane”, 28/02/1970, “20552”)
    };
    listView.ItemsSource = people;
  }
}
```

You can modify the appearance of the data by using a Data Template. If you have a collection of objects from a ListView control, you can define the property within a Data Template. The template will bind the property values to the corresponding elements, which will then be rendered appropriately.

```
<ListView x:Name=”listView”>
  <ListView.ItemTemplate>
    <DataTemplate>
      <ViewCell>
        <Grid>
          ...
        <Label Text=”{Binding Name}” FontAttributes=”Bold”
/>
          <Label Grid.Column=”1” Text=”{Binding DoB}” />
         <Label Grid.Column=”2” Text=”{Binding postCode}”
... />
        </Grid>
      </ViewCell>
    </DataTemplate>
  </ListView.ItemTemplate>
</ListView>
```

#9 TIP

Using .NET Core

THROUGHOUT THE PREVIOUS TIPS, we've discussed a lot about the APIs that are available for each platform, as well as the packages that you can integrate with NuGet. It is important to understand when you should use .NET Core, a native API, or a sourced package.

Use .NET when possible

Most issues that you will typically run into during development with Xamarin fall somewhere between the .NET logic and using a native API. The more .NET that you can use as part of the app, the better it will perform, and the easier it will be to troubleshoot. Using .NET also maximizes the amount of code that you can share, and minimizes the amount of platform-specific development that is needed for each platform.

For example, you shouldn't develop logic that manipulates the NSDate class for iOS just because you can. Instead, you should use the DateTime class in the .NET Framework, and then convert it to NSDate only when communicating with that specific API. Any time a .NET API exists, you should use it over a native API. The exception to this is when you are trying to highlight or take advantage of a certain aspect of the native API that the .NET framework might not be able to.

Understand XAML performance on different platforms

It's also a common misconception that Xamarin.Forms is a silver bullet for developing cross-platform apps. Of course, this isn't exactly the case. Using Xamarin.Forms helps solve some common problems of developing for individual platforms, but by the same token it also introduces its own complexities.

An example of this is using XAML for Windows Presentation Foundation (WPF) and Universal Windows Platform (UWP). The XAML that you develop for Android and iOS might not behave the same on WPF or UWP. You can avoid a lot of headaches by learning how XAML works differently on Windows devices compared to other platforms.

Share logic

Two common options for sharing logic are Shared Projects and Portable Class Libraries (PCLs).

- Using a Shared Project is a simple method of sharing and referencing code files. This enables you to reference platform specific logic in classes without using dependency interjection. However, this can also have the downside of leading to poor logic decisions.
- PCLs enable sharing by building a library that is valid for a certain set of .NET platforms. There are individual APIs that you can use depending on the platform that you are developing for. PCLs are very portable and can be referenced from any platform that is a member of the PCL profile. This ensure that the logic is performed accurately, as it is defined as a dependency rather than woven in. However, the more platform-specific dependencies that are required by an app, the more complex and larger the app becomes.

.NET Core assists in the development of cross-platform apps by providing an open source and open module .NET platform. Using .NET Core gives you the ability to deploy applications while supporting multiple versions of the framework..NET Core is composed with NuGet packages, as discussed in a previous section. The packages can either be compiled directly into the application, or included as files within the app.

.NET Core also includes a version of ASP.NET, named ASP.NET Core. This too is a smaller, modular version of the framework for web developers. High performance web apps can be deployed using ASP.NET Core to cloud or on-premises solutions. ASP.NET Core can be run on both.NET Core and the full .NET Framework. When using .NET Core, ASP.NET Core applications can be used across multiple platforms.

Releasing an Android app

THERE ARE MULTIPLE STEPS to preparing and publishing an application. When publishing an Android application, these steps can include:

1. Prepare the app for release.
2. Sign the app package.
3. Publish the app to a store
   - Google Play store
     a. Use Google Licensing services.
     b. Upload APK expansion files.
   - Publish the app to the Amazon App Store.
   - Publish the app independently.

Prepare the app for release

After you have developed and sufficiently tested an app and are ready to release it, you should configure some attributes and settings, including the following:

- **Application icon.** Define an icon for the app (required by the Google Play store).
- **Application version.** Initialize or update the app version for both internal tracking and user notification.
- **Shrink the APK.** Use the profiler and link the Xamarin.Android packages to reduce the size of the solution.
- **Protect the application.** Prevent reverse engineering by disabling debugging, adding anti-debug or anti-tamper code, and using native compilation.
- **Compile.** Compile the code and verify that it builds in release mode.

Sign the app package

To sign an APK, you first need to create a new security certificate. You can create a certificate from Visual Studio from the app details when distributing the package. When you click the green plus to add a signing key, you will be prompted to create the Android key store with the required information. By default, the keystore will be saved in the AppData folder of the local user account.

To assign the certificate with a package, select the signing identity while publishing the application. You will be prompted to select the directory where to save the APK file. The certificate will be saved with the package, and will be used to sign the package.

Publish the application to the Google Play store

Android offers several app markets that apps can be pushed to. The most popular, and available on all Android devices, is the Google Play store. To publish apps to the Google Play store, you must first obtain a developer account with Google. This requires a one-time fee, which at the time of this writing is $25 USD. All applications that are submitted to the Google Play store must have a signing key that expires after October 22, 2033.

The maximum size for an APK in the Google Play store is 100 MB. If an application exceeds 100 MB, additional app content can be delivered by using APK Expansion Files. Each Android app can have up to two expansion files, with each file being up to 2 GB. The Google Play store hosts and distributes the files at no cost, regardless of size.

After creating a Google developer account, create and register for the Google API access. You can use the API access to publish the application to the Google Play store.

Upload APK expansion files

As mentioned in the previous section, APK expansion files are necessary for files that are larger than 100 MB. Each expansion file can be up to 2 GB in size, for a total of just over 4 GB. On most Android devices, the expansion files are saved to removable storage, and not to internal storage. There are two valid expansion types:

- **Main expansion.** This is for primary files that will not simply fit within the APK size limit.
- **Patch expansion.** This is designed for updates or patches to an application.

Using Google Licensing services

Google Licensing is a network-based service that enables you to determine whether an application is licensed to run on a specific device. You can configure the Android APK to check whether a license exists to use the app on a specific device, as well as how often to check and how to handle various responses from the device.

Google Licensing uses an RSA key pair that is shared between the application and the Google Play store. Developers for the Google Play store receive a public key that is embedded within the Android application and used to authenticate the responses.

Google Licensing is a key component of APK expansion files, as the services obtain the URLs that the additional data should be downloaded from. If an app is published or obtained through a source other than the Google Play store, the Google Licensing services are not used.

Publish the app to the Amazon Appstore

Amazon offers a competing app marketplace to the Google Play store, and has a separate distribution program to go along with it. The Amazon Appstore does not have a specific APK file size limit like the Google Play store does. However, APKs that are larger than 30 MB use only FTP as a transfer method.

When you submit an app to the Amazon Appstore, you must define the following package assets:

- **App icon.** A 114 px × 114 px .png file.
- **App thumbnail.** A 512 px × 512 px .png file, typically a larger version of the icon.
- **Screenshots.** From three to ten screenshots that depict the app user interface.
- **Promotional image.** If the app can be used in a promotion, then a 1024 px × 500 px image must be provided for promotional purposes.
- **Videos.** Up to five videos can be included with the app that previews the app in the store.

After you submit an app, Amazon must approve the app before it is posted in the Amazon Appstore. After approval, Amazon will send you a notice and make the APK available for download.

Publish the app independently

You can publish an application independently of a specific app marketplace. By default, Android devices prevent users from installing applications from unknown third-party sources. To install an independent app, the device must be configured to allow the installation of apps from unknown sources. This setting is located in the Security section of the device settings.

There are a few methods of distribution that can be used:

- **Email.** You can email the APK. When the APK is opened on the device, an Install button will be displayed.
- **Web.** When a supported device downloads the APK from a web browser, it will be automatically installed after the download is complete.
- **Manual installation.** If you have a device attached to a computer, you can install the APK directly to the device.

#11 BONUS TIP

Using the Prism framework

PRISM PROVIDES A METHOD of building XAML applications for the WPF, UWP, and Xamarin.Forms. Each platform has an separate release that is built individually on its own timeline. Prism enalbes you to implement a collection of design patterns when developing XAML applications. These include MVVM, dependency injection, commands, EventAggregator, and more by using a shared code base in a PCL for each platform. Prism 6 is open source and can be downloaded from the GitHub repository at *https://github.com/PrismLibrary/Prism*.

Key Concepts

Prism enhances capabilities and design patterns for composite application development. Some of the key concepts that are defined in Prism include:

- **Modules.** Modules are packages that are used for specific functionality. Modules can also wrap application infrastructure or servers to be reused in multiple applications.
- **Module catalog.** For composite applications, the modules are discovered and loaded at run time. Prism uses a catalog to specify the modules that are to be loaded, and which order they must be loaded in.
- **Shell.** The modules are loaded into the shell. This defines the layout and structure of the application, but is not aware of the individual modules that are being used.
- **Views.** Views are UI controls for a feature of an application. You can use a MVVM pattern in conjunction with a view
- **View models.** View models are the classes that wrap the presentation logic and state. These define the properties, events, and commands to control the view.
- **Models.** As part of the MVVM pattern, modules wrap any data for validation and rules for consistency and integrity.
- **Commands.** Commands can be objects or methods in a view model. Prism uses the **DelegateCommand** and **CompositeCommand** classes.

- **Regions.** Regions updates an application's UI without requiring any changes to the underlying logic. The **ContentControl**, **ItemsControl**, **ListBox**, and **TabControl** controls can be used as a region.
- **Navigation.** Prism uses two methods of navigation: state-based and view-switching. State-based navigation uses an existing view that is updated. View-switching navigation creates new views and replaces existing views.
- **EventAggregator.** Prism uses the EventAggregator components to enable components to publish events and other components to subscribe to events.
- **Dependency injection container.** Prism uses dependency injection to manage dependences between components. This enables dependencies to be added at run time with Unity, MEF, or other dependency injection container.
- **Services.** Services define non-UI related aspects of an application. This can include logging, data access, and exception management. Services can be registered with a dependency injection container and used by other components.
- **Controllers.** Controllers use presentation logic to display information, including the Prism view-switching navigation.
- **Bootstrapper.** Bootstrapper is another component that can be used by an application to initialize Prism component's and services.

A Developers Guide for the PWPF Prism Library is available on MSDN for version 5. It offers a variety of information on using Prism, and can be found at *https://msdn.microsoft.com/en-us/library/gg406140.aspx*.

Summary

THAT WRAPS UP 10 of the most common tips, tricks, and traps that you will typically encounter when developing cross-platform apps using Xamarin. Xamarin (and Xamarin.Forms) is not a one-size-fits-all tool that magically makes mobile app development a seamless and painless process. Xamarin addresses some issues that have proven difficult in the past for developing cross-platform apps, but also introduces its own complexities and requirements. If you become aware of these tips, tricks, and traps before starting your first development project, you'll be ahead of the curve when you encounter your first issue.

Infragistics aims to make Xamarin and Xamarin.Forms easier to use and manage by introducing the Moo2U component. This component addresses some of the pitfalls discussed in this eBook by providing additional management options and GUI tools for use during development. Visit *http://www.infragistics.com* for the latest updates for Xamarin components.

Charles Pluta is a technical consultant and Microsoft Certified Trainer (MCT) who has authored several certification exams, lab guides, and learner guides for various technology vendors. As a technical consultant, Charles has assisted small, medium, and large organizations deploy and maintain their IT infrastructure. He is also a speaker, staff member, or trainer at several large industry conferences every year. Charles has a degree in Computer Networking, and holds more than 15 industry certifications. He makes a point to leave the United States to travel to a different country once every year. When not working on training or traveling, he plays pool in Augusta, Georgia.