Chapter No. 2
"Xamarin.Android Architecture"
In this package, you will find:
A Biography of the author of the book
A preview chapter from the book, Chapter NO.2 "Xamarin.Android Architecture"
A synopsis of the book’s content
Information on where to buy this book

About the Author

Mark Reynolds is a software enthusiast who has worked in the industry for nearly 30 years. He began his career with Electronic Data Systems, building and supporting systems for the manufacturing sector. Over the years, he has worked with companies ranging in size from startups to Fortune 500 across a diverse set of industries including manufacturing, entertainment, financial services, government, and telecom. In 1993, Mark started a consulting practice focused on delivering training and mentoring services in the areas of software architecture, design, and implementation. With the rise of mobile computing, Mark has returned to what he loves the most, designing, developing, and delivering software solutions, now focusing in the mobile computing space. He continues his private consulting practice based in Allen, TX, where he also resides with his wife and son.

Mark works as an independent consultant through his own private consulting practice (RSEG) based in Allen, TX, a community located north of Dallas. You can find out more about the services he offers from his website, rseg.net.

For More Information:
I would like to say thank you to my wonderful, God-given wife for all her encouragement and support, to my wonderful, God-given son for his creative inspiration, to all my customers who fund my interest in mobile computing, and to all the supporting staff and reviewers associated with Packt Publishing—they've had a big impact on the content and usability of this book.

For More Information:
Xamarin Mobile Application Development for Android

In the fall of 2013, when Packt Publishing first approached me about writing this book, it was a no brainer; of course I would. Why? Why not? A book about mobile development using Xamarin.Android; I'm in! I have to admit, I didn't start here; it was a journey for me.

I've always been keen on cross-platform development environments. I'm not really fond of learning new ways to do the same thing using a different syntax unless there is significant productivity gains associated with it. Ten years ago, the foregone conclusion for most was that cross-platform development belonged to Java; I was on board with that. Had anyone told me that in 2013, with the rise of mobile computing, I would be doing all my cross-platform mobile development with C#, I would have laughed. So, how did I get here?

It started in 2010. I was struggling with Objective-C, trying to get an iOS prototype up and running. One day, I described the prototype to a colleague, Ed Tighe. Ed suggested that I look into MonoTouch. As I recall, the conversation went something as follows:

Me: MonoTouch? What is that?
Ed: A Mono-based development environment for iOS.
Me: Mono… you mean Mono; as in the open source cross-platform C# thing?
Ed: That's the one.
Me: Is that still around? Who would trust Mono with a mission critical solution? What's the likelihood they will be around in four to five years?

Sometimes Ed says funny things; I chalked this one up to that. I was completely dismissive of the idea. It wasn't that I didn't respect what the Mono project and contributors had achieved; it was more about the overriding belief that at some point Microsoft would decide Mono did not need to exist and would work against its continued progress. I also had serious reservations about whether support and long term commercial viability was there. However, the one thing Mono had in its favour was a mass of C# developers that could adopt their platform with minimal investment.

In early 2012, I was approached by Andy LaBrunda, VP of IT, for a telecom-based on Guam, about developing mobile apps for prepaid customers. I knew they were a .NET shop and were looking for both iOS and Android apps with the possibility of a Windows Phone app in the future. I also knew they had a relatively small set of developers, who would be tasked with supporting the apps, and they already knew C#, .NET, and rich client development using WPF.

For More Information:
With all this in mind, it only made sense to consider MonoTouch and Mono for Android. The GTA staff would not have to learn Objective-C and Java, and we would achieve some level of reuse between the two apps, so we framed up a small proof of concept effort, the goal being to build two apps with only a few screens, hook the apps up to RESTful services, and share some code between the apps. As always, when I get to play with new technology, I am excited so I approached this effort with great optimism. I wasn't disappointed; the Xamarin products delivered on everything we set out to prove. I was sold and have never looked back. We built out the two prepaid apps and moved on to build out two postpaid apps.

Since then, I have spent significant time and energy building out my Xamarin practice, including writing this book. I've also taken what we learned at GTA and I'm now working with a company in the Dallas area in the entertainment industry building customer facing apps. With the recent strengthening of the relationship between Xamarin and Microsoft, I believe that Mono and the Xamarin product line have a bright future.

The idea behind this book was to bring the base set of knowledge required to build Android apps with Xamarin.Android together in a convenient, concise, productive format that could be used by those looking to get started with the product. I have always been a fan of learning experiences structured around building solutions, or examples, incrementally throughout the book so that the approach we settled on for this book. We begin with two chapters of general Android and Xamarin architecture and then step through building a Point of Interest app that demonstrates the basics of building Android apps, including some of the more interesting features such as integration with location services, the map app, and the camera app. Our goal has been to provide you with a productive learning experience; I hope we have achieved that and I thank you for taking the time to read it.

Oh! And one more thing on this topic; Ed, you were right.
What This Book Covers

Chapter 1, The Anatomy of an Android App, provides an overview of the Android platform and what Android apps are composed of.

Chapter 2, Xamarin.Android Architecture, describes the use of Mono, describes how Mono and the Dalvik runtime work together, and the Android platform coexist and allow developers to build Android apps using C#.

Chapter 3, Creating the Points of Interest App, walks the reader through creating a new app and running the app within the Android emulator.

Chapter 4, Creating a Data Storage Mechanism, presents a number of options for storing data on an Android device and steps the reader through creating a JSON-based solution.

Chapter 5, Adding a List View, describes Android's AdapterView architecture and steps the reader through using ListView and creating a custom adapter.

Chapter 6, Adding a Detail View, walks the reader through creating a detail view to view a point of interest, adding navigation from the list view, and adding actions for saving and deleting information.

Chapter 7, Making POIApp Location Aware, presents the various options that developers have to make their apps location aware and walks the reader through adding logic to determine a device's location and the address of a location, and displaying a location within the map app.

Chapter 8, Adding Camera App Integration, presents the various options that developers have to add integration with the device camera and walks the reader through adding integration with camera apps on device.

Chapter 9, Deploying Your App, discusses the various options for distributing Android apps and walks the reader through preparing a Xamarin.Android app for distribution.

For More Information:
Xamarin.Android Architecture

Now that we have an understanding of the Android platform, let's talk about Xamarin. In this chapter, we will look at the architecture of Xamarin.Android and how it facilitates the development of Android apps using C# and .NET. This chapter covers the following topics:

- The benefits and drawbacks of adopting Xamarin.Android
- What is Mono?
- Mono and Dalvik side by side (peer objects)
- Xamarin.Android binding libraries
- IDE choices

Xamarin is a company which provides commercial software development tools that leverage the Mono open source project in order to allow you to develop applications for Android, iOS, and OS X using C# and the .NET framework. The product that is used to develop Android apps is Xamarin.Android.

Why Xamarin.Android?

Before we take a dive into the architecture of Xamarin.Android, let's first discuss the question of why Xamarin.Android is our choice. Like any significant platform decision, one size does not fit all, and there are a number of things that should be considered. The following two lists identify some of the key benefits and drawbacks of using Xamarin.Android.

For More Information:
Benefits of using Xamarin.Android:

- Leverages existing C# and .NET skills
  Developers invest a great deal of time and energy in mastering the many features of the C# language and the effective use of the .NET framework. Yes, Java and all OO languages have many similarities, but there is a real cost associated with going from being proficient in C# and .NET to making the same claim in Java. Individuals and groups that have made a significant investment in C# and .NET and have a need to develop Android apps would be well served to at least consider Xamarin.Android.

- Reuse in cross-platform development
  While Xamarin.Android will not allow you to build a single app that can be deployed to Android, iOS, and WP8, it does give you the capability to reuse large portions of your code base across all of these platforms. In general, the user interface code and the code that deals with the device capabilities tend to be written for each platform, while things such as service client logic, client side validation, data caching, and client side data storage can potentially be reused, saving a significant amount of time.

Drawbacks of using Xamarin.Android:

- Licensing requirement
  Xamarin.Android as well as Xamarin.iOS and Xamarin.Os X are all commercial tools and must be licensed, so there is a tangible cost of entry. Check the Xamarin's website for current pricing.

- Waiting for updates
  There is some lag time between a new release of the Android platform and the corresponding release of Xamarin.Android.

- Performance and memory management
  In some cases, Xamarin.Android allocates both Java and C# objects to achieve some of the "magic" of developing in C#/ .NET on an Android device. This has an impact on both the memory footprint and execution performance. Unfortunately, at this time, I do not have any objective data to quantify this impact. What I can report is that with two apps built and another one underway, my Android customers have not reported any concerns in this area.

For More Information:
• Distribution size

There are a number of runtime libraries that must be distributed with a Xamarin.Android application. We will discuss the actual size and strategies for minimizing the distribution size in the last chapter.

While the list of drawbacks may seem extensive, in most cases, the impact of each can be minimized. If you are a group or individual that places a high value on the benefits, you should seriously consider Xamarin.Android.

**What is Mono?**

Mono is an open source, cross-platform implementation of a C# compiler, and a **Common Language Runtime (CLR)** that is binary compatible with Microsoft .NET. The Mono CLR has been ported to many platforms, including Android, most Linux distributions, BSD, OS X, Windows, Solaris, and even some game consoles such as Wii and Xbox 360. In addition, Mono provides a static compiler that allows apps to be compiled for environments such as iOS and PS3.

**Mono and Dalvik side by side**

As you can recall from *Chapter 1, The Anatomy of an Android App*, Android apps run within the Dalvik VM, and we now know that Mono apps run within the Mono CLR. So how does a Xamarin.Android app run? A simple answer is that it uses both the Mono CLR and the Dalvik VM. The following diagram depicts how the runtimes co-exist:

So, how do the Mono CLR and Dalvik VM work together in a Xamarin.Android app? The magic is accomplished through a concept called **peer objects** and a framework called the **Java Native Interface (JNI)**.

---

For More Information:

The Java Native Interface

Java Native Interface (JNI) is a framework that allows a non-Java code (such as C++ or C#) to call or be called by a Java code running inside a JVM. As you can see from the preceding diagram, JNI is a critical component in the overall Xamarin.Android architecture.

Peer objects

Peer objects are a pair of objects consisting of a managed object residing in the Mono CLR and a Java object residing in the Dalvik VM, which work together to perform the functions of a Xamarin.Android app.

Xamarin.Android is delivered with a set of assemblies called the Android binding libraries. Classes in the Android binding libraries correspond to the Java classes in the Android application framework, and the methods on the binding classes act as wrappers to call corresponding methods on Java classes. Binding classes are referred to as Managed Callable Wrappers (MCW). Anytime you create a C# class that inherits from one of these binding classes, a corresponding Java proxy class is generated at build time. The Java proxy contains a generated override for each overridden method in your C# class and acts as a wrapper to call the corresponding method on the C# class.

The creation of peer objects can be initiated from within the Dalvik VM by the Android application framework or from within the Mono CLR by the code you write in the overridden methods. A reference between the two peer objects is kept by each instance of a MCW and can be accessed through the Android.Runtime.IJavaObject.Handle property.

The following diagram depicts how peer objects collaborate:

For More Information:
Xamarin.Android application packaging

In Chapter 1, The Anatomy of an Android App, we discussed Android packages (.apk files). Xamarin.Android creates the .apk files but also includes the following additional types of files:

- The C# code is stored as assemblies in the assembly folder of the archive
- The Mono runtime is packaged as native libraries

The Android bindings design

Core parts of Xamarin.Android are the bindings for the Android APIs. The Xamarin team focused a great deal in developing a consistent approach to creating the bindings so that a C# .NET developer would feel at home when using them. This has resulted in a number of key benefits as follows:

- The Android API feels natural to a C# .NET developer and allows the developer to explore the API using code completion and pop-up documentation from within the IDE
- C# developers can leverage the vast array of Java/Android examples and documentation that can be easily transformed for use with C# and Xamarin.Android

Design principles

A complete set of design principles can be found on the Xamarin website; we have included only a few for discussion:

- Allowing developers to subclass Java classes from the Android application framework
- Exposing a strongly typed API
- Exposing JavaBean properties as C# properties
- Exposing Java event listeners as C# delegates

C# properties

The JavaBean properties, the getter and setter methods, are transformed to C# properties, when appropriate. The following rules are used to determine when properties should be created:

- Read/write properties are created for the getter and setter method pairs

For More Information:
Read-only properties are created for getters without corresponding setter methods.
No write-only properties are created for the rare case that only a setter exists.
Properties are not created when the type would be an array.

As you may be aware, Java does not have a property construct but instead follows a design pattern defined in the JavaBean specification. In order to define a property, a developer simply creates the public getter and setter methods with read-only properties that only provide a getter method.

Delegates

The Android APIs follow the Java pattern for defining and hooking up event listeners. The C# developers are more familiar with using delegates and events, so the Android bindings attempt to facilitate this using the following rules:

- When a listener callback has a void return, an event is generated based on the EventHandler delegate.
- When a listener callback does not have a void return, a specific delegate is generated that supports the appropriate signature.

These events or properties are only created under the following conditions:

- The Android event handling method has a prefix, for example, setOnClickListener.
- The Android event handler has a void return type.
- The Android event handler has a single parameter.

Constants to enumerations

It is common in the Android APIs to see methods that accept or return an int type that must be mapped to a constant to determine its meaning. When possible, the Xamarin team creates a .NET enumeration to replace the constants and adjusts the appropriate methods to work with the enumerations. This provides a significant productivity gain by being able to use IntelliSense from within the IDE as well as enhancing the type safety of the methods.

For More Information:
Development environments
Developers have two choices when it comes to IDEs, Xamarin Studio or Visual Studio.

Xamarin Studio
Xamarin Studio is a customized version of the MonoDevelop IDE, which can be used to develop Android, iOS, and OS X applications. Xamarin Studio is available on both OS X and Windows and has many advanced features as follows:

- Code completion
- Smart syntax highlighting
- Code navigation
- Code tooltips
- Integrated debugging for mobile apps running in emulators or on devices
- Source control integration with Git and subversion built-in

The following screenshot shows Xamarin Studio with the Android user interface designer opened:

For More Information:  
Xamarin for Visual Studio

Xamarin for Visual Studio is an add-in that supports the development of the Xamarin.Android apps. If you already have a license to Visual Studio and are comfortable with the environment, the add-in will likely be more appealing than Xamarin Studio because of the simplicity of adoption. The following screenshot shows Visual Studio 2012 with the Android user interface designer opened:

For More Information:
IDE comparison

The following table summarizes some of the pros and cons of adopting each IDE:

<table>
<thead>
<tr>
<th>IDE</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xamarin Studio</td>
<td>It comes with Xamarin.Android</td>
<td>It does not support the use of TFS for source control</td>
</tr>
<tr>
<td></td>
<td>No additional license is required</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Runs on Windows and OS X</td>
<td></td>
</tr>
<tr>
<td>Visual Studio</td>
<td>Most of the C# developers are already familiar and comfortable with Visual Studio</td>
<td>It requires an addition license</td>
</tr>
<tr>
<td></td>
<td>It allows the use of TFS for source control, which is used in many .NET shops</td>
<td>It runs on Windows only</td>
</tr>
</tbody>
</table>

Compatibility

The solution and project files created and updated by Xamarin Studio are compatible with Visual Studio, making it easy to switch between the two environments throughout the duration of a project. This also allows the team members to adopt the tool that they are most comfortable with or that runs on their platform of choice.

Summary

In this chapter, we have discussed the architecture of Xamarin.Android and the magic of how it facilitates the creation of Android apps using C# and .NET. We also reviewed a set of benefits and drawbacks of adopting Xamarin.Android. In the next chapter, we will install Xamarin.Android and create a project that we will build on for the remainder of the book.

For More Information:
Where to buy this book


Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.