Java Integration Overview

Related Articles:

- Architecture
- Binding a Java Library
- Working with JNI
- Sharpen
- Sharpen on Github

Related SDK:

- Java Native Interface

The Java ecosystem includes a diverse and immense collection of components. Many of these components can be used to reduce the time it takes to develop an Android application. This document will introduce and provide a high-level overview of some of the ways that developers can use these existing Java components to improve their Xamarin.Android application development experience.

Overview

Given the extent of the Java ecosystem, it is very likely that any given functionality required for a Xamarin.Android application has already been coded in Java. Because of this, it is appealing to try and reuse these existing libraries when creating a Xamarin.Android application.

There are three possible ways to reuse Java libraries in a Xamarin.Android application:

- **Create a Java Bindings Library** – With this technique, a Xamarin.Android project is used to create C# wrappers around the Java types. A Xamarin.Android application can then reference the C# wrappers created by this project, and then use the .jar file.
- **Java Native Interface** – The *Java Native Interface* (JNI) is a framework that allows non-Java code (such as C++ or C#) to call or be called by Java code running inside a JVM.
- **Port the Code** – This method involves taking the Java source code, and then converting it to C#. This can be done manually, or by using an automated tool such as Sharpen.

At the core of the first two techniques is the *Java Native Interface* (JNI). JNI is a framework that allows applications not written in Java to interact with Java code running in a Java Virtual Machine. Xamarin.Android uses JNI to create *bindings* for C# code.

The first technique is a more automated, declarative approach to binding Java libraries. It involves using either Xamarin Studio or a Visual Studio project type that is provided by Xamarin.Android—the Java Bindings Library. To successfully create these bindings, a Java Bindings Library may still require some manual modifications, but not as many as would a pure JNI approach.

The second technique, using JNI, works at a much lower level, but can provide for finer control and access to Java methods that would not normally be accessible through a Java Binding Library.

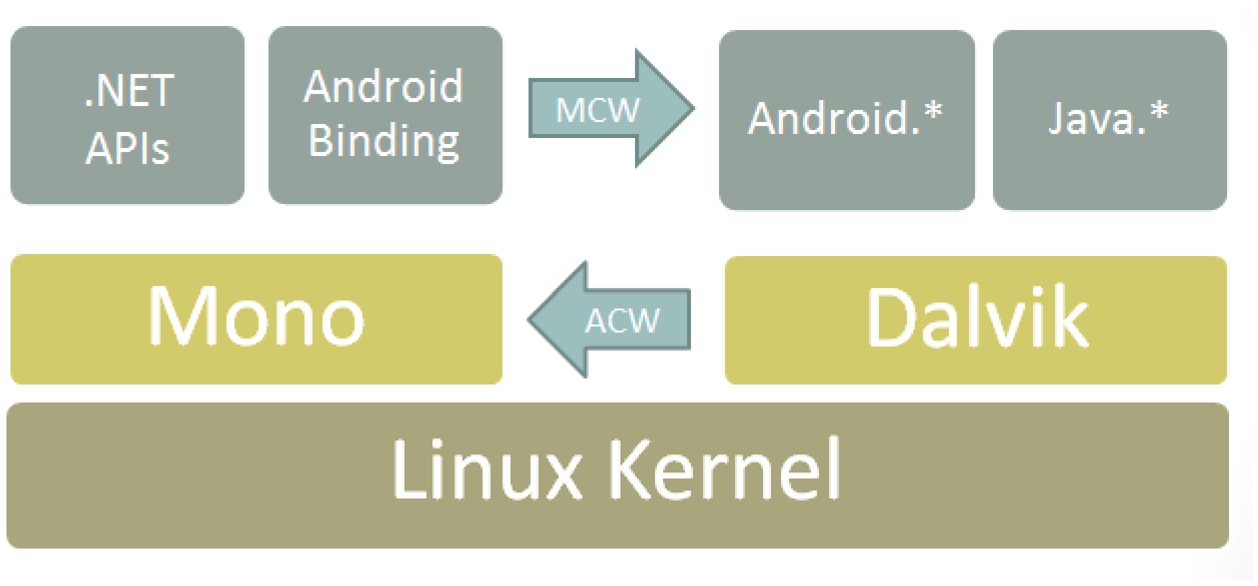
The third technique is radically different from the previous two: porting the code from Java to C#. Porting code from one language to another can be a very laborious process, but it is possible to reduce that effort with the help of a tool called *Sharpen*. Sharpen is an open source tool that is a Java-to-C# converter.

The concepts for each of these techniques will be discussed in more detail in the following sections.

Binding a Java Library

JNI was created solely to allow Java to interact with “native” code. Xamarin.Android leverages JNI to provide bindings for some of the Android libraries. For example, Mono.Android.dll contains the bindings for android.jar. Xamarin.Android accomplishes this by using *Managed Callable Wrappers* (MCW). MCW is a JNI bridge that is used every time managed code needs to invoke Java code. Managed callable wrappers also provide support for subclassing Java types and for overriding virtual methods on Java types. Likewise, anytime Dalvik code wishes to invoke managed code, it does so via another JNI bridge known as Android Callable Wrappers (ACW).

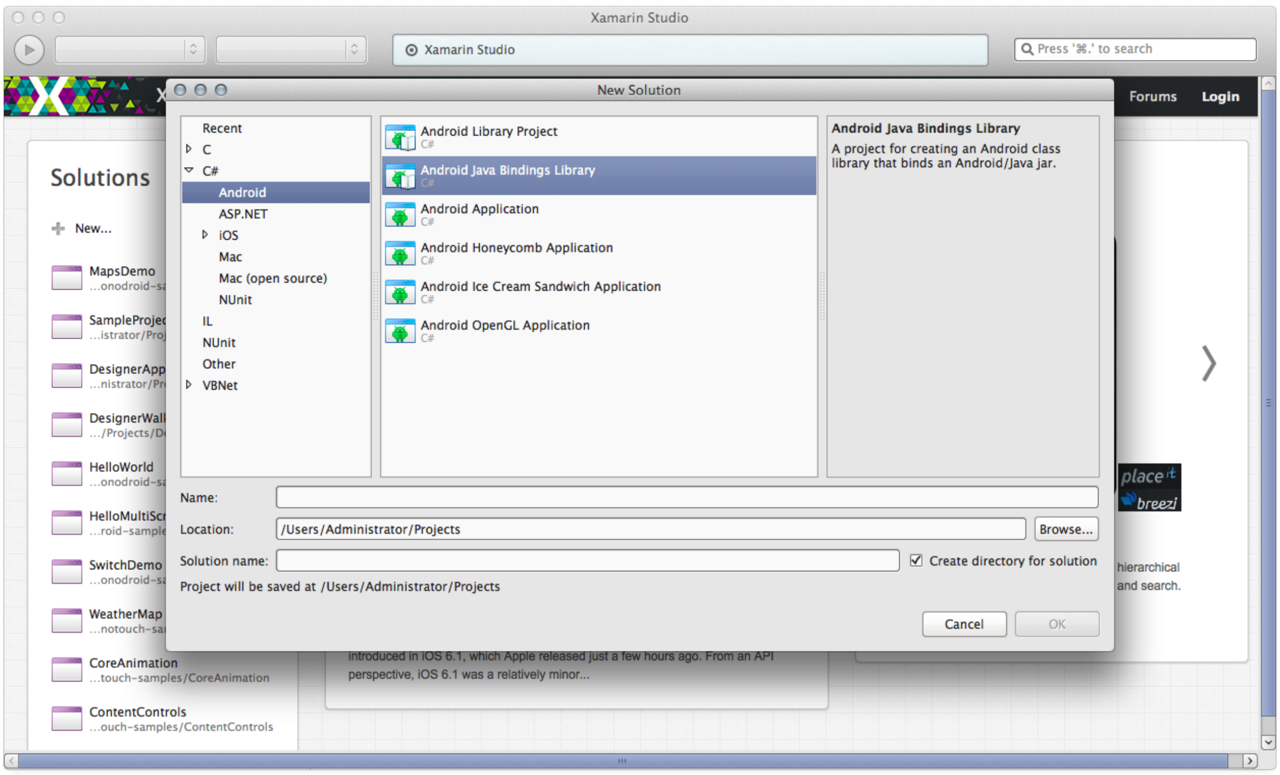
This architecture is illustrated in the following diagram:



Although MCW have always been a part of Xamarin.Android, it was only possible to create custom MCW starting in Xamarin.Android 4.0. Prior to that release, only non-virtual methods, static methods, or virtual methods without supporting overrides could be bound by the Android.Runtime.JNIEnv namespace.

Xamarin.Android 4.2 introduced a new project template called the *Java Bindings Library*. This new project type creates a .NET assembly with the bindings for.jar files for Android Library projects and will embed the necessary .jar files into the assembly. By referencing this assembly, a Xamarin.Android project may reuse an existing Java library.

The following two screenshots show how to create a new Java Bindings Library in Xamarin.Android and Visual Studio 2010:

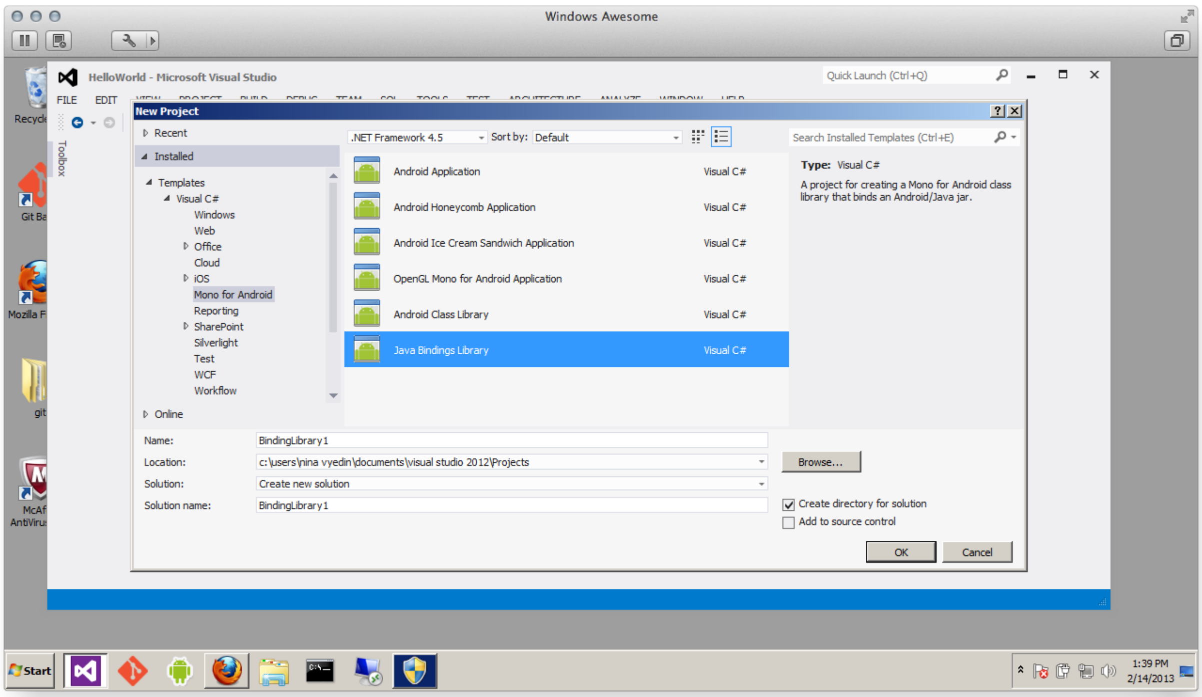

New Solution

Recent
C
C#
Android
ASP.NET
iOS
Mac
Mac (open source)
NUnit
IL
NUnit
Other
VBNet

Android Library Project C#
Android Java Bindings Library C#
Android Application C#
Android Honeycomb Application C#
Android Ice Cream Sandwich Application C#
Android OpenGL Application C#

Android Java Bindings Library
A project for creating an Android class library that binds an Android/Java jar.

Name:
Location: /Users/Administrator/Projects
Solution name:
Create directory for solution [✓]
Project will be saved at /Users/Administrator/Projects


New Project

Recent
Installed
Templates
Visual C#
Windows
Web
Office
Cloud
iOS
Mono for Android
Reporting
SharePoint
Silverlight
Test
WCF
Workflow
Online

.NET Framework 4.5 Sort by: Default

Android Application Visual C#
Android Honeycomb Application Visual C#
Android Ice Cream Sandwich Application Visual C#
OpenGL Mono for Android Application Visual C#
Android Class Library Visual C#
Java Bindings Library Visual C#

Type: Visual C#
A project for creating a Mono for Android class library that binds an Android/Java jar.

Name: BindingLibrary1
Location: c:\users\nina vyedin\documents\visual studio 2012\Projects
Solution: Create new solution
Solution name: BindingLibrary1
Create directory for solution [✓]
Add to source control

In general, creating a Java binding library is the recommended approach for using a .jar file. The document [Binding a Java Library] provides more detail and a walkthrough of the steps involved with this technique.

Working with JNI

It is not always necessary or possible to create an MCW to invoke Java code. In many cases, “inline” JNI is perfectly acceptable and useful for one-off use of unbound Java members. It is often simpler to use JNI to invoke a single method on a Java class than to generate an entire .jar binding.

The document Working with JNI provides a detailed and in-depth explanation about how to use JNI in a Xamarin.Android application.

Porting Java to C#

A third option for using Java in a Xamarin.Android application is to port the Java source code to C#. This approach may be of interest to organizations that:

- **Are switching technology stacks from Java to C#.**
- **Must maintain a C# and a Java version of the same product.**
- **Wish to have a .NET version of a popular Java library.**

There are two ways to port Java code to C#. The first way is to port the code manually. This involves skilled developers who understand both .NET and Java and are familiar with the proper idioms for each language. A future document from Xamarin will provide some guidance and advice for manually porting code. This approach makes the most sense for small amounts of code, or for organizations that wish to completely move away from Java to C#.

The second porting methodology is to try and automate the process by using a code converter, such as Sharpen. Sharpen is an open source converter from Versant that was originally used to port the code for *db4o* from Java to C#. db4o is an object-oriented database that Versant developed in Java, and then ported to .NET. Using a code converter may make sense for projects that must exist in both languages and that require some parity between the two.

An example of when an automated code conversion tool makes sense can be seen in the ngit project. Ngit is a port of the Java project jgit . Jgit itself is a Java implementation of the Git source code management system. To generate C# code from Java, the ngit programmers use a custom automated system to extract the Java code from jgit, apply some patches to accommodate the conversion process, and then run Sharpen, which generates the C# code. This allows the ngit project to benefit from the continuous, ongoing work that is done on jgit.

There is often a non-trivial amount of work involved with bootstrapping an automated code conversion tool, and this may prove to be a barrier to use. In many cases, it may be simpler and easier to port Java to C# by hand.

Summary

This document provided a high-level overview of some of the different ways that libraries from Java can be reused in a Xamarin.Android application. It introduced the concepts of bindings and managed callable wrappers, and discussed options for porting Java code to C#.

Source URL: http://docs.xamarin.com/guides/android/advanced_topics/java_integration_overview