Austin Caldwell

11/29/2016

CSC 483

**Publishing Mobile Apps for Android OS Devices Using Xamarin**

*This guide outlines the steps to configure an app for packaging, creating an Android OS mobile app package (a .apk file), creating a Google Developer account, and uploading the app for distribution on Google Play.*

*(It is assumed the developer is running Xamarin for Visual Studio with Microsoft Visual Studio.)*

When developing an Android mobile app using **Xamarin**, developers can test the app using various emulators running on a computer. To run the app on a mobile device (*i.e.* a smartphone or tablet), the app must be “packaged” and “distributed.” The most popular way of “distributing” Android applications is to release apps on Google Play.

1. **Preparing the Android App for Packaging**

The first step in releasing a mobile app for the Android OS is to adjust the app settings to prepare for packaging. The settings allow developers to protect the application code and optimize the app for various devices. Per (Xamarin Inc., n.d.), the steps taken to prepare a mobile application for packaging include:

1. **“Specify the Application Icon” –** The application icon is the image that users see on the home screen of their mobile devices. The icon should represent the mobile app’s brand or main purpose. The image file must be included as part of the Xamarin.Android project, and can be set “through the **Android Manifest** section of project **Properties” in Visual Studio 2013 and 2015.**[[1]](#footnote-1)
2. **“Version the Application” –** The Android app includes two different ways to track the app “version,” which are used to record-keeping and determining which features are included in each app package. The “Version Number” is an integer used only *internally* and “is incremented with each build.”[[2]](#footnote-2) It is important to keep in mind that Xamarin says “Applications and publishing services should not display this value to users.”[[3]](#footnote-3) The “Version Name” is a string used *externally* which lets the user know which version of an application is running on the user’s device. The “Version Name” is also listed on Google Play. The “Version Number” and “Version Name” attributes are set using the **Android Manifest**.
3. **Configure Required Permissions –** Developers need to manually set permissions for the Android app being developed. Permissions are set in the **Android Manifest**. It is important to note **permissions can be set differently for “Debug” and “Release” builds, and only the permissions set for “Release” builds will be included in the packaged/published app.** Permissions include options such as INTERNET and READ\_EXTERNAL\_STORAGE which must be enabled for the app to access a mobile device’s Internet connection or storage (for reading files), respectively.[[4]](#footnote-4)
4. **“Shrink the APK” –** Developers should make app packages as small as possible. Users do not like to download excessively large apps because large app downloads consume cellular data plans and device storage space. There are many technologies used to “shrink the APK,” and these technologies include:

* **Linker:** “Uses static analysis to determine which assemblies, types, and type members are used or referenced by a Xamarin.Android application. The linker then discards all the unused assemblies, types, and members that are not used (or referenced [at runtime]). This can result in a significant reduction in the package size.”[[5]](#footnote-5) This option is set using the **Android Options**. The **linker** has three options: *None*, *SDK Assemblies Only*, and *SDK and* *User Assemblies*. *None* disables the **linker** entirely. *SDK Assemblies Only* includes “assemblies that are required by Xamarin.Android” itself and ignores all other assemblies.[[6]](#footnote-6) *SDK and User Assemblies* includes all assemblies “that are required by the application, and not just the ones required by Xamarin.Android.”[[7]](#footnote-7)
* **ProGuard:** A feature that “obfuscates Java code” and “removes unused Java bytecode,” thus reducing the size of even small apps.[[8]](#footnote-8) This feature is configured using the **Android Options** and “is disabled by default.”[[9]](#footnote-9) It is important to note “The Xamarin.Android ProGuard configuration does not obfuscate the APK, and it is not possible to enable obfuscation through ProGuard, even with custom configuration files.”[[10]](#footnote-10) Xamarin.Android apps must use *Dotfuscator* for code obfuscation.

1. **“Protect the Application” –** Developers must also protect their mobile applications from unwanted tampering by malicious or curious end users. Using **Android Options** in the mobile app’s properties, Xamarin.Android allows developers to disable debugging and obfuscate code using *Dotfuscator,* which will prevent re-packaging of the app. Re-packaging the app would allow malicious parties to debug the app or change app permissions even if debugging was originally disabled in the released app.[[11]](#footnote-11) Developers should also consider “bundling assemblies into native code” using shared libraries, but “this option requires an Enterprise license [for Visual Studio].”[[12]](#footnote-12)
2. **“Set Packaging Properties” –** The mobile app has many packaging properties that can be adjusted by developers. The settings “determine how the app is optimized for size and execution speed, how it is protected from tampering, and how it can be packaged to support different architectures and size restrictions.”[[13]](#footnote-13) These options are set in the **Android Options** tab of the mobile app project’s properties. Developers will need to specify options for:

* **Supported Architectures:** Apps can support many different CPU architectures, including x86 and x86-64 (32-bit and 64-bit), and ARM processors. Xamarin.Android defaults to using the*armeabi-v7a* architecture.[[14]](#footnote-14) An app targeting a 32-bit CPU can also run on 64-bit devices, but will not be able to address as much memory as a 64-bit app running on a 64-bit CPU. The 64-bit ARM CPU is the *arm64-v8a* architecture.[[15]](#footnote-15) Xamarin.Android’s developer help states “In some situations, you may need to create a separate APK for each architecture (to reduce the size of your APK, or because your app has shared libraries that are specific to a particular CPU architecture).”[[16]](#footnote-16)
* **Packages per ABI:** If the developers choose to create “one package per ABI,” the app will have one .apk file for each supported CPU architecture. If the “one package per ABI” option is unchecked, only one large .apk file will be created and will support all selected CPU architectures.
* **Multi-Dex:** The *Multi-*Dex option “bypass[es] the 65K method limit of the .dex file format.”[[17]](#footnote-17)Xamarin.Android apps are limited to 65K (65,000) Java method “*references* (including those in any libraries that you use) [… and] is not based on the number of methods that you *define.*”[[18]](#footnote-18) *ProGuard* can be used to remove unnecessary Java code and reduce the number of method references in the app. Xamarin recommends developers enable *ProGuard* before enabling *Multi-Dex*.

1. **“Compile” –** Developers must clean the solution and build it again, to ensure that the entire application builds correctly with all the adjusted settings.
2. **“Archive for Publishing” –** To prepare to publish the app in its current state, including all current settings in the **Android Manifest**, developers must “archive” the app project. To archive an app project, developers simply right-click on the name of the project and click “Archive.” This action opens the **Archive Manager** through which archives are distributed.

At this point, the app has been built, configured, and prepared for package creation. The next step is to create the .apk file (the package), which is copied to an Android device in order to install the mobile app.

1. **Packaging the Android App**

The Xamarin website gives instructions for developers on how to package Android applications. The specific steps used to create an app package include:

1. Once an **Archive** has been created, developers must enter the **Archive** Manager in Visual Studio and then click “Distribute ….” A screen will appear, presenting two options for app distribution: “Ad Hoc” and “Google Play.” The developers should select “Google Play” to prepare their app for release on the Google Play Store.
2. The app distribution wizard will ask for a **Signing Identity**. The **Signing Identity** is an **Android (Java) Keystore** used to virtually sign the app with a digital certificate, to assure users of the app’s origin and the app developer’s identity. If no signing keys exist, the developers must create a new key. Click the “+” icon to create a new key.
3. The developers must create an “Android Key Store” by giving the key an alias/name, a password, and a length of time to be valid (currently, apps posted to Google Play must have keys valid until at least October 23, 2033). The developers must also enter information about their name, department, organization, city, state, and country so that users can identity the app’s creator.
4. Once the new key is created, the developers must “sign” the APK. To do this, developers open the app distribution wizard and select the name of the key they wish to use for signing the app. Developers then select a location to which the .apk file should be saved. The final step before the APK is created is to “sign” the app with the key by entering the key’s password. Once the app is signed, the APK is created and stored in the specified folder.

At this point, the app has been packaged into an APK, or Android app package. The .apk file can now be copied/pasted to a mobile device or emulator to install and run the app. The app can also be distributed via an app store, such as Google Play (as described next).

1. **Distributing the Android App via Google Play**

To distribute a mobile app via Google Play, developers must 1) register for a **Google Publisher Account**, 2) meet all quality criteria for the application itself, and 3) push the app package to Google Play for distribution.

1. To create a **Publisher Account**, the developers must designate a Google account to represent their organization. (An individual developer can register himself for a **Publisher Account** if desired.) The organization’s developer account will need to include personal contact information and will require acceptance of the “Developer Distribution Agreement” that specifies the rules and regions for how and where Google Play apps can be distributed. Finally, the developer or organization must pay $25.00 USD to register. If the developer/organization desires to “sell priced apps, in-app products, or subscriptions,” they will “need a Google payments merchant account” as well.[[19]](#footnote-19) Developers registered as Google Publishers will be able to manage their apps using the **Developer Console**.
2. Before a mobile app can be published to Google Play, the app must meet a list of criteria and follow certain publishing steps established by Google. The various criteria and steps to follow include (but are not limited to):

* The key used to sign the app must expire after October 22, 2033.[[20]](#footnote-20)
* The mobile app APK package must be 100 MB or less.[[21]](#footnote-21)
* The APK can “have 2 additional files, each of them up to 2GB in size” hosted by Google Play using the Android Expansion system.[[22]](#footnote-22)
* Apps on Google Play should be rated. Developers can have their apps rated by “fill[ing] out a rating questionnaire on the Google Play **Developer Console** about the nature of [the] apps’ content and [receiving] a content rating from multiple rating authorities.”[[23]](#footnote-23)
* The application should include a **launcher icon** for the device to display to represent the app – “a 32-bit PNG with an alpha channel for transparency.”[[24]](#footnote-24) The application should *also* include a high-resolution application icon for display on Google Play.
* The developers should include between two (2) (minimum) and eight (8) (maximum) screenshots to represent the app on Google Play. Developers may also include promotional and/or featured graphics and a 30 second (minimum) to 2 minute (maximum) video hosted on YouTube to “showcase the best parts of [the] application.”[[25]](#footnote-25)
* The app must integrate **Google Licensing Services** if the application is a priced app. **Google Licensing Services** requires an Internet connection and compares the app on a user’s device to a license on Google’s servers. The app will only function properly if the license is valid for the app in question. Free apps “only require Google Licensing when the application uses APK expansion files.”[[26]](#footnote-26)
* Developers should follow Google’s published **Launch Checklist** for releasing Android apps via Google Play. The checklist and additional materials involve app testing, app quality standards, app payment processing, localization, and release strategies (Alpha, Beta, or Full Release). The checklist can be found at <https://developer.android.com/distribute/tools/launch-checklist.html>.

1. The final step in releasing an app to Google Play is to push the APK file to Google Play. The first time this step is done, it must be completed through the online **Developer Console** accessed through the developer’s or organization’s **Publisher Account** (set up earlier). After the app is released the first time, the developers can use the distribution wizard included in Visual Studio for faster app releases. To configure the distribution wizard, developers must “Register Google API Access” in Visual Studio to connect the app project to an **API project** on the **Publisher Account**. Once the API is configured, new app APKs can be published straight from Visual Studio to Google Play. Publishing to Google Play from Visual Studio creates an APK from an app **Archive** and uploads the APK directly to Google Play (similarly to how .apk files are created using the distribution wizard’s “Ad Hoc” mode and saved on a local computer as described in step II.4 of this guide).

At this point, the mobile app is uploaded to Google Play and is ready for users to download, install, and use the app on their devices.

*In review, this guide outlined the steps to configure an app for packaging, create an Android OS mobile app package (a .apk file), create a Google Developer account, and upload the app for distribution on Google Play.*

# **References**

Google. (n.d., n.d. n.d.). *Content Ratings for Apps & Games.* Retrieved November 26, 2016, from Developer Console Help: https://support.google.com/googleplay/android-developer/answer/188189

Google. (n.d., n.d. n.d.). *Get Started with Publishing.* Retrieved November 26, 2016, from Android - Developers: https://developer.android.com/distribute/googleplay/start.html

Xamarin Inc. (n.d., n.d. n.d.). *CPU Architectures.* Retrieved November 25, 2016, from Xamarin Developers: https://developer.xamarin.com/guides/android/application\_fundamentals/cpu\_architectures/

Xamarin Inc. (n.d., n.d. n.d.). *Part 1 - Preparing an Application for Release.* Retrieved November 22, 2016, from Xamarin Developers: https://developer.xamarin.com/guides/android/deployment,\_testing,\_and\_metrics/publishing\_an\_application/part\_1\_-\_preparing\_an\_application\_for\_release/

Xamarin Inc. (n.d., n.d. n.d.). *Part 3 - Publishing an Application on Google Play.* Retrieved November 26, 2016, from Xamarin Developers: https://developer.xamarin.com/guides/android/deployment,\_testing,\_and\_metrics/publishing\_an\_application/part\_3\_-\_publishing\_an\_application\_on\_google\_play/

Xamarin Inc. (n.d., n.d. n.d.). *Part 4 - Google Licensing Services.* Retrieved November 26, 2016, from Xamarin Developers: https://developer.xamarin.com/guides/android/deployment,\_testing,\_and\_metrics/publishing\_an\_application/part\_4\_-\_google\_licensing\_services/

Xamarin Inc. (n.d., n.d. n.d.). *Permissions.* Retrieved November 26, 2016, from Xamarin Developers: https://developer.xamarin.com/guides/android/application\_fundamentals/permissions/

1. (Xamarin Inc., n.d.) [↑](#footnote-ref-1)
2. (Xamarin Inc., n.d.) [↑](#footnote-ref-2)
3. (Xamarin Inc., n.d.) [↑](#footnote-ref-3)
4. (Xamarin Inc., n.d.) [↑](#footnote-ref-4)
5. (Xamarin Inc., n.d.) [↑](#footnote-ref-5)
6. (Xamarin Inc., n.d.) [↑](#footnote-ref-6)
7. (Xamarin Inc., n.d.) [↑](#footnote-ref-7)
8. (Xamarin Inc., n.d.) [↑](#footnote-ref-8)
9. (Xamarin Inc., n.d.) [↑](#footnote-ref-9)
10. (Xamarin Inc., n.d.) [↑](#footnote-ref-10)
11. (Xamarin Inc., n.d.) [↑](#footnote-ref-11)
12. (Xamarin Inc., n.d.) [↑](#footnote-ref-12)
13. (Xamarin Inc., n.d.) [↑](#footnote-ref-13)
14. (Xamarin Inc., n.d.) [↑](#footnote-ref-14)
15. (Xamarin Inc., n.d.) [↑](#footnote-ref-15)
16. (Xamarin Inc., n.d.) [↑](#footnote-ref-16)
17. (Xamarin Inc., n.d.) [↑](#footnote-ref-17)
18. (Xamarin Inc., n.d.) [↑](#footnote-ref-18)
19. (Google, n.d.) [↑](#footnote-ref-19)
20. (Xamarin Inc., n.d.) [↑](#footnote-ref-20)
21. (Xamarin Inc., n.d.) [↑](#footnote-ref-21)
22. (Xamarin Inc., n.d.) [↑](#footnote-ref-22)
23. (Google, n.d.) [↑](#footnote-ref-23)
24. (Xamarin Inc., n.d.) [↑](#footnote-ref-24)
25. (Xamarin Inc., n.d.) [↑](#footnote-ref-25)
26. (Xamarin Inc., n.d.) [↑](#footnote-ref-26)