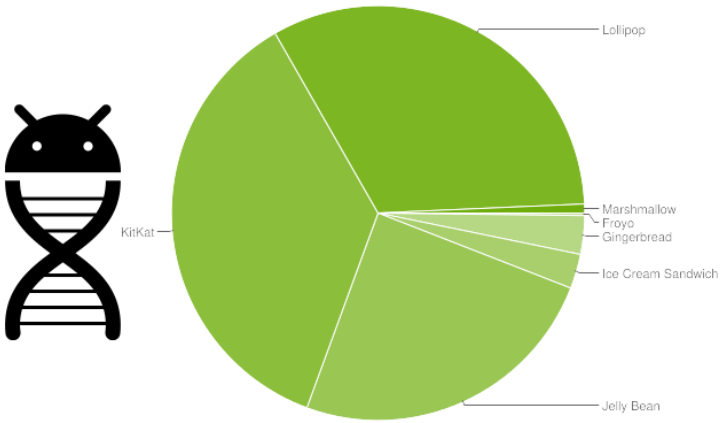
Ian Lake
Android Framework Developer at Google and Runner www.google.com/+IanLake
Jan 6, 2016 · 5 min read



Picking your compileSdkVersion, minSdkVersion, and targetSdkVersion

Depending on the time of the year, it might only be a few months after you release an app that a new version of Android is announced. What does that mean for your app though—is everything going to break?

You'll be happy to know that **forward compatibility** is a strong focus of Android—existing apps built against prior SDKs should not break when the user updates to a new version of Android. This is where **compileSdkVersion**, **minSdkVersion**, and **targetSdkVersion** come in: they control what APIs are available, what the required API level is, and what compatiblity modes are applied, respectively.

compileSdkVersion

compileSdkVersion is your way to tell Gradle what version of the Android SDK to compile your app with. Using the new Android SDK is a requirement to use any of the new APIs added in that level.

It should be emphasized that **changing your compileSdkVersion does not change runtime behavior**. While new compiler warnings/errors may be present when changing your compileSdkVersion, your compileSdkVersion is not included in your APK: it is purely used at compile time. (You should really fix those warnings though—they were added for a reason!)

Therefore it is strongly recommended that you **always compile with the latest SDK**. You'll get all the benefits of new compilation checks on existing code, avoid newly deprecated APIs, and be ready to use new APIs.

Note that if you use the Support Library, compiling with the latest SDK is a *requirement* for using the latest Support Library releases. For example, to use the 23.1.1 Support Library, you must have a compileSdkVersion of at least 23 (those first numbers need to match!). In general, a new version of the Support Library is released alongside a new platform version, providing compatibility shims to newly added APIs as well as new features.

minSdkVersion

If compileSdkVersion sets the newest APIs available to you, **minSdkVersion is the lower bound for your app**. The minSdkVersion is one of the signals the Google Play Store uses to determine which of a user's devices an app can be installed on.

It also plays an important role during development: by default lint runs against your project, warning you when you use any APIs above your minSdkVersion, helping you avoid the runtime issue of attempting to call an API that doesn't exist. Checking the system version at runtime is a common technique when using APIs only on newer platform versions.

Keep in mind that libraries you use, such as any of the Support Libraries or Google Play services, may have their own minSdkVersion —your app's minSdkVersion must be at least as high as your dependencies' minSdkVersion—if you have libraries that require 4, 7, and 9, your minSdkVersion must be at least 9. In rare cases where you want to continue to use a library with a higher minSdkVersion than your app (and deal with all edge cases/ensure the library is only used on newer platform versions), you can use the tools:overrideLibrary marker, but make sure to test thoroughly!

When deciding on a minSdkVersion, you should **consider the stats on the Dashboards**, which give you a global look on all devices that visited the Google Play Store in the prior 7 days—that's your potential audience when putting an app on Google Play. It is ultimately a business decision on whether supporting an additional 3% of devices is worth the development and testing time required to ensure the best experience.

Of course, if a new API is key to your entire app, then that makes the minSdkVersion discussion quite a bit easier. Just remember that even 0.7% of 1.4 billion devices is *a lot* of devices.

targetSdkVersion

The most interesting of the three, however, is targetSdkVersion. **targetSdkVersion is the main way Android provides forward compatibility** by not applying behavior changes unless the targetSdkVersion is updated. This allows you to use new APIs (as you did update your compileSdkVersion right?) prior to working through the behavior changes.

Much of the behavior changes that targetSdkVersion implies are documented directly in the VERSION_CODES, but all of the gory details are also listed on the each releases' platform highlights, nicely linked in the API Levels table.

For example, the Android 6.0 changes talk through how targeting API 23 transitions your app to the runtime permissions model and the Android 4.4 behavior changes detail how targeting API 19 or higher changes how alarms set with *set()* and *setRepeating()* work.

With some of the behavior changes being very visible to users (the deprecation of the menu button, runtime permissions, etc), **updating to target the latest SDK should be a high priority** for every app. That doesn't mean you have to use every new feature introduced nor should you blindly update your targetSdkVersion without testing—**please, *please* test before updating your targetSdkVersion**! Your users will thank you.

Gradle and SDK versions

So setting the correct compileSdkVersion, minSdkVersion, and targetSdkVersion is important. As you might imagine in a world with Gradle and Android Studio, these values are integrated into the tools system through inclusion in your module's build.gradle file (also available through the Project Structure option in Android Studio):

```
android {
  compileSdkVersion 23
  buildToolsVersion “23.0.1”

  defaultConfig {
    applicationId “com.example.checkyourtargetsdk"
```

```
      minSdkVersion 7
      targetSdkVersion 23
      versionCode 1
      versionName "1.0"
    }
  }
```

The compileSdkVersion, being a compile time thing (who would have guessed!), is one of the android settings alongside with your build tools version. The other two are slightly differently in that they are declared at the build variant level—the *defaultConfig* is the base for all build variants and where'd you put default values for these, but you could imagine a more complicated system where specific versions of your app have a different minSdkVersion for example.

minSdkVersion and targetSdkVersion also differ from compileSdkVersion in that they are included in your final APK—if you were to look at the generated *AndroidManifest.xml*, you'd see a tag such as:

```
<uses-sdk android:targetSdkVersion="23"
android:minSdkVersion="7" />
```

You'll find if you manually put this in your manifest, it'll be ignored when you build with Gradle (although other build systems might certainly rely on it being there).

Putting it all together

If you made it through the **bolded notes**, you'll notice a relationship between the three values:

```
minSdkVersion <= targetSdkVersion <= compileSdkVersion
```

This intuitively makes sense—if compileSdkVersion is your 'maximum' and minSdkVersion is your 'minimum' then your maximum must be at least as high as your minimum and the target must be somewhere in between.

Ideally, the relationship would look more like this in the steady state:

```
minSdkVersion (lowest possible) <=
    targetSdkVersion == compileSdkVersion (latest SDK)
```

You'll hit the biggest audience with a low minSdkVersion and look and act the best by targeting and compiling with the latest SDK—a great way to #BuildBetterApps.

Join the discussion on the Google+ post and follow the Android Development Patterns Collection for more!