Building the To Do application

for Android

by Trevin Beattie

Table of Contents

1 Building from the Source

When making any changes to the code, you also need to change the release date in the "About..." dialog (`res/values/strings.xml` / `InfoPopupText`) as well as the application's internal version number (`AndroidManafest.xml` / `android:versionCode` and `android:versionName`).

Building the application `.apk` was originally done up until 2014 using:

- A Java 1.6 compiler
- Eclipse integrated development environment (IDE)
- The Android software development kit (SDK) release 12.

To build the old code on a newer system requires a few different tools, but not *too* modern. The following development environment was set up and tested in 2025 on Fedora Linux 40 with only minor bug fixes to the code:

- Java (1.)8 *(Do* **not** *use a newer version, as Gradle 3.5 is not compatible with Java 11 or higher.)*
- Android Studio 2.3.3, downloaded from the archive (`https://developer.android.com/studio/archive`).
- Gradle 3.5, installed by setting up the Gradle Wrapper for the project and then modifying `gradle/wrapper/gradle-wrapper.properties` to set `distributionUrl` to `https\://services.gradle.org/distributions/gradle-3.5-all.zip`.
- The Android Gradle Plugin version 2.3.3, which had to be downloaded along with most of its dependencies from a 3rd-party mirror `https://repository.axelor.com/nexus/service/rest/repository/browse/maven-public/` as it does not exist in Maven Central nor in Google's Maven repository.

See `http://developer.android.com/sdk/index.html` for information on how to set up a project using the Android SDK.

1.1 Generating Icons

Some of the application icons are generated from 3-D image description files using Persistence of Vision (`http://www.povray.org/`). The image files are included with the source code (in `app/src/main/res/drawable*/`, but in case you want to tweak or change any image you can use `povray` to generate new ones. The `.pov` sources are under `app/src/main/graphics/`.

Because some of the icons contain small details that may get lost if rendered at a low resolution, it is recommended that you render the initial image at a size which is the least common multiple of all icon sizes — 288×288 — and then use a raster graphics program such as the GIMP (`http://www.gimp.org/`), `convert` (`http://www.imagemagick.org/`), or `pamscale` (`http://netpbm.sourceforge.net/`). For each image file

`foo.png`, you need to create five icons: a 16×16 icon in `res/drawable/foo_16.png`, a 24×24 icon in `res/drawable-mdpi/foo.png`, a 32×32 icon in `res/drawable/foo_32.png`, a 36×36 icon in `res/drawable/foo_36.png`, and a 48×48 icon in `res/drawable-hdpi/foo.png`.

The command for generating an image from one of the `.pov` descriptions is:

```
povray +FN +AM3 +A0.3 +UA +W288 +H288 foo.pov
```

The main application icon was drawn by hand in the GIMP, and can be found in `IconMaster.xcf`. This is scaled down to 48×48 for `res/drawable-mdpi/icon.png`, and to 72×72 for `res/drawable-hdpi/icon.png`.

1.2 Testing

Currently, there are no automated tests for the source code; testing must be done by running the application either on an emulator or real Android device which has USB Debugging enabled.

1.2.1 Using Virtual Devices

If you want to test the application on different versions of Android or different types of devices than you have physical devices for (or don't want to use real devices for testing), you will need to configure emulators using the Android Virtual Device Manager.

In Android Studio, go to Tools → Android → AVD Manager to open the Android Virtual Device Manager. Click on "Create Virtual Device" to add a new emulator. On the first page you will select the type and screen size of the device, e.g. small phone or large tablet, along with its pixel resolution.

On the next page choose which version of Android the emulator will run. The ABI determines what type of CPU the system will run on: "arm", "arm64", "x86", or "x86_64". In order for the emulator to run the ABI **must** match (or be compatible with) your computer's host CPU; for example, an "arm" image *will not run* on an "x86_64" computer. (This means that you cannot test on Android 2.2 (Froyo) or earlier in emulation, since there are no x86 builds for those versions of Andriod.)

The system image list shows both the Android version (e.g. 6.0), code name (e.g. Marshmallow), and API level (e.g. 23). You should create at least enough emulators to cover the minimum and target SDK versions specified in `AndroidManifest.xml`, and ideally the latest release of Android and a few intermediate versions to make sure the app is compatible across a wide range of device ages. It is *highly recommended* to test against the following API's, since the code has branches that follow different paths for each of these versions:

- Froyo (Android 2.2, API 8), which is the current minimum version
- Gingerbread (Android 2.3, API 9–10)

- Honeycomb (Android 3.0, API 11) through Jelly Bean (Android 4.3, API 18)
- KitKat (Android 4.4, API 19) through Lollipop (Android 5.1, API 22)
- Marshmallow (Android 6, API 23), which is the current target version

Testing the following API's is suggested to ensure compatibility with new devices:

- Nougat (Android 7, API 24–25)
- Oreo (Android 8, API 26) through Snow Cone (Android 12, API 32)
- Tiramisu (Android 12, API 33)
- Upside-down Case (Android 14, API 34) and up

On the last page finalize the details of the virtual device. Give it a descriptive name which will distinguish it from other virtual devices; for example, include the type/size of the device and the Android version and/or API level. Under "Advanced Settings" you can configure the system's CPU and memory usage.

To run the virtual device for testing, click on its play button in the device list. The emulated device should start up in a new window, acting as if it were powered on.

1.2.2 Enabling USB Debugging

You will need to do this whether you are using a virtual or real device.

First, if your device runs Android 4.2 (Jelly Bean) or higher you will need to enable Developer Mode. To do this, go to the system Settings and under "About phone" look for the "Build number". (This may be nested under "Software information".) Tap on the build number **seven times**; you should see the message "You are now a developer!".

Next find the "Developer options" in the system settings and scroll down to the "USB debugging" switch. Turn that on whenever you need to test the application, and plug the phone into your computer that is running Android Studio. For security you should always turn this setting back off when you are finished testing.

1.2.3 Debugging the Application

In Android Studio (with the ToDo project open), ensure you have a valid build. If you will be testing on a virtual device be sure that the emulator is running. See Section 1.2.1 [Using Virtual Devices], page 2, for how to configure and run the virtual device.

Click the debug icon or go to Run → Debug 'app', then in the "Select Deployment Target" window select the device under "Connected Devices". If that section shows "<none>" then either the device is not running, not connected (by a USB cable if it's a physical device), doesn't have USB debugging enabled, or may have some other problem with it. When you click

"OK" then Studio should install the current app build onto the device and launch it.

The "Android Monitor" panel may be used to view any log messages from the application or the Android system. Be sure the correct device is selected. When the application is started in debug mode, Android will wait for Android Studio's debugger to connect before the application will run so that it can catch any startup errors. If the application crashes, check then panel for an exception error message which should show the point in the code where the error occurred.

1.3 Generating the Application Package (APK)

In Android Studio, click on the Build menu then "Build APK". If there were no errors, this should produce `app/app-release.apk`. You should rename this file to a more descriptive name like `todo-1.2.0.apk`.

1.4 Generating Documentation

Lastly, if you need to generate a new edition of this manual, you will need `texinfo` (`http://www.gnu.org/software/texinfo/`) and `texi2pdf` (`https://www.gnu.org/software/texinfo/manual/texinfo/html_node/Format-with-texi2dvi-or-texi2pdf.html`).

To generate the manual in PDF, simply run:

```
texi2pdf ToDo.texinfo
```

2 To Do

- Document how the code works
- Document the data structures
- Document the encryption algorithms