(This wiki page applies to people migrating code written before February 2017. It is unlikely to still be relevant. The `flutter` tool directs people to this page when a non-Gradle project is detected.)

Introduction

Prior to Pull Request 7902 – which was merged to master on February 6th 2017 – we used a custom build process when building for Android. We now build for Android using the same standard 'gradle-based' build process that Android Studio project uses. This makes it easier to edit the native Android code; after the upgrade you can open the `android` folder in Android Studio, and it will know how to build the project, and will allow you to add java code.

If you have a project that was created prior to this date, please follow these steps to switch to building with gradle. This is required as we will be removing the custom build support shortly.

Note: These steps apply to projects created with `flutter create` prior to February 6th 2017. If your project was based on a copy of `/examples/hello_services/`, then you just need to synchronize the contents of `/android/build.gradle` and `/android/app/build.gradle`.

Upgrading an existing project

The steps below use `<existing-app-dir>` as a placeholder for the directory containing your existing app, e.g. `~/dev/flutter/awesomeapp`.

Step 0: Ensure flutter is up to date

Make sure you flutter SDK is up to date. You need a recent (March 1st, 2017 later) SDK from the master branch of https://github.com/flutter/flutter.

You can upgrade by running `flutter upgrade` in a terminal.

Step 1: Create a new project with the new template

```
flutter create <appname>
mv <appname>/lib <appname>/lib.template
```

Step 2: Merge over your Flutter UI code

Main Dart UI code:

```
cp -r <existing-app-dir>/lib <appname>/
```

If `<existing-app-dir>/assets` exists:

```
cp -r <existing-app-dir>/assets <appname>/
```

Step 3: Synchronize AndroidManifest.xml

Note: This step is only required if you made any changes to`<existing-app-dir>android/AndroidManifest.xm`

If that is the case, apply those changes to the new manifest, `<appname>/android/app/src/main/AndroidManife`

Step 4: Synchronize pubspec.yaml

Note: This step is only required if you made any changes to`<existing-app-dir>/pubspec.yaml`

If that is the case, apply those changes to the new manifest,`<appname>/pubspec.yaml`

Step 5: Move icon resources, and other resources as applicable

Note: This step is only required if your app has custom launcher icons.

```
cp -r <existing-app-dir>/android/res <appname>android/app/src/main/
```

Repeat this for any other resources you may have.

Step 6: Move over iOS code if applicable

Note: This step is only required if your app has any custom iOS code.

```
mv <appname>/ios <appname>/ios.template
cp -r <existing-app-dir>/ios/ <appname>
```

Step 7: Validate you can build, and clean-up

When you are done, the structure of your `android` folder should match that of a new flutter project created with `flutter create` using a recent framework.

Validate that you can build for both Android and iOS. *Note*: The first build will take a little longer (as it will download and then cache a few gradle files).

```
cd <appname>
flutter build apk
flutter build ios
```

Finally, once you are sure everything is working as intended, consider removing the `.template` folders.

Working with the project after upgrading

You can still build with `flutter build apk` and run with `flutter run`.

In addition, you can edit and build from Android Studio:

1. Start Android Studio
2. Invoke File > Open...
3. Select the `android` folder inside your folder, and press OK

4. Press the Run button, or use the Run > Run menu item to build and run the app
5. Edit java code by locating the .java file in the file tree on the right hand side.