Barcode Detection with the Mobile Vision API – Google Codelabs

Link: <https://codelabs.developers.google.com/codelabs/bar-codes/>

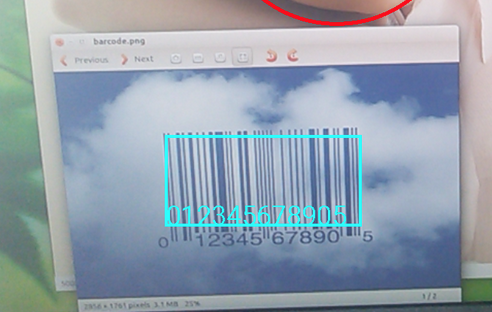
## **1. Overview**

With the release of Google Play services 7.8 we're excited to announce that we've added new Mobile Vision APIs which provide bar code detection APIs that read and decode a myriad of different bar code types, quickly, easily and locally.

## **Barcode detection**

Classes for detecting and parsing bar codes are available in the com.google.android.gms.vision.barcode namespace. The BarcodeDetector class is the main workhorse -- processing Frame objects to return a SparseArray<Barcode> types.

The Barcode type represents a single recognized barcode and its value. In the case of 1D barcode such as UPC codes, this will simply be the number that is encoded in the bar code. This is available in the rawValue property, with the detected encoding type set in the format field.



For 2D bar codes that contain structured data, such as QR codes -- the valueFormat field is set to the detected value type, and the corresponding data field is set. So, for example, if the [URL](https://developers.google.com/android/reference/com/google/android/gms/vision/barcode/Barcode.html#constants) type is detected, the constant URL will be loaded into the valueFormat, and the [Barcode.UrlBookmark](https://developers.google.com/android/reference/com/google/android/gms/vision/barcode/Barcode.UrlBookmark" \t "_blank) will contain the URL value. Beyond URLs, there are lots of different data types that the QR code can support -- check them out in the documentation [here](https://developers.google.com/android/reference/com/google/android/gms/vision/barcode/Barcode).

When using the Mobile Vision APIs, you can read barcodes in any orientation - they don't always need to be the straight on, and oriented upwards!

Importantly, all bar code parsing is done locally, so you don't need to do a server round trip to read the data from the code. In some cases, such as [PDF-417](https://en.wikipedia.org/wiki/PDF417), which can hold up to 1kb of text, you may not even need to talk to a server at all to get all the information you need.

You can learn more about using the API by checking out the Multi-Detector sample on GitHub. This uses the Mobile Vision APIs along with a Camera preview to detect both faces and bar codes in the same image.

## **Prerequisites**

Before beginning, check that you have all the necessary pre-requisites. These include:

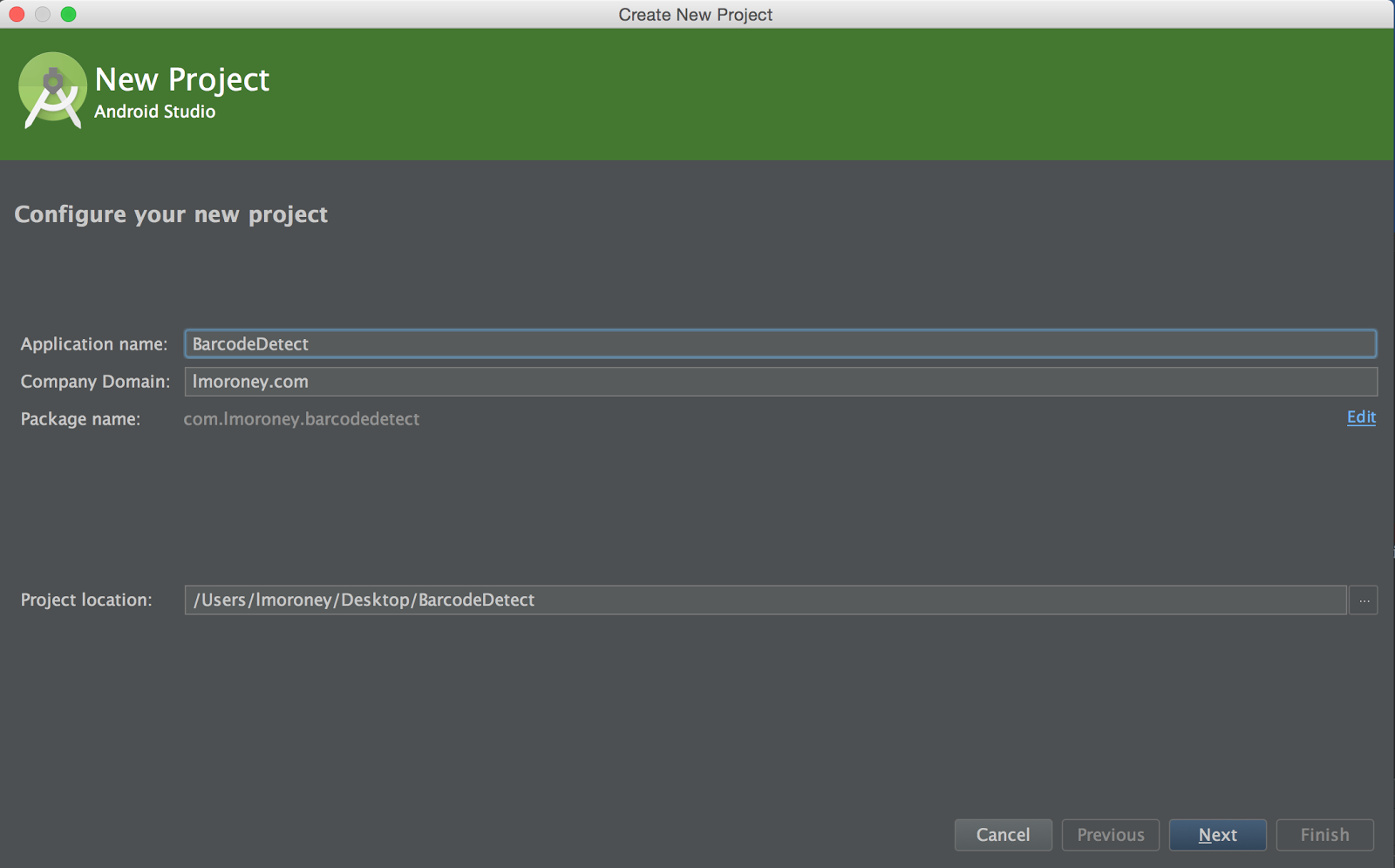
* Android Studio
* An Android Device that runs Android 4.2.2 or later -or- A configured Android Emulator (this is available in Android Studio)
* The latest version of the Android SDK including the SDK tools component. You can get this from the Android SDK Manager in Android Studio.
* The Google Play Services SDK. You can get this from the Android SDK Manager in Android Studio.

## **2. Creating an App That Supports Barcodes**

In this step you'll create the basic skeleton of an app that you'll fill in later by adding the sign in code.

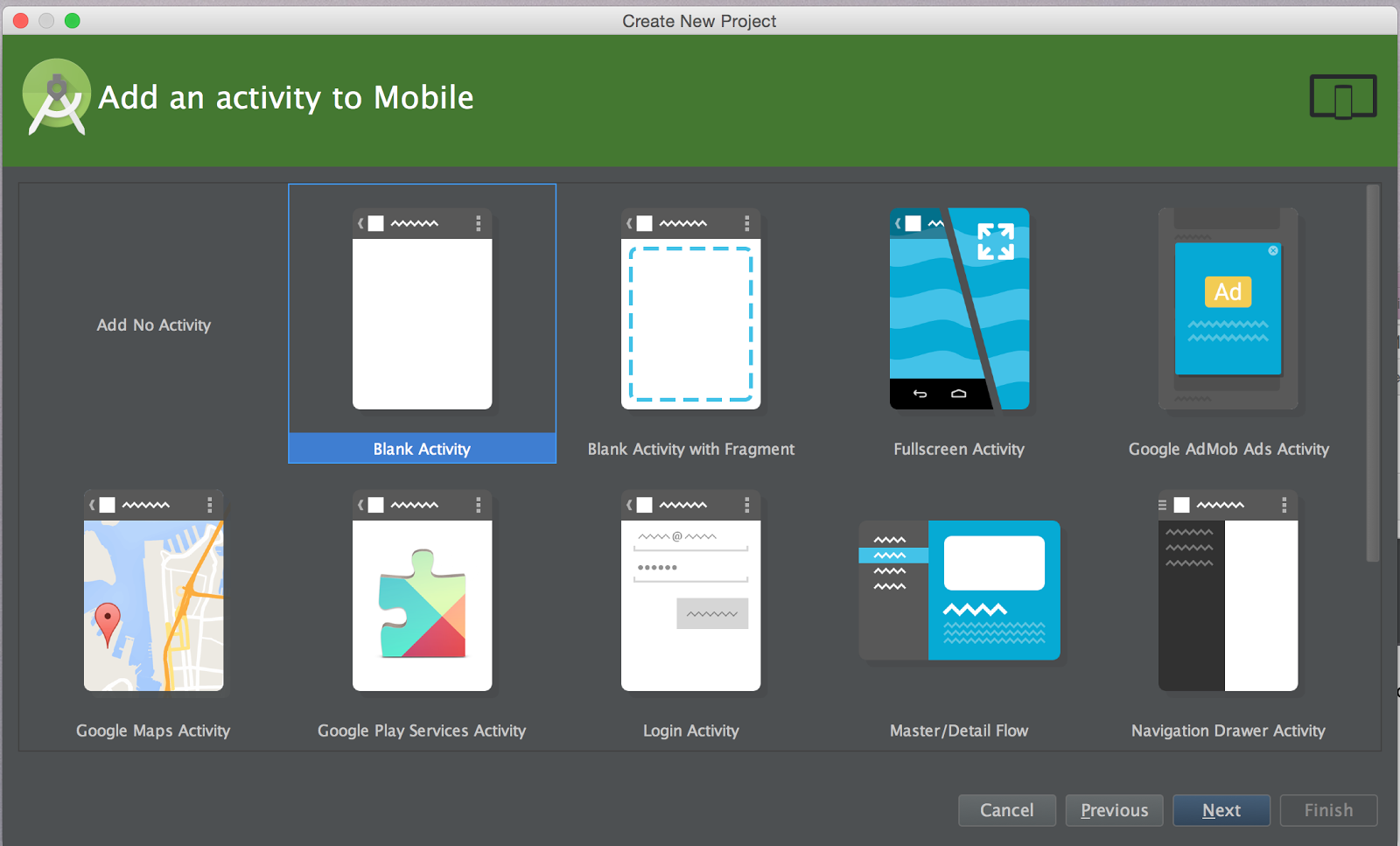
## **Create the new App**

Open **Android Studio**. When you launch Android Studio, you'll see an option to ‘Start a new Android Studio project'. Select this. You'll see the ‘New Project' dialog. Enter the details for your app.



Press Next, and you'll get a screen asking to ‘Target Android Devices'. Accept the default here, and press Next to the next screen.

You'll see ‘Add an activity to Mobile' dialog. **Ensure that you select ‘Blank Activity' here.**

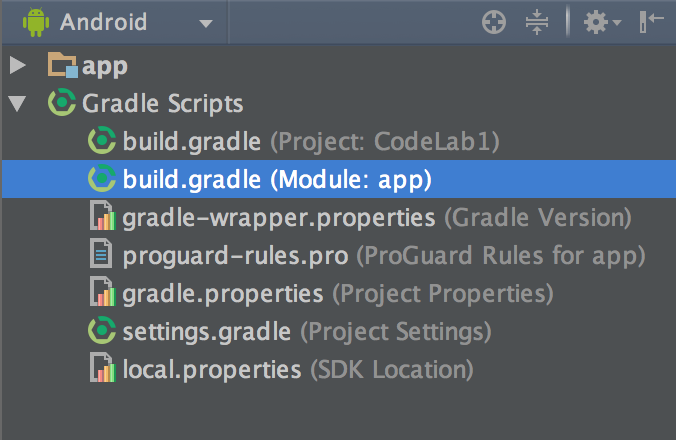


Click ‘Next' and you'll be asked to customize the Activity. Just accept the defaults, and press ‘Finish'.

## **3. Configure Build.gradle**

In this step you'll ensure that your app can use Google Play services, in which the Mobile Vision APIs reside. To do this, you'll first update your build.gradle file.

In Android Studio, open the Gradle Scripts node, and select build.gradle (Module App) as shown:



This will open your build.gradle file, at the bottom of which will be code like this:

dependencies {

compile fileTree(dir: 'libs', include: ['\*.jar'])

}

Add a dependency for play services like this:

dependencies {

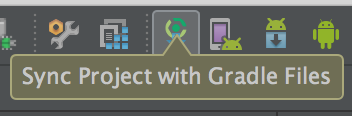
compile fileTree(dir: 'libs', include: ['\*.jar'])

compile 'com.android.support:appcompat-v7:22.0.0'

compile 'com.google.android.gms:play-services:7.8+'

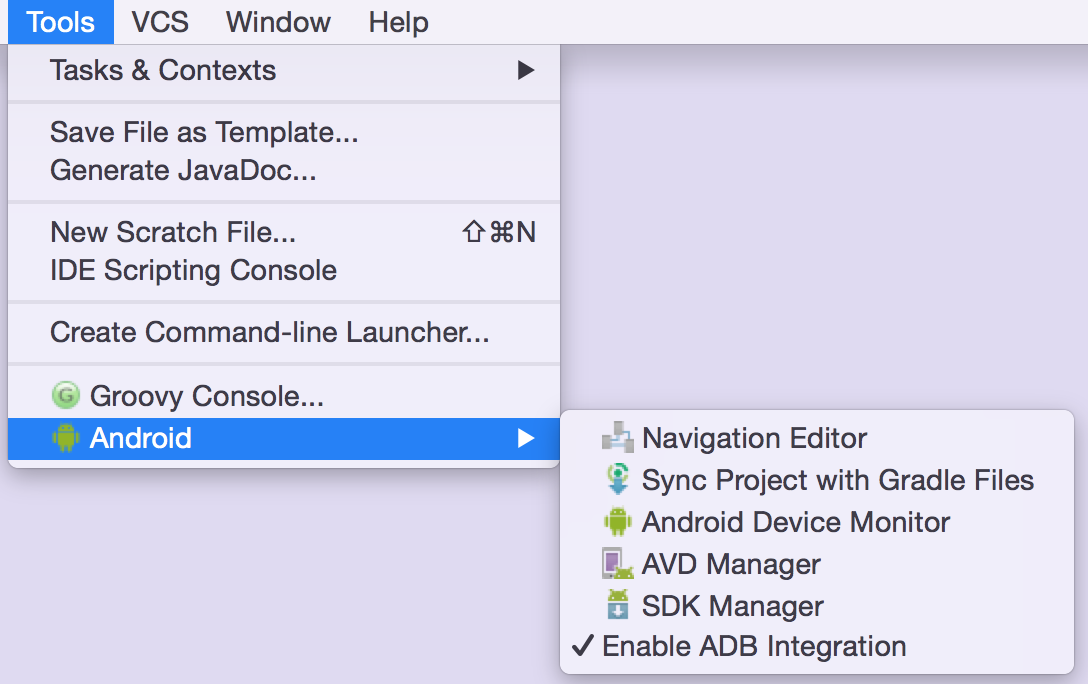
}

If you are asked to perform a gradle sync, do so. Otherwise, find the Gradle Sync button on the toolbar and press it to trigger a sync. It looks like this:

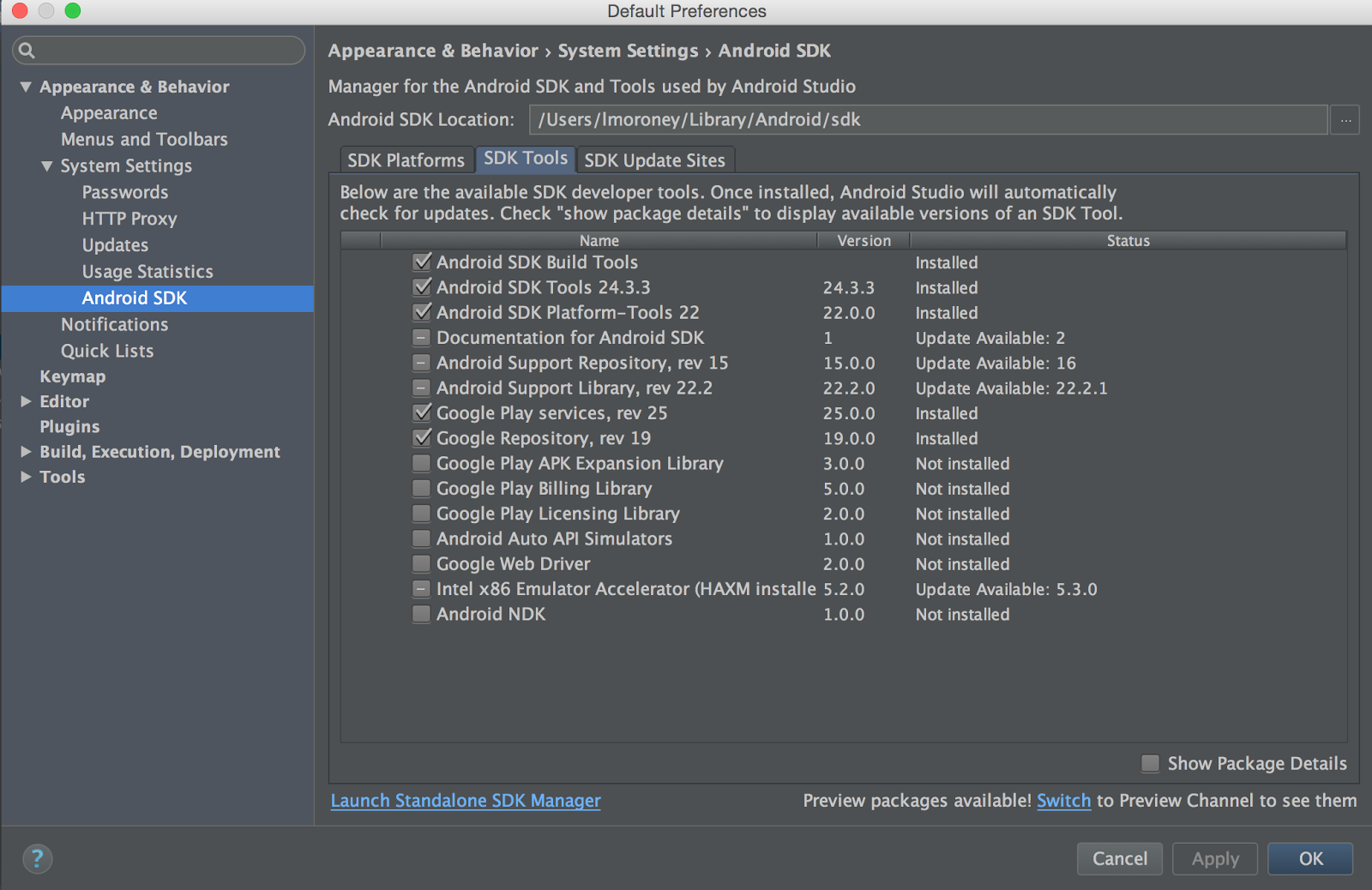


## **4. Update Google Play Services**

Google Play services is frequently updated and this codelab assumes you have a recent version. To get the latest version, in Android Studio click **Tools > Android > SDK Manager**:



Then find the entry for **Google Play Services** and make sure you have version 26 or higher:

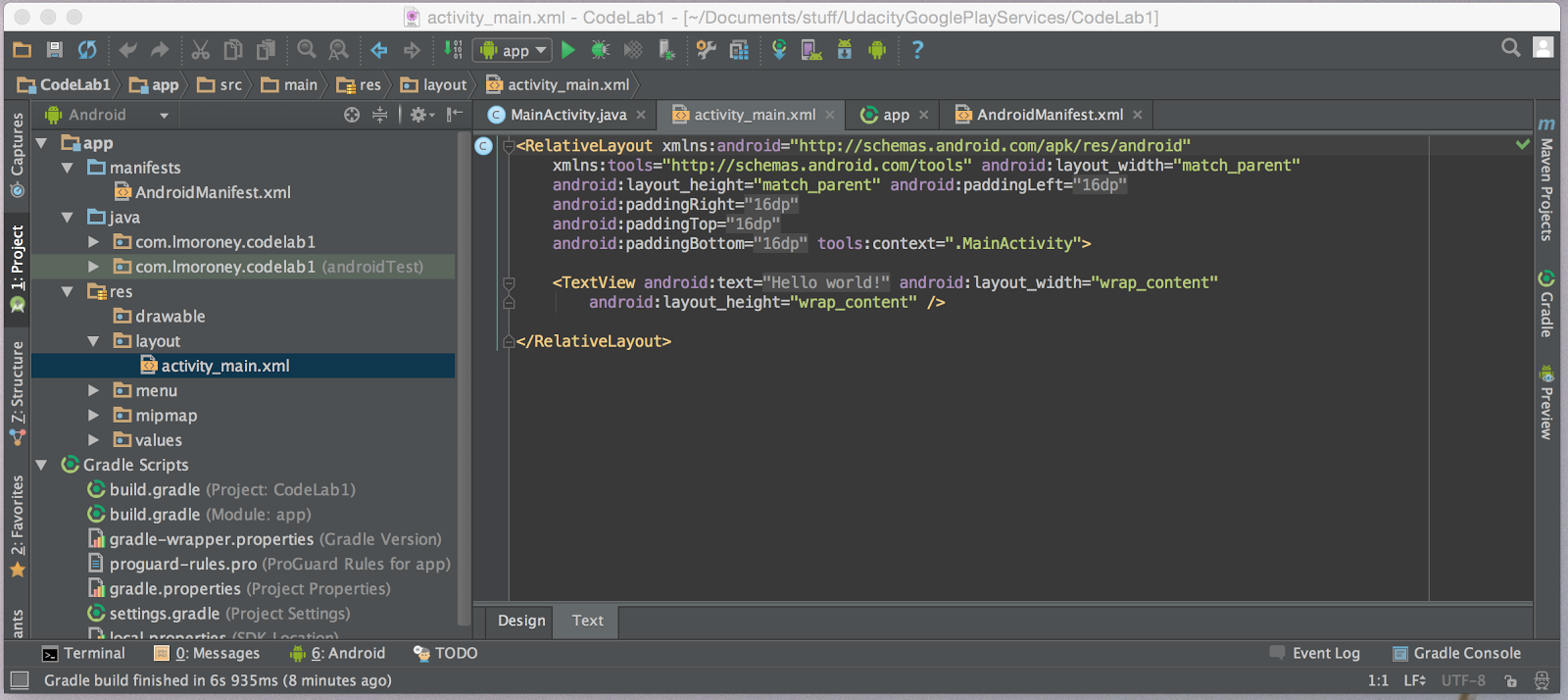


## **5. Adding a UI to your App**

Now that your app is fully configured, it's time to build a UI that lets the user detect a face in an image, and then overlay that face with a bounding box.

In Android Studio, select the ‘**res**' folder, and open its ‘**layout**' subfolder. In here you'll see ‘activity\_main.xml'.

Double click to open it in the editor, and be sure to select the ‘**Text'** tab at the bottom of the editor to get the XML text view of your Layout. Android Studio Should look something like this:



You can see that your layout contains a single <TextView> node. Delete this and replace with:

<LinearLayout

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:orientation="vertical">

<TextView android:text=""

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:id="@+id/txtContent"/>

<Button

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Process"

android:id="@+id/button"

android:layout\_alignParentTop="true"

android:layout\_alignParentStart="true" />

<ImageView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:id="@+id/imgview"/>

</LinearLayout>

This layout gives you a button which will loading and then processing an image, which will appear in the ImageView. After processing is complete, the data from the Barcode will render in the TextView.

## **Adding a Barcode to your app**

Typically you would take pictures of bar codes with the device's camera, or maybe process the camera preview. That takes some coding, and in later steps you'll see a sample that does this. To keep things simple, for this lab, you're just going to process an image that is already present in your app.

Here's the image:



Name it puppy.png, and add it to the res/drawable directory on your file system. You'll see that Android Studio adds it to the drawable directory. It also makes the file accessible as a resource, with the following ID: R.drawable.puppy

With the image in place, you can now begin coding your application.

## **6. Coding your Application**

When you created the app with a single view activity, the template created a menu on the app. You're not going to need or use it, so find the code (in MainActivity) that handles these. These are called **onCreateOptionsMenu** and **onOptionsItemSelected**. Go ahead and delete these.

This application has a single button that will load the image, detect any faces on it, and draw a red rectangle around them when it does. Let's write the code to achieve this:

## **Includes**

In case you need them, here's the full set of includes that this app uses.

import android.app.Activity;

import android.graphics.Bitmap;

import android.graphics.BitmapFactory;

import android.graphics.drawable.BitmapDrawable;

import android.os.Bundle;

import android.util.SparseArray;

import android.view.View;

import android.widget.Button;

import android.widget.ImageView;

import android.widget.TextView;

import com.google.android.gms.vision.Frame;

import com.google.android.gms.vision.barcode.Barcode;

import com.google.android.gms.vision.barcode.BarcodeDetector;

## **Wiring up the Button**

In your MainActivity.java in your onCreate method, add the following code:

Button btn = (Button) findViewById(R.id.button);

btn.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

}

});

This sets up the event handler (onClick) for when the user presses the button. When they do that, we want to load the bar code, process it for data, and write that data to the TextView.

## **Load the Image**

Let's start with loading the image. We do this first by getting a reference to our image view, that we call ‘myImageView'. Then you use a BitMapFactory to decode the R.drawable.puppy resources into a Bitmap. Then you set it to be the bitmap for myImageView.

ImageView myImageView = (ImageView) findViewById(R.id.imgview);

Bitmap myBitmap = BitmapFactory.decodeResource(

getApplicationContext().getResources(),

R.drawable.puppy);

myImageView.setImageBitmap(myBitmap);

## **Setup the Barcode Detector**

Next we'll setup the detector we're going to use to detect a barcode.

We create our new BarcodeDetector using a builder, and tell it to look for QR codes and Data Matrices (there are a lot of other barcode types we could also look for).

It's possible that, the first time our barcode detector runs, Google Play Services won't be ready to process barcodes yet. So we need to check if our detector is operational before we use it. If it isn't, we may have to wait for a download to complete, or let our users know that they need to find an internet connection or clear some space on their device.

BarcodeDetector detector =

new BarcodeDetector.Builder(getApplicationContext())

.setBarcodeFormats(Barcode.DATA\_MATRIX | Barcode.QR\_CODE)

.build();

if(!detector.isOperational()){

txtView.setText("Could not set up the detector!");

return;

}

## **Detect the Barcode**

Now that our detector is set up and we know it's operational, we'll detect the barcode. This code is pretty straightforward -- it creates a frame from the bitmap, and passes it to the detector. This returns a SparseArray of barcodes.

Note that the API is capable of detecting multiple bar codes in the same frame. In this case our entire image is a bar code, but consider what would happen with a camera preview type app where you are looking at multiple bar codes -- in this case the SparseArray<Barcode> would be populated with multiple entries.

Frame frame = new Frame.Builder().setBitmap(myBitmap).build();

SparseArray<Barcode> barcodes = detector.detect(frame);

## **Decode the Barcode**

Typically in this step you would iterate through the SparseArray, and process each bar code independently. Usually, we need to allow for the possibility that there won't be any barcodes, or there might be several. Because for this sample, I know I have 1 and only 1 bar code, I can hard code for it. To do this, I take the Barcode called ‘thisCode' to be the first element in the array. I then assign it's rawValue to the textView -- and that's it -- it's that simple!

Barcode thisCode = barcodes.valueAt(0);

TextView txtView = (TextView) findViewById(R.id.txtContent);

txtView.setText(thisCode.rawValue);

## **7. Results**

Now all you have to do is run the app. So, for example, if you use the puppy.png from earlier, you'll see that the data encoded in the QR Code was some text and a URL as shown here:



In this lab you went end-to-end in creating a simple app that uses the Mobile Vision API to decode a bar code. You did it using a static image, which will give you a start on the principles and how they work. For your next step you can explore the [official sample](https://github.com/googlesamples/android-vision) which takes this to the next level, detecting multiple bar codes and faces and tracking them in the live camera preview.

## **8. Learn More**

* [Watch the Google Play services Dev Byte](https://www.youtube.com/watch?v=nZjY_DPO1qs)
* [Read the Documentation](https://developers.google.com/vision/barcodes-overview)
* [Download the Android Vision API sample](https://github.com/googlesamples/android-vision)