

ECE480 Design Team 6
Application Note Assignment
Developing an Android-based Barcode Scanner
Matt Rasmussen

Introduction

This application note will walk you through the steps of first creating a blank Android application, then it will detail the steps required to successfully recognize a barcode (from a choice of several types) and then send this UPC value to a webserver. The details on the setup of the webserver are to be explained in a separate application note, so it is assumed that you will have one readily available for use.

Background

UPC, also known as Universal Product Code, is used on nearly every food product found at grocery stores in the United States, and they are used by stores to easily identify to the computer the item the user has brought up. This concept was first developed by IBM in the late 1960's, but due to issues with accurately printing readable labels, they were not introduced into stores until the end of 1972. There have been many different types of UPC over the years, and with each one was a different method of detecting them. The most commonly used barcode today, known as UPC-A has the ability to store 12 numerical digits. This means there are about one trillion (10^{12}) total barcodes possible. As many of you are likely familiar, a UPC-A barcode consists of several strips of black bars on a white background. An example is shown below.¹



The code follows the pattern of a pair of bars at the beginning and end, another pair in the exact center, and six digits worth on either side of these center bars. These outside and center bars are consistently the same on every UPC-A barcode, and are how the scanner knows when to

start and stop scanning for values.² Every number is represented by a distinct group of bars, and therefore no number can be represented multiple ways. In addition to this, no barcode can be read in different ways, so each one is distinct and unique. The location of the numbers on the barcode itself do not necessarily line up with the set up bars that represent them. Those are there for a user to error-check and confirm that the code read is correct. This method is used instead of recognizing the numbers themselves because it is much less resource-intensive to follow the widths of the bars instead of identifying each number. Therefore this can be done much faster, and for a fraction of the cost. Our final UPC barcode scanner will be able to detect a wide range of types of UPC as well as QR codes.

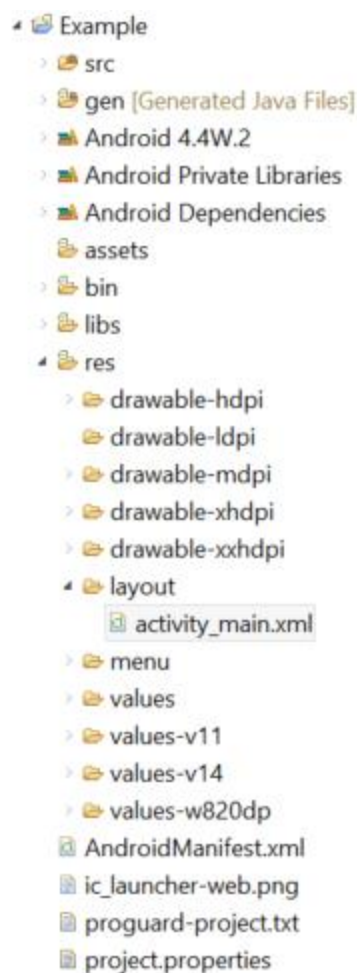
Introduction to Eclipse for Android

For those of you without any experience in developing an application for Android, the place to start would be to download Eclipse with the Android SDK plug-in. The windows copy can be found [here](#). The first thing you will need to do following the installation is to create a new blank project. This can be done by pressing File>New>Android Application Project. Next a box will pop up, and you must fill in your application, project, and package name, followed by the versions of android the app will be compatible with, and targeting. At the time of this paper, the newest version is 4.4.4 (Android Kit Kat) so that is what we will be targeting. You can change any of the options on the next pages to your liking, but we will assume you did not change the name of any of the files created here (activity_main, mainactivity, etc.). Upon completing the

1: Image taken from <http://fashionbrainacademy.com/wp-content/uploads/2014/10/UPC-Code-sample.jpg>

2: http://en.wikipedia.org/wiki/Universal_Product_Code

setup for the project, a box on the left side of the window, similar to the image on the next page should appear. This tutorial will assume you have some familiarity with Java, because if not this paper would take well over the 10 page suggested limit to properly explain the language. The first file that will be opened by default will likely be named `activity_main.xml`. This file represents the starting point of your application, and when it is opened by the user, this activity will be the first thing the user sees. There are two options for viewing this file, both the graphical



layout, and the standard xml layout. The graphical layout allows the programmer to easily see exactly what the activity will look like on a user's phone, while the xml layout is the code that represents that graphical layout. For this particular example, we will only need one activity, as the user will not need multiple screens to scan a barcode. On this activity we will need one button, and four TextViews. The button will be used to initiate our scan, while the TextViews will be used to display to the user the UPC barcode scanned, as well as the type of code (UPC-A, UPC-E, EAN, QR, etc.). The bare

minimum code for this activity is shown below. If you would like to change font, text size, text color, or anything similar that can be added, but it is not present here for simplicity.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">

    <Button
        android:id="@+id/btnScan"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Scan" />

    <TextView
        android:id="@+id/tvStatuslbl"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Status"/>
    <TextView
        android:id="@+id/tvStatus"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>

    <TextView
        android:id="@+id/tvResultlbl"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Result" />

    <TextView
        android:id="@+id/tvResult"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
    />
</LinearLayout>
```

The section marked “Button” represents the scan button, and is identified throughout the code by the ID “btnScan”. The first and third TextView sections are hard-coded to only ever display the words “Status” and “Result”. These will serve as titles to describe the second and fourth TextViews, which will contain the UPC type, and code itself. The lines that describe the height and width are set there by default by Eclipse. The idea that the width “matches parent” simply means that this component will attempt to match the size of display that the layout needs to fill.

While activity_main.xml represents what the user sees, MainActivity.java represents the actual code that is going to happen when the app is run, and when buttons are pressed on screen.

In larger applications, there will be several java files, and you can switch in between them, as well as pass data back and forth through the use of intents. We will use these to “jump” over to a library we must include on our project, but this will be outlined later on. When we first set up this application, we set MainActivity.java to be out “main activity” meaning that it will be the first java file to be run when the app is run. No other one will be run unless explicitly told to somewhere with the code. This is the only one that runs automatically with the app starting. Here is the code for MainActivity.java, with comments (following “//” in green) that explain each section:

```
package com.example.example;

//This section imports files from Android that will be needed in order to
properly run the app
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends Activity {
    TextView tvStatus;
    TextView tvResult;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //Initializes the TextViews and Button for use in this file.
        tvStatus = (TextView) findViewById(R.id.tvStatus);
        tvResult = (TextView) findViewById(R.id.tvResult);
        Button scanBtn = (Button) findViewById(R.id.btnScan);
        scanBtn.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                try {
                    //Accesses ZXing, the library needed for this app.
                    Intent intent = new Intent(

                        "com.google.zxing.client.android.SCAN");
                    intent.putExtra("SCAN_MODE",
                        "QR_CODE_MODE, PRODUCT_MODE");
                    startActivityForResult(intent, 0);
                }
            }
        });
    }
}
```

```
        //Creates a screen popup in case of an error
    } catch (Exception e) {
        e.printStackTrace();
        Toast.makeText(getApplicationContext(),
"ERROR:" + e, 1).show();
    }
    });
}
//Retrieves the results:
public void onActivityResult(int requestCode, int resultCode, Intent
intent) {
    if (requestCode == 0) {
        if (resultCode == RESULT_OK) {
            //Pulls the type and UPC value from ZXing

            tvStatus.setText(intent.getStringExtra("SCAN_RESULT_FORMAT"));

            tvResult.setText(intent.getStringExtra("SCAN_RESULT"));
        } else if (resultCode == RESULT_CANCELED) {
            //Sets the TextViews on Activity_main.xml in case of
an error

            tvStatus.setText("Press a button to start a scan.");
            tvResult.setText("Scan cancelled.");
        }
    }
}
}
```

This code must access a library named ZXing (“Zebra Crossing”) to properly identify the barcodes themselves. This library is available [here](#), as well as credit to the author. This library must be imported into Eclipse as a project, as well as made into a library, then referenced by our example app. It can be imported by pressing File>Import>Android>Existing Android Code Into Workspace. Once added in, you should see it in the package explorer (left side, similar to the picture from a few pages ago). To make sure Eclipse knows this project is a library, right-click on the name of the project (ZXing) then click properties>Android, then check the box marked “Is Library”. Finally, we need to go back into our original project, and tell it that ZXing is a library that we need to access. This is done by accessing properties>Android for your project, then

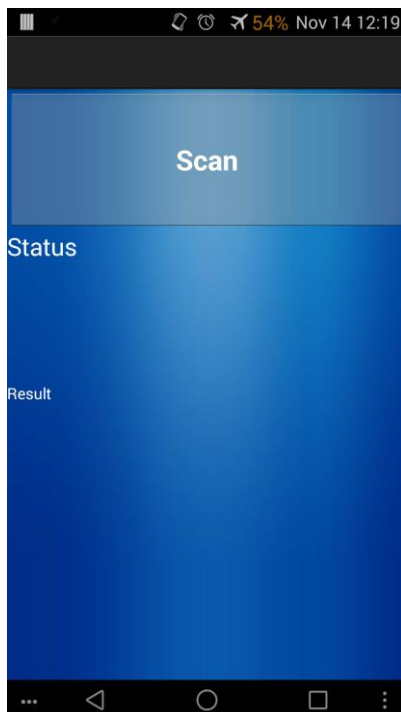
clicking “Add...”, then clicking on “ZXing”, then apply. This will allow your application to access ZXing as needed to complete the scanning of a UPC barcode.

Assuming everything has been correctly outlined and added to the project, it should build automatically, and upon saving the project it will create a file in your “bin” folder in the package explorer named “Example.apk” (or whatever you named your app). This is the android equivalent to an .exe file, and is what will ultimately be run on the phone itself. It can be thought of as a compacted down version of your eclipse project. When it comes to testing your application, you have several options. The first (and likely the easiest) is to turn on debugging mode on your Android phone, then plug it into your computer. To turn on debugging mode, you must first go into your settings>about phone, then click on the “Build number” option seven times. This sounds a little strange, but after seven presses, a toast message will popup saying “You are now a developer!” or something like that, and this will add a new section of settings to the previous paged called “Developer Options”. Inside this section are a number of new options for you to use, one of which is “Debugging mode”. This options are originally hidden from view because there are several advanced options that the user would likely never need to use, so leaving them out in the open could result in problems to the phone caused by an inexperienced user clicking randomly on things they do not know about. Now that you are in Debugging mode, when you press the green play button at the top of Eclipse in it will pop up. Simply select the phone and press OK. This will cause the phone to run your application. If you do not have an Android smartphone, or you wish to test the application across multiple devices (ideal for people who plan on selling an application) you can run an emulated Android phone within windows. There are two main options for this: Either use the built-in emulator within Eclipse, or download a 3rd-party emulator and plug-in. The first option can be done by pressing the play button without

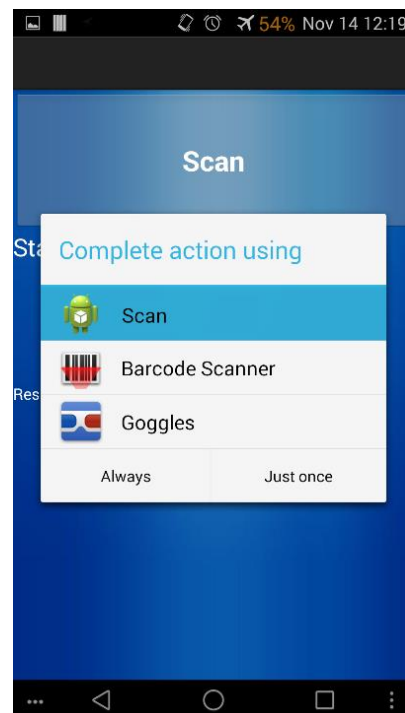
any device connected to the computer. This will give you a popup that will prompt you to set up a virtual device. Configure to this emulator to your liking, and a few minutes after that a popup with your emulated phone's name should appear, then run your application. For many Eclipse users, the built-in emulator simply is too slow and weak to properly emulate their app. In addition to this, it is missing key features like simulated connections, hardware controls like the camera, etc. As an alternative, we can use GenyMotion, which is arguably the most popular 3rd-Party emulator. It is available for download [here](#). This requires a little more setup than the built-in emulator, but it is definitely worth it for its ease of use, and better loading speed.

Now that we have several ways to run our application, we can see it in action. The next few photos will show what the application will look like on a Nexus 5 running Android 4.4.4.

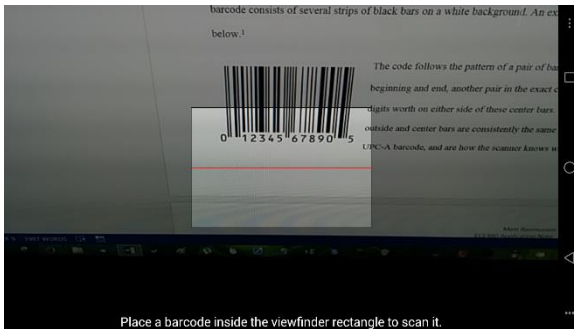
1)



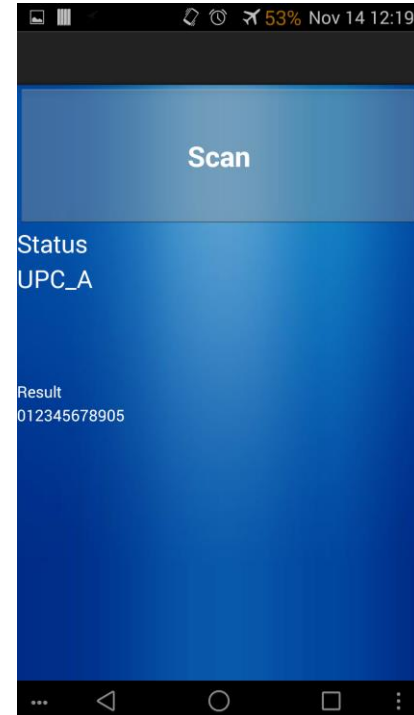
2)



3)



4)



As you can see in picture (1), the application opens up with a Scan button as outlined in our Activity_Main.xml, with four TextViews beneath it. Two of them are blank as there has been no code yet scanned. Upon pressing the scan button we get the option of which application to use for the scanning (2). We select our application name (In this case, the application is appropriately named “Scan”) to prove we are only using one application the whole time. Then, in picture (3), the camera opens up and we have a box appear in which we will place the UPC barcode. In this case we used the code from earlier in the introduction section. Finally, in picture (4) we see our result, which as we know from earlier it is of type UPC-A, and has the number value of “012345678905”. This means we have successfully created an Android-based application that uses the rear-view camera on the phone to scan a UPC generated barcode and print that result.