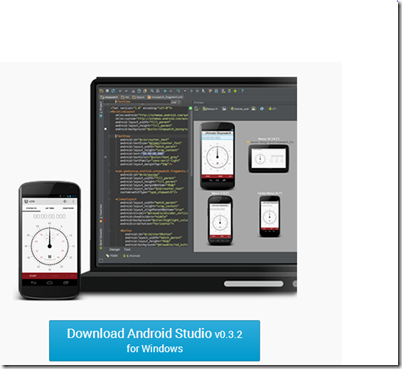
**Creating a Weather App in Android Studio**

In a previous [blog](http://www.todroid.com/android-development-environments/), we summarized different development environments that would allow you to develop android apps for your phone, tablet, and other devices supporting the android OS. In this blog post we will introduce you to one of the newer android development IDE’s, Android Studio.

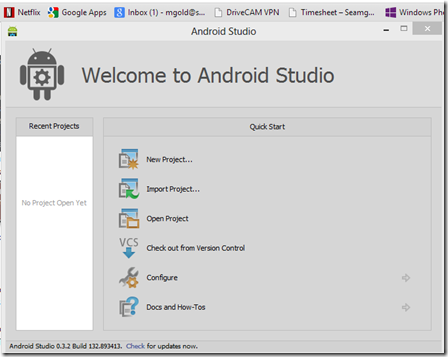
**Installing**

[](http://www.todroid.com/wp-content/uploads/2013/12/image.png)

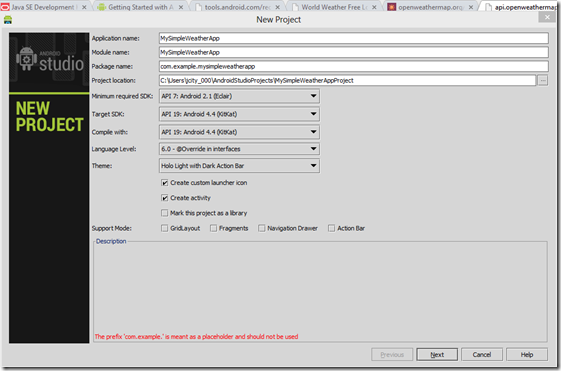
You can get started with Android Studio, but downloading the software [here](http://developer.android.com/sdk/installing/studio.html).  There are installs for Windows, Maxx OS X, and even Linux.  As of this blog post, the android studio platorm for Windows is an early access preview.  Since I’m writing this article on a Windows platform, I’ll go through the Windows installation.  Click on the Download Android Studio button and run the installation.  You’ll need to agree to the terms and conditions for installing this software.  The download is about 462 MB, so if you have a slow connection, be prepared to wait a while. Also keep in mind the Android Studio installation requires that you have Java [JDK](http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html) 1.7 installed on your computer.

**Running Android Studio**

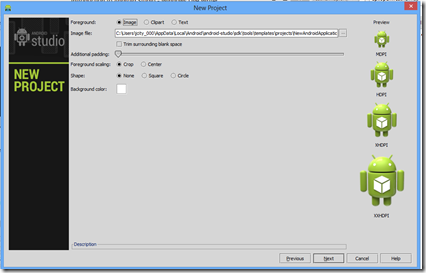
We are now ready to create our first Android App.  In order to give this application a little more practical purpose than your simple “Hello World”, I’ve chosen to step you through creating a simple weather application.  After you install Android Studio, it will launch and prompt you for a new or existing project.  Choose **New Project** from the menu to create a brand new Android project.

[](http://www.todroid.com/wp-content/uploads/2013/12/image1.png)

After we click on new project, we’ll be prompted with an application project template.  In our example we are going to create a weather app, so we’ll name the application MySimpleWeatherApp to get started.  Then we’ll click next:

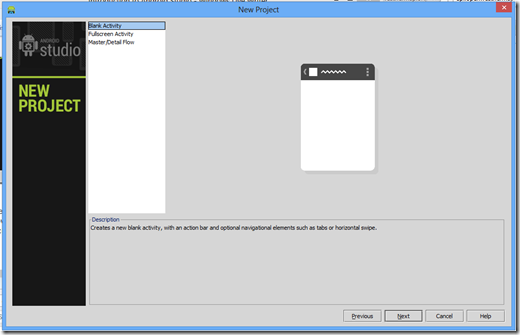
[](http://www.todroid.com/wp-content/uploads/2013/12/image2.png)

The next step will prompt you to choose an image file.

[](http://www.todroid.com/wp-content/uploads/2013/12/image3.png)

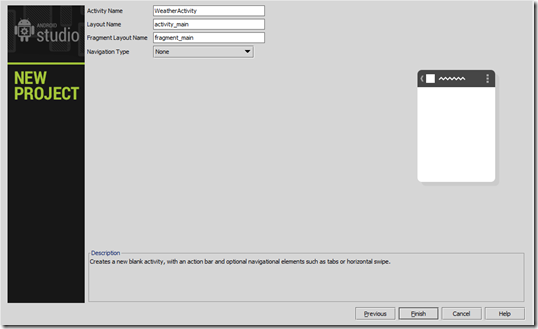
For now, we’ll just use the default image provided by android studio.  In the future, you can change the image files, supplying one file for each required size (MDPI, HDPI, XHDPI, and XXHDPI).

Click next to get to the activity template screen.

[](http://www.todroid.com/wp-content/uploads/2013/12/image4.png)

Android Studio continues to guide you through creating the application and gives you a choice of 3 possible activity templates: a blank template,  a master detail template, or a full screen template.  In our example, choose blank template, as we are creating a very simple weather application to display the current weather and won’t require any intricate navigation.

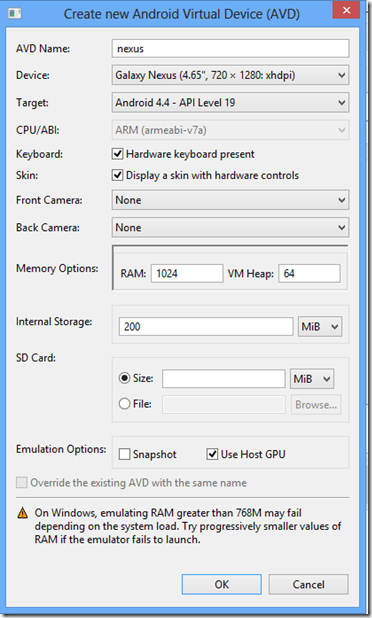
Hitting next brings us to filling in the name of our main activity which we shall call WeatherActivity.  In Android development, an activity can simply be thought of as a view.  Our app has only one view, and that view is the WeatherActivity which we will type into our template.

[](http://www.todroid.com/wp-content/uploads/2013/12/image5.png)

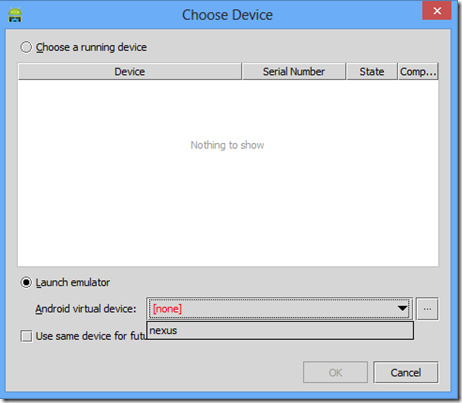
Finally we can hit the finish button and have Android Studio generate our project with the desired wizard choices.  Android Studio will precede to use a java utility called Maven to download the necessary java libraries for the project we just created.

**Building and Running**

We can actually build and run the template we just created so far.  Choose Build –> Make from the file menu to build the project.  There should be no errors.  We can now choose Run from the Run menu and choose an emulator.  Initially there will be no emulator to choose from, so you will need to create one.  Click the ellipsis …  and choose New.  Fill out the emulation profile as shown below:

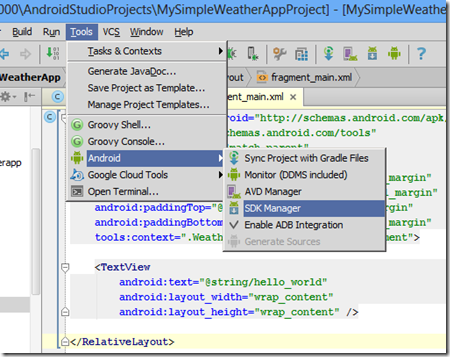
[](http://www.todroid.com/wp-content/uploads/2013/12/image6.png)

After clicking okay, you can select the new emulator in the Choose Device Dialog:

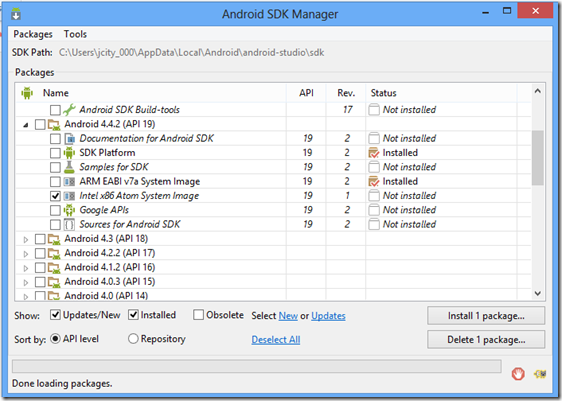
[](http://www.todroid.com/wp-content/uploads/2013/12/image7.png)

**Installing the Intel Emulator**

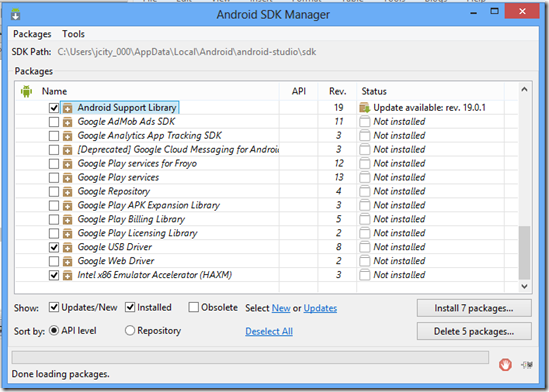
I actually prefer using the Intel Emulator, because its much faster. The Arm Emulator is painfully slow to develop with.  Here are the steps. First go the Tools->Android->SDK Manager in the Android Studio Menu:

[](http://www.todroid.com/wp-content/uploads/2013/12/image8.png)

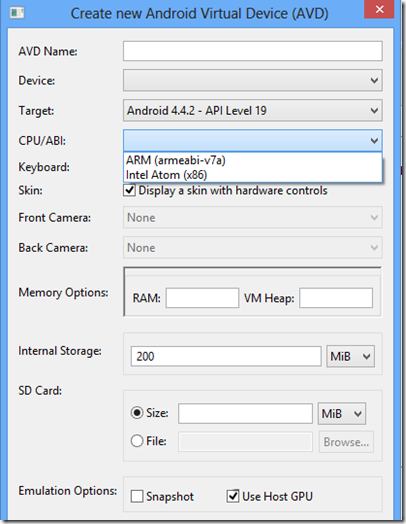
This will bring up the SDK Manager that allows you to not only manage Android SDK’s (and there are many versions!) but also allows you to manager emulator and device connectivity.  Go to the particular SDK version we are developing against.  In this case, its version 4.4.2.  Under this category we have the various emulators.  We want to install the Intel x86 Atom emulator image, so check the matching checkbox shown below:

[](http://www.todroid.com/wp-content/uploads/2013/12/image9.png)

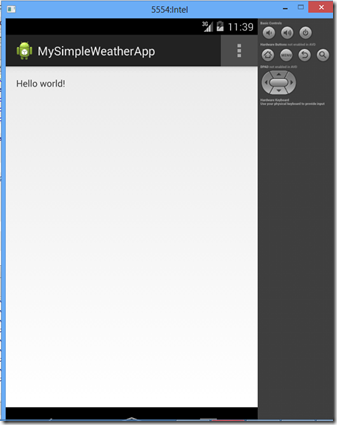
The very last section of the Extras Category contains an Intel Emulator Accelerator (HAXM).  Check this check box as well and click Install.

[](http://www.todroid.com/wp-content/uploads/2013/12/image10.png)

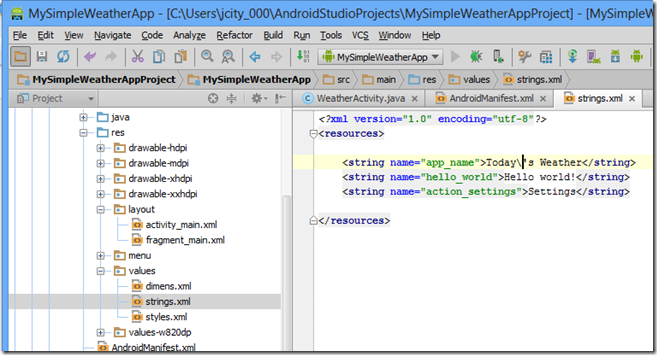
Once this step is done, we can create an emulator that uses the intel as a CPU.  If you are using an intel laptop or desktop to run Android Studio, this is magnitudes faster than the ARM emulator.  Now when you are prompted for an emulator, choose intel as the CPU and create a new virtual device that utilizes the (x86) image.  Be sure to check **Use Host GPU**, to get better performance from the emulator.  Also make sure you set your RAM less than 768.  More than 768 may cause the Intel Emulator to fail in the Windows environment.

[](http://www.todroid.com/wp-content/uploads/2013/12/image11.png)

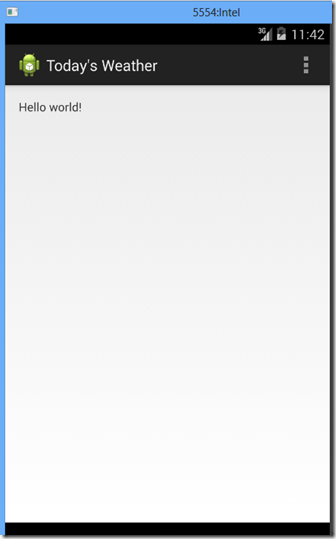
Once the emulator is set up, you can run the app.   When  you start the app from Android Studio, and choose Run-> Run MySampleWeatherApp, Studio will prompt you for a device. You should be able to start your newly created Intel Virtual Device if all has gone well. If it doesn’t run properly, start up the emulator image first and make sure its fully launched.  Then run the app again from the Studio menu and pick the already loaded emulator image as your device.  You can continue to use the loaded emulator, so don’t close it while you are developing and adding code.  Below is the results of what we have created so far:

[](http://www.todroid.com/wp-content/uploads/2013/12/image12.png)

Android Studio has already created our main activity for us and a simple Hello World! text message.  We are going to alter this slightly to  get our weather information for the day.  First lets fix the title of the app.  The main title can be found in the strings resource file in the  project, strings.xml.  We will need to alter the title from **MySimpleWeatherApp** to something less kludgy such as **Today’s Weather.**

[](http://www.todroid.com/wp-content/uploads/2013/12/image13.png)

Now when we rerun from Android Studio, we see a nicer title at the top of our activity:

[](http://www.todroid.com/wp-content/uploads/2013/12/image14.png)

**Displaying Today’s Weather**

In order to get weather data, we are going to use a freely available open source weather service.  In our example, we will make it easy, and retrieve only the weather for austin, texas.  If you put the following url in your browser, it will spit out a JSON object containing Austin weather data for the  day:

<http://api.openweathermap.org/data/2.5/weather?q=austin,tx>

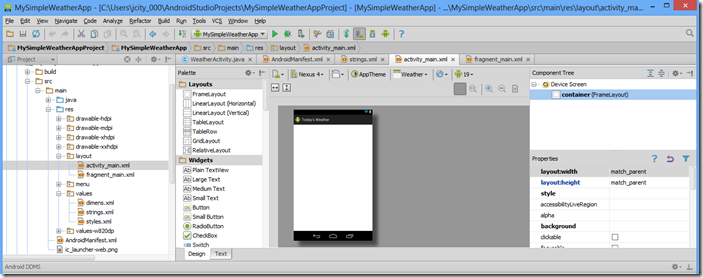
This URL returns the following JSON data as an example:

{"coord":{"lon":-97.74,"lat":30.27},"sys":{"message":0.2297,"country":"United States of America","sunrise":1388064350,"sunset":1388101090},"weather":[{"id":804,"main":"Clouds","description":"overcast clouds","icon":"04n"}],"base":"gdps stations","main":{"temp":281.79,"humidity":72,"pressure":1029,"temp\_min":280.37,"temp\_max":283.15},"wind":{"speed":3.96,"deg":351.5},"rain":{"3h":0},"clouds":{"all":92},"dt":1388033870,"id":4671654,"name":"Austin","cod":200}

We are going to use this weather information to create our Weather in Austin for the Day App.  To keep everything fairly simple for this blog post, we will display the following field:  temperature, humidity, pressure, and description.

**Laying out the Fields**

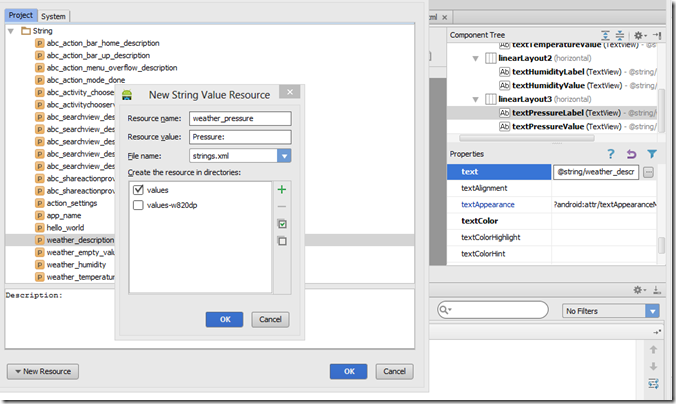
If we open up the activity\_main.xml file, we can see two tabs at the bottom: **Design** and **Text**.  If we choose Design, a Design editor pops up and allows us to start adding layout widgets and controls to the application:

[](http://www.todroid.com/wp-content/uploads/2013/12/image15.png)

I’m going to use a Vertical Linear Layout at the top level, and a Horizontal linear layout for each label/textbox pair.  I’m not going to go into the details of Linear Layout here, but I will say that Studio makes it very easy to layout the Activity by dragging and dropping the Linear Layouts onto the design view.  After you drop the first layout pair of label and value, you can simply copy and paste the Horizontal Layout Group 3 additional times inside the component tree and alter the labels and ids of the TextView widgets. Note that we are using the same exact widget (TextView) for both the label and value pair since both are non-editable text fields displayed to the user.

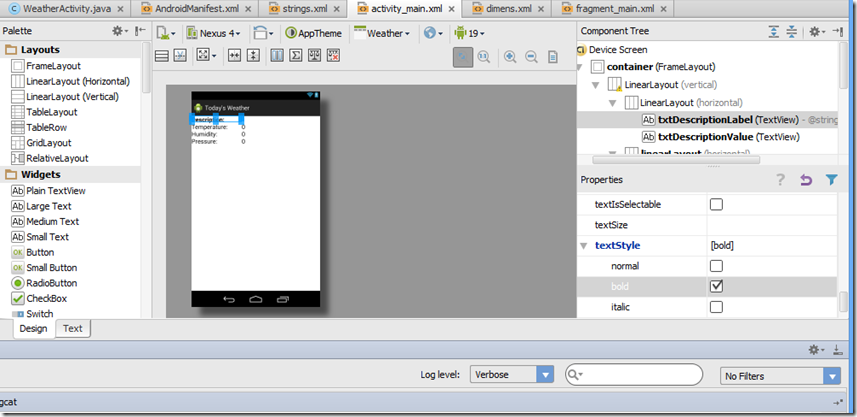
**Adding String Resources**

For all of the labels in our app, we need to add strings to strings.xml to represent the label names.  If we go to the property window of the TextView and choose the text property, it gives us an … button to press that allows us to enter a new resource into the system and assigns the resource to our textView control.  For example, below is the case where we create a new label called Pressure as a resource.  Android Studio automatically puts the resource inside the strings.xml file and adjusts the resource properly inside of the widget.  Also, any id we assign to a widget, is automatically generated in the proper places in the resource file by Studio, so there is less for the programmer to worry about when doing these tedious tasks.

[](http://www.todroid.com/wp-content/uploads/2013/12/image16.png)

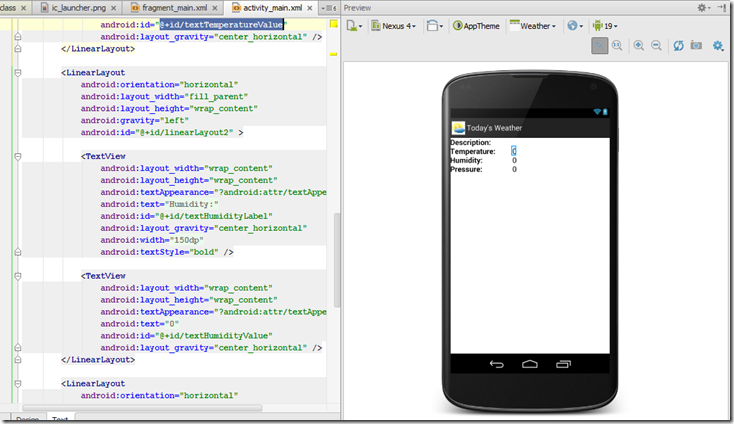
**Styling Labels**

It would be nice to style the labels for each of the weather fields.  In the case of the weather app, we would like to make the labels a fixed width and stand out a bit by making them bold.  To make the labels bold, we simply check the textStyle in the label:

[](http://www.todroid.com/wp-content/uploads/2013/12/image17.png)

To fix the width, we need to change the width field of the labels to a fixed value.  For this example, we set each field width to 150 dp (dp stand for density-independent pixels).  Setting the width of the labels all to 150dp has the added benefit of justifying all the weather values for each field at the same point horizontally.

Here is what are design view looks like with all the widgets laid out to display our weather information.  Notice the xml on the left hand side represents the layout compiled by Android Studio and the visual design is rendered on the right hand side of the IDE.

[](http://www.todroid.com/wp-content/uploads/2013/12/image18.png)

**Adding the Code**

Now that we have added every visual component we need for our activity, we need to do some java programming to get the values to show up inside the widgets.  To do this, we need to accomplish the following

1.  Call the URL service that will return our weather information from Java

2.  Parse out the data we need from the JSON object that is returned (description, temperature, humidity, pressure)

3.  Convert any units that need converting. (For example, temperature is returned in Kelvin and needs to be converted to Fahrenheit.)

4.  We need to populate the  TextView widgets in the activity with the retrieved weather data.

**Calling the Weather Service from a Task**

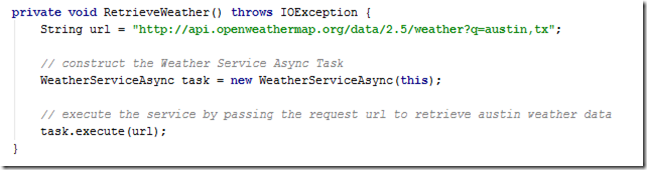
The visual information displayed in our activity such as the labels and weather values all run inside a process called the main thread.  When we retrieve our data for the weather over the internet, we do not want to interfere with this process, or it can cause problems with how the weather data is displayed to the user.  To avoid running in the main thread, we need to spawn another process that will retrieve the weather information.  In order to accomplish this, we will use what is called an AsyncTask.  The AsyncTask will run separately, and when the data is returned, it will populate the data inside the activity.  The activity code does not need to wait for the data to return, it will receive the data when the AsyncTask is ready to populate it.  Below is the code for retrieving the weather data from the internet.  The WeatherServiceAsync extends the class AsyncTask to create a aysnchronous task specific to retrieving weather data.  The constructor call WeatherServiceAsync takes an activity as a parameter allowing us to keep the activity in memory to later populate when the response returns from the http get call. The doinbackground method in our AsyncTask receives the url we pass in from the execute command which is called inside our activity.  The url is used to make a request over the internet using the DefaultHttpClient class.  The url is wrapped in an HttpGet request object and passed into the execute method of the DefaultHttpClient.  The response is returned in the execute call as a stream, and the stream data is read into a memory buffer and turned into a string.  The string returned is the JSON string described before as shown below.

{"coord":{"lon":-97.74,"lat":30.27},"sys":{"message":0.2297,"country":"United States of America","sunrise":1388064350,"sunset":1388101090},"weather":[{"id":804,"main":"Clouds","description":"overcast clouds","icon":"04n"}],"base":"gdps stations","main":{"temp":281.79,"humidity":72,"pressure":1029,"temp\_min":280.37,"temp\_max":283.15},"wind":{"speed":3.96,"deg":351.5},"rain":{"3h":0},"clouds":{"all":92},"dt":1388033870,"id":4671654,"name":"Austin","cod":200}

Here is the java class for WeatherServiceAsync.

|  |
| --- |
| package com.example.mysimpleweatherapp;  import android.os.AsyncTask;  import org.apache.http.HttpResponse; import org.apache.http.client.methods.HttpGet; import org.apache.http.impl.client.DefaultHttpClient; import org.json.JSONException; import org.json.JSONObject;  import java.io.BufferedReader; import java.io.InputStream; import java.io.InputStreamReader;  /\*\* \* Created by jcity\_000 on 12/27/13. \*/  public class WeatherServiceAsync extends AsyncTask<String, Void, String> { private final WeatherActivity WeatherActivity;  *// this constructor takes the activity as the parameter. // that way we can use the activity later to populate the weather value fields // on the screen*      public WeatherServiceAsync(WeatherActivity weatherActivity) { this.WeatherActivity = weatherActivity; }  @Override protected String doInBackground(String… urls) {  *// this weather service method will be called after the service executes. // it will run as a seperate process, and will populate the activity in the onPostExecute // method below*  String response = “”; // loop through the urls (there should only be one!) and call an http Get using the URL passed // to this service  for (String url : urls) { DefaultHttpClient client = new DefaultHttpClient(); HttpGet httpGet = new HttpGet(url); try {  *// make the http request for the weather data* HttpResponse execute = client.execute(httpGet);  // get the content of the result returned when the response comes back // it should be a json object InputStream content = execute.getEntity().getContent();  BufferedReader buffer = new BufferedReader(new InputStreamReader(content)); String s = “”;  // populate the response string which will be passed later into the post execution while ((s = buffer.readLine()) != null) { response += s; }  } catch (Exception e) { e.printStackTrace(); } } return response; }  @Override protected void onPostExecute(String result) { String test = result; try { *// parse the json result returned from the service* JSONObject jsonResult = new JSONObject(test);  *// parse out the temperature from the JSON result* double temperature = jsonResult.getJSONObject(“main”).getDouble(“temp”); temperature = ConvertTemperatureToFarenheit(temperature);  *// parse out the pressure from the JSON Result*             double pressure = jsonResult.getJSONObject(“main”).getDouble(“pressure”);  *// parse out the humidity from the JSON result*             double humidity = jsonResult.getJSONObject(“main”).getDouble(“humidity”);  *// parse out the description from the JSON result* String description = jsonResult.getJSONArray(“weather”).getJSONObject(0).getString(“description”);  *// set all the fields in the activity from the parsed JSON* this.WeatherActivity.SetDescription(description); this.WeatherActivity.SetTemperature(temperature); this.WeatherActivity.SetPressure(pressure); this.WeatherActivity.SetHumidity(humidity); } catch (JSONException e) { e.printStackTrace(); } }  private double ConvertTemperatureToFarenheit(double temperature) { return (temperature – 273)\* (9/5) + 32; } } |

The execute call inside the WeatherActivity that kicks off the asynchronous request for the weather data is shown below.  The RetrieveWeather call constructs the Weather Service Asynchronous Task and executes it with the URL for retrieving weather data from Austin.   The RetrieveWeather data method is called from within the onCreate method of the WeatherActivity so its called as soon as the activity is available.

[](http://www.todroid.com/wp-content/uploads/2013/12/image19.png)

**Parsing out the JSON Response Weather Data**

When the asynchronous task is finished and ready to bring the data into the activity, onPostExecute is called.  This method parses out the JSON data returned in the response and sticks it in the activity using methods that we created in the activity (such as SetPressure, SetTemperature) for populating the screen.  Below is the SetTemperature method inside the WeatherActivity:

|  |
| --- |
| [image](http://www.todroid.com/wp-content/uploads/2013/12/image20.png) |

The SetTemperature method retrieves the temperature value TextView widget from the activity, formats the temperature into a 2 digit precision string, and populates the TextView with the temperature value.

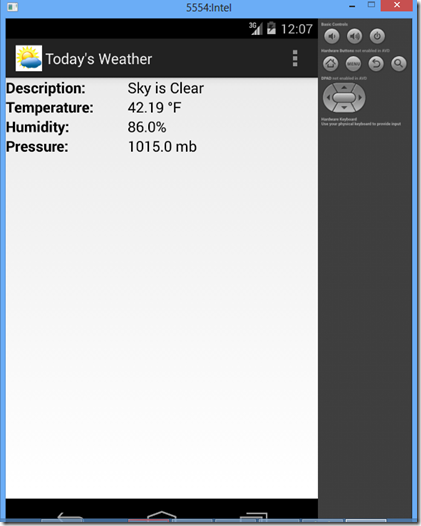
**Allowing Internet Access**

There is one more step to getting the data over the internet for our activity.  We need to give the application permission to use internet access.  The permissions for Android Activity are added to the Android Manifest (AndroidManifest.xml).  We need to add a permission clause to the manifest to allow the weather application to use the internet.  The listing below shows the manifest with the permission added to the manifest.

|  |
| --- |
| <?xml version=”1.0″ encoding=”utf-8″?> <manifest xmlns:android=”http://schemas.android.com/apk/res/android” package=”com.example.mysimpleweatherapp” android:versionCode=”1″ android:versionName=”1.0″ >  <uses-sdk android:minSdkVersion=”7″ android:targetSdkVersion=”19″ /> **<uses-permission android:name=”android.permission.INTERNET” />**     <application android:allowBackup=”true” android:icon=”@drawable/ic\_launcher” android:label=”@string/app\_name” android:theme=”@style/AppTheme” >  <activity android:name=”com.example.mysimpleweatherapp.WeatherActivity” android:label=”@string/app\_name” > <intent-filter> <action android:name=”android.intent.action.MAIN” />  <category android:name=”android.intent.category.LAUNCHER” /> </intent-filter> </activity> </application>  </manifest> |

**Running the Application**

If we did everything correctly, all our weather values should appear next to their corresponding labels as soon as we run the application.  Build the application and choose Run from the Run menu in Android Studio.  The emulator screen should show you the current weather in Austin.

[](http://www.todroid.com/wp-content/uploads/2013/12/image21.png)

**Debugging the Weather App on Your Android Device**

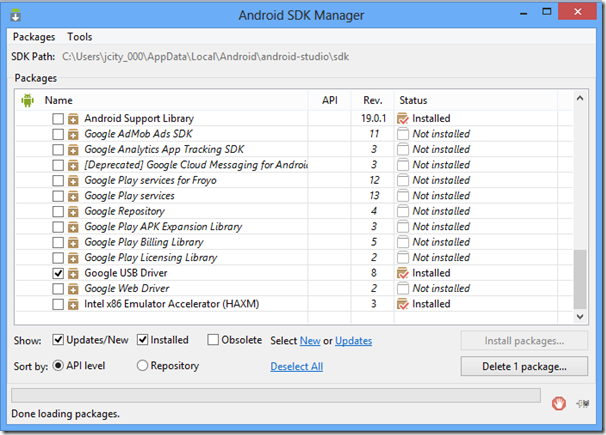
In order to debug the weather app on your device  instead of the emulator, you’ll need to make sure

1) Your device is set up to allow debugging

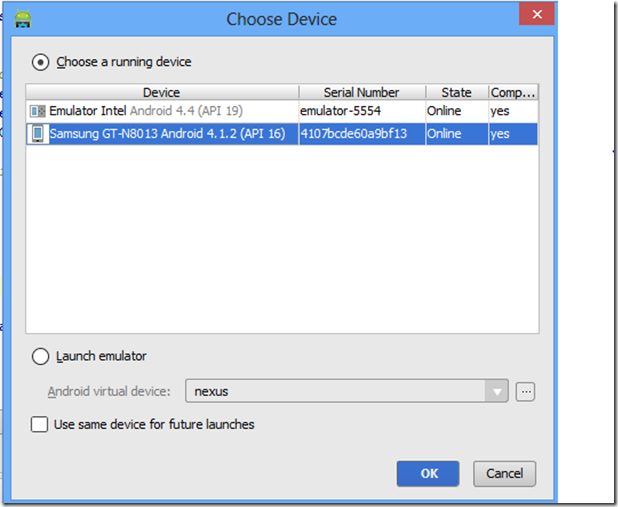
2) You have installed the USB Device Driver on your PC.

In order to setup your Android Device for debugging, go to Settings –> Developer Options on the Android.  Select USB debugging. This should make your android device available to the debugger through the USB connection.

To install the USB Device Driver, go to Tools –> Android –> SDK Manager in Android Studio.  Select Google USB Driver under the Extras Category and choose install if not already installed.

[](http://www.todroid.com/wp-content/uploads/2013/12/image22.png)

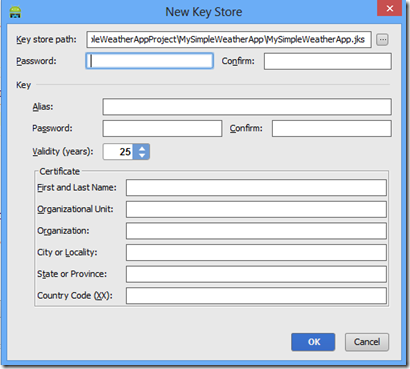
Connect your Android device to the PC through a USB cable and start the debugger.  This should give you a choice of whether you want to debug on the emulator or your device.  Select the device and click okay:

[](http://www.todroid.com/wp-content/uploads/2013/12/image23.png)

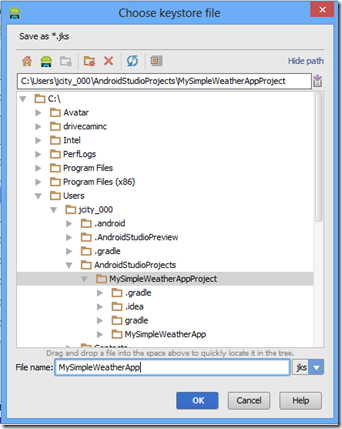
If you did everything correctly, the weather application is now running in the debugger on your Android device.

**Generating a Signed APK file for Application Deployment**

In order to deploy your application to the Android Device, you’ll want to create a signed APK file of your app.  In Android Studio, go to Build –> Generate Signed APK… in the menu.  Click on the ellipsis and Choose Create New… to create a new key.

[](http://www.todroid.com/wp-content/uploads/2013/12/image24.png)

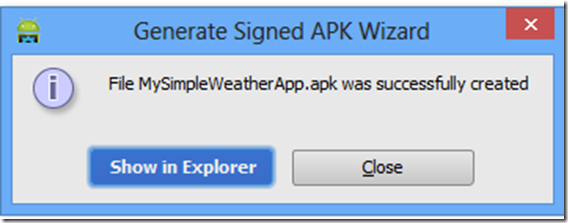
Choose a path to save the key and type in a filename for the key and click OK.

[](http://www.todroid.com/wp-content/uploads/2013/12/image25.png)

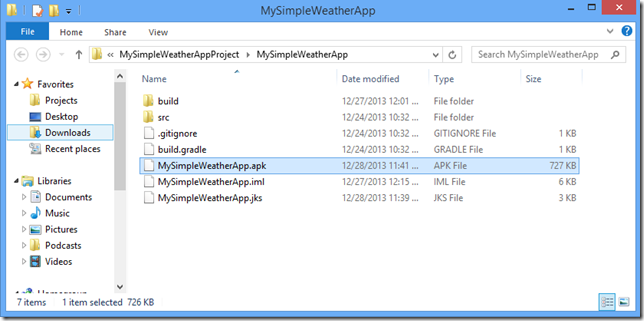
Fill in the remaining fields and click Next, type in the password and click Finish to finish generating the APK file.

[](http://www.todroid.com/wp-content/uploads/2013/12/image26.png)

When the APK wizard is finished generating the APK, Android Studio will prompt you to open the location of the signed apk file on your PC in Window’s Explorer

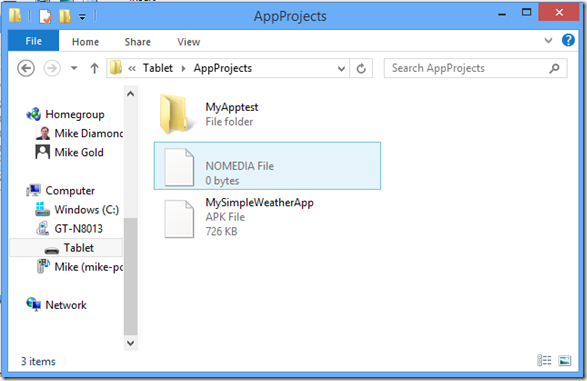
[](http://www.todroid.com/wp-content/uploads/2013/12/image27.png)

You can now open the file explorer and locate the signed APK file of your weather application which you are now ready to deploy to your android device.

[](http://www.todroid.com/wp-content/uploads/2013/12/image28.png)

**Deploying to the Android**

When you plug in your USB connection from the PC into the Android, it should install drivers that allow you to treat the Android like a hard drive and copy your Android apk file from the PC over to the Android device using the File Manager.  Drag the MySimpleWeatherApp.apk onto your android device into a folder you wish to install from.

[](http://www.todroid.com/wp-content/uploads/2013/12/image29.png)

In order to install the file, you are going to need to install an APK installer app onto your Android.  I used an app called **File Manager** that lets you navigate to the app in an explorer-like folder tree and click on the apk app and installs the app.  Another option is to use an App called **Easy Installer**, which locates the apk files on your android device and prompts you for installation.

Note: If you already deployed the weather app using the debugger in the previous section, you will need to uninstall the app through the Application Manager on your device before deploying through the signed APK file.

**Conclusion**

Android Studio gives you all the tools you need to develop an Android application in Java.  In this blog, we went through the process of using Android Studio to layout, code, and deploy a weather application to run on an Android Device.  The application retrieved weather data over a free internet service and displayed it inside the Android Device.  Android Studio has several nice features that makes creating this application a breeze, such as autocompleting code, automatically bringing in the libraries needed for retrieving data from the internet and allowing us to visually layout our Android Widgets.  Even though Android Studio is in prerelease, it is looking very promising as being the defacto tool for Android development.