Arduino Android Blueprints

Get the best out of Arduino by interfacing it with Android to create engaging interactive projects

Marco Schwartz
Stefan Buttigieg
In this package, you will find:

- The authors biography
- A preview chapter from the book, Chapter 3 "Bluetooth Weather Station"
- A synopsis of the book’s content
- More information on *Arduino Android Blueprints*

**About the Authors**

**Marco Schwartz** is an electrical engineer, entrepreneur, and blogger. He has a Master's degree in Electrical Engineering and Computer Science from Supélec in France and a Master's degree in Micro Engineering from the EPFL in Switzerland.

He has more than 5 years of experience working in the domain of electrical engineering. His interests gravitate around electronics, home automation, the Arduino and Raspberry Pi platforms, open source hardware projects, and 3D printing.

He runs several websites around Arduino, including the Open Home Automation website, which is dedicated to building home automation systems using open source hardware.

He has written another book on home automation and Arduino, called *Home Automation with Arduino* and another book on how to build the Internet of Things projects with Arduino, called *Internet of Things with the Arduino Yún*, Packt Publishing.

**Stefan Buttigieg** is a medical doctor, mobile developer, and entrepreneur. He graduated as a Doctor of Medicine and Surgery at the University of Malta, and he is currently enrolled at the University of Sheffield where he is undertaking a Master's degree in Health Informatics.

He has more than 5 years of experience working in various technical positions in international and local student organizations, and has founded MD Geeks, an online community that brings health professionals, developers, and entrepreneurs together from around the world to share their passion for the intersection of healthcare and information technology.
His main interests are in mobile development, specifically, Android and iOS, open source healthcare projects, user interface design, mobile user experience, and project management.

I would like to thank Angelika Biernacka-Buttigieg, my wife, for her unconditional support and patience throughout the creation of this book.

My parents, Joseph Buttigieg and Anne Buttigieg, for their support for this book project.

Christopher Svanefalk for his amazing insight and patience with code reviews and support.

Don Coleman, for his invaluable help in Chapter 8, Control an Arduino Board via NFC. We recommend his expertise on near field communication technologies.
Arduino Android Blueprints

When directly comparing Arduino and Android, one can see that they are two incredibly different platforms with different targets. Arduino is mostly focused on connecting physical everyday objects to embedded microcontrollers. On the other hand, Android intends to provide the necessary operating system and framework to operate countless smartphones around the world.

This reality also reflects the contrasting realities of the authors, who come from very different backgrounds and cultures; Stefan hails from the Island of Malta, and he brings with him a medical background and passion for the intersection of technology and medicine, whereas Marco originates from France and has an electrical engineering background.

The power of combining the efforts of Arduino and Android platforms bring about incredibly implemented practical projects that enhance daily life. Keeping this motivation in mind is what brought two authors from contrasting backgrounds together to work on this book. We believe in the intersection of technology and real life and visualize a future where technology will keep on forming an integral part of our day-to-day life.

What This Book Covers

Chapter 1, Setting Up Your Workspace, covers the necessary steps that you will have to take in order to build all the projects of the book. You will learn how to set up the Android development environment. We will also build our first Arduino project.

Chapter 2, Controlling an Arduino Board via Bluetooth, teaches us how to link Arduino and Android for the first time. We will build an Arduino system with a Bluetooth Low Energy module, and control a simple LED from an Android application.

Chapter 3, Bluetooth Weather Station, teaches us how to build our first useful application using Arduino and Android. We will build a weather measurement station, and visualize the measurements via an Android application, which we will build from scratch.

Chapter 4, Wi-Fi Smart Power Plug, teaches us how to build a DIY version of a popular device: a wireless power switch. We will use an Android application to communicate with the switch via Wi-Fi, control it, and measure the energy consumption of the connected device.

Chapter 5, Wi-Fi Remote Security Camera, introduces a powerful Arduino board, the Arduino Yún, to build a DIY wireless security camera. We will also build an Android application to monitor this camera remotely from an Android phone.
Chapter 6, Android Phone Sensor, explains how to turn things around, and use the sensors from our phone to control the Arduino board. Applying this, we will use the gyroscope of the phone to control the angle of a servomotor.

Chapter 7, Voice-activated Arduino, teaches us how to use the powerful Android speech API to control an Arduino board via Bluetooth.

Chapter 8, Control an Arduino Board via NFC, shows how to use the NFC chip present in many Android phones to activate a relay connected to an Arduino board.

Chapter 9, Bluetooth Low Energy Mobile Robot, uses everything we learned so far in the book to build a mobile robot based on the Arduino. The robot will be controlled via Wi-Fi from an Android application.

Chapter 10, Pulse Rate Sensor, is dedicated to a medical application that measures the heart rate. We will connect a heart rate sensor to Arduino and monitor the measurements via Bluetooth Low Energy.
Bluetooth Weather Station

In this chapter, we will build the first complete application of this book using Arduino and Android. We will build a small weather station using Arduino, which will be accessed by an Android app via Bluetooth.

On the Arduino side, we will build a simple weather station using a temperature and humidity sensor along with an ambient light-level sensor. We will connect a Bluetooth Low Energy (BLE) module to the project so that the Android phone can access the measurements wirelessly.

We will develop a simple Android app with an interface that allows us to:

- Access all the measurements performed by the weather station with the tap of a button
- Display each measurement within an enlarged text view

**Hardware and software requirements**

The first thing you will need for this project is an Arduino Uno board.

Then, you need a BLE module. We chose the Adafruit nRF8001 chip because it comes with a nice Arduino library, and it already has existing examples of Android apps to control the module. This is the same module that we used in the previous chapter.

For the sensors, I chose a DHT11 sensor to measure the temperature and the ambient humidity. DHT11 is a digital temperature and humidity sensor that is really easy to integrate with Arduino. There are several solutions available for Arduino, but this sensor was chosen because it is one of the easiest to interface with Arduino. To make the sensor work with Arduino, we will also need a 4.7K Ohm resistor.
We will also use a photocell in series with a 10K Ohm resistor to measure the ambient light level. The photocell is basically a resistor that will change its resistance depending on the incoming light on the cell. It will be connected to the Arduino analog input to measure the ambient light level.

Finally, you will need a breadboard and some jumper wires to make the different connections.

The following is a list of all hardware parts you will need for this project, along with links to find these parts on the Web:

- The DHT11 sensor and 4.7K Ohm resistor ([https://www.adafruit.com/products/386](https://www.adafruit.com/products/386))
- The photocell ([https://www.sparkfun.com/products/9088](https://www.sparkfun.com/products/9088))
- The 10K Ohm resistor ([https://www.sparkfun.com/products/8374](https://www.sparkfun.com/products/8374))
- Adafruit nRF8001 breakout board ([https://www.adafruit.com/products/1697](https://www.adafruit.com/products/1697))
- The breadboard ([https://www.adafruit.com/product/64](https://www.adafruit.com/product/64))

On the software side, you will need the Arduino IDE as usual, and the Arduino aREST library, which is found at [https://github.com/marcoschwartz/aREST/](https://github.com/marcoschwartz/aREST/).

The photocell make measurements from the DHT11 sensor, you will need the DHT library found at [https://github.com/adafruit/DHT-sensor-library](https://github.com/adafruit/DHT-sensor-library).

For the BLE chip, you will also need the nRF8001 Arduino library found at [https://github.com/adafruit/Adafruit_nRF8001](https://github.com/adafruit/Adafruit_nRF8001).

To install a given library, simply extract the folder in your Arduino /libraries folder (or create this folder if it doesn't exist yet).
Hardware configuration

We will now build the hardware for this project. To help you out, here is a schematic of the project:

Now, we will perform the following steps:

1. The first step is to place the Bluetooth module, the DHT11 sensor, and the photocell on the breadboard.
2. Then, connect the power supply from the Arduino board to the breadboard: 5V of the Arduino board goes to the red power rail, and GND goes to the blue power rail.
3. We will now connect the BLE module. First, connect the power supply of the module: GND goes to the blue power rail, and VIN goes to the red power rail.
4. After that, you need to connect the different wires responsible for the SPI interface: SCK to Arduino pin 13, MISO to Arduino pin 12, and MOSI to Arduino pin 11.
5. Then, connect the **REQ** pin to Arduino pin **10**. Finally, connect the **RDY** pin to Arduino pin **2**, and the **RST** pin to Arduino pin **9**.

For the DHT sensor, this is the function of each pin on the sensor:

![DHT Sensor Diagram](image)

6. You need to first connect the power supply: the **VCC** pin goes to the red power rail on the breadboard, and the **GND** pin goes to the blue power rail.

7. You also need to connect the **DATA** pin to pin number **7** of the Arduino board.

8. Finally, place the 4.7K Ohm resistor between the **VCC** and the **DATA** pin of the sensor.

9. For the photocell, connect the 10K Ohm resistor in series with the photocell. This means that one pin of the photocell should be in contact (on the same row on the breadboard) with one pin of the resistor.

10. Then, connect the other pin of the resistor to the blue power rail, and the other pin of the photocell to the red power rail of the breadboard.
11. Finally, connect the common pin between the photocell and resistor to the analog pin A0 of the Arduino board.

This is an image of the completely assembled project:

![Image of the completely assembled project]

**Testing the sensors**

We will now write a simple Arduino sketch to test all the sensors of the project. This will ensure that all the connections were made correctly before writing our Android app using Bluetooth. This is the complete sketch for testing the sensors:

```cpp
#include "DHT.h"

// DHT sensor
#define DHTPIN 7
#define DHTTYPE DHT11

// DHT instance
DHT dht(DHTPIN, DHTTYPE);

void setup()
{
    // Initialize the Serial port
    Serial.begin(9600);
}
```
Bluetooth Weather Station

    // Init DHT
    dht.begin();
}

void loop()
{
    // Measure from DHT
    float temperature = dht.readTemperature();
    float humidity = dht.readHumidity();

    // Measure light level
    float sensor_reading = analogRead(A0);
    float light = sensor_reading/1024*100;

    // Display temperature
    Serial.print("Temperature: ");
    Serial.print((int)temperature);
    Serial.println(" C");

    // Display humidity
    Serial.print("Humidity: ");
    Serial.print(humidity);
    Serial.println("%");

    // Display light level
    Serial.print("Light: ");
    Serial.print(light);
    Serial.println("%");
    Serial.println("*");

    // Wait 500 ms
    delay(500);
}

Let's now look at this sketch in more detail. It starts by including the DHT11 library:

    #include "DHT.h"

We also declare that the sensor is attached to pin number 7, and that the DHT sensor we are using is a DHT11 sensor by declaring constants:

    #define DHTPIN 7

    #define DHTTYPE DHT11
After that, we can declare an instance of the DHT sensor:

```cpp
DHT dht(DHTPIN, DHTTYPE);
```

In the `setup()` function of the sketch, we will start the serial communications:

```cpp
Serial.begin(9600);
```

We will also initialize the DHT sensor:

```cpp
dht.begin();
```

In the `loop()` function of the sketch, we will perform the temperature and humidity measurements from the sensor:

```cpp
float temperature = dht.readTemperature();
float humidity = dht.readHumidity();
```

We will also read out from the photocell, and convert this reading to a percentage of illumination. To do so, we must know that the analog input of the Arduino returns a value going from 0 to 1,023 (10 bits). Therefore, we need to divide the reading from the input by 1,023. Then, to get a result in percent, we will multiply this value by 100:

```cpp
float sensor_reading = analogRead(A0);
float light = sensor_reading/1024*100;
```

When the measurements are done, we print out the value of each of them on the serial port so that we can visualize the data. This is for example the code that prints out the temperature:

```cpp
Serial.print("Temperature: ");
Serial.print((int)temperature);
Serial.println(" C");
```

We will also repeat each `loop()` function every 500 ms:

```cpp
delay(500);
```

Note that all the code for this chapter can be found inside the GitHub repository of the book at https://github.com/marcoschwartz/arduino-android-blueprints.
It's now time to test this simple Arduino sketch to check if our sensors are working. Upload the sketch to the Arduino board, and open the serial monitor (making sure the serial speed is set to 9,600). You should get a similar result inside the serial monitor, depending on your surroundings:

Temperature: 26°C
Humidity: 35%
Light: 75.42%

**Writing the Arduino sketch**

Now that we know that our sensors are working correctly, we can write the final sketch that allows the Arduino board to be accessed by the Android application we will write later on. The following is the complete sketch for this part:

```cpp
// Control Arduino board from BLE

// Enable lightweight
#define LIGHTWEIGHT 1

// Libraries
#include <SPI.h>
#include "Adafruit_BLE_UART.h"
#include <aREST.h>
#include "DHT.h"

// Pins
#define ADAFRUITBLE_REQ 10
#define ADAFRUITBLE_RDY 2
#define ADAFRUITBLE_RST 9

// DHT sensor
#define DHTPIN 7
#define DHTTYPE DHT11

// DHT instance
DHT dht(DHTPIN, DHTTYPE);

// Create aREST instance
aREST rest = aREST();

// BLE instance
Adafruit_BLE_UART BTLEserial = Adafruit_BLE_UART(ADAFRUITBLE_REQ, ADAFRUITBLE_RDY, ADAFRUITBLE_RST);
```
// Variables to be exposed to the API
int temperature;
int humidity;
int light;

void setup(void)
{
    // Start Serial
    Serial.begin(9600);

    // Start BLE
    BTLEserial.begin();

    // Give name and ID to device
    rest.set_id("001");
    rest.set_name("weather_station");

    // Expose variables to API
    rest.variable("temperature", &temperature);
    rest.variable("humidity", &humidity);
    rest.variable("light", &light);

    // Init DHT
    dht.begin();

    // Welcome message
    Serial.println("Weather station started");
}

void loop()
{
    // Measure from DHT
    float t = dht.readTemperature();
    float h = dht.readHumidity();
    temperature = (int)t;
    humidity = (int)h;

    // Measure light level
    float sensor_reading = analogRead(A0);
    light = (int)(sensor_reading/1024*100);

    // Tell the nRF8001 to do whatever it should be working on.
    BTLEserial.pollACI();
Bluetooth Weather Station

// Ask what is our current status
aci_evt_opcode_t status = BTLEserial.getState();

// Handle REST calls
if (status == ACI_EVT_CONNECTED) {
    rest.handle(BTLEserial);
}
}

Now, let's look at this sketch in more detail. Some of the parts are similar to the sketch we saw earlier to test the sensor; we will not detail these parts again. It starts by declaring that we want to use the lightweight mode of the aREST library:

#define LIGHTWEIGHT 1

Then, we will define that we want to use the library for the Bluetooth chip, the aREST library, and the library for the DHT sensor:

#include <SPI.h>
#include "Adafruit_BLE_UART.h"
#include <aREST.h>
#include "DHT.h"

After this, we will define the pins on which we connected the BLE module:

#define ADAFRUITBLE_REQ 10
#define ADAFRUITBLE_RDY 2
#define ADAFRUITBLE_RST 9

We need to create an instance of the aREST library:

aREST rest = aREST();

We also need to create an instance of the BLE module:

Adafruit_BLE_UART BTLEserial = Adafruit_BLE_UART(ADAFRUITBLE_REQ, ADAFRUITBLE_RDY, ADAFRUITBLE_RST);

Just before the setup() function of the sketch, we will declare the following three variables that contain the measurements coming from the sensor:

    int temperature;
    int humidity;
    int light;

Then, in the setup() function of the sketch, we will initialize the BLE module:

    BTLEserial.begin();
After that, we will set an ID and a name for our project:

```java
rest.set_id("001");
rest.set_name("weather_station");
```

We also have to expose the different measurement variables to the aREST API so that they can be accessed by the Android app:

```java
rest.variable("temperature", &temperature);
rest.variable("humidity", &humidity);
rest.variable("light", &light);
```

In the `loop()` function of the sketch, we will poll the status of the BLE module:

```java
BTLEserial.pollACI();
```

We will also get the state of the module and store it in a variable:

```c
aci_evt_opcode_t status = BTLEserial.getState();
```

If this status indicates that the Bluetooth module is connected to another device, we will process the incoming request with the aREST library:

```java
if (status == ACI_EVT_CONNECTED) {
    rest.handle(BTLEserial);
}
```

Note that all the code for this chapter can be found inside the GitHub repository of the book at https://github.com/marcoschwartz/arduino-android-blueprints.

It's now time to upload the sketch to your Arduino board. When this is done, you can move on to the development of the Android app to control the Arduino board via the BLE sketch.

**Wireframing our Android application and modifying the layout files**

We will start off our BLE weather station project by creating a new project in Android Studio with a blank activity.

We will target our project for a minimum SDK of 18 and a maximum SDK of 19.
Bluetooth Weather Station

We will first start off by drawing a paper prototype of how our application will work and the basic user flow, as shown in the following image. This will help us understand how the application will work as well as facilitating our development process.

![Paper prototype image]

Upon analyzing the preceding image, we can see that this design will require two TextView objects. The upper TextView object will show all the Bluetooth callbacks, state changes, and characteristics written to the BLE module, while the lower TextView object will show the output from the temperature, light, and humidity sensor depending on which button was tapped.

The TextView objects will give them the following IDs:

- connectionStatusView
- dataOutputTextView

In the lower part of the layout, we will have three buttons reflecting the three parameters that we will be requesting, that is, temperature, light, and humidity. We will name the buttons as follows:

- The temperature button will be named as follows:
  - **Text**: Temperature
  - **ID**: temperatureButton
• The humidity button will be named as follows:
  - **Text**: Humidity
  - **ID**: humidityButton

• The light button will be named as follows:
  - **Text**: Light
  - **ID**: lightButton

**Implementing Android layouts in the main activity**

Before we embark on this project, we will enable the Auto-Import function, which will enable us to compile our project even more effectively and gives us one thing less to worry about.

You can enable Auto-Import by going to the Preferences option and selecting all the available options. The Auto-Import preferences are available on Mac and Windows as follows:

- On a Mac, navigate to **Android Studio > Preferences > Editor > Auto-Import**
- On Windows, navigate to **File > Settings > Editor > Auto-Import**

With all the necessary settings in place, we will first start off by creating a new project, where we will choose the following within the New Project setup:

- **Name**: Bluetooth Weather Station
- **Minimum SDK**: 18
- **Project**: Blank Activity
- **Activity Name**: MainActivity
- **Domain**: arduinoandroid.com

We will build on our previous project in *Chapter 2, Controlling an Arduino Board via Bluetooth*, that is, the Arduino BLE Android project will start off by importing the arduinoBLE project from the Github repository and clone it to our desktop or download it as a ZIP file as explained in *Chapter 2, Controlling an Arduino Board via Bluetooth*. 
Once imported, we will open `MainActivity.java`, select all the code below the `import` statement and copy it. When all the code has been copied, we will open our current project (Android Bluetooth Weather Station), go into `MainActivity.java`, delete all the code below the `import` statement, and paste the code.

In case you get stuck at this stage of the project, our code will be available in the repository in two stages, the version with all the necessary code that needs to be modified and the completed project. These are all available in the GitHub repository at https://github.com/marcoschwartz/arduino-android-blueprints.

Once the code is in our project, we will proceed by changing references to the UI elements to reflect our latest additions to the Android layout file in the `onCreate()` method:

```java
dataOutput = (TextView) findViewById(R.id.dataOutputTextView);
connectionOutput = (TextView) findViewById(R.id.connectionStatusView);

adapter = BluetoothAdapter.getDefaultAdapter();
temperature = (Button) findViewById(R.id.temperatureButton);
light = (Button) findViewById(R.id.lightButton);
humidity = (Button) findViewById(R.id.humidityButton);
```

In this project, we will modify `onClickListeners` to connect to the buttons that we have included in the Android layout file:

```java
temperature.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        String setTempMessage = "/temperature /";
        tx.setValue(setTempMessage.getBytes(Charset.forName("UTF-8")));
        if (gatt.writeCharacteristic(tx)) {
            writeLine("Sent: " + setTempMessage);
        } else {
            writeLine("Couldn't write TX characteristic!");
        }
    }
});
```

```java
light.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        String setLightMessage = "/light /
        tx.setValue(setLightMessage.getBytes(Charset.forName("UTF-8")));
        if (gatt.writeCharacteristic(tx)) {
```

We will also modify the code that deals with writing remoteCharacteristics, namely, the writeLine() method, and in addition, we will add another method known as writeSensorData(), which will deal with the remote data arriving from our different sensors:

private void writeSensorData(final CharSequence text) {
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            connectionOutput.setText("\n");
            connectionOutput.append(text);
            connectionOutput.append("\n");
        }
    });
}

//Implement the method below to output temperature/humidity/light readings to dataOutputView

private void writeSensorData(final CharSequence text) {
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
Before we are able to move ahead with compiling the project, we need to work on the `onCharacteristicChanged` method so that the data that is received from the sensor data will be set to the `dataOutput` text view:

```java
public void onCharacteristicChanged(BluetoothGatt gatt,
        BluetoothGattCharacteristic characteristic) {
    super.onCharacteristicChanged(gatt, characteristic);
    writeSensorData(characteristic.getStringValue(0));
}
```

At this point in time, the project will be unable to function as the necessary permissions have not been implemented yet. User permissions are necessary as it allows the application to access different capabilities of the device. In this case, we will need to add the following two permissions within the `AndroidManifest.xml` file, which you will find by navigating to `app > src > main > AndroidManifest.xml`:

```xml
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
```

When we perform all these changes, we should expect the rudimentary user interface to look as follows, with the sensor data showing up after tapping on the different parameters:
Enhancing the user interface

The current user interface requires further enhancements to make it user friendly. One can easily notice that the sensor data output needs to be enlarged and centered and the buttons can definitely be more attractive. Also, we want to make sure that our Weather Station app stands out from the user's current list of apps, so our app would definitely benefit from a change in the icon.

We will work on the following main tasks:

- Creating and adding our very own Android app icon
- Centering and enlarging the data output text
- Modifying the buttons and adding some color to our text

Creating and adding our very own app icon

One of our very first steps to enhance the user experience is to have our very own icon.

First, we will start off by downloading the image asset. This is available publicly at http://bit.ly/chapter3-iclauncher.
Bluetooth Weather Station

You should navigate using the project tree, followed by a right-click on app, as shown in the following screenshot:

After you right-click on app, create a new image asset by going to **New > Image Asset**, as shown in the following screenshot:
You will then be shown an **Asset Studio** pop-up window, which will allow you to choose your very own image file. For optimization purposes, we recommend going for a `.png` file with a resolution of 144 pixels by 144 pixels. Android Studio automatically does all the resizing and resource creation to adapt your graphic to different screens, as shown in the following screenshot:
Bluetooth Weather Station

Once you choose the `ic_launcher` image file, which we have provided you with, you will be shown a screen with the icon in different sizes. Click on Next, where you will see the following screen:

![Asset Studio](image)

The preceding screen warns you that previous files will be overwritten and shows you the image launcher file in a number of different resolutions once again. Click on Finish, then compile the app, launch it on your physical device, and you should see something as pleasant as the following in your app tray and in the app's action bar:
Here's what the app's action bar will look like:

![Bluetooth Weather Station]

Service discovery completed!

**Centering and enlarging the data output text**

In order to edit the layout for the main text output where the sensor data will be shown, we will need to open the project tree and navigate towards the layout file, which is available at `app > src > main > res > layout > activity_main_screen.xml`.

Once in this view, we recommend that you modify the text using the text view. This will allow you finer control and will also get you used to the different conventions used when editing Android layout files programmatically.

When opening the `activity_main_screen.xml` file, we will be seeing the different XML codes for the buttons and text views. At this point, look out for the code that takes care of the Sensor Data Output TextView and add the following code:

```xml
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:id="@+id/dataOutputTextView"
    android:layout_gravity="center_vertical"
    android:textSize="200dp"
    android:gravity="center"
    android:text="99" />
```

In this block of code, we have temporarily used the placeholder text 99 so that we can approximate how it will look with the Android layout designer. With this modification, the sensor data output is now big enough to be seen by the user, thus enhancing the user experience.
Modifying the buttons and adding some color to our text

Finally, we will modify our buttons and add some color to the text by performing the following steps:

1. We will follow these two steps to create new buttons:
   1. Create a drawable folder with a new XML drawable file known as buttonshape.xml.
   2. We will then connect the drawable resource file to the main Android layout file.
2. Create the drawable folder by right-clicking on the res folder, which is available by navigating to App > src > main > res.
3. After creating the drawable folder within the res folder, we need to once again right-click on the new drawable folder and click on New and choose Drawable resource file, as shown in the following screenshot:

4. Name the file buttonshape and type down shape as the Root element followed by clicking on OK, as shown in the following screenshot:
5. Within the `buttonshape.xml` file, we will add the following code:

```xml
<?xml version="1.0" encoding="utf-8"?>

<shape xmlns:android="http://schemas.android.com/apk/res/android">
  android:shape="rectangle">
    <corners android:radius="10dp"/>
    <solid android:color="#FFFFFF"/>
    <padding android:left="0dp" android:top="0dp" android:right="0dp" android:bottom="0dp"/>
    <size android:width="85dp" android:height="99dp"/>
    <stroke android:width="2dp" android:color="#4A90E2"/>
  </shape>
```

6. Then, we go towards the `activity_main_screen.xml` file and refer to this drawable by including the following line of code within the button modules:

```xml
android:background="@drawable/buttonshape"
```

7. We will also add some flavor by adding the following line of code to the button and TextView modules within the `activity_main_screen.xml` file:

```xml
android:textColor="#4A90E2"
```

In the preceding code, #4A90E2 refers to the hex code of the main color used in the app icon so that we maintain some consistency with the main user interface.
Bluetooth Weather Station

The final layout will look as follows on a Nexus 5 smartphone:

It's important to note that different Android devices have different dimensions. So, for your specific Android device, you might need to do further optimizations within the Android layout files to improve the interface.

**How to go further**
A large number of improvements could be done towards improving the user interface process within the Android app. Currently, service discovery is refreshed only by physically rotating the device, as the `onResume()` method is called upon rotation of the device. This could easily be improved by adding a refresh icon in the action bar and connecting this icon to the code, so that this method is called when the icon is tapped.
In addition, further user interface customizations can make it possible to personalize the app to your own liking; with regards to this app, you can get an idea of the possibilities by looking at the following links from the Android developers site:


You can even expand the app further with real-time monitoring, statistics, and trends.

**Summary**

In this chapter, we built a simple weather station using Arduino and Android. We attached several sensors to our Arduino board, along with a Bluetooth Low Energy module. We also built the corresponding Android app so that we can access all the data measured by the Arduino board just by tapping on a button of the phone.

In the next chapter, we will use a different technology to interact with an Arduino board via Android: Wi-Fi. We will build a smart power switch, to control an electrical device remotely, and also to measure the device power consumption via Wi-Fi.
Where to buy this book

You can buy Arduino Android Blueprints from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.