

COM413 – Mobile App Development – Workbook Practical 5

Introduction

In this weeks practical, you will find out how to use the HTTP protocol to talk to web servers so that you can download binary data and text.

You will also learn how to use API's within your application to display information via web services, this includes downloading and parsing the data and then displaying this information to your users.

Basic:

Create an app which will:

- Download and display an image
- Download and display plain text
- Download and display sensor data

Advanced:

Creating a weather app on Android is easy and a cool thing you do within your app. You can get weather information including Temperature, Pressure, Humidity, Weather status by using a weather API.

There are many websites you can get by searching on the Google which supply free weather API, but in this practical we are going to use an API provided by OpenWeatherMap.

Additional:

Choose an API to use from online and customise your app to display the information to the user. The app doesn't need to be complex. Display the information in an eye-catching way.

Basic Functionality:

Part 1.

In this practical, you will find out how to use the HTTP protocol to talk to web servers so that you can download binary data and text.

How It Works

Before you start the app, let's discuss what's happened at this point. You are using the HTTP protocol to connect to the web, your application needs the **INTERNET** permission. That means the first thing you do is add the permission in the **AndroidManifest.xml** file.

```
<uses-permission android:name="android.permission.INTERNET" />
```

You then define the `OpenHttpConnection ()` method, which takes a URL string and returns an `InputStream` object. Using an `InputStream` object, you can download the data by reading bytes from the stream object. In this method, you make use of the `HttpURLConnection` object to open an HTTP connection with a remote URL. You set all the various properties of the connection, such as the request method, and so on:

```
HttpURLConnection httpConn = (HttpURLConnection) conn;
httpConn.setAllowUserInteraction(false);
httpConn.setInstanceFollowRedirects(true);
httpConn.setRequestMethod("GET"); // GET is a method to get the data
```

After trying to establish a connection with the server, the HTTP response code is returned. If the connection is established (via the response code `HTTP_OK`), then you proceed to get an `InputStream` object from the connections:

```
response = httpConn.getResponseCode();
if (response == HttpURLConnection.HTTP_OK) {
    in = httpConn.getInputStream();
}
```

Using the `InputStream` object, you can then start to download the data from the server.

Consuming Web Services Using HTTP

One common way to communicate with the outside world is through HTTP. HTTP is the protocol that drives much of the web's success. Using the HTTP protocol, you can perform a variety of tasks, such as downloading web pages from a web server, downloading binary data and more.

In this practical, we will create an Android project to **use the HTTP protocol to connect to the web to download binary data content.**

1. Using Android Studio, create a new Android project with an **Empty Activity** and name it **Networking Binary**.

Note: keep the company domain as **example.com**

2. Add the following `uses-permission` statement to the **AndroidManifest.xml** file.

```
<uses-permission android:name="android.permission.INTERNET" />
```

```
<application
    android:allowBackup="true"
```

3. Import the following packages to the **MainActivity.java** (i.e. you need to type the following import statements)

```
import java.io.IOException;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLConnection;

public class MainActivity extends AppCompatActivity {
```

4. Define the **OpenHttpConnection ()** method in the **MainActivity.java** file:

```
private InputStream OpenHttpConnection(String urlString) throws IOException{
    InputStream in = null;
    int response = -1;

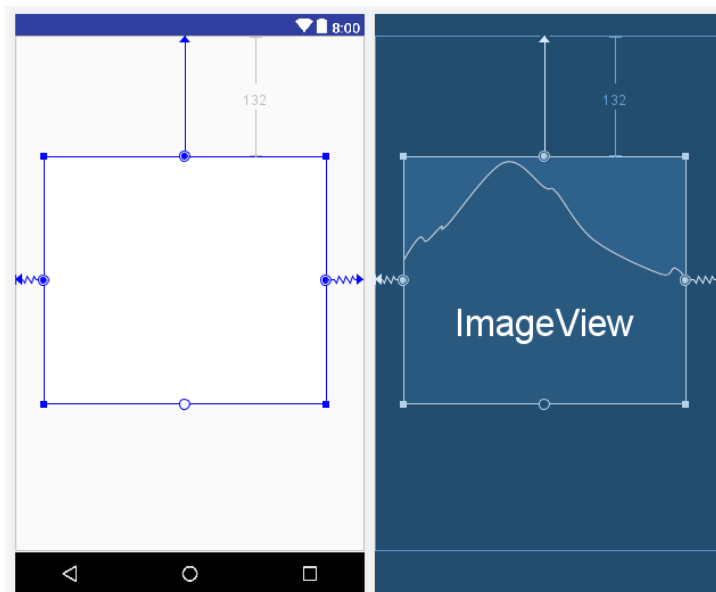
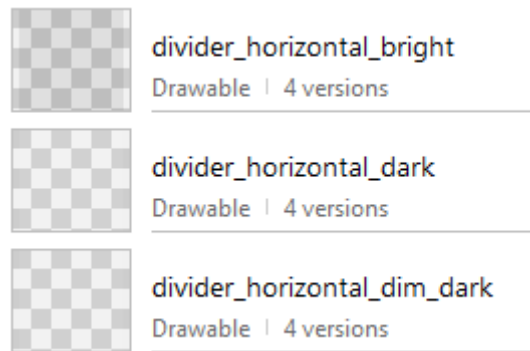
    URL url = new URL(urlString);
    URLConnection conn = url.openConnection();

    if (! (conn instanceof HttpURLConnection) )
        throw new IOException("Not an HTTP connection");
    try {
        HttpURLConnection httpConn = (HttpURLConnection) conn;
        httpConn.setAllowUserInteraction(false);
        httpConn.setInstanceFollowRedirects(true);
        httpConn.setRequestMethod("GET");
        httpConn.connect();
        response = httpConn.getResponseCode();
        if (response == HttpURLConnection.HTTP_OK){
            in = httpConn.getInputStream();
        }
    }
    catch (Exception ex){
        Log.d( tag: "Networking", ex.getLocalizedMessage());
        throw new IOException("Error connecting");
    }
    return in;
}
```

Downloading and displaying Binary Data

Now we can actually use the internet to download information. A common task you can perform is downloading binary data from the web. For example, you might want to download an image from a server so that you can display it in your application.

1. Replace the default **TextView** with the **ImageView** in the **activity_main.xml** file and choose one of the following for the default view.



2. Declare **img** and **REQUEST_INTERNET** in the **MainActivity.java** file:

```
public class MainActivity extends AppCompatActivity {  
  
    ImageView img;  
    final private int REQUEST_INTERNET = 123;
```

3. Currently, the **MainActivity.java** file includes **onCreate** and **OpenHttpConnection** methods. Before download the image, we need to check the permissions for **INTERNET** using **checkSelfPermission**. Therefore, add the following statements to the **onCreate()**.

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    if (ContextCompat.checkSelfPermission( context: this, Manifest.permission.INTERNET)
        != PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions( activity: this, new String[]{
            Manifest.permission.INTERNET}, REQUEST_INTERNET);
    }
    else{
        new DownloadImageTask().execute
            ("https://upload.wikimedia.org/wikipedia/en/9/99/Ulster_University_Logo.png");
    }
}

```

4. We haven't created the **DownloadImageTask()** method yet. Therefore, it may be displayed in red. Don't worry about this as we will be creating the **DownloadImageTask()** method soon.
5. After the **checkSelfPermission**, we need to call **onRequestPermissionsResult** method. Therefore, add the following in the **MainActivity.java** file:

```

@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults){
    switch (requestCode){
        case REQUEST_INTERNET:
            if (grantResults[0] == PackageManager.PERMISSION_GRANTED){
                new DownloadImageTask().execute
                    ("https://upload.wikimedia.org/wikipedia/en/9/99/Ulster_University_Logo.png");
            }
            else{
                Toast.makeText( context: MainActivity.this, text: "Permission Denied", Toast.LENGTH_LONG).show();
            }
            break;
        default:
            super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    }
}

```

6. Now we will create the **DownloadImageTask()** method in the **MainActivity.java** file.

```

private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {
    protected Bitmap doInBackground(String... urls){
        return DownloadImage(urls[0]);
    }

    protected void onPostExecute(Bitmap result){
        ImageView img = (ImageView) findViewById(R.id.imageView);
        img.setImageBitmap(result);
    }
}

```

7. The next task is to create the **DownloadImage()** method in the **MainActivity.java** file.

```
private Bitmap DownloadImage(String URL) {  
    Bitmap bitmap = null;  
    InputStream in = null;  
    try{  
        in = OpenHttpConnection(URL);  
        bitmap = BitmapFactory.decodeStream(in);  
        in.close();  
    }  
    catch (IOException e1){  
        Log.d( tag: "NetworkingActivity", e1.getLocalizedMessage());  
    }  
    return bitmap;  
}
```

8. Run the program using an Emulator or an android mobile phone. The figure below shows the image downloaded from the web and then displayed in the **ImageView**.



How It Works

The **DownloadImage()** method takes the URL of the image to download and then opens the connection to the server using the **OpenHttpConnection()**. Using the **InputStream** object returned by the connection, the **decodeStream()** method from the **BitmapFactory** class is used to download and decode the data into a **Bitmap** object. The **DownloadImage()** method returns a **Bitmap** object.

To download an image and display it on the activity, you call the **DownloadImage()** method. The **DownloadImage()** method is synchronous – that is, it will not return control until the image is downloaded – calling it directly freezes the UI of your activity. All synchronous code must be wrapped using an **AsyncTask** class. Using **AsyncTask** enables you to perform background

tasks in a separate thread and then return the result in a UI thread. That way, you can perform background operations without needing to handle complex threading issues.

To call the `DownloadImage ()` method asynchronously, you need to wrap the code in a subclass of the `AsyncTask` class, as shown here:

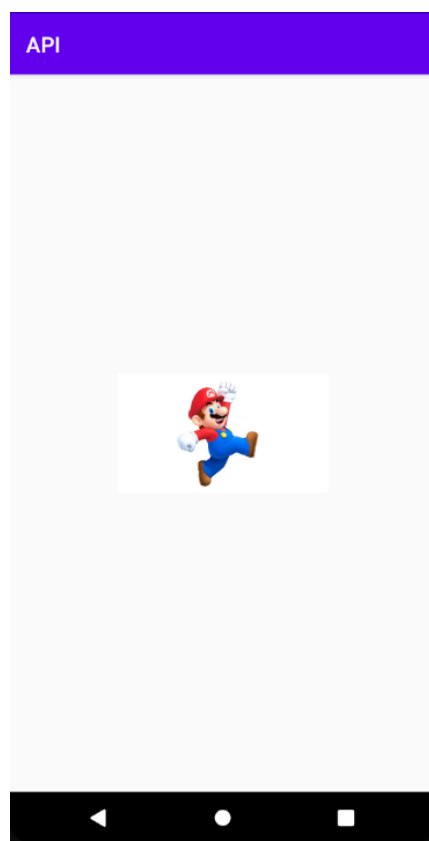
```
private class DownloadImageTask extends AsyncTask<String, Void, Bitmap>{  
    protected Bitmap doInBackground(String... urls){  
        return DownloadImage(urls[0]);  
    }  
  
    protected void onPostExecute(Bitmap result){  
        ImageView img = (ImageView) findViewById(R.id.imageView);  
        img.setImageBitmap(result);  
    }  
}
```

Basically, you define a class (`DownloadImageTask`) that extends the `AsyncTask` class. In this case, there are two methods within the `DownloadImageTask` class: `doInBackground ()` and `onPostExecute ()`.

You put all the code that needs to be run asynchronously in the `doInBackground ()` method. When the task is completed, the result is passed back via `onPostExecute ()` method.

In this case, you use the `ImageView` to display the downloaded image.

You need to find the address of the image, but once you do this you can change the image to anything you want:



Part 2.

In this part of the practical, you find out how to use the HTTP protocol to download **plain-text** content. For example, you might want to access a web service that returns a string of random quotes. Create a new project but keep the old project working... Or take screenshots and just change the bits you need to.

Downloading Plain-Text Content

We will create an Android project to use the HTTP protocol to connect to the web to download plain-text content.

1. Using Android Studio, create a new Android project with **Empty Activity** and name it **Networking Text**. (**Note:** keep the company domain as **example.com**)
2. Add the following statement to the **AndroidManifest.xml** file.

```
<uses-permission android:name="android.permission.INTERNET" />

<application
    android:allowBackup="true"
```

3. Import the following packages to the **MainActivity.java** (i.e. you need to type the following import statements)

```
import java.io.IOException;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLConnection;

public class MainActivity extends AppCompatActivity {
```

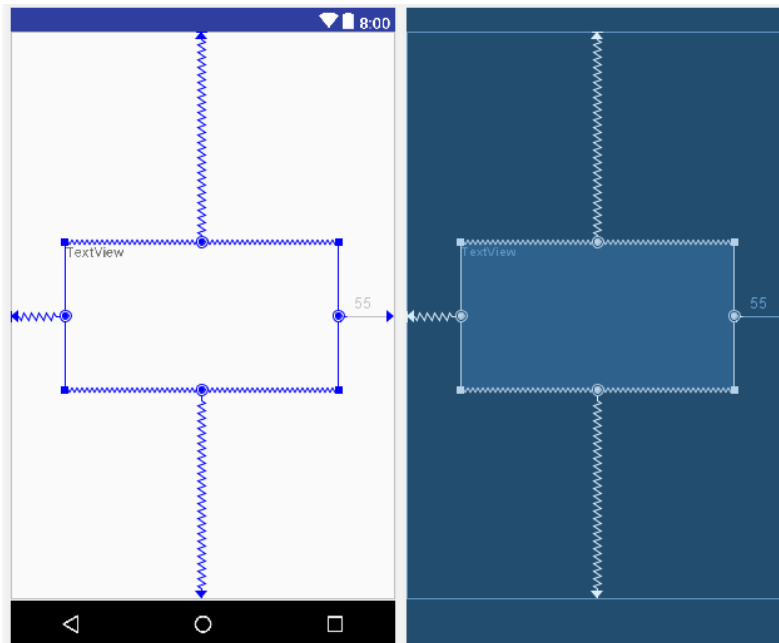
4. Define the **OpenHttpConnection ()** method in the **MainActivity.java** file:

```
private InputStream OpenHttpConnection(String urlString) throws IOException{
    InputStream in = null;
    int response = -1;

    URL url = new URL(urlString);
    URLConnection conn = url.openConnection();

    if (! (conn instanceof HttpURLConnection) )
        throw new IOException("Not an HTTP connection");
    try {
        HttpURLConnection httpConn = (HttpURLConnection) conn;
        httpConn.setAllowUserInteraction(false);
        httpConn.setInstanceFollowRedirects(true);
        httpConn.setRequestMethod("GET");
        httpConn.connect();
        response = httpConn.getResponseCode();
        if (response == HttpURLConnection.HTTP_OK){
            in = httpConn.getInputStream();
        }
    }
    catch (Exception ex){
        Log.d( tag: "Networking", ex.getMessage());
        throw new IOException("Error connecting");
    }
    return in;
}
```


- Replace the default **TextView** with a new **TextView** in the **activity_main.xml** file. It may look like something similar to the image shown below.



- Declare **txtView** and **REQUEST_INTERNET** in the **MainActivity.java** file:

```
public class MainActivity extends AppCompatActivity {
    final private int REQUEST_INTERNET = 123; //123 is just an ID for the request,
    TextView txtView;
```

- Currently, the **MainActivity.java** file includes **onCreate** and **OpenHttpConnection** methods. Before download the text, we need to check the permission of **INTERNET** using **checkSelfPermission**. Therefore, add the following statements to the **onCreate()**.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    txtView=(TextView)findViewById(R.id.txtView);
    if (ContextCompat.checkSelfPermission( context: this, Manifest.permission.INTERNET)
        != PackageManager.PERMISSION_GRANTED){
        ActivityCompat.requestPermissions( activity: this, new String[]{Manifest.permission.INTERNET},
            REQUEST_INTERNET);
    }
    else{
        new DownloadTextTask().execute("https://www-eng-x.llnl.gov/documents/a_document.txt");
    }
}
```

We haven't created the **DownloadTextTask()** method yet. Therefore, it may be displayed in red. Again, we will be creating the **DownloadTextTask()** method soon.

8. After the **checkSelfPermission**, we need to call **onRequestPermissionsResult** method. Therefore, add the following in the **MainActivity.java** file:

```
@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults){
    switch (requestCode){
        case REQUEST_INTERNET:
            if (grantResults[0] == PackageManager.PERMISSION_GRANTED){
                new DownloadTextTask().execute("https://www-eng-x.llnl.gov/documents/a_document.txt");
            }
            else{
                Toast.makeText( context MainActivity.this, text: "Permission Denied", Toast.LENGTH_LONG).show();
            }
            break;
        default:
            super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    }
}
```

9. Now we will create the **DownloadTextTask()** method in the **MainActivity.java** file.

```
private class DownloadTextTask extends AsyncTask<String, Void, String> {
    protected String doInBackground(String... urls){
        return DownloadText(urls[0]);
    }
    @Override
    protected void onPostExecute(String result){
        Toast.makeText(getBaseContext(), result, Toast.LENGTH_LONG).show();
        txtView.setText(result);
    }
}
```

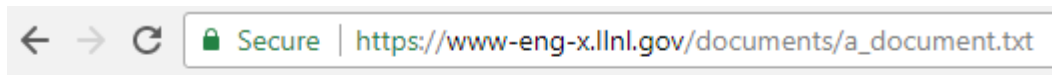
10. Next task is to create the **DownloadText()** method in the **MainActivity.java** file

```
private String DownloadText(String URL){
    int BUFFER_SIZE = 2000;
    InputStream in = null;

    try{
        in = OpenHttpConnection(URL);
    }
    catch (IOException e1){
        Log.d( tag: "Networking", e1.getLocalizedMessage());
        return "";
    }
    InputStreamReader isr = new InputStreamReader(in);
    int charRead;
    String str = "";
    char[] inputBuffer = new char[BUFFER_SIZE];
    try{
        while ((charRead = isr.read(inputBuffer)) > 0){
            //-----convert the chars to a string -----
            String readString = String.valueOf(inputBuffer, offset: 0, charRead);
            str += readString;
            inputBuffer = new char[BUFFER_SIZE];
        }
        in.close();
    } catch (IOException e){
        Log.d( tag: "Networking", e.getLocalizedMessage());
        return "";
    }
    return str;
}
```

11. Run the program using an Emulator or an android mobile phone. Here, we are going to test with a web address - https://www-eng-x.llnl.gov/documents/a_document.txt .

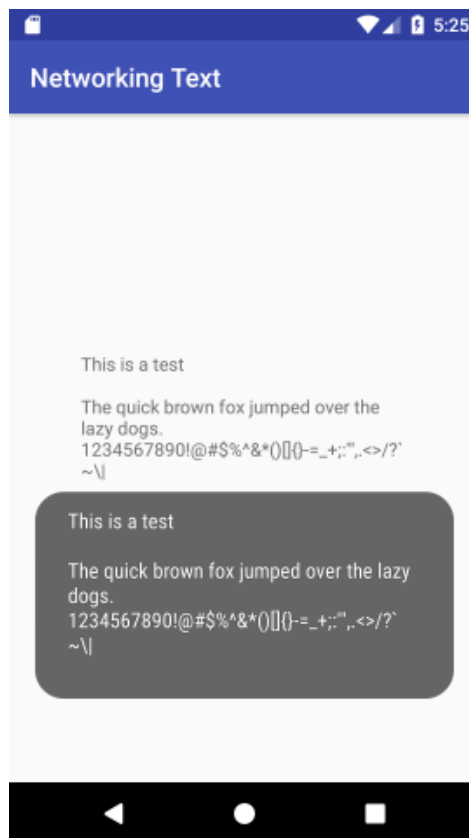
12. If this doesn't work, try: https://www.w3.org/TR/PNG/iso_8859-1.txt



This is a test

The quick brown fox jumped over the lazy dogs.

1234567890!@#\$%^&*()[]{}~=-_+;:'.<>/?'`~\|



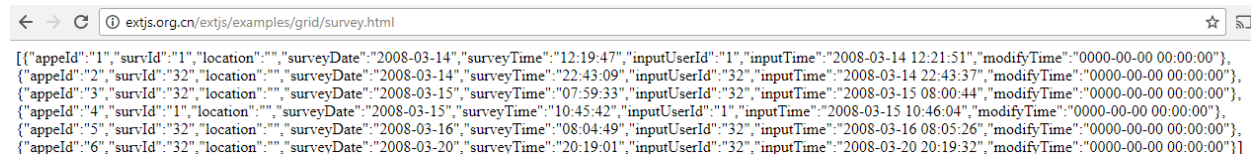
How It Works

The DownloadText () method access the text file's URL, downloads the text file, and then returns the desired string of text. It basically opens an HTTP connection to the server and then uses an InputStreamReader object to read each character from the stream and save it in a String object. As shown in the previous section, you had to create a subclass of the AsyncTask class to call the DownloadText () method asynchronously.

Part 3.

An efficient way to represent information exists in the form of JSON (JavaScript Object Notation). JSON is a lightweight data-interchange format that is easy for humans to read and write. It is also easy for machines to parse and generate. The following show what a JSON message looks like:

<http://extjs.org.cn/extjs/examples/grid/survey.html>



The screenshot shows a web browser window with the address bar displaying "extjs.org.cn/extjs/examples/grid/survey.html". The main content area shows a JSON array of six objects, each representing a survey entry with fields like "appeld", "survld", "location", "surveyDate", "surveyTime", "inputUserId", "inputTime", and "modifyTime".

The above details can be easily represented as given below:

```
[
  {
    "appeld": "1",
    "survld": "1",
    "location": "",
    "surveyDate": "2008-03-14",
    "surveyTime": "12:19:47",
    "inputUserId": "1",
    "inputTime": "2008-03-14 12:21:51",
    "modifyTime": "0000-00-00 00:00:00"
  }
]
```

The preceding block of lines represents a set of data taken for a survey. Note that the information is represented as a collection of key/value pairs (e.g. "appeld": "1"), and that each key/value pair is grouped into an ordered list of objects. Unlike XML, there are no lengthy tag names. Instead, there are only brackets and braces.

We will learn about how to process JSON messages using the **JSONArray** and **JSONObject** classes available in the Android SDK.

In this example, we will see how to extract the values of "appeld" and "inputTime" from the above example.

1. Using Android Studio, create a new Android project with an **Empty Activity** and name it **Networking JSON**. (**Note:** keep the company domain as **example.com**)
2. Add the following statement to the **AndroidManifest.xml** file.

```
<uses-permission android:name="android.permission.INTERNET"/>
```

```
<application
    android:allowBackup="true"
    android:usesCleartextTraffic="true"
```

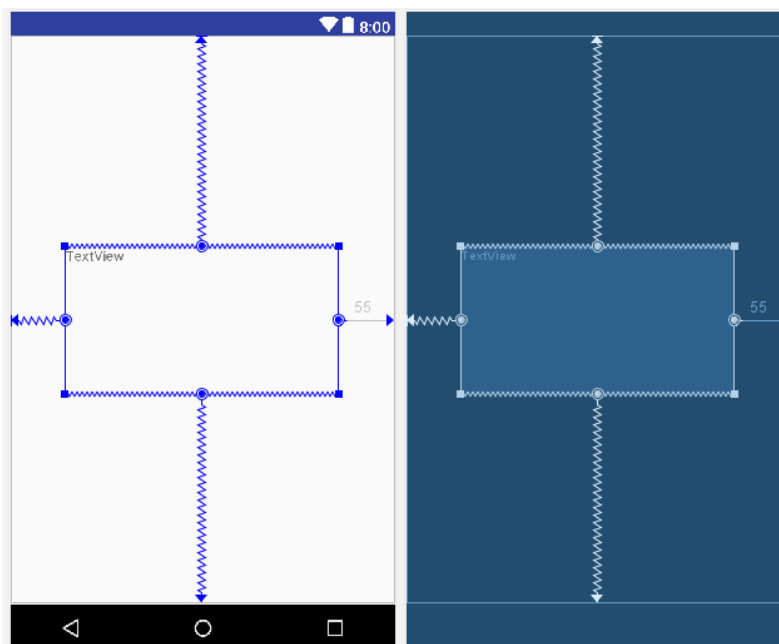
3. Define the **OpenHttpConnection ()** method in the **MainActivity.java** file:

```
private InputStream OpenHttpConnection(String urlString) throws IOException{
    InputStream in = null;
    int response = -1;

    URL url = new URL(urlString);
    URLConnection conn = url.openConnection();

    if (! (conn instanceof HttpURLConnection) )
        throw new IOException("Not an HTTP connection");
    try {
        HttpURLConnection httpConn = (HttpURLConnection) conn;
        httpConn.setAllowUserInteraction(false);
        httpConn.setInstanceFollowRedirects(true);
        httpConn.setRequestMethod("GET");
        httpConn.connect();
        response = httpConn.getResponseCode();
        if (response == HttpURLConnection.HTTP_OK) {
            in = httpConn.getInputStream();
        }
    }
    catch (Exception ex){
        Log.d( tag: "Networking", ex.getLocalizedMessage());
        throw new IOException("Error connecting");
    }
    return in;
}
```

4. Replace the default **TextView** with a new **TextView** in the **activity_main.xml** file. . It may look like something similar to the image shown below.



5. Declare **txtView** in the **MainActivity.java** file:

```
public class MainActivity extends AppCompatActivity {  
    TextView txtView;
```

6. Currently, the **MainActivity.java** file includes **onCreate** and **OpenHttpConnection** methods. To get the data from <http://extjs.org.cn/extjs/examples/grid/survey.html>, we need to add the following statements to the **onCreate()**.

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
  
    txtView=(TextView)findViewById(R.id.txtView);  
    new ReadJSONFeedTask().execute("http://extjs.org.cn/extjs/examples/grid/survey.html");  
}
```

We haven't created the **ReadJSONFeedTask()** method yet. Therefore, it may be displayed in red color. Don't worry about this as we will be creating the **ReadJSONFeedTask()** method soon.

7. Now we will create the **ReadJSONFeedTask()** method in the **MainActivity.java** file.

```
private class ReadJSONFeedTask extends AsyncTask<String, Void, String> {  
  
    protected String doInBackground(String... urls){  
        return readJSONFeed(urls[0]);  
    }  
  
    protected void onPostExecute(String result){  
        try{  
            JSONArray jsonArray = new JSONArray(result);  
            Log.i( tag: "JSON", msg: "Number of surveys in feed: " + jsonArray.length());  
            String str = "";  
            //----print out the content of the json feed ----  
            for (int i = 0; i < jsonArray.length(); i++){  
                JSONObject jsonObject = jsonArray.getJSONObject(i);  
                Toast.makeText(getBaseContext(), text: "appeId: " + jsonObject.getString( name: "appeId") +  
                    " , inputTime: " + jsonObject.getString( name: "inputTime"),  
                    Toast.LENGTH_SHORT).show();  
                str += "appeId: " + jsonObject.getString( name: "appeId") +  
                    " , inputTime: " + jsonObject.getString( name: "inputTime") + "\n";  
            }  
            txtView.setText(str);  
        } catch (JSONException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

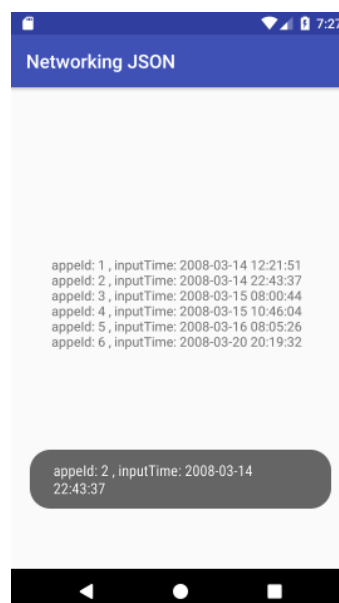
8. Next task is to create the **readJSONFeed()** method in the **MainActivity.java** file.

```

public String readJSONFeed(String address){
    URL url = null;
    try{
        url = new URL(address);
    }catch (MalformedURLException e){
        e.printStackTrace();
    }
    StringBuilder stringBuilder = new StringBuilder();
    HttpURLConnection urlConnection = null;
    try{
        urlConnection = (HttpURLConnection) url.openConnection();
    }catch (IOException e){
        e.printStackTrace();
    }
    try{
        InputStream content = new BufferedInputStream(urlConnection.getInputStream());
        BufferedReader reader = new BufferedReader(new InputStreamReader(content));
        String line;
        while((line = reader.readLine()) != null){
            stringBuilder.append(line);
        }
    }catch (IOException e){
        e.printStackTrace();
    }finally {
        urlConnection.disconnect();
    }
    return stringBuilder.toString();
}

```

9. Run the program using an Emulator or an android mobile phone. The figure below shows the output displayed in the **textView** and **Toast**.



How It Works

You call the `readJSONFeed()` method in the `doInBackground` method, and the JSON string that you fetch is passed in through the `onPostExecute()` method. The JSON string used in this example is from <http://extjs.org.cn/extjs/examples/grid/survey.html>.

To obtain the list of objects in the JSON string, you use the `JSONArray` class, passing it the JSON feed as the constructor for the class:

```
JSONArray jsonArray = new JSONArray(result);
Log.i("JSON", "Number of surveys in feed: " + jsonArray.length());
```

The `length()` method returns the number of objects in the `jsonArray` object. With the list of objects stored in the `jsonArray` object, you iterate through it to obtain each object using the `getJSONObject()` method:

```
//----print out the content of the json feed ----
for (int i = 0; i < jsonArray.length(); i++){
    JSONObject jsonObject = jsonArray.getJSONObject(i);
    Toast.makeText(getBaseContext(), "appeId: " + jsonObject.getString("appeId") +
        " , inputTime: " + jsonObject.getString("inputTime"),
        Toast.LENGTH_SHORT).show();
    str += "appeId: " + jsonObject.getString("appeId") +
        " , inputTime: " + jsonObject.getString("inputTime") + "\n";
}
```

The `getJSONObject()` method returns an object of type `JSONObject`. To obtain the value of the key/value pair stored inside the object, you use the `getString()` method (you can also use the `getInt()`, `getLong()`, and `getBoolean()` methods for other data types).

Finally, you access the JSON feed asynchronously using the `execute()` method:

```
new
ReadJSONFeedTask().execute("http://extjs.org.cn/extjs/examples/grid/survey.html");
```


Advanced Functionality:

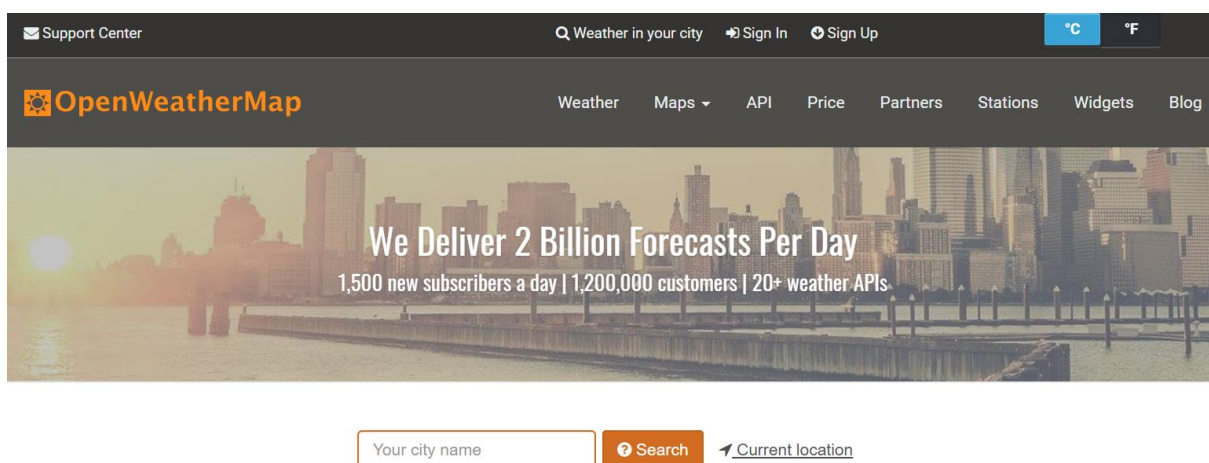
In this part of the practical, we're going to pull weather data from a free weather API called **OpenWeatherMap** and display the current weather for a given location.

Creating a weather app on Android is easy and a cool thing you can use by yourself. You can get weather information including Temperature, Pressure, Humidity, Weather status by using a weather API.

There are many websites you can get by searching on the Google which supply free weather API, but in this practical we are going to use an API provided by [OpenWeatherMap](https://openweathermap.org/). So, let's start creating a weather app.

Steps to Create a Weather App on Android

1. First, create a Weather API from OpenWeatherMap (<https://openweathermap.org/>).



2. Click on the API.

Weather API

[Home](#) / [Weather API](#)

Please **sign up** and use our fast and easy-to-work weather APIs for free. Look at our **monthly subscriptions** for more options than Free account can provide you. Read **How to start** first and enjoy using our powerful weather APIs.

Current weather data

[API doc](#) [Subscribe](#)

- Access current weather data for any location including over 200,000 cities

5 day / 3 hour forecast

[API doc](#) [Subscribe](#)

- 5 day forecast is available at any location or city

16 day / daily forecast

[API doc](#) [Subscribe](#)

- 16 day forecast is available at any location or city

3. Click on the Subscribe button

Price

[Home](#) / [Price](#)

Subscribe to current weather, forecasts, and historical data collections and enjoy our fast simple API!
Please, read [How to buy](#) before you subscribe.

Current weather and forecasts collection

	Free	Startup	Developer	Professional	Enterprise
Price Price is fixed, no other hidden costs (VAT is not included)	Free	40 USD / month	180 USD / month	470 USD / month	2,000 USD / month
Subscribe	Get API key and Start	Subscribe	Subscribe	Subscribe	Subscribe
Calls per minute (no more than)	60	600	3,000	30,000	200,000

4. Click on “Get API key and Start” to get an API key.

It is quite easy to work with Openweather API. Just sign up to get your API key and then call any weather API. And mind using API key in every API call whatever account you choose from Free to Enterprise.

How to start in 3 simple steps

1 [Sign up](#) and get an API key (APPID) on your account page.

It takes up to 1 hour to activate your API key. We send you a confirmation email as your API key is ready to work.

2 Start using API for **free**.

Find the complete description of API calls with a list of parameters and examples of responses in [API documentation](#).

Please, use API key in each API call.

3 If you need more features than Free account can give you, look at the options of our monthly subscriptions [here](#).

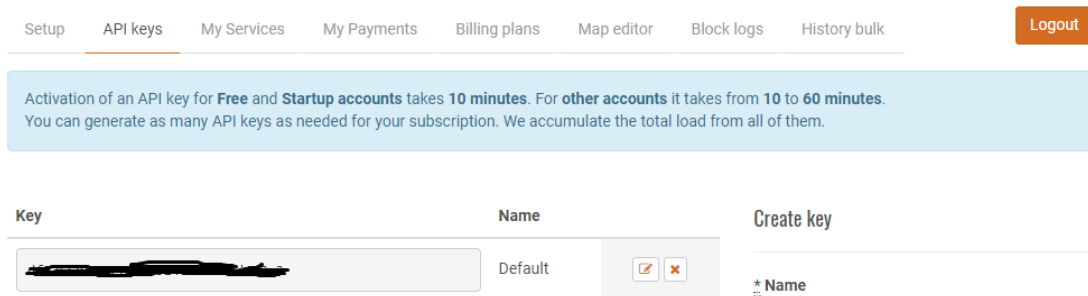
Choose your subscription depending on a number of calls per sec, API availability, service provided, and other features.

Contact us via [Support Center](#).

5. Click on the Sign up and create an account to get an API key.

Create New Account

- Once you created an account, click on the API keys to get the key for the app development.

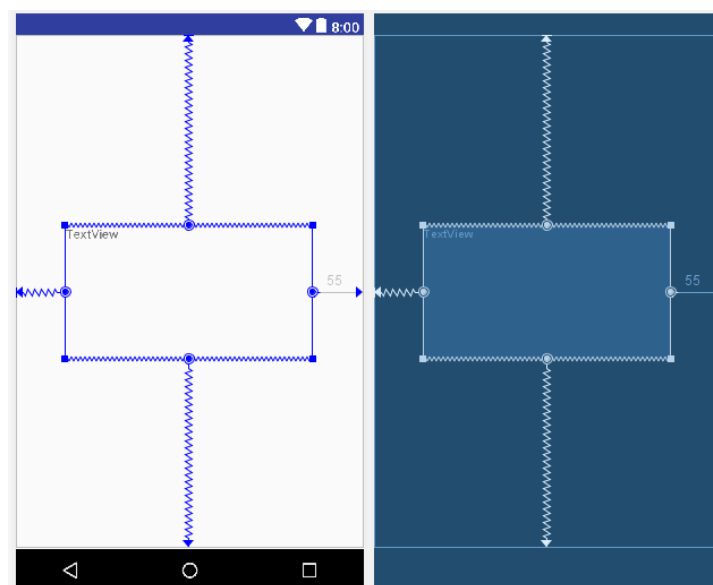


- Using Android Studio, create a new Android project with an **Empty Activity** and name it **My Weather App**. (**Note:** keep the company domain as **example.com**)
- Add the following statement to the **AndroidManifest.xml** file.

```
<uses-permission android:name="android.permission.INTERNET" />

<application
    android:allowBackup="true"
    android:usesCleartextTraffic="true"
```

- Replace the default **TextView** with a new **TextView** in the **activity_main.xml** file. . It may look like something similar to the image shown below.



10. Declare **txtView**, **OPEN_WEATHER_MAP_URL** and **OPEN_WEATHER_MAP_API_KEY** in the **MainActivity.java** file:

```
public class MainActivity extends AppCompatActivity {

    private static final String OPEN_WEATHER_MAP_API_KEY = "2a58a392-428a-432d-8a2d-8a2d8a2d8a2d";
    TextView txtView;
```

11. Currently, the **MainActivity.java** file includes **onCreate** and **OpenHttpConnection** methods. To get the weather data, we need to add the following statements to the **onCreate()**.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    txtView = (TextView) findViewById(R.id.textView);
    // Coordinates for New York City Hall, New York, USA
    // Feel free to replace these with where you live!
    double lat = 40.712774, lon = -74.006091;
    String units = "metric";
    String url = String.format(

    new GetWeatherTask().execute(url);
}
```

```
String url =  
String.format("http://api.openweathermap.org/data/2.5/weather?q=London,uk&A  
PPID= OPEN WEATHER MAP API KEY ");
```

We haven't created the `GetWeatherTask()` method yet. Therefore, it may be displayed in red color. Don't worry about this as we will be creating the `GetWeatherTask()` method next.

12. Now we will create the **GetWeatherTask()** method in the **MainActivity.java** file

```
private class GetWeatherTask extends AsyncTask<String, Void, String> {

    @Override
    protected String doInBackground(String... strings) {
        String weather = "UNDEFINED";
        try {
            URL url = new URL(strings[0]);
            HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();

            InputStream stream = new BufferedInputStream(urlConnection.getInputStream());
            BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(stream));
            StringBuilder builder = new StringBuilder();

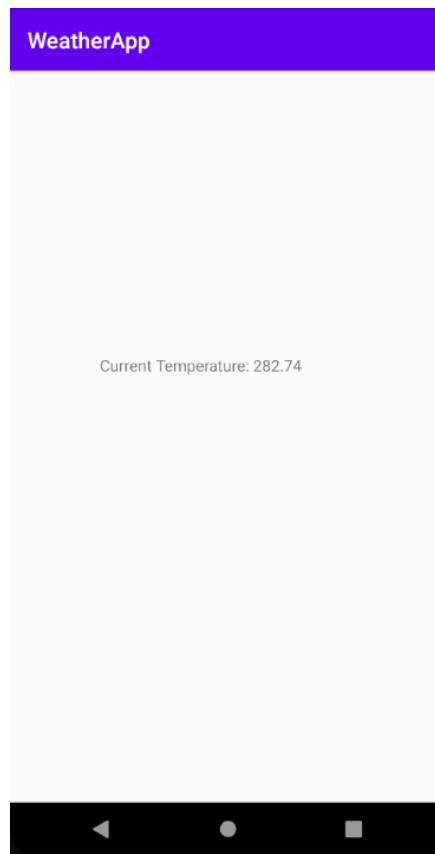
            String inputString;
            while ((inputString = bufferedReader.readLine()) != null) {
                builder.append(inputString);
            }

            JSONObject topLevel = new JSONObject(builder.toString());
            JSONObject main = topLevel.getJSONObject("main");
            weather = String.valueOf(main.getDouble("temp"));

            urlConnection.disconnect();
        } catch (IOException | JSONException e) {
            e.printStackTrace();
        }
        return weather;
    }

    @Override
    protected void onPostExecute(String temp) {
        txtView.setText("Current Temperature : " + temp);
    }
}
```

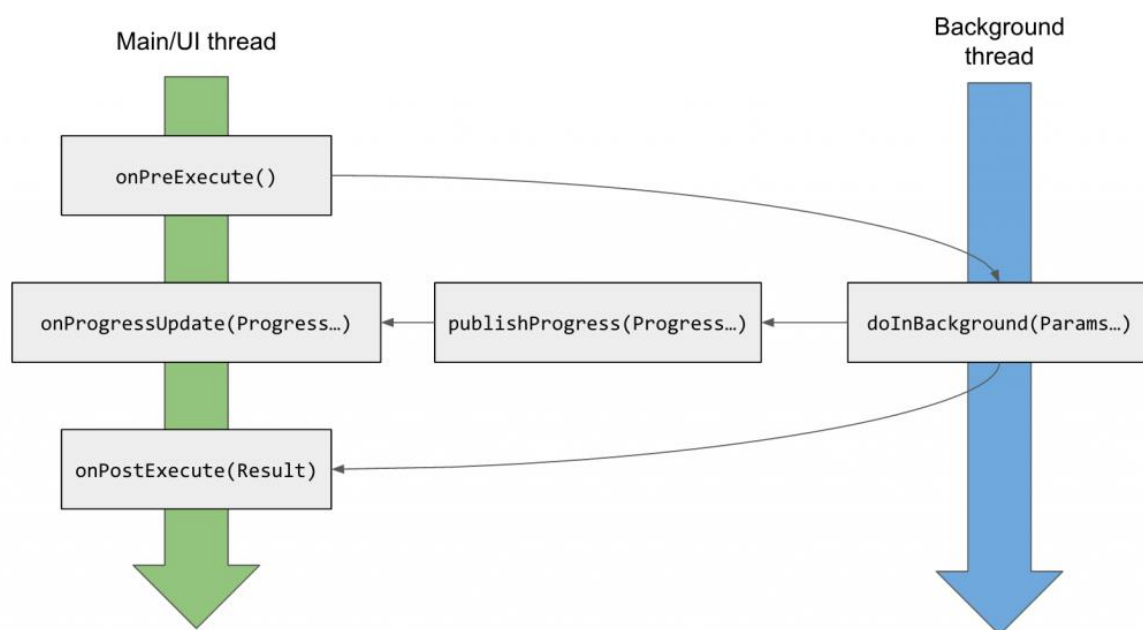
13. Run the program using an Emulator or an android mobile phone. The figure below shows the sample output displayed in the **textView**



How it works

Before we get into the actual networking, let us first look at threads and threading. A thread is simply a sequence of execution handled independently of the main sequence. For example, Android has a main thread that we have been working with. Most of our code runs on main thread. In fact, we can only access UI elements on the main thread, hence it's also called the UI thread sometimes.

But, we have a problem if we want to do networking. If we try to run networking calls on the main/UI thread, our entire app will freeze until we are done with the networking request. Imagine you are waiting for your news feed to load on a news app on your phone. If we did not have threading, your phone would freeze until the stories are ready to go! To prevent this, we can create another thread, grab the data from the network from that thread, and give it to our UI thread to update the UI elements. In fact, Android is now more proactive and forces you to take this approach so an error will be shown if you try to access the network on the UI thread!



Instead of creating and managing these threads manually, Android provides us with an **AsyncTask** parameterized class to help us do exactly this! Above is a graphic of the different method calls and the order in which they are called and the thread they run on.

First, `onPreExecute()` is called and then `doInBackground(...)` is called on the background thread immediately after. We can push progress updates to `onProgressUpdate(...)` by calling `publishProgress(...)` in `doInBackground(...)`. Finally, after `doInBackground(...)` is finished, we run `onPostExecute(...)`.

Let's create an inner subclass of **AsyncTask** to handle all of our networking after the `onCreate(...)` method. Notice that we are accepting a **String** as an input, leaving the progress parameter unused (put in **Void**), and returning a **String**. Since **AsyncTask** is an abstract class, we'll need to implement `doInBackground(...)` at least. But we also need to override `onPostExecute(...)`. We are going to be using aliasing to our benefit to populate the `TextView`. We are going to pass in a reference to the `TextView` we want to populate in the constructor. Since we're getting a **temperature**, we can simply set the `TextView`'s text to say that temperature in `onPostExecute(...)` with what we'll soon get in `doInBackground(...)`.

```

private class GetWeatherTask extends AsyncTask<String, Void, String> {
    @Override
    protected String doInBackground(String... strings) {
        return null;
    }
    @Override
    protected void onPostExecute(String temp) {
        textView.setText("Current Temperature: " + temp);
    }
}

```

Now that we have done this, we can look at what we need to do in the background thread. Let me outline the approach we are going to take in this background thread. Note that we get a varargs list, but we only need the first argument from that. However, we will need to grab just the first from that list. We need to create a valid URL from it and open up a connection. Then we will get some JSON back and we need to convert that input stream into a String and convert that into a JSON object. Then we can extract our value from the JSON. Finally, we need to **close** the connection.

The reason we use a String is so that in case we do not get a response back, we can account for that. We need to surround most of this in a try-catch since many of these instance methods throw exceptions. In the next two lines, we create a new url and connect using Android's awesome HttpURLConnection class. The next three lines essentially convert the input stream to a buffered format so we can read it a line at a time. If we wanted to be more robust, we could check the status code of the URL connection to make sure it is an HTTP OK before grabbing an input stream. Then we have a StringBuilder that we append each line of the response to the StringBuilder. Converting to a JSONObject is a piece of cake since one of the parameterized constructors for JSONObject takes a String and parses it for us.

JSON stands for Javascript Object Notation and it is a way to format and organize information in key-value pairs. We can essentially have JSON Objects with key-value pairs. We can also have JSON Arrays of objects, each with key-value pairs as well. Look at this link for more information about JSON and examples of it compared to XML. Looking at the OpenWeatherMap API, we know that we will get a top-level JSONObject. The temperature we are looking for is another JSON object whose name is "main". Then we can extract the temperature inside that object called "temp" and get the value of it. Finally, we close the connection.

Now that we have our AsyncTask subclass created, it's very simple to execute it in our onCreate(...) method.

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    double lat = 40.712774, lon = -74.006091;
    String units = "metric";
    String url =
String.format("http://api.openweathermap.org/data/2.5/weather?lat=%f&lon=%f&units=%s&appid=%s", lat, lon, units, APP_ID);

    TextView textView = (TextView) findViewById(R.id.textView);

```

```
new GetWeatherTask().execute(url);
}
```

Feel free to replace the lat and lon with your location! The one provided is that of New York City, New York, USA. The units can be either “imperial” or “metric” depending on your taste of units. If you want to see the raw JSON, copy the url link and put it in your browser and substitute the lat, lon, API key, and units. Your browser should show raw JSON text. You can format it by pasting into a JSON formatter [here](#).

In this code, we grab a reference to the TextView to change and pass it to the AsyncTask to populate after the network call. In the final line, we create a new task and call execute with the single URL. We don't need to store it in a variable since we won't be doing anything with it besides calling execute(...) on it. We can see the current weather at our provided location below!

To do

1. Go to: <https://openweathermap.org/current#data>
2. Change the metrics so that temperature is displayed in degrees Celsius.
3. Display other information such as wind, pressure e.g.

```
"temp": 282.55, "feels_like": 281.86, "temp_min": 280.37,
"temp_max": 284.26, "pressure": 1023, "humidity": 100
```

4. Use the longitude and latitude to get the weather information instead of using any city name.

```
String url =
String.format("http://api.openweathermap.org/data/2.5/weather?lat=%f&lon=%f&units=%s
&appid=%s", lat, lon, units, APP_ID);
```

5. Change the city name:

<http://api.openweathermap.org/data/2.5/weather?q=London,uk&APPID=2c5cacc84098ec6fdcfaeda8aa90dc1f>

Then you will get these details



```
{
  "coord": {
    "lon": -0.13,
    "lat": 51.51
  },
  "weather": [
    {
      "id": 521,
      "main": "Rain",
      "description": "shower rain",
      "icon": "09d"
    },
    {
      "id": 701,
      "main": "Mist",
      "description": "mist",
      "icon": "50d"
    },
    {
      "id": 300,
      "main": "Drizzle",
      "description": "light intensity drizzle",
      "icon": "09d"
    }
  ],
  "base": "stations",
  "main": {
    "temp": 282.55,
    "pressure": 1023,
    "humidity": 100,
    "temp_min": 280.37,
    "temp_max": 284.26
  },
  "visibility": 1000,
  "wind": {
    "speed": 1
  },
  "clouds": {
    "all": 90
  },
  "dt": 1539782400,
  "sys": {
    "type": 1,
    "id": 5091,
    "message": 0.0025,
    "country": "GB",
    "sunrise": 1539757717,
    "sunset": 1539795721
  },
  "id": 2643743,
  "name": "London",
  "cod": 200
}
```

This document explains how we can get those details.

When we try to get the weather information using a city name, for example London,uk

<http://api.openweathermap.org/data/2.5/weather?q=London,uk&APPID=2c5cacc84098ec6fdcfaeda8aa90dc1f>

Then we will get these details in JSON format.

← → ↻ ⓘ Not secure | api.openweathermap.org/data/2.5/weather?q=London,uk&APPID=2c5cacc84098ec6fdcfaeda8aa90dc1f ☆

```
{"coord":{"lon":-0.13,"lat":51.51},"weather":[{"id":521,"main":"Rain","description":"shower rain","icon":"09d"}, {"id":701,"main":"Mist","description":"mist","icon":"50d"}, {"id":300,"main":"Drizzle","description":"light intensity drizzle","icon":"09d"}],"base":"stations","main":{"temp":287.75,"pressure":1019,"humidity":87,"temp_min":287.15,"temp_max":288.15},"visibility":10000,"wind":{"speed":1},"clouds":{"all":90},"dt":1539782400,"sys":{"type":1,"id":50991,"message":0.0025,"country":"GB","sunrise":1539757717,"sunset":1539795721},"id":2643743,"name":"London","cod":200}
```

To get the temperature detail, we have used temp

```
JSONObject topLevel = new JSONObject(builder.toString());
JSONObject main = topLevel.getJSONObject("main");
weather = String.valueOf(main.getDouble("temp"));
```

Similar to this, we can get pressure, humidity, etc.

Note:

- if the value is in decimal format then you can use `getDouble`
- if the value is in integer format then you can use `getInt`
- if the value is in string format then you `getString`
- if the value is long then you can use `getLong`

Display the following information for your Workbook:

Weather2021

Current Temp: 8.72

Feels Like: 7.65

Pressure:1032

Humidity:78

Additional:

Choose an API to use from online and customise your app to display the information to the user. The app doesn't need to be complex. Display the information in an eye-catching way.

Cool Android APIs to Play

If you are currently not developing any particular app though want to try your hands on some interesting projects, here's a list of some of the coolest APIs found on the web.

1. [Marvel API](#) to create some awesome stuff with the world's greatest comic API.
2. [International Space Station API](#) empowers you to know the exact location of the ISS.
3. [Pokemon API](#) is a RESTful API which serves all the Pokemon data you will ever need.
4. [Brewery DB](#) this houses the world's most complete database for the beer industry.
5. [Lob API](#) helps you in sending personalized postcards programmatically to your customers.

Apart from this, [APIList.Fun](#) has a number of API for different genres. Feel free to explore and try on your own. We at Simform make sure that our Android Developers are ahead at their Enterprise App Development game. Are we missing anything or have something to share? Drop us a comment.

<https://www.simform.com/blog/android-app-developers-top-api/>