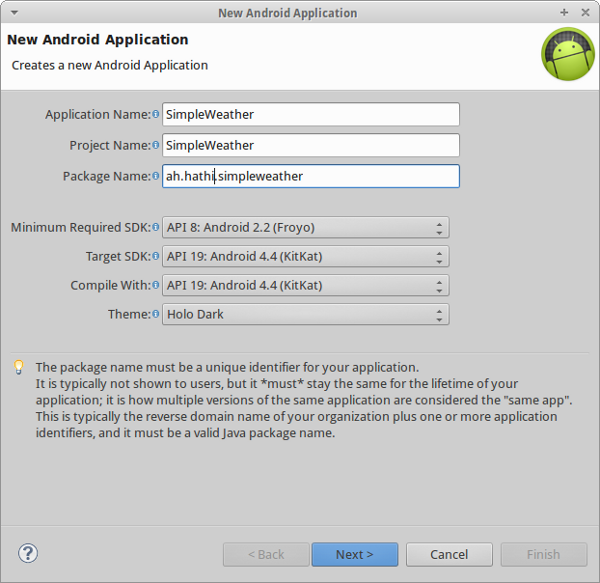
**1. Prerequisites**

Before you continue, double-check that you have the following set up:

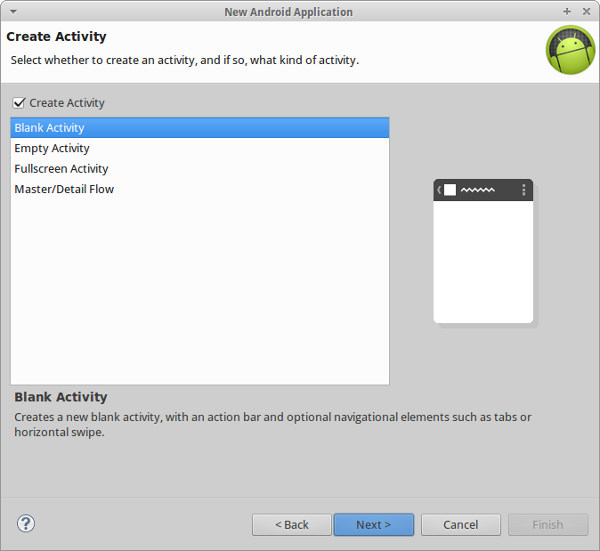
* **Eclipse ADT Bundle**: You can download it at the [Android Developer website](http://developer.android.com/sdk/index.html).
* **OpenWeatherMap API Key**: This isn't required to complete the tutorial, but it's free. You can obtain one by signing up at the [OpenWeatherMap website](http://openweathermap.org/appid).
* **Icons**: I recommend you download the weather icons font created by [Erik Flowers](https://github.com/erikflowers/weather-icons). You need to download the [TTF](https://github.com/erikflowers/weather-icons/blob/master/fonts/weathericons-regular-webfont.ttf?raw=true) file, because we'll be using it in a native app. We'll use the font to render various icons depending on the weather conditions.

**2. Create a New Project**

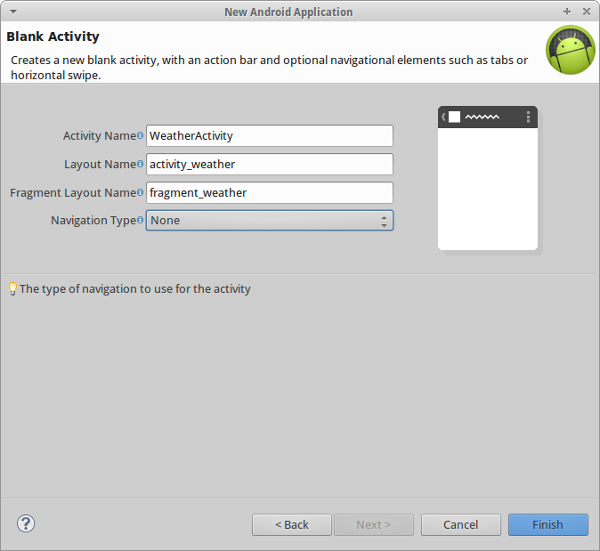
I'm going to call this app **SimpleWeather**, but feel free to give it any name you like. Enter a unique package name, set the minimum required SDK to **Android 2.2**, and set the target SDK to **Android 4.4**. You can leave the theme at **Holo Dark**.



This app will only have one Activity and it will be based on the **Blank Activity**template as shown below.



Name the Activity **WeatherActivity**. We'll be using a Fragment inside thisActivity. The layout associated with the Activity is **activity\_weather.xml**. The layout associated with the Fragment is **fragment\_weather.xml**.



**3. Add the Custom Font**

Copy **weathericons-regular-webfont.ttf**to your project's **assets/fonts** directory and rename it to **weather.ttf**.

**4. Edit the Manifest**

The only permission this app needs is android.permission.INTERNET.

|  |  |
| --- | --- |
| 1 | <uses-permission android:name="android.permission.INTERNET"/> |

To keep this tutorial simple, we're only going to support **portrait** mode. The activity node of the manifest should look like this:

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10 | <activity      android:name="ah.hathi.simpleweather.WeatherActivity"      android:label="@string/app\_name"      android:screenOrientation="portrait"      >      <intent-filter>        <action android:name="android.intent.action.MAIN" />        <category android:name="android.intent.category.LAUNCHER" />      </intent-filter>  </activity> |

**5. Edit the Activity's Layout**

There isn't much to change in **activity\_weather.xml**. It should already have aFrameLayout. Add an extra property to change the color of the background to#FF0099CC.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <FrameLayout xmlns:android="<http://schemas.android.com/apk/res/android>"      xmlns:tools="<http://schemas.android.com/tools>"      android:id="@+id/container"      android:layout\_width="match\_parent"      android:layout\_height="match\_parent"      tools:context="ah.hathi.simpleweather.WeatherActivity"      tools:ignore="MergeRootFrame"      android:background="#FF0099CC" /> |

**6. Edit the Fragment's Layout**

Edit **fragment\_weather.xml** by adding five TextView tags to show the following information:

* city and country
* current temperature
* an icon showing the current weather condition
* a timestamp telling the user when the weather information was last updated
* more detailed information about the current weather, such as description and humidity

Use a RelativeLayout to arrange the text views. You can adjust the textSize to suit various devices.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56 | <RelativeLayout xmlns:android="<http://schemas.android.com/apk/res/android>"      xmlns:tools="<http://schemas.android.com/tools>"      android:layout\_width="match\_parent"      android:layout\_height="match\_parent"      android:paddingBottom="@dimen/activity\_vertical\_margin"      android:paddingLeft="@dimen/activity\_horizontal\_margin"      android:paddingRight="@dimen/activity\_horizontal\_margin"      android:paddingTop="@dimen/activity\_vertical\_margin"      tools:context="ah.hathi.simpleweather.WeatherActivity$PlaceholderFragment" >        <TextView          android:id="@+id/city\_field"          android:layout\_width="wrap\_content"          android:layout\_height="wrap\_content"          android:layout\_alignParentTop="true"          android:layout\_centerHorizontal="true"          android:textAppearance="?android:attr/textAppearanceLarge" />        <TextView          android:id="@+id/updated\_field"          android:layout\_width="wrap\_content"          android:layout\_height="wrap\_content"          android:layout\_below="@+id/city\_field"          android:layout\_centerHorizontal="true"          android:textAppearance="?android:attr/textAppearanceMedium"          android:textSize="13sp" />        <TextView          android:id="@+id/weather\_icon"          android:layout\_width="wrap\_content"          android:layout\_height="wrap\_content"          android:layout\_centerVertical="true"          android:layout\_centerHorizontal="true"          android:textAppearance="?android:attr/textAppearanceLarge"          android:textSize="70sp"          />        <TextView          android:id="@+id/current\_temperature\_field"          android:layout\_width="wrap\_content"          android:layout\_height="wrap\_content"          android:layout\_alignParentBottom="true"          android:layout\_centerHorizontal="true"          android:textAppearance="?android:attr/textAppearanceLarge"          android:textSize="40sp" />        <TextView          android:id="@+id/details\_field"          android:layout\_width="wrap\_content"          android:layout\_height="wrap\_content"          android:layout\_below="@+id/weather\_icon"          android:layout\_centerHorizontal="true"          android:textAppearance="?android:attr/textAppearanceMedium"          />    </RelativeLayout> |

**7. Edit strings.xml**

This file contains the strings used in our app as well as the Unicode character codes that we'll use to render the weather icons. The application will be able to display eight different types of weather conditions. If you want to handle more, then refer to [this cheat sheet](http://erikflowers.github.io/weather-icons/cheatsheet/). Add the following to **values/strings.xml**:

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22 | <?xml version="1.0" encoding="utf-8"?>  <resources>        <string name="app\_name">Simple Weather</string>      <string name="change\_city">Change city</string>        <!-- Put your own APP ID here -->      <string name="open\_weather\_maps\_app\_id">11111</string>        <string name="weather\_sunny">&#xf00d;</string>      <string name="weather\_clear\_night">&#xf02e;</string>        <string name="weather\_foggy">&#xf014;</string>      <string name="weather\_cloudy">&#xf013;</string>      <string name="weather\_rainy">&#xf019;</string>      <string name="weather\_snowy">&#xf01b;</string>      <string name="weather\_thunder">&#xf01e;</string>      <string name="weather\_drizzle">&#xf01c;</string>        <string name="place\_not\_found">Sorry, no weather data found.</string>    </resources> |

**8. Add a Menu Item**

The user should be able to choose the city whose weather they want to see. Edit**menu/weather.xml** and add an item for this option.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12 | <menu xmlns:android="<http://schemas.android.com/apk/res/android>"      xmlns:app="<http://schemas.android.com/apk/res-auto>"      xmlns:tools="<http://schemas.android.com/tools>"      tools:context="ah.hathi.simpleweather.WeatherActivity" >        <item          android:id="@+id/change\_city"          android:orderInCategory="1"          android:title="@string/change\_city"          app:showAsAction="never"/>    </menu> |

Now that all the XML files are ready to use, let's move on and query the OpenWeatherMap API to fetch weather data.

**9. Fetch Data From OpenWeatherMap**

We can get the current weather details of any city formatted as JSON using the OpenWeatherMap API. In the query string, we pass the city's name and the metric system the results should be in.

For example, to get the current weather information for Canberra, using the metric system, we send a request to <http://api.openweathermap.org/data/2.5/weather?q=Canberra&units=metric>

The response we get back from the API looks like this:

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | {      "base": "cmc stations",      "clouds": {          "all": 90      },      "cod": 200,      "coord": {          "lat": -35.28,          "lon": 149.13      },      "dt": 1404390600,      "id": 2172517,      "main": {          "humidity": 100,          "pressure": 1023,          "temp": -1,          "temp\_max": -1,          "temp\_min": -1      },      "name": "Canberra",      "sys": {          "country": "AU",          "message": 0.313,          "sunrise": 1404335563,          "sunset": 1404370965      },      "weather": [          {              "description": "overcast clouds",              "icon": "04n",              "id": 804,              "main": "Clouds"          }      ],      "wind": {          "deg": 305.004,          "speed": 1.07      }  } |

Create a new Java class and name it **RemoteFetch.java**. This class is responsible for fetching the weather data from the OpenWeatherMap API.

We use the HttpURLConnection class to make the remote request. The OpenWeatherMap API expects the API key in an HTTP header named x-api-key. This is specified in our request using the setRequestProperty method.

We use a BufferedReader to read the API's response into a StringBuffer. When we have the complete response, we convert it to a JSONObject object.

As you can see in the above response, the JSON data contains a field named cod. Its value is 200 if the request was successful. We use this value to check whether the JSON response has the current weather information or not.

The **RemoteFetch.java** class should look like this:

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49 | package ah.hathi.simpleweather;    import java.io.BufferedReader;  import java.io.InputStreamReader;  import java.net.HttpURLConnection;  import java.net.URL;    import org.json.JSONObject;    import android.content.Context;  import android.util.Log;    public class RemoteFetch {        private static final String OPEN\_WEATHER\_MAP\_API =              "[http://api.openweathermap.org/data/2.5/weather?q=%s&units=metric](http://api.openweathermap.org/data/2.5/weather?q=%25s&units=metric)";        public static JSONObject getJSON(Context context, String city){          try {              URL url = new URL(String.format(OPEN\_WEATHER\_MAP\_API, city));              HttpURLConnection connection =                      (HttpURLConnection)url.openConnection();                connection.addRequestProperty("x-api-key",                      context.getString(R.string.open\_weather\_maps\_app\_id));                BufferedReader reader = new BufferedReader(                      new InputStreamReader(connection.getInputStream()));                StringBuffer json = new StringBuffer(1024);              String tmp="";              while((tmp=reader.readLine())!=null)                  json.append(tmp).append("\n");              reader.close();                JSONObject data = new JSONObject(json.toString());                // This value will be 404 if the request was not              // successful              if(data.getInt("cod") != 200){                  return null;              }                return data;          }catch(Exception e){              return null;          }      }  } |

**10. Store the City as a Preference**

The user shouldn't have to specify the name of the city every time they want to use the app. The app should remember the last city the user was interested in. We do this by making use of SharedPreferences. However, instead of directly accessing these preferences from our Activity class, it is better to create a separate class for this purpose.

Create a new Java class and name it **CityPreference.java**. To store and retrieve the name of the city, create two methods setCity and getCity. TheSharedPreferences object is initialized in the constructor. The **CityPreference.java** class should look like this:

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | package ah.hathi.simpleweather;    import android.app.Activity;  import android.content.SharedPreferences;    public class CityPreference {        SharedPreferences prefs;        public CityPreference(Activity activity){          prefs = activity.getPreferences(Activity.MODE\_PRIVATE);      }        // If the user has not chosen a city yet, return      // Sydney as the default city      String getCity(){          return prefs.getString("city", "Sydney, AU");      }        void setCity(String city){          prefs.edit().putString("city", city).commit();      }    } |

**11. Create the Fragment**

Create a new Java class and name it **WeatherFragment.java**. This fragment uses**fragment\_weather.xml** as its layout. Declare the five TextView objects and initialize them in the onCreateView method. Declare a new Typeface object named weatherFont. The TypeFace object will point to the web font you downloaded and stored in the **assets/fonts** folder.

We will be making use of a separate Thread to asynchronously fetch data from the OpenWeatherMap API. We cannot update the user interface from such a background thread. We therefore need a Handler object, which we initialize in the constructor of the WeatherFragment class.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | public class WeatherFragment extends Fragment {      Typeface weatherFont;        TextView cityField;      TextView updatedField;      TextView detailsField;      TextView currentTemperatureField;      TextView weatherIcon;        Handler handler;        public WeatherFragment(){          handler = new Handler();      }        @Override      public View onCreateView(LayoutInflater inflater, ViewGroup container,              Bundle savedInstanceState) {          View rootView = inflater.inflate(R.layout.fragment\_weather, container, false);          cityField = (TextView)rootView.findViewById(R.id.city\_field);          updatedField = (TextView)rootView.findViewById(R.id.updated\_field);          detailsField = (TextView)rootView.findViewById(R.id.details\_field);          currentTemperatureField = (TextView)rootView.findViewById(R.id.current\_temperature\_field);          weatherIcon = (TextView)rootView.findViewById(R.id.weather\_icon);            weatherIcon.setTypeface(weatherFont);          return rootView;      }  } |

Initialize the weatherFont object by calling createFromAsset on the Typefaceclass. We also invoke the updateWeatherData method in onCreate.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @Override  public void onCreate(Bundle savedInstanceState) {      super.onCreate(savedInstanceState);      weatherFont = Typeface.createFromAsset(getActivity().getAssets(), "fonts/weather.ttf");      updateWeatherData(new CityPreference(getActivity()).getCity());  } |

In updateWeatherData, we start a new thread and call getJSON on theRemoteFetch class. If the value returned by getJSON is null, we display an error message to the user. If it isn't, we invoke the renderWeather method.

Only the main Thread is allowed to update the user interface of an Android app. Calling Toast or renderWeather directly from the background thread would lead to a runtime error. That is why we call these methods using the handler's postmethod.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22 | private void updateWeatherData(final String city){      new Thread(){          public void run(){              final JSONObject json = RemoteFetch.getJSON(getActivity(), city);              if(json == null){                  handler.post(new Runnable(){                      public void run(){                          Toast.makeText(getActivity(),                                  getActivity().getString(R.string.place\_not\_found),                                  Toast.LENGTH\_LONG).show();                      }                  });              } else {                  handler.post(new Runnable(){                      public void run(){                          renderWeather(json);                      }                  });              }          }      }.start();  } |

The renderWeather method uses the JSON data to update the TextView objects. The weather node of the JSON response is an array of data. In this tutorial, we will only be using the first element of the array of weather data.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | private void renderWeather(JSONObject json){      try {          cityField.setText(json.getString("name").toUpperCase(Locale.US) +                  ", " +                  json.getJSONObject("sys").getString("country"));            JSONObject details = json.getJSONArray("weather").getJSONObject(0);          JSONObject main = json.getJSONObject("main");          detailsField.setText(                  details.getString("description").toUpperCase(Locale.US) +                  "\n" + "Humidity: " + main.getString("humidity") + "%" +                  "\n" + "Pressure: " + main.getString("pressure") + " hPa");            currentTemperatureField.setText(                      String.format("%.2f", main.getDouble("temp"))+ " ℃");            DateFormat df = DateFormat.getDateTimeInstance();          String updatedOn = df.format(new Date(json.getLong("dt")\*1000));          updatedField.setText("Last update: " + updatedOn);            setWeatherIcon(details.getInt("id"),                  json.getJSONObject("sys").getLong("sunrise") \* 1000,                  json.getJSONObject("sys").getLong("sunset") \* 1000);        }catch(Exception e){          Log.e("SimpleWeather", "One or more fields not found in the JSON data");      }  } |

At the end of the renderWeather method,  we invoke setWeatherIcon with theid of the current weather as well as the times of sunrise and sunset. Setting the weather icon is a bit tricky, because the OpenWeatherMap API supports more weather conditions than we can support with the web font we're using. Fortunately, the weather ids follow a pattern, which you can read more about on the[OpenWeatherMap website](http://openweathermap.org/weather-conditions).

This is how we map a weather id to an icon:

* the weather codes in the 200 range are related to thunderstorms, which means we can use R.string.weather\_thunder for these
* the weather codes in the 300 range are related to drizzles and we use R.string.weather\_drizzle for these
* the weather codes in the 500 range signify rain and we use R.string.weather\_rain for them
* and so on ...

We use the sunrise and sunset times to display the sun or the moon, depending on the current time of the day and only if the weather is clear.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | private void setWeatherIcon(int actualId, long sunrise, long sunset){      int id = actualId / 100;      String icon = "";      if(actualId == 800){          long currentTime = new Date().getTime();          if(currentTime>=sunrise && currentTime<sunset) {              icon = getActivity().getString(R.string.weather\_sunny);          } else {              icon = getActivity().getString(R.string.weather\_clear\_night);          }      } else {          switch(id) {          case 2 : icon = getActivity().getString(R.string.weather\_thunder);                   break;          case 3 : icon = getActivity().getString(R.string.weather\_drizzle);                   break;          case 7 : icon = getActivity().getString(R.string.weather\_foggy);                   break;          case 8 : icon = getActivity().getString(R.string.weather\_cloudy);                   break;          case 6 : icon = getActivity().getString(R.string.weather\_snowy);                   break;          case 5 : icon = getActivity().getString(R.string.weather\_rainy);                   break;          }      }      weatherIcon.setText(icon);  } |

Of course, you can handle more weather conditions by adding morecase statements to the switch statement of the setWeatherIcon method.

Finally, add a changeCity method to the fragment to let the user update the current city. The changeCity method will only be called from the main Activity class.

|  |  |
| --- | --- |
| 1  2  3 | public void changeCity(String city){      updateWeatherData(city);  } |

**12. Edit the Activity**

During the project's setup, Eclipse populated **WeatherActivity.java** with some boilerplate code. Replace the default implementation of the onCreatemethod with the one below in which we use the WeatherFragment. The onCreate method should look like this:

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11 | @Override  protected void onCreate(Bundle savedInstanceState) {      super.onCreate(savedInstanceState);      setContentView(R.layout.activity\_weather);        if (savedInstanceState == null) {          getSupportFragmentManager().beginTransaction()                  .add(R.id.container, new WeatherFragment())                  .commit();      }  } |

Next, edit the onOptionsItemSelected method and handle the only menu option we have. All you have to do here is invoke the showInputDialog method.

In the showInputDialog method, we use AlertDialog.Builder to create aDialog object that prompts the user to enter the name of a city. This information is passed on to the changeCity method, which stores the name of the city using the CityPreference class and calls the Fragment's changeCity method.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | @Override  public boolean onOptionsItemSelected(MenuItem item) {      if(item.getItemId() == R.id.change\_city){          showInputDialog();      }      return false;  }    private void showInputDialog(){      AlertDialog.Builder builder = new AlertDialog.Builder(this);      builder.setTitle("Change city");      final EditText input = new EditText(this);      input.setInputType(InputType.TYPE\_CLASS\_TEXT);      builder.setView(input);      builder.setPositiveButton("Go", new DialogInterface.OnClickListener() {          @Override          public void onClick(DialogInterface dialog, int which) {              changeCity(input.getText().toString());          }      });      builder.show();  }    public void changeCity(String city){      WeatherFragment wf = (WeatherFragment)getSupportFragmentManager()                              .findFragmentById(R.id.container);      wf.changeCity(city);      new CityPreference(this).setCity(city);  } |

Your weather app is now ready. Build the project and deploy it to an Android device for testing.