# Creating a User Interface using XML

## Prerequisites

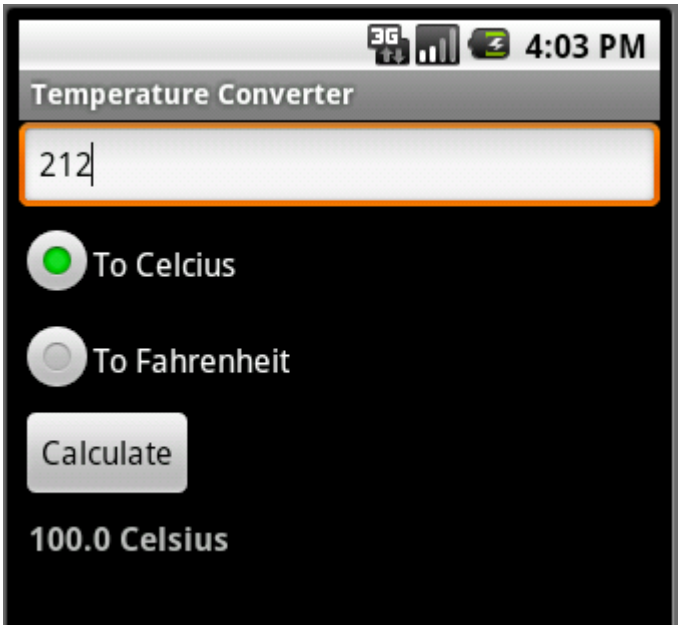
Before starting with this tutorial you should have read “Android SDK and Eclipse IDE”, or some other “Hello World” like tutorial on Android. If you use a computer at Campus Haninge, read appendix A in the tutorial mentioned above.

Although you can find a project with the solution to this exercise at Bilda, the best way to learn the basics on creating user interfaces in Android is to do it yourself, using the tools in Eclipse as described in the text.

## Introduction

When developing an Android application you should define the components and their properties in your user interface using XML. Although you can define the components in the source code, it’s good practice to separate this from the rest of the code (the code dealing with the applications event handling and logic).

In this exercise you will define a simple user interface in XML, using the layout editor in Eclipse. The goal is a user interface, UI, like the one below.



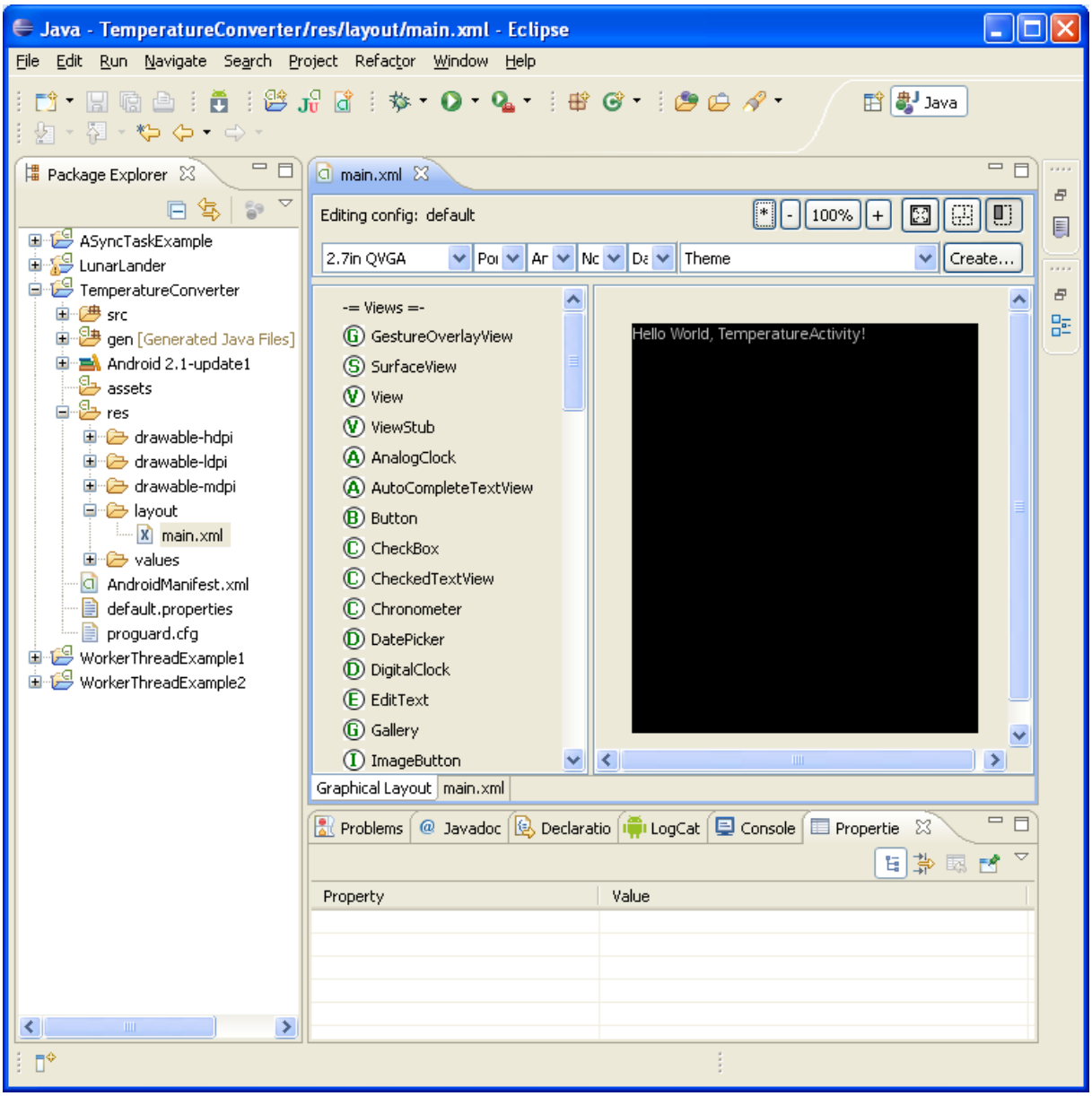

## Create a project

Create a new Android project with the following properties:
Project name, TemperatureConverter
Build target, Android 2.1
Application Name, Temperature Converter

Package Name, se.kth.temperature
Create Activity, TemperatureActivity
Min SDK version, 7

## Building the UI using the XML editor in Eclipse

Open the file res/layout/main.xml. This is where the layout of your (main) UI is defined. Use the tabs at the bottom to switch between the Graphical layout and the XML-code defining the layout.




The UI contains a LinearLayout with a TextView presenting a text (@string/hello is defined in res/values/string.xml).

Switch to XML (the tab main.xml at the *bottom* of the panel). It should look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
```

```
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
</LinearLayout>
```

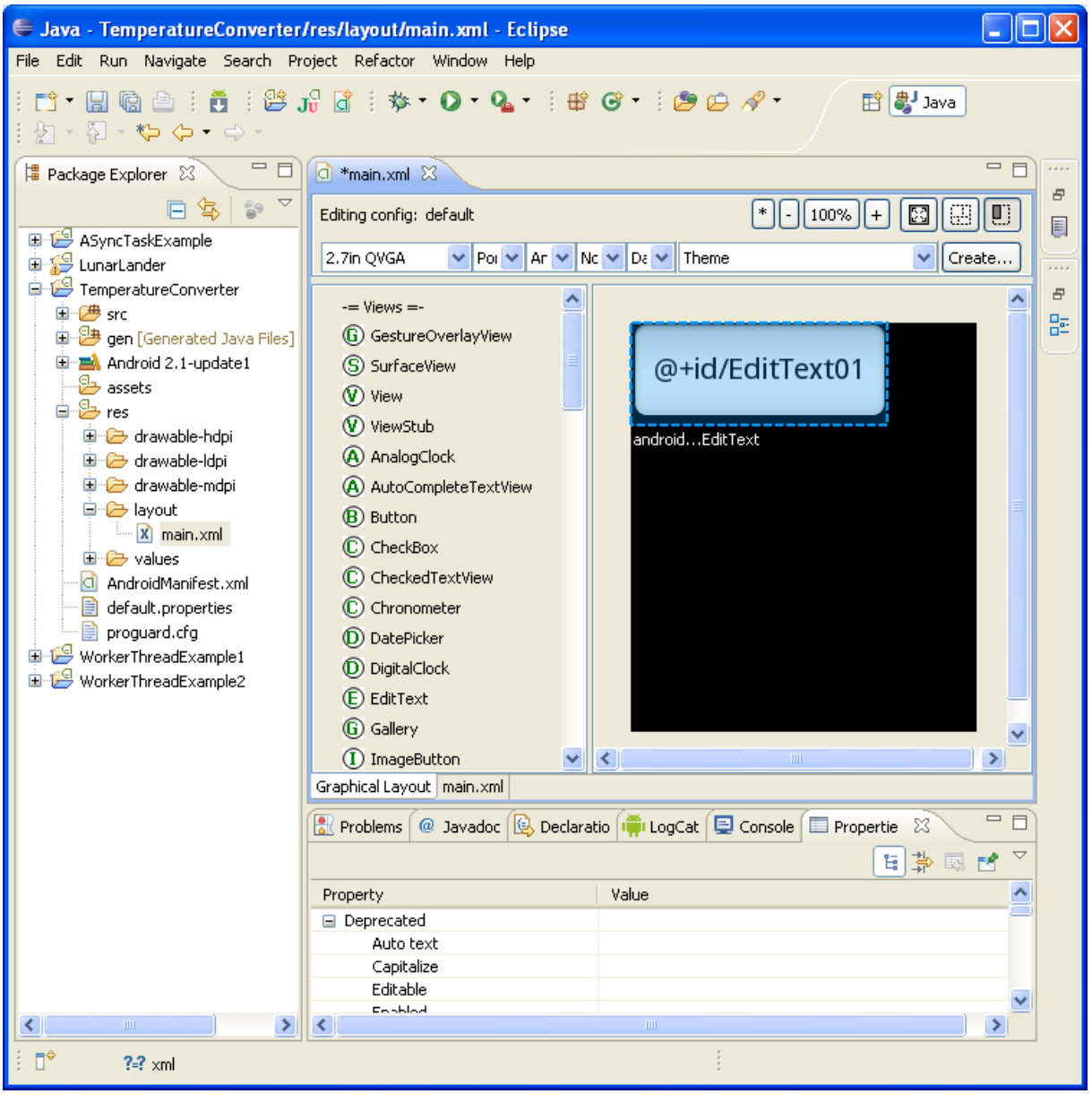
Switch back to graphical layout and remove the TextView; right-click it and select delete. Switch to XML-view again and check that the TextView is removed.

## Adding Views

Remember, in an Android, a View can be an atomic UI component, a Button, TextView or such, or it can be a container, ViewGroup, for example a layout component.

Let's add some view components in our LinerLayout. Switch to graphical layout.

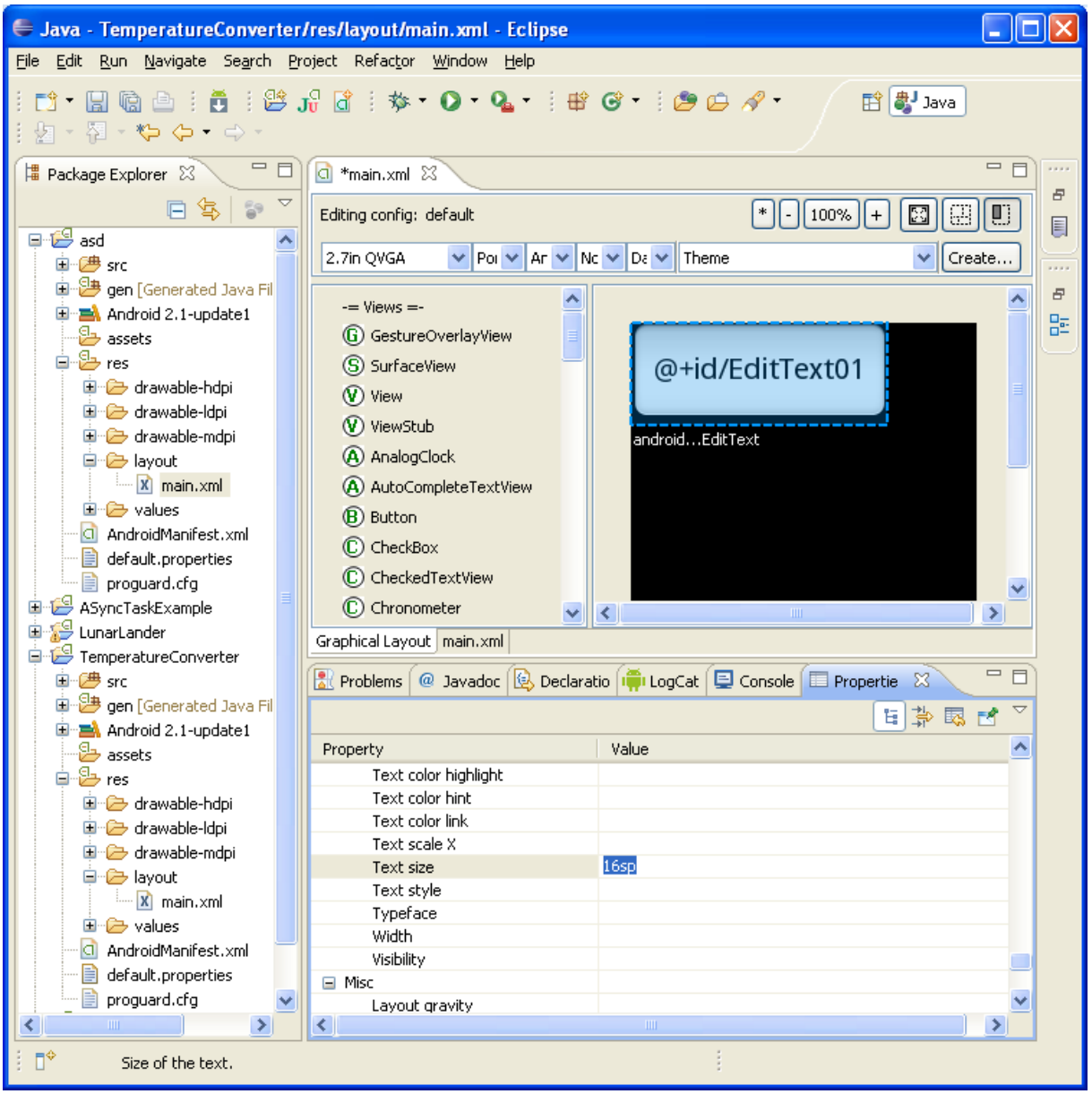
First add a TextView for text input; click on EditText in the right panel, then drag and drop the component onto the layout.




## Managing View properties

To manage the properties for this EditText, right-click the component and select Show In…/Properties. A new tab, Properties, is opened below the graphical layout. Here, you can set properties for the EditText.

For example, to change the text size, browse to the corresponding line and set the value “16sp” (scaled pixels).




Also set/change these properties for the EditText component:

| | | |
|---|---|---|
| Hint | Input a temperature | Text displayed when input is empty |
| Id | @+id/TextInput | This is the id used when referencing this view from your source code. *Must begin with "@+id/"* |
| Input type | numberDecimal | This allows only decimal number input |
| Lines | 1 | Single line |
| Text | | The displayed texts, in this case none. |
| Text size | 16sp | *Must* have a unit, sp = scale-independent pixels. |
| Layout width | fill_parent | Use the drop down menu to browse the alternatives |

Your XML-file should contain this information (though the line breaks and the indentation might be different in your version):

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
```

```
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
>

  <EditText
    android:layout_height="wrap_content"
    android:textSize="16sp"
    android:lines="1"
    android:id="@+id/TextInput"
    android:layout_width="fill_parent"
    android:hint="Input a temperature"
    android:inputType="numberDecimal"
  >
  </EditText>

</LinearLayout>
```

## Adding more views

In the same way as described above, add the following using drag and drop.

- A RadioGroup (a layout)
- In the RadioGroup, two RadioButtons. Make sure you drop them inside the RadioGroup (check the xml-code)
- A button
- A TextView (for the output)

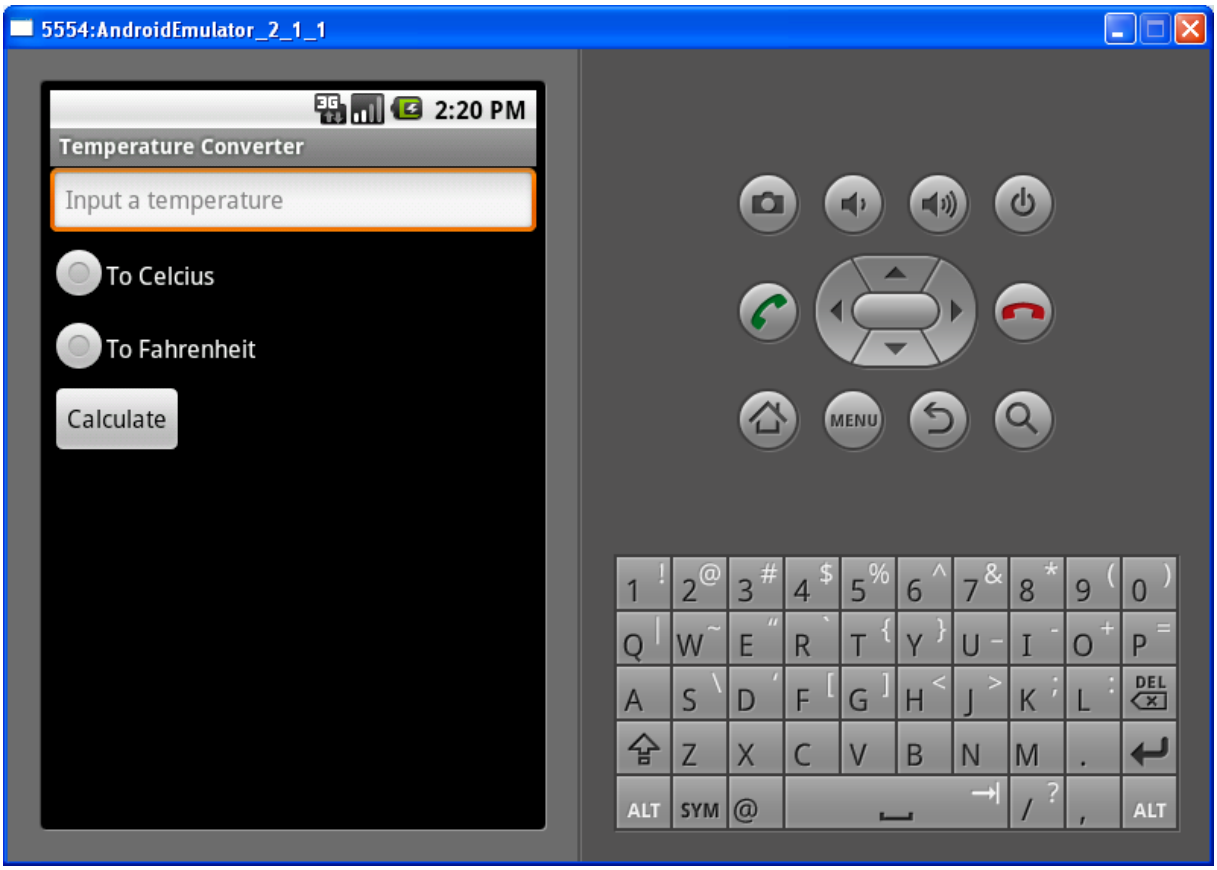
For the RadioButtons, set the Id, Text and Text size properties to "@+id/CelciusButton","To Fahrenheit", "16sp" and "@+id/FahrenheitButton", "To Celcius" , "16 sp" respectively.
Set the same properties for the Button to "@+id/CalculateButton","Calculate" and "16sp" respectively.
For the TextView, set "@+id/TextOutput","" and "16sp".

The XML-code should, more or less, look like below. Make sure the RadioButtons are defined inside the RadioGroup.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
>
  <EditText
    android:layout_height="wrap_content"
    android:textSize="16sp"
    android:lines="1"
    android:id="@+id/TextInput"
    android:layout_width="fill_parent"
    android:hint="Input a temperature"
    android:inputType="numberDecimal"
  >
  </EditText>

  <RadioGroup
    android:id="@+id/RadioGroup01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
  >
    <RadioButton
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textSize="16sp"
      android:text="To Celcius"
      android:id="@+id/CelciusButton"
    >
    </RadioButton>

    <RadioButton
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/FahrenheitButton"
      android:text="To Fahrenheit"
      android:textSize="16sp"
    >
    </RadioButton>
  </RadioGroup>

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="16sp"
    android:text="Calculate"
    android:id="@+id/CalculateButton"
  >
  </Button>

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/TextOutput"
    android:textSize="16sp"
  >
  </TextView>

</LinearLayout>
```

## Run the application

To check out what your UI looks like, run the application; right-click the project header and select Run As…/Android Application. The UI should look like below.




At runtime, the user interface components are created from the information in the XML-file.
The line

```
setContentView(R.layout.main);
```

in the TemperatureActivity.java tells the application which layout file to use (the name of our layout file is main.xml).
You can have multiple layout files in our application and thus define multiple layouts.

## Accessing UI components in the source code

You can access the UI components, defined in the XML-file, from your application code through their id's (e.g. "@+id/*TextInput*" for the EditText component) and the findViewById-method.

Open TemperatureActivity.java and add a member representing the ExitText and then call findViewById to get a reference to the component.

```
public class TemperatureActivity extends Activity {
  private EditText textInput;

  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    textInput = (EditText) findViewById(R.id.TextInput);
  }
```

Add code to reference the buttons as well.

We then have to define what should happen when the calculate button is pressed (get the input from the EditText and convert the value to degrees Celsius or Fahrenheit). This is done in a class implementing the call back interface View.OnClickListener. Below you find the complete code.

```
public class TemperatureActivity extends Activity {

   private EditText textInput ;
   private RadioButton celciusButton;
   private RadioButton fahrenheitButton;
   private Button calculateButton;
   private TextView textOutput;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      textInput  = (EditText) findViewById(R.id.TextInput);
      celciusButton = (RadioButton) findViewById(R.id.CelciusButton);
      fahrenheitButton = (RadioButton) findViewById(R.id.FahrenheitButton);
      calculateButton = (Button) findViewById(R.id.CalculateButton);
      textOutput = (TextView) findViewById(R.id.TextOutput);

      // Create a listener instance and associate it with the button
      OnClickListener listener = new ButtonListener();
      calculateButton.setOnClickListener(listener);
   }

   /**
    * This class defines the action to take when the calculate button
    * is clicked, in this case by calling buttonClickHandler().
    */
   private class ButtonListener implements OnClickListener {
      @Override
      public void onClick(View v) {
         buttonClickHandler();
      }
   }

   private void buttonClickHandler() {
      String text = textInput.getText().toString();
      if(text.length() == 0) {
         showToast("Please enter a a valid number");
         return;
      }

      float value = Float.parseFloat(text);
      String result = "";
      if(celciusButton.isChecked()) {
         float c = convertFahrenheitToCelsius(value);
         result = "" + c + " Celsius";
      }
      else if(fahrenheitButton.isChecked()) {
         float f = convertCelsiusToFahrenheit(value);
         result = "" + f + " Fahrenheit";
      }
      else {
         showToast("Please select a radio button");
```

```
            return;
        }

        textOutput.setText(result);
    }

    private float convertFahrenheitToCelsius(float fahrenheit) {
        return ((fahrenheit - 32) * 5 / 9);
    }

    private float convertCelsiusToFahrenheit(float celsius) {
        return ((celsius * 9) / 5) + 32;
    }

    private void showToast(String msg) {
        Toast toast = Toast.makeText(this, msg, Toast.LENGTH_SHORT);
        toast.show();
    }
}
```

## Where to go from here…

I recommend you to continue with a more elaborate tutorial, the Note Pad Tutorial (1-3) at http://developer.android.com/resources/tutorials/notepad/index.html .

There are a lot of good tutorials on user interfaces, and other topics, at http://developer.android.com/resources/tutorials/ .

## Declaring the call back method in the XML-file

Instead of defining an OnClickListener in your code, you may reference the method to call by setting the onClick property for the button.

In this case, we want to reference the buttonClickHandler method from the XML-file. In the layout manager, select the button and set the property “onClick” to “buttonClickHandler”.

In this case, the method must have public visibility, and an argument of type view, i.e.

```
public void buttonClickHandler(View view) {…
```

There’s no need for an explicit OnClickListner in this case; remove the lines defining the class implementing the OnClickListener interface.

```
public class TemperatureActivity2 extends Activity {

   private EditText textInput ;
   private RadioButton celciusButton;
   private RadioButton fahrenheitButton;
   private Button calculateButton;
   private TextView textOutput;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      textInput  = (EditText) findViewById(R.id.TextInput);
      celciusButton = (RadioButton) findViewById(R.id.CelciusButton);
      fahrenheitButton = (RadioButton) findViewById(R.id.FahrenheitButton);
      calculateButton = (Button) findViewById(R.id.CalculateButton);
      textOutput = (TextView) findViewById(R.id.TextOutput);
   }

   /**
    * Because we set the property onClick for the Button to reference this
    * method we don't have to implement the OnClickListener explicitly.
    * NB: The method to handle the event must not be private and
    * must have an argument of type View.
    */
   public void buttonClickHandler(View view) {
      String text = textInput.getText().toString();
      if(text.length() == 0) {
         showToast("Please enter a a valid number");
         return;
      }
(…continues as in previous example…)
```

The project TemperatureConverter2 contains the complete implementation of this version of the application.

## Multiple layouts or other resources

To add additional XML-files defining layouts, or other resources, right click on the project header and select Android Tools…/New Resource File.