**CSCI 322/522 3. Creating a First Android App**

* The preceding chapter covered the steps necessary to configure an environment suitable for the development of Android applications using the Android Studio IDE.
* Before moving on to slightly more advanced topics, now is a good time to validate that all of the required development packages are installed and functioning correctly.
* The best way to achieve this goal is to create an Android application and compile and run it.
* This chapter will cover the creation of an Android application project using Android Studio.
* Once the project has been created, a later chapter will explore the use of the Android emulator environment to perform a test run of the application.

**About the Project**

* The project created in this chapter takes the form of a rudimentary currency conversion calculator (so simple, in fact, that it only converts from Dollars to Euros using an estimated conversion rate).
* The project will also make use of one of the most basic of Android Studio project templates.
* This simplicity allows us to introduce some of the key aspects of Android app development without overwhelming the beginner by trying to introduce too many concepts, such as the recommended app architecture and Android architecture components, at once.
* When following the tutorial in this chapter, rest assured that all of the techniques and code used in this initial example project will be covered in much greater detail in later chapters.
* Note that the finished project – with a few changes – is included in the chapter's code examples on Blackboard.

**Creating a New Android Project**

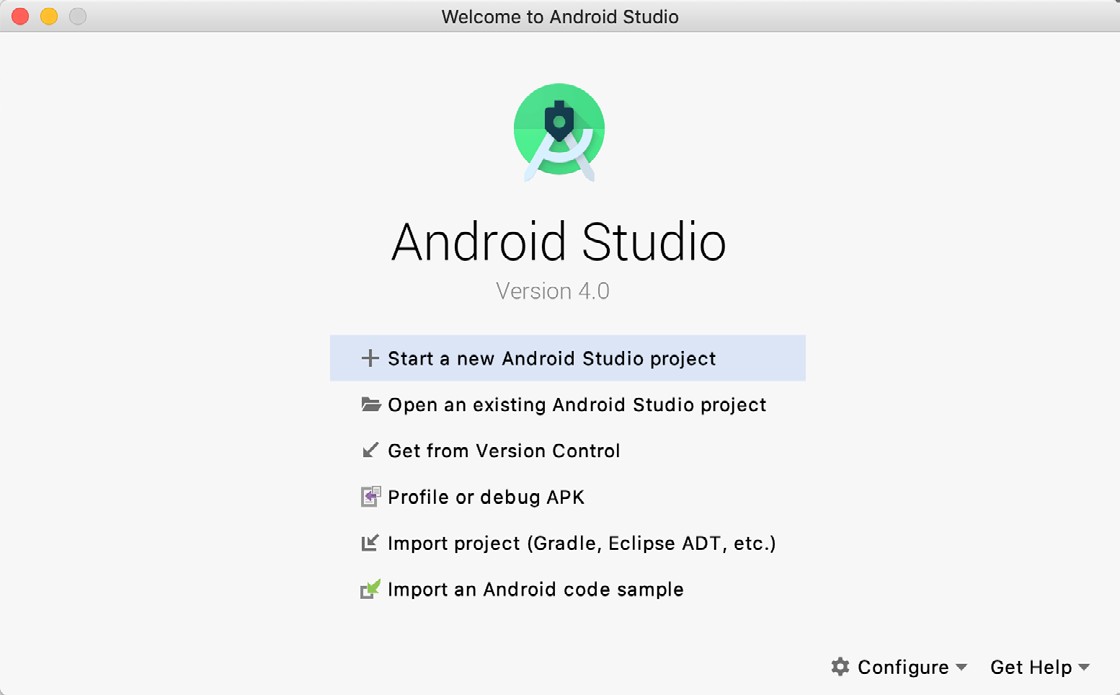
* The first step in the application development process is to create a new project within the Android Studio environment.
* Begin by launhing Android Studio so that the *Welcome to Android Studio* dialog appears as illustrate[d in Figure 3-1:](#_bookmark20)

Figure 3-1

* Once this window appears, Android Studio is ready for a new project to be created.
* To create the new project, simply click on the Start a new Android Studio project option to display the first screen of the New Project wizard.

**Creating an Activity**

* The first step is to define the type of initial activity that is to be created for the application.
* Options are available to create projects for Phone and Tablet, Wear OS, TV, Android Audio or Android Things.
* A range of different activity types is available when developing Android applications, many of which will be covered extensively in later chapters.
* For the purposes of this example, however, simply select the option to create an Empty Activity on the Phone and Tablet screen.
* The Empty Activity option creates a template user interface consisting of a single TextView object.

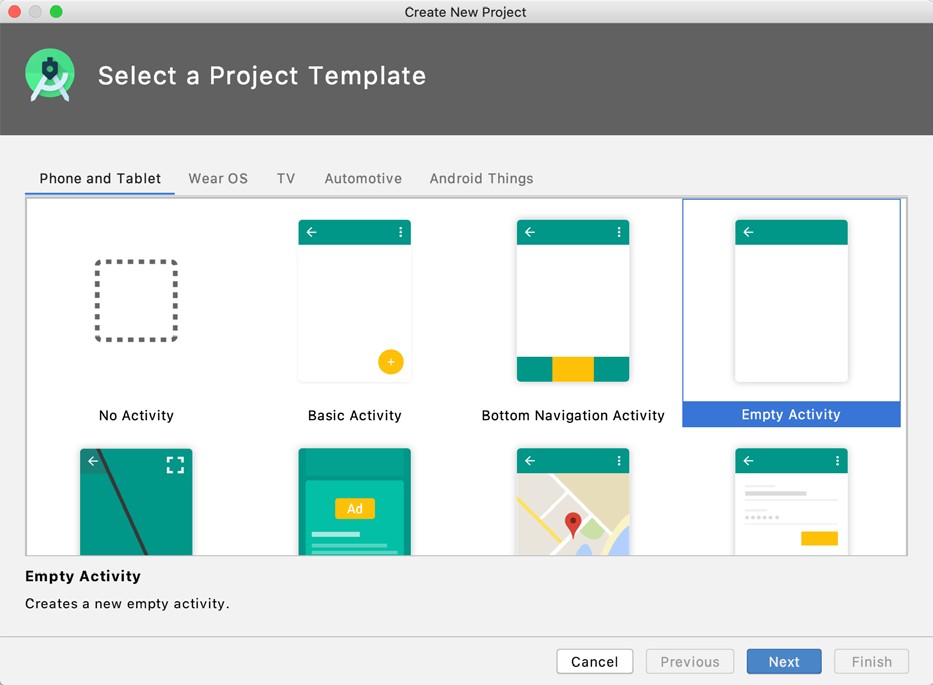


Figure 3-2

* With the Empty Activity option selected, click Next to continue with the project configuration.

**Defining the Project and SDK Settings**

* In the project configuration window [(Figure](#_bookmark23) 3-3), set the Name field to Dollars to Euros.
* The application name is the name by which the application will be referenced and identified within Android Studio and is also the name that would be used if the completed application were to go on sale in the Google Play store.
* The Package name is used to uniquely identify the application within the Android application ecosystem.
* Although this can be set to any string that uniquely identifies your app, it is traditionally based on the reversed URL of your domain name followed by the name of the application.
* For example, if your domain is [www.](http://www/)mycompany.com, and the application has been named Dollarstoeuros, then the package name might be specified as follows:

com.mycompany.dollarstoeuros

* For all projects in CSCI 322/522, use the package name:

edu.niu.*your z-id*.*assignment or in-class exercise number*

* The Save location setting will default to a location in the folder named AndroidStudioProjects located in your home directory and may be changed by clicking on the folder icon to the right of the text field containing the current path setting.
* Set the minimum SDK setting to API 16: Android 4.1 (Jelly Bean). (This older version will run on 99.8% of all devices out there.)
* This is the SDK that will be used in most of our projects unless a necessary feature is only available in a more recent version.

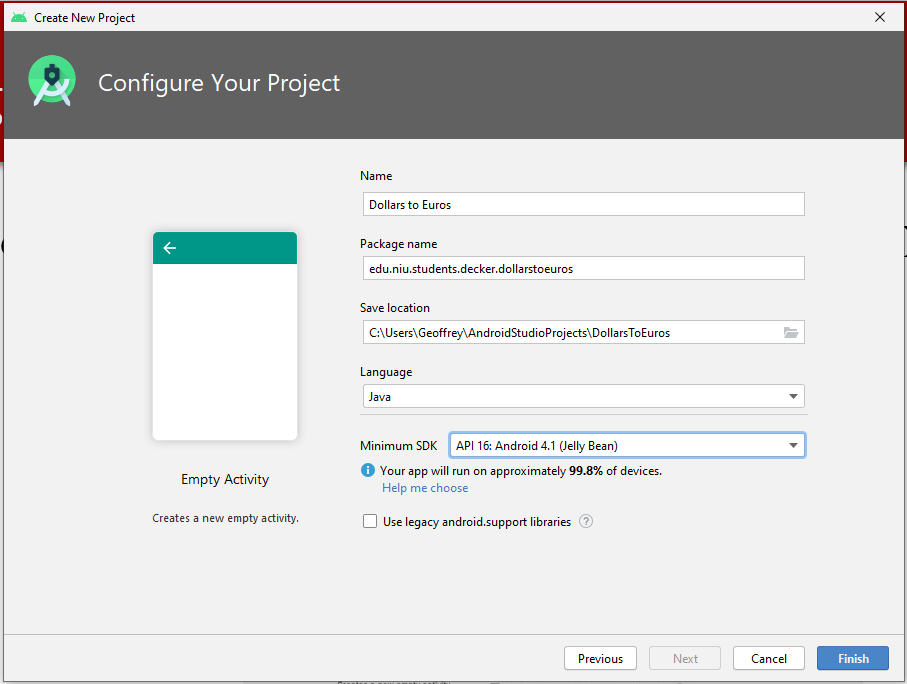


Figure 3-3

* Finally, change the Language menu to Java and make sure the Use legacy android.support libraries option is turned off before clicking on Finish to initiate the project creation process.

**Modifying the Example Application**

* At this point, Android Studio has created a minimal example application project and opened the main window.

A screenshot of a cell phone

Description automatically generated

Figure 3-4

* The newly created project and references to associated files are listed in the Project tool window located on the left-hand side of the main project window.
* The Project tool window has a number of modes in which information can be displayed.
* By default, this panel should be in Android mode.
* This setting is controlled by the menu at the top of the panel as highlighted in [Figure 3-5.](#_bookmark25)
* If the panel is not currently in Android mode, use the menu to switch mode:

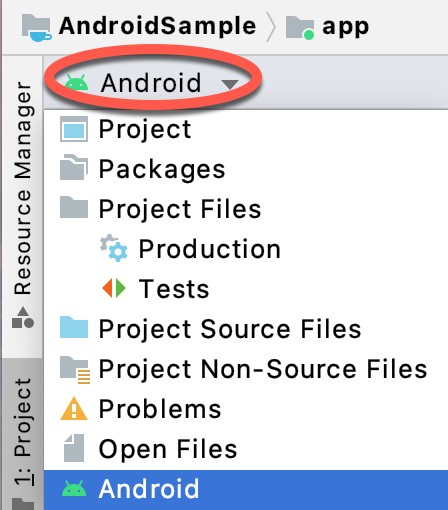


Figure 3-5

**Modifying the User Interface**

* The user interface design for our activity is stored in a file named activity\_main.xml which, in turn, is located under app > res > layout in the project file hierarchy.
* Once located in the Project tool window, double-click on the file to load it into the user interface Layout Editor tool which will appear in the center panel of the Android Studio main window:

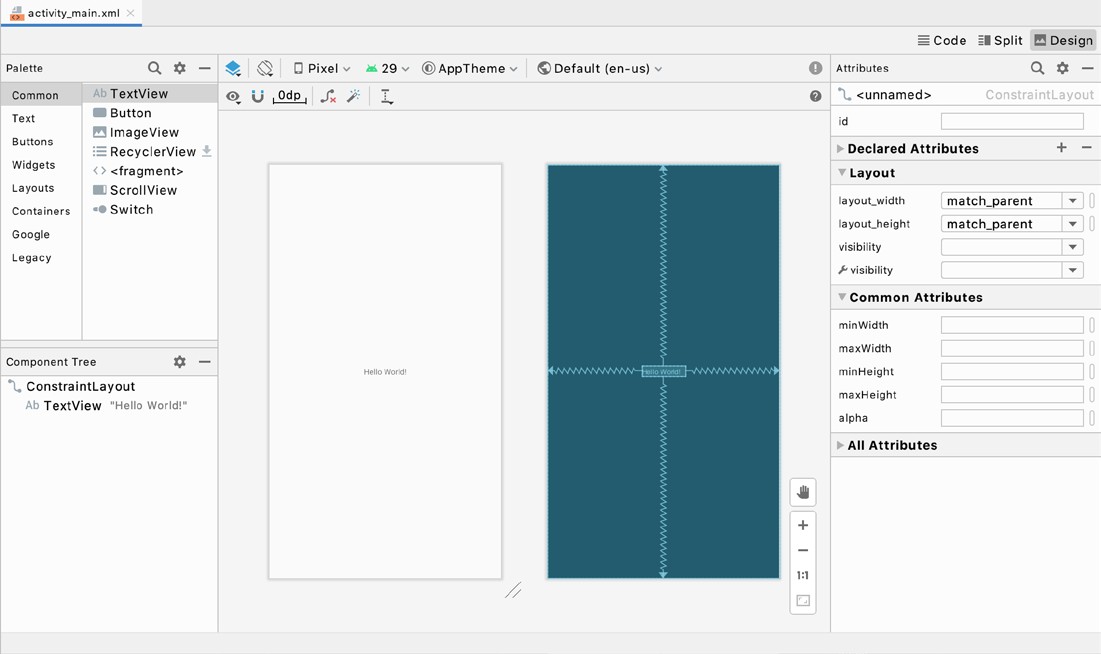


Figure 3-6

* In the toolbar across the top of the Layout Editor window is a menu (currently set to Pixel in the above figure) which is reflected in the visual representation of the device within the Layout Editor panel.
* A wide range of other device options are available for selection by clicking on this menu.
* To change the orientation of the device representation between landscape and portrait simply use the drop down menu immediately to the left of the device selection menu showing the  icon.
* As can be seen in the device screen, the content layout already includes a label that displays a “Hello World!” message.
* Running down the left-hand side of the panel is a palette containing different categories of user interface components that may be used to construct a user interface, such as buttons, labels and text fields.
* It should be noted, however, that not all user interface components are obviously visible to the user.
* One such category consists of layouts.
* Android supports a variety of layouts that provide different levels of control over how visual user interface components are positioned and managed on the screen.
* Though it is difficult to tell from looking at the visual representation of the user interface, the current design has been created using a ConstraintLayout.

**Creating an Example Android App in Android Studio**

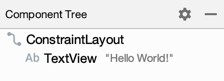
* This can be confirmed by reviewing the information in the Component Tree panel which, by default, is located in the lower left-hand corner of the Layout Editor panel and is sho[wn in Figure 3-7:](#_bookmark30)

Figure 3-7

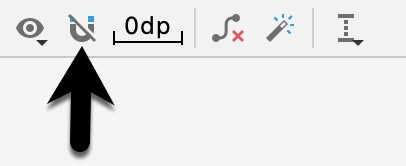
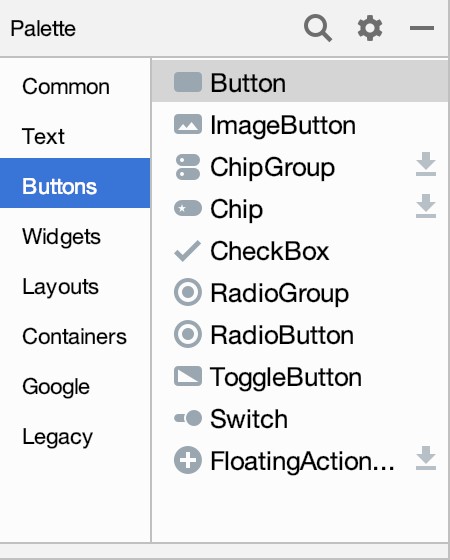
* As we can see from the component tree hierarchy, the user interface layout consists of a ConstraintLayout parent and a TextView child object.
* Before proceeding, also check that the Layout Editor’s Autoconnect mode is enabled.
* This means that as components are added to the layout, the Layout Editor will automatically add constraints to make sure the components are correctly positioned for different screen sizes and device orientations (a topic that will be covered in much greater detail in future chapters).
* The Autoconnect button appears in the Layout Editor toolbar and is represented by a magnet icon.
* When disabled the magnet appears with a diagonal line through it [(Figure](#_bookmark29) [3-8).](#_bookmark29)
* If necessary, re-enable Autoconnect mode by clicking on this button.

Figure 3-8

* The next step in modifying the application is to add some additional components to the layout, the first of which will be a Button for the user to press to initiate the currency conversion.
* The Palette panel consists of two columns with the left-hand column containing a list of view component categories.
* The right-hand column lists the components contained within the currently selected category.
* In [Figure 3-9,](#_bookmark28) for example, the Button view is currently selected within the Buttons category.

Figure 3-9

* Click and drag the Button object from the Buttons list and drop it in the horizontal center of the user interface design so that it is positioned beneath the existing TextView widget:

Figure 3-10

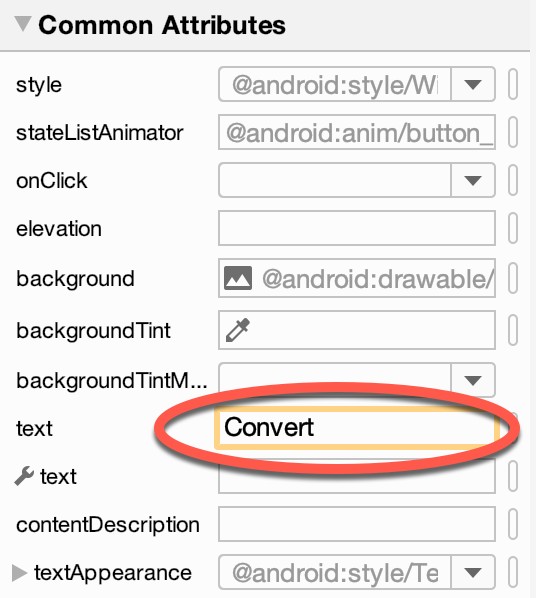
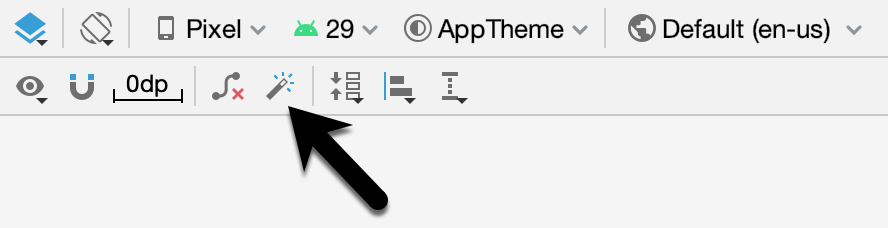
* The next step is to change the text that is currently displayed by the Button component.
* The panel located to the right of the design area is the Attributes panel.
* This panel displays the attributes assigned to the currently selected component in the layout.
* Within this panel, locate the text property in the Common Attributes section and change the current value from “Button” to “Convert” as sho[wn in Figure 3-11:](#_bookmark31)

Figure 3-11

* The second text property with a wrench next to it allows a text property to be set which only appears within the Layout Editor tool but is not shown at runtime.
* This is useful for testing the way in which a visual component and the layout will behave with different settings without having to run the app repeatedly.
* Just in case the Autoconnect system failed to set all of the layout connections, click on the Infer constraints butto[n (Figure 3-12)](#_bookmark35) to add any missing constraints to the layout:



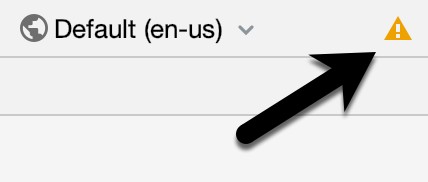
* Figure 3-12
* At this point it is important to explain the warning button located in the top right-hand corner of the Layout Editor tool as indicated in [Figure 3-13.](#_bookmark34)
* Obviously, this is indicating potential problems with the layout. For details on any problems, click on the button:

Figure 3-13

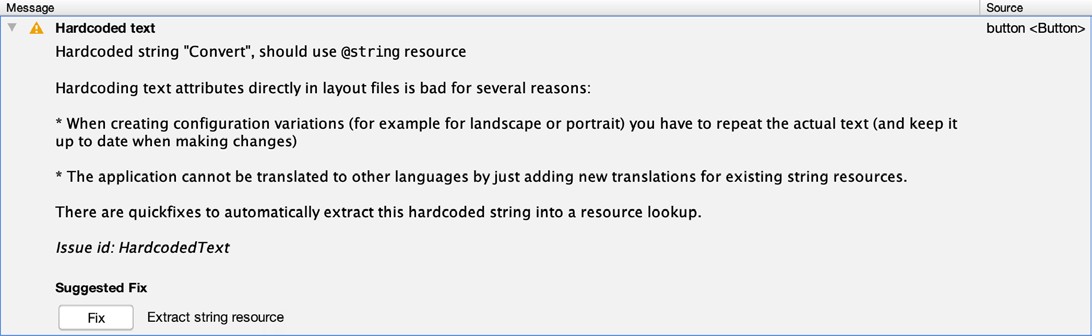
* When clicked, a panel [(Figure 3-14)](#_bookmark33) will appear describing the nature of the problems and offering some possible corrective measures:

Figure 3-14

* Currently, the only warning listed reads as follows:

Hardcoded string "Convert", should use @string resource

* This I18N message is informing us that a potential issue exists with regard to the future internationalization of the project (“I18N” comes from the fact that the word “internationalization” begins with an “I”, ends with an “N” and has 18 letters in between).
* The warning is reminding us that when developing Android applications, attributes and values such as text strings should be stored in the form of resources wherever possible.
* Doing so enables changes to the appearance of the application to be made by modifying resource files instead of changing the application source code.
* This can be especially valuable when translating a user interface to a different spoken language.
* If all of the text in a user interface is contained in a single resource file, for example, that file can be given to a translator who will then perform the translation work and return the translated file for inclusion in the application.
* This enables multiple languages to be targeted without the necessity for any source code changes to be made. In this instance, we are going to create a new resource named convert\_string and assign to it the string “Convert”.
* Click on the Fix button in the Issue Explanation panel to display the Extract Resource panel [(Figure 3-15).](#_bookmark37)

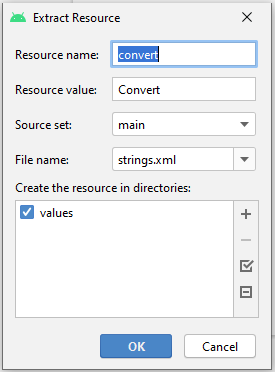


Figure 3-15

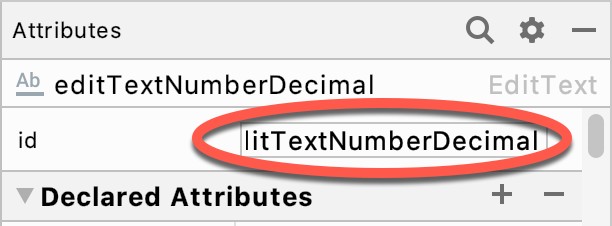
* Within this panel, change the resource name field to convert\_string and leave the resource value set to Convert before clicking on the OK button.
* It is also worth noting that the string could also have been assigned to a resource when it was entered into the Attributes panel.
* This involves clicking on the narrow button to the right of the property field in the Attributes panel and selecting the Add new resource > New String Value… menu option from the resulting Resources dialog.
* In practice, however, it is often quicker to simply set values directly into the Attributes panel fields for any widgets in the layout, then work sequentially through the list in the warnings dialog to extract any necessary resources when the layout is complete.
* The next widget to be added is an EditText widget into which the user will enter the dollar amount to be converted.
* From the widget palette, select the Text category and click and drag a Number (Decimal) component onto the layout so that it is centered horizontally and positioned above the existing TextView widget.
* With the widget selected, use the Attributes tools window to set the hint property to “Dollars to Euros”.
* Click on the warning icon and extract the string to a resource named dollars\_hint.
* The code written later in this chapter will need to access the dollar value entered by the user into the EditText field.
* It will do this by referencing the id assigned to the widget in the user interface layout.
* The default id assigned to the widget by Android Studio can be viewed and changed from within the Attributes tool window when the widget is selected in the layout as sho[wn in Figure 3-16:](#_bookmark36)

Figure 3-16

* Change the id to dollarText before proceeding.
* Add any missing layout constraints by clicking on the Infer constraints button. At this point the layout should resemble that sho[wn in Figure 3-17:](#_bookmark39)

Figure 3-17

**Reviewing the Layout and Resource Files**

* Before moving on to the next step, we are going to look at some of the internal aspects of user interface design and resource handling.
* In the previous section, we made some changes to the user interface by modifying the activity\_main.xml file using the Layout Editor tool.
* In fact, all that the Layout Editor was doing was providing a user-friendly way to edit the underlying XML content of the file.
* In practice, there is no reason why you cannot modify the XML directly in order to make user interface changes and, in some instances, this may actually be quicker than using the Layout Editor tool.
* In the top right-hand corner of the Layout Editor panel are three buttons as highlight[ed in Figure 3-18](#_bookmark38) below:

Figure 3-18

* By default, the editor will be in Design mode whereby just the visual representation of the layout is displayed.
* The left-most button will switch to Code mode to display the XML for the layout, while the middle button enters Split mode where both the layout and XML are displayed, as sho[wn in Figure 3-19:](#_bookmark41)

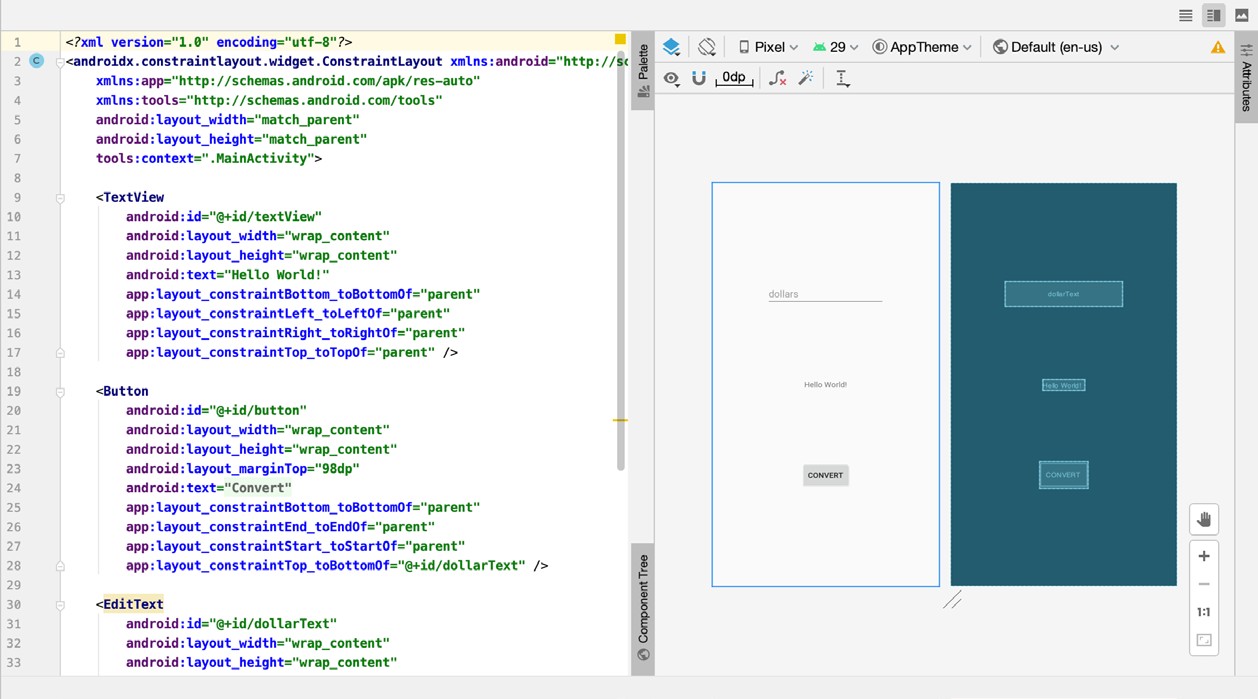


Figure 3-19

* As can be seen from the structure of the XML file, the user interface consists of the ConstraintLayout component, which in turn, is the parent of the TextView, Button and EditText objects.
* We can also see, for example, that the text property of the Button is set to our convert\_string resource.
* Although varying in complexity and content, all user interface layouts are structured in this hierarchical, XML based way.
* As changes are made to the XML layout, these will be reflected in the layout canvas.
* The layout may also be modified visually from within the layout canvas panel with the changes appearing in the XML listing.
* To see this in action, switch to Split mode and modify the XML layout to change the background color of the ConstraintLayout to a shade of red as follows:

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

<androidx.constraintlayout.widget.ConstraintLayout

xmlns:android="http://schemas.android.com/apk/res/android"

[xmlns:app="http://schemas.android.com/apk/res-auto"](http://schemas.android.com/apk/res-auto)

[xmlns:tools="http://schemas.android.com/tools"](http://schemas.android.com/tools)

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context=".MainActivity" android:background="#ff2438" >

.

.

</androidx.constraintlayout.widget.ConstraintLayout>

* Note that the color of the layout changes in real-time to match the new setting in the XML file.
* Note also that a small red square appears in the left-hand margin (also referred to as the gutter) of the XML editor next to the line containing the color setting.
* This is a visual cue to the fact that the color red has been set on a property.
* Before proceeding, delete the background property from the layout file so that the background returns to the default setting.
* Finally, use the Project panel to locate the **app > res > values > strings.xml** file and double-click on it to load it into the editor.
* Currently the XML should read as follows:

<resources>

<string name="app\_name">Dollars to Euros</string>

<string name="convert\_string">Convert</string>

<string name="dollars\_hint">Dollars to Euros</string>

</resources>

* As a demonstration of resources in action, change the string value currently assigned to the convert\_string resource to “Convert to Euros”.
* Then return to the Layout Editor tool by selecting the tab for the layout file in the editor panel.
* Note that the layout has picked up the new resource value for the string.
* There is also a quick way to access the value of a resource referenced in an XML file.
* With the Layout Editor tool in Split or Code mode, click on the “@string/convert\_string” property setting so that it highlights and then press Ctrl-B on the keyboard (Cmd-B on macOS).
* Android Studio will subsequently open the strings.xml file and take you to the line in that file where this resource is declared.
* Use this opportunity to revert the string resource back to the original “Convert” text and to add the following additional entry for a string resource that will be referenced later in the app code:

<resources>

.

.

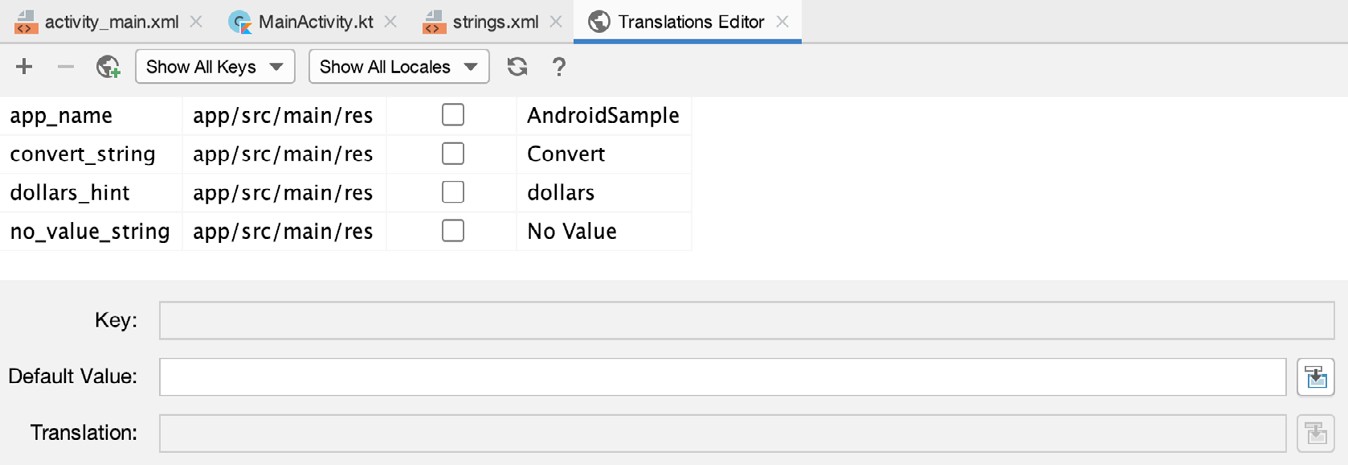
<string name="convert\_string">Convert</string>

<string name="dollars\_hint">Dollars to Euros</string>

<string name="no\_value\_string">No Value</string>

</resources>

* Resource strings may also be edited using the Android Studio Translations Editor.
* To open this editor, right- click on the app > res > values > strings.xml file and select the Open Translations Editor menu option.
* This will display the Translation Editor in the main panel of the Android Studio window:

Figure 3-20

* This editor allows the strings assigned to resource keys to be edited and for translations for multiple languages to be managed.

**Adding Interaction**

* The final step in this example project is to make the app interactive so that when the user enters a dollar value into the EditText field and clicks the convert button the converted euro value appears on the TextView.
* This involves the implementation of some event handling on the Button widget.
* Specifically, the Button needs to be configured so that a method in the app code is called when an onClick event is triggered.
* Event handling can be implemented in a number of different ways and is covered in detail in a later chapter entitled [“An Overview and](#_bookmark346) [Example of Android Event Handling”](#_bookmark346).
* Return the layout editor to Design mode, select the Button widget in the layout editor, refer to the Attributes tool window and specify a method named convertCurrency as shown below:

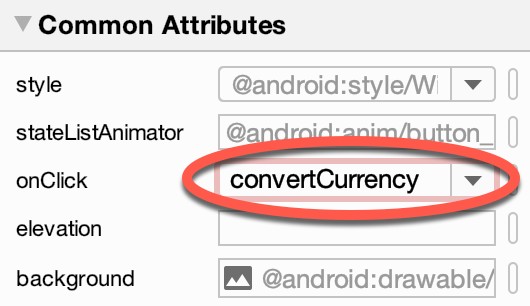


Figure 3-21

* Note that the text field for the onClick property is now highlighted with a red border to warn us that the button has been configured to call a method which does not yet exist.
* To address this, double-click on the MainActivity.java file in the Project tool window (app > java > <package name> > MainActivity) to load it into the code editor and add the code for the convertCurrency method to the class file so that it reads as follows, noting that it is also necessary to import some additional Android packages:

package com.example.dollarstoeuros;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

import android.view.View;

import android.widget.EditText;

import android.widget.TextView;

public class MainActivity extends AppCompatActivity

{

@Override

protected void onCreate(Bundle savedInstanceState)

{

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

}

public void convertCurrency(View view)

{

EditText dollarText = findViewById(R.id.dollarText);

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

if (!dollarText.getText().toString().equals(""))

{

float dollarValue = Float.valueOf(dollarText.getText().toString());

float euroValue = dollarValue \* 0.84F;

textView.setText(String.format(Locale.getDefault(), "%.2f Euros",

euroValue));

}

else

{

textView.setText(R.string.no\_value\_string);

}

}

}

* The method begins by obtaining references to the EditText and TextView objects by making a call to a method named findViewById, passing through the id assigned within the layout file.
* A check is then made to ensure that the user has entered a dollar value and if so, that value is extracted, converted from a String to a floating point value to two decimal places and converted to euros.
* Finally, the result is displayed on the TextView widget.
* If any of this is unclear, rest assured that these concepts will be covered in greater detail in later chapters.

**Summary**

* While not excessively complex, a number of steps are involved in setting up an Android development environment.
* Having performed those steps, it is worth working through an example to make sure the environment is correctly installed and configured.
* In this chapter, we have created an example application and then used the Android Studio Layout Editor tool to modify the user interface layout.
* In doing so, we explored the importance of using resources wherever possible, particularly in the case of string values, and briefly touched on the topic of layouts.
* Next we looked at the underlying XML that is used to store the user interface designs of Android applications.
* Finally, an onClick event was added to a Button connected to a method that was implemented to extract the user input from the EditText component, convert from dollars to euros and then display the result on the TextView.
* With the app ready for testing, the steps necessary to set up an emulator for testing purposes will be covered in detail in the next chapter.