

COMPONENT IDS

- Here is the line of code from themain.xml file:
android:id="@+id/centertext"
The @ sign in the ID is a signal to the XML parser how to deal with the ID string. The + indicates that this is an ID the user has created; it is not part of the Android framework namespace.
- Suppose we write an application with a text field where the user would enter her name. The programmer would need a way to identify that particular text field to extract the entered name to do something with it. There must be a way to link a component from the static design of the screen during development (the XML file) to the application at run-time (the Java code).
- In the Java code, we declare and assign an instance of an object matching the class of the element in the XML.
- It is a good idea to put an android:id attribute on your layouts. For example, you could identify the lowest level layout, the one represented with the first opening tag, as "base." When we added two TableLayouts to the underlying LinearLayout, the LinearLayout could be IDed as "base," the first TableLayout as level2a or table_1.

A FEW SIMPLE CONTROLS

- So far, we have used TextView objects to display some simple text. Two more basic but useful controls, or widgets as they are called, are the EditText and the Button.
- Java programmers are already familiar with Buttons, and EditText controls are similar to TextFields. (The Android TextView is similar to the Java Label class.)
- Let's begin a simple application that will take the contents of a predefined TextView and use a button to cause the application to take that text, convert it to uppercase, and display it in an EditText field.
- A basic main.xml file could look like the following:

```
<?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"
```

```

android:id="@+id/base"
>
<TextView
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="My first android application"
android:id="@+id/my_TextView"
/>
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Touch me"
android:id="@+id/my_Button"
/>

<EditText
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:id="@+id/my_EditText"
/>
</LinearLayout>

```

- Look at how the id attributes are interpreted in the

R.java file:

```

/* AUTO-GENERATED FILE. DO NOT MODIFY.
*
* This class was automatically generated by the
* aapt tool from the resource data it found. It
* should not be modified by hand.
*/
package com.sheusi.DemoApp;
public final class R {
public static final class attr {
}
public static final class drawable {
public static final int icon=0x7f020000;
}
public static final class id {
public static final int base=0x7f060000;
public static final int my_Button=0x7f060002;
public static final int my_EditText=0x7f060003;
public static final int my_TextView=0x7f060001;
}

```

```

public static final class layout {
public static final int main=0x7f030000;
}
public static final class string {
public static final int app_name=0x7f040001;
public static final int hello=0x7f040000;
}
public static final class style {
public static final int shout=0x7f050000;
}
}

```

- Our first step is to decide what our application is designed to do. This is our model in the MVC paradigm.
- MVC is the separation of model, view and controller — nothing more, nothing less. It's simply a **paradigm**; an ideal that you should have in the back of your mind when designing classes. Avoid mixing code from the three categories into one class.
- As stated, our application is meant to take the contents of a text field, convert them to uppercase, and place the results in another text field.
- Our view includes the original text in a text field, a button to start the conversion, and a text field to hold the results.
- Let's look at the Java code and peel it apart:

```

package com.sheusi.DemoApp;
import android.app.Activity;
import android.os.Bundle;
import android.view.*;
import android.widget.*;
import android.view.View.OnClickListener;
public class Demo extends Activity implements OnClickListener{
/** Called when the activity is first created. */
Button b=null;
EditText et=null;
TextView tv=null;
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
b=(Button)findViewById(R.id.my_Button);
et=(EditText)findViewById(R.id.my_EditText);
tv=(TextView)findViewById(R.id.my_TextView);
b.setOnClickListener(this);
}
}

```

```

}
public void onClick(View v){
String temp=tv.getText().toString();
temp=temp.toUpperCase();
et.setText(temp);
}
}

```

- The first statement starts with the word package.
package com.sheusi.DemoApp;
You chose the package name when you created the project.
- Next, notice the statements that begin with the word import.
Java, like other object oriented languages, uses groups of predefined classes called packages.
- Packages generally have multipart names separated by periods, such as android.view.View.Button.
- When packages are built, they have a tree-style organization, and the periods let the Java compiler navigate through the structure.
- The import statements are a convenience that relieves us from having to type the whole package name each time we use a class from the package.
- Here is an example.
Without the import statement, if we wanted to declare and assign a couple of buttons, we would need these statements:
android.widget.Button b1;
android.widget.Button b2;
b1= new android.widget.Button("click me");
b2= new android.widget.Button("quit");
- Using the import statement,
import android.widget.Button;
our declarations and assignments need only the class name:
Button b1;
Button b2;
b1= new Button("click me");
b2= new Button("quit");
- To indicate all classes in a given package, we can use an asterisk (*) as a wild card. For example:
import android.widget.*;
- How do we know which packages to include?

Whatever reference you use, the class documentation will always tell you the package name. You simply write an import statement for that particular package.

- The next line names the public class. Again, you provided the name for the public class when you created the project:
public class Demo **extends** Activity **implements** OnClickListener{
- The way classes are structured in terms of syntax requires knowledge of Java or C++.
- The core of an Android application is built on the Activity class. When we create an application, we customize the Activity class, or extend it in Java-ese.
- We add the words implements OnClickListener as we customize the application.
- The class OnClickListener, as the name implies, lets our application “listen” or check for actions on the user interface, namely the Android device screens, hardware buttons, and so on.
- Only certain objects can send click messages, but the Android device knows what these are, and the Android application programming interface (API) tells the programmer what they are.
- The next three lines declare the text areas and Button for the application’s user interface.
- A TextView is similar to a Java Label class in that it contains text but cannot be edited by simply typing into it at run-time. For this, we use the EditText class.
- The next section is where we define whatever we want to happen when the application starts, including laying out the screen, setting initial values to variables, and so on.
- In our application, we assign values to our control (widget) objects, and we connect the application’s “listener” code to the Button:

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.main);  
    b=(Button)findViewById(R.id.my_Button);  
    et=(EditText)findViewById(R.id.my_EditText);  
    tv=(TextView)findViewById(R.id.my_TextView);  
    b.setOnClickListener(this);  
}
```

- Notice all the statements that include R.id. You will recognize these as the id attribute values we added to the sections of the main.xml file.
- We then looked at how they manifested themselves in the R.java file. These statements connect our screen objects to the Java code.
- Finally, the Button variable, b, connects the listener code.


```
public void onClick(View v){
String temp=tv.getText().toString();
temp=temp.toUpperCase();
et.setText(temp);
}
```
- This last section defines what we want to happen when we touch the button on the screen.
- In plain English, we create a text string and assign to it the contents of the TextView on the screen. The TextView got its original text from the android:text statement in the main.xml file.
- Then, through a built-in method of Java's String class, we convert all the letters to uppercase.
- Finally, we assign the converted text string to the text property of the EditText field.
- The emulator should look like Figure 3.6 at start-up.



Figure 3.6
Emulator image of the application designed earlier.

- To round out this simple application, let's add a second Button to close the application.
- Go back to the main.xml file and add another button directly below the EditText set of tags.
- Set the android:text attribute to Quit and the android:id to @+id/quit

```

package com.sheusi.DemoApp;
import android.app.Activity;
import android.os.Bundle;
import android.view.*;
import android.widget.*;
import android.view.View.OnClickListener;
public class Demo extends Activity implements OnClickListener{
/** Called when the activity is first created. */
    Button b=null;

    EditText et=null;
    TextView tv=null;
    Button quitbutton=null;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        b=(Button)findViewById(R.id.my_Button);
        quitbutton=(Button)findViewById(R.id.quit);
        et=(EditText)findViewById(R.id.my_EditText);
        tv=(TextView)findViewById(R.id.my_TextView);
        b.setOnClickListener(this);
        quitbutton.setOnClickListener(this);
    }
    public void onClick(View v){
        if(v==b){
            String temp=tv.getText().toString();
            temp=temp.toUpperCase();
            et.setText(temp);
        }
        if(v==quitbutton){
            this.finish();
        }
    }
}

```

- We have declared an additional Button object and assigned it using the same syntax as the original Button object.
- We have also assigned the same OnClickListener to the second Button.
- It may strike you that if we assign the same listener, the application would not know which Button was touched or clicked. We solve that problem with decision statements based on the parameter View v in the onClick() statement.
- Because the Button class is a child class of the View class in the Android SDK, the parameter can represent the buttons. The decision statements merely assess which Button was clicked or touched.
- The .finish() method is a method of the Activity class that ends the activity.
- Because our application extends the Activity class, we can use this method to end the application.
- When working with EditText controls on the screen, you may want the application to respond to a particular keystroke rather than requiring the user to touch a button or take some other action. For instance, you may want to respond to keypad input as soon as the user touches the Enter key. We can do that by associating a KeyListener with a given input field. We accomplish that by making some modifications to the previous example.

Look at this revision of the previous code:

```

package com.sheusi.DemoApp;
import android.app.Activity;
import android.os.Bundle;
import android.view.*;
import android.widget.*;
import android.view.View.OnClickListener;
import android.view.View.OnKeyListener;
public class DemoAppActivity extends Activity implements
OnClickListener,
OnKeyListener{
/** Called when the activity is first created. */
Button b=null;
EditText et=null;
TextView tv=null;
Button quitbutton=null;
@Override
public void onCreate(Bundle savedInstanceState) {

```

```

super.onCreate(savedInstanceState);
setContentView(R.layout.main);
b=(Button)findViewById(R.id.my_Button);
quitbutton=(Button)findViewById(R.id.quit);
et=(EditText)findViewById(R.id.my_EditText);
tv=(TextView)findViewById(R.id.my_TextView);
b.setOnClickListener(this);
quitbutton.setOnClickListener(this);
et.setOnKeyListener(this);
}
public void onClick(View v){
if(v==b){
String temp=tv.getText().toString();
temp=temp.toUpperCase();
et.setText(temp);
}
if(v==quitbutton){
this.finish();
}
}
public boolean onKey(View v, int keyCode, KeyEvent event){
if(event.getAction()==KeyEvent.ACTION_DOWN){
if(keyCode==KeyEvent.KEYCODE_ENTER){
String temp=et.getText().toString();
temp=temp.toUpperCase();

et.setText(temp);
}
}
return false;
}
}

```

- First, notice there is a new import statement to add the OnKeyListener interface to our namespace.
- If you are familiar with Java, you will know that implementation of interfaces requires that certain methods be defined in the code.
- In the case of the OnKeyListener interface, it is the onKey() method. The method takes three arguments, or parameters, a View object, an integer representing the key you want to respond to, and finally a KeyEvent object. The KeyEvent object will represent one of two actions on a given key, when it is touched or pushed, and when it is released.

- These are represented by symbolic constants that can be found in the KeyEvent's documentation, namely ACTION_DOWN and ACTION_UP.
- The keyCode integer variable also represents a set of symbolic constants defined in the KeyEvent's documentation; there is a constant for each of the keys on the keypad.
- Some of the common noncharacter keys on the Android devices and their corresponding symbolic constants are listed in Table 3.5.

Table 3.5 Keystrokes and Their Symbolic Constants

Device's Key	KeyEvent Symbolic Constant
Power button	KEYCODE_POWER
Back key	KEYCODE_BACK
Menu key	KEYCODE_MENU
Camera button	KEYCODE_CAMERA
Home key	KEYCODE_HOME
Search key	KEYCODE_SEARCH

- In our example, we want to respond to the user's touch on the Enter or Return key on the keypad; hence, the symbolic constant KEYCODE_ENTER.
- During run-time, the application assesses each keystroke entered into the EditText control we assigned the listener to. If the action matches ACTION_DOWN, the application goes on to check which key was pressed.
- If it was indeed the Enter key, we proceed with an action—in this case to convert all the characters typed in to uppercase—and then replace them in the EditText field.

For any doubt contact 9873961590