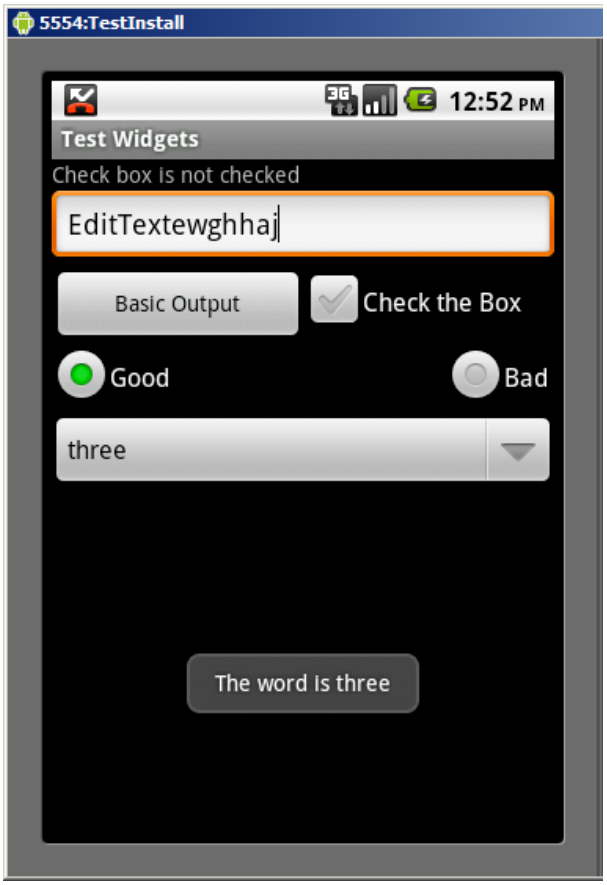
Linking Android XML Widgets to Java Code

There are other options for entering text besides the EditText and other actions besides a Button click. This document will consider the following widgets and how to interact with them.

- CheckBox
- RadioButton
- Spinner
- Toast
- AlertDialog



These all allow a limited range of input choices. A Spinner will offer a list to be selected from. A RadioButton allows on 1 of each set of RadioButtons to be selected. A CheckBox can be checked or not but unlike the RadioButton several can be selected at the same time.

The Toast is used for outputting messages in an alert box for a short period of time. It is also handy for debugging.

The Activity to the left has used a table layout in xml to organise the widgets and fit them on the rather small screen real estate.

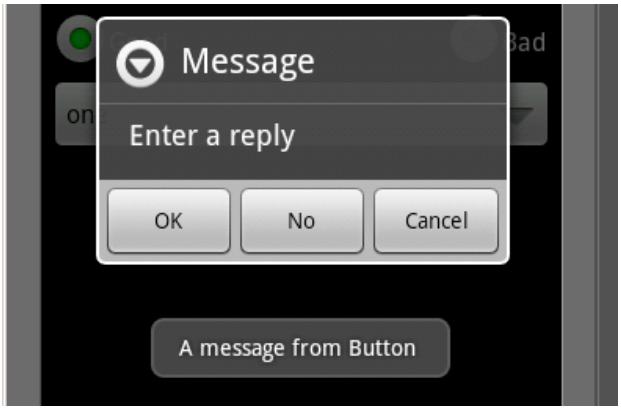
The key lines to attach an onClickListener to a Button and pick up the click in code were:

```
import android.view.View;

implements View.OnClickListener

bSimple.setOnClickListener(this);
```

```
public void onClick(View view)
{
      if (bSimple.isPressed())
          ;
}
```

The Toast is simple; the class can be accessed and used in 1 line. There is no need to create an instance of a Toast. The length refers to the time that the Toast is displayed not how long the message bar is on screen.

```
Toast.makeText(this, "A message from Button ",
Toast.LENGTH_LONG).show();
```



The AlertDialog places a message in a box but unlike the Toast can have up to 3 Buttons allowing some feedback to be grabbed from the user. If an AlertDialog is displayed without any Buttons the back key on the keyboard will need to be used to remove it.

Here is a simple Alert with one Button, not all of the Buttons need to be used.

```
AlertDialog.Builder theAlert = new AlertDialog.Builder(this);
theAlert.setTitle("Message").setMessage("Enter a reply");
theAlert.setNegativeButton("Cancel", null);
theAlert.setPositiveButton("OK", null);
theAlert.setNeutralButton("No", null);
theAlert.show();
```

At this point the AlertDialog is diplayed but there is no code to pick up the Button click actions. A Listener can be added to any or all of these Buttons.

Here is the adapted code to pick up a message from the ‘Cancel’ Button. Note that the entire onClick listener method is within the .setNegativeButton method. Each AlertDialog Button will need its own onClick() event.

```
theAlert.setNegativeButton("Cancel",
new DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialogInterface, int i) {
                Toast.makeText(getApplicationContext(), "this
is a negative onClick", Toast.LENGTH_LONG).show();
                }
});
```

A CheckBox can be linked to the same onClickListener and onClick method as a Button but the event will fire when either is clicked so the code will have to cope with this and work out if the click is a click to check or uncheck the CheckBox.

As long as the CheckBox controls have different names they can all be linked to the onClickListener and handled within onClick(). These might be used to choose several out of a list of options all of which can be applied.

```
import android.widget.CheckBox;

CheckBox ckMe;
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        ckMe = (CheckBox)findViewById(R.id.ckBox);
        ckMe.setOnClickListener(this);

    }
    public void onClick(View view)
    {
      if (((CheckBox) ckMe).isChecked())
      {
           tOut.setText("Check box is checked");
      }
      else
      {
           tOut.setText("Check box is not checked");
      }
      if (bSimple.isPressed())
          tOut.setText("Message from Button");
    }
```

RadioButtons need adding as part of a RadioGroup so that only 1 of the group can be selected at once. Only the individual RadioButtons need their id setting in XML if there is only 1 RadioGroup. This is easier to handle within the xml than using the graphical designer.

```
<RadioGroup android:id="@+id/radioGroup1"
android:layout_width="wrap_content"
android:layout_height="wrap_content" android:layout_weight="1"
android:orientation="horizontal">

     <RadioButton android:layout_height="wrap_content"
     android:layout_width="wrap_content" android:text="Good"
     android:checked="true" android:id="@+id/rbGood"
     android:layout_weight="50"></RadioButton>
     <RadioButton android:layout_width="wrap_content"
     android:layout_height="wrap_content" android:text="Bad"
     android:layout_weight="1"
     android:id="@+id/rbBad"></RadioButton>
</RadioGroup>
```

The RadioButtons work in the same way as CheckBoxes here the onClick() code has been modified to handle both events:

```
rGood = (RadioButton)findViewById(R.id.rbGood);
rBad = (RadioButton)findViewById(R.id.rbBad);
rGood.setOnClickListener(this);
rBad.setOnClickListener(this);
```

These are handled within onClick like a Button or CheckBox

```
public void onClick(View view)
{
 if (((CheckBox) ckMe).isChecked())
 {
      tOut.setText("Check box is checked");
       if (rGood.isChecked())
            tOut.setText("A good thing the box is checked");
      if (rBad.isChecked())
            tOut.setText("A bad thing the box is checked");
 }
}
```

The key pressed within an EditText can be picked up in a similar yet subtly different arrangement. The EditText is picked up the key down action and this requires use - of an onKeyListener.

A class can only extend one other class but luckily it can implement several interfaces. An interface has some similarity to a class but must implemeent some methods and cannot be instantiated on its own.

```
//additional listener
public class InputOutput extends Activity implements
View.OnClickListener, View.OnKeyListener {
    EditText tIn;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        tIn = (EditText)findViewById(R.id.tInput);
        tIn.setOnKeyListener(this);

    }
     public boolean onKey(View v, int keyCode, KeyEvent event)
     {
           // key listener
             if ((event.getAction() == KeyEvent.ACTION_DOWN)
                && (keyCode == KeyEvent.KEYCODE_ENTER))
             {
                   tOut.setText(tIn.getText());
                   return true;
             }
           return false;
     }
}
```

The enter key needs to be picked up from the dummy keyboard on the virtual device, not the keyboard on the computer used to host the virtual device.

The Spinner is a drop down list; the user can pick one item from the list. In this Android 2.2 implementation it had to be set to Visibility = Visible in the XML to get the program to load on the emulator.

Here a String Array will be used to fill the Spinner:

```
Spinner sList;
String [] items ={"one","two","three","four","five"};
```

No additional interfaces are required in the implements to access the Spinner data. The ArrayAdapter is part of Android and is used to map the array of Strings (`items`) to the Spinner.

```
sList = (Spinner)findViewById(R.id.spList);
sList.setOnItemSelectedListener(new
MyOnItemSelectedListener());
ArrayAdapter<String> aa=new ArrayAdapter<String>(this,
                android.R.layout.simple_spinner_item,
                items);
aa.setDropDownViewResource(
                android.R.layout.simple_spinner_dropdown_item);
sList.setAdapter(aa);
```

MyOnItemSelectedListener is a brand new class nested within the project class that handles the selection event.

```
public class MyOnItemSelectedListener implements
OnItemSelectedListener {
public void onItemSelected(AdapterView<?> parent,View view,
int pos, long id) {
          Toast.makeText(parent.getContext(), "The word is " +
          parent.getItemAtPosition(pos).toString(),
          Toast.LENGTH_LONG).show();
    }
public void onNothingSelected(AdapterView parent) {
           // Do nothing - this method is required.
}
```

We can get the same effect by loading the Spin from the strings.xml file in the res/values folder:

```
<resources>
    <string name="hello">Hello World, InputOutput!</string>
    <string name="app_name">Test Widgets</string>
        <string name="nums_prompt">Choose a number</string>
    <string-array name="nums">
        <item>one</item>
        <item>two</item>
        <item>three</item>
        <item>four</item>
        <item>five</item>
        <item>six</item>
    </string-array>
</resources>
```

The Android code then needs modifying as follows:

```
/*ArrayAdapter<String> aa=new ArrayAdapter<String>(this,
       android.R.layout.simple_spinner_item,
                items);*/
ArrayAdapter<CharSequence> aa =
     ArrayAdapter.createFromResource(this, R.array.nums,
     android.R.layout.simple_spinner_item);
```

Generally it is better to load values such the array of words from the xml rather than from the Java code. Modifying the xml is less technical so should be easier to set up. The Java code can be left alone and the xml changed; different builds can then be made to handle different language markets.

The /res folders can be used to load data many purposes. Here a String array models a set of questions and answers.

```
<string-array name="quests">
<item>"What is the capital of France?,Paris,Berlin,Moscow,1"</item>
<item>"What is the capital of Germany?,Paris,Berlin,Moscow,2"</item>
</string-array>
```

This can be picked up in the Java and assigned to a Java array

```
String []questions;
Resources res = getResources();
questions = res.getStringArray(R.array.quests);
qCount = 0;
```

The array can be split along the ‘,’ token and the members placed in appropriate controls.

```
if(qCount<questions.length)
{
    String [] token = questions[qCount].split(",");
    txtQuest.setText(token[0]);
    txtAns1.setText(token[1]);
    txtAns2.setText(token[2]);
    txtAns3.setText(token[2]);
    qCount ++;
}
```

We see Berlin twice and no Moscow because the last TextView should be set to `token[3]`.