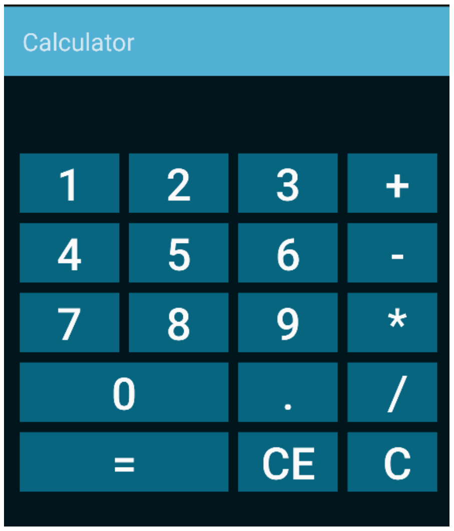
Xamarin.Android – Creating a Simple Calculator Layout

In this exercise, you will create a layout for a simple calculator, and wire up the necessary events to make it functional. There is already boiler plate code in place within the Activity to handle the events and calculate the expression. This demo uses an open-source library, called NCalc to perform the calculations.

To learn more about the topics discussed in this workshop, refer to the Resources links available in the XamarinAndroidFundamentals_Layouts.pdf in this directory and read through the online Android Developer documentation.

The resulting user interface will look similar to the image below.



Open the solution in the Start folder

1. In the Main.axml file, modify the GridLayout root element to specify its width and height

   Hint: android:layout_width and android:layout_height settings will be used for this

2. Set the GridLayout's column count to match the number of columns shown in calculator image above.

   Hint: Refer to the Android documentation on GridLayout to help with this: https://developer.android.com/reference/android/widget/GridLayout.html

3. A TextView is needed to display the values as they are entered in the calculator, and to display the end result when the expression is calculated. Add a TextView as the first child element of the GridLayout. A style has already been created for you which defines the default width, height and text size properties for the TextView. To apply the default style, set the style attribute on the TextView to:

```
style="@style/DefaultCalcStyle"
```

The style is located in the Resources/values folder within the Styles.xml file. Open this file, and explore the styles that are there. Feel free to change the values accordingly.

4. We will need to reference the TextView in the Activity's code, so be sure to provide it with a unique Id. Hint: set the android:id attribute.

   For example: `android:id="@+id/ExpressionText"`
5. The TextView will need to span the entire first row. To do this, we can set the column span on the element to span across all 4 columns in the GridLayout.

   Hint: `android:layout_columnSpan="4"`

6. The TextView will need to span the entire first row. To do this, we can set the column span on the element to span across all 4 columns in the GridLayout.

7. In the OnCreate method in the MainActivity.cs file, load the layout by calling SetContentView and passing in the Id of the layout.

   Hint: SetContentView(Resource.Layout.Main);

8. In the MainActivity.cs file, after the call to load the layout, retrieve a handle to the ExpressionText TextView in the layout and store that in the variable _expressionText.

9. Return to the Main.axml layout. Add the necessary buttons to make up the calculator view in the image above. For each button, you can apply the

included style, CalcButtonStyle which will take care of setting the Button’s default width and height values, as well as margins and text size. Also include the button’s onClick event and reference the existing method CalcButtonPressed. For example:

```
<Button
        android:text="1"
        android:onClick="CalcButtonPressed"
        style="@style/CalcButtonStyle"
        />
```

Every button will execute the CalcButtonPressed event when clicked, so be sure to set that on all buttons.

For the zero (0) and equal (=) button, you will need to set its column span to take up 2 columns. Also you will need to set its layout to fill the space of both columns. To do this set the attribute, android:layout_gravity to fill_horizontal, as follows: `android:layout_gravity="fill_horizontal"`

Once you have your layout in place, and all of the necessary code wired up, test your application by running it in an emulator or on an attached Android device.