Build a simple user interface

In this lesson, you learn how to use the Android Studio Layout Editor (/studio/write/layout-editor) to create a layout that includes a text box and a button. This sets up the next lesson, where you learn how to make the app send the content of the text box to another activity when the button is tapped.

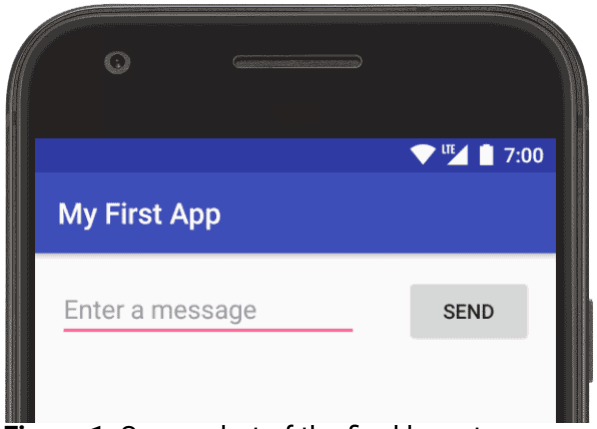

Figure 1. Screenshot of the final layout

The user interface (UI) for an Android app is built as a hierarchy of *layouts* and *widgets*. The layouts are `ViewGroup` (/reference/android/view/ViewGroup) objects, containers that control how their child views are positioned on the screen. Widgets are `View` (/reference/android/view/View) objects, UI components such as buttons and text boxes.

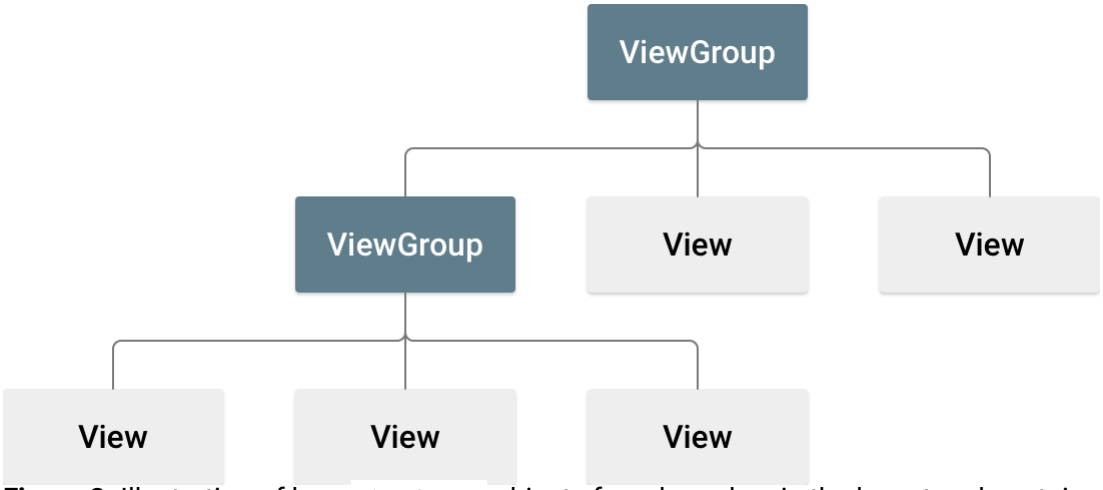

Figure 2. Illustration of how `ViewGroup` objects form branches in the layout and contain `View` objects.

Android provides an XML vocabulary for `ViewGroup` and `View` classes, so most of your UI is defined in XML files. However, rather than teach you to write XML, this lesson shows you how to create a layout using Android Studio's Layout Editor. The Layout Editor writes the XML for you as you drag and drop views to build your layout.

This lesson assumes that you use Android Studio v3.0 (/studio) or higher and that you've completed the create your Android project (/training/basics/firstapp/creating-project) lesson.

Open the Layout Editor

To get started, set up your workspace as follows:

1. In the Project window, open **app > res > layout > activity_main.xml**.
2. To make room for the Layout Editor, hide the **Project** window. To do so, select **View > Tool Windows > Project**, or just click **Project**

on the left side of the Android Studio screen.

3. If your editor shows the XML source, click the **Design** tab at the bottom of the window.
4. Click **Select Design Surface**
   
   and select **Blueprint**.
5. Click **Show**
   
   in the Layout Editor toolbar and make sure that **Show All Constraints** is checked.
6. Make sure Autoconnect is off. A tooltip in the toolbar displays **Enable Autoconnection to Parent**
   
   when Autoconnect is off.
7. Click **Default Margins** in the toolbar and select **16**. If needed, you can adjust the margins for each view later.
8. Click **Device for Preview**
   
   in the toolbar and select **5.5, 1440 × 2560, 560 dpi (Pixel XL)**.

Your Layout Editor now looks as shown in figure 3.

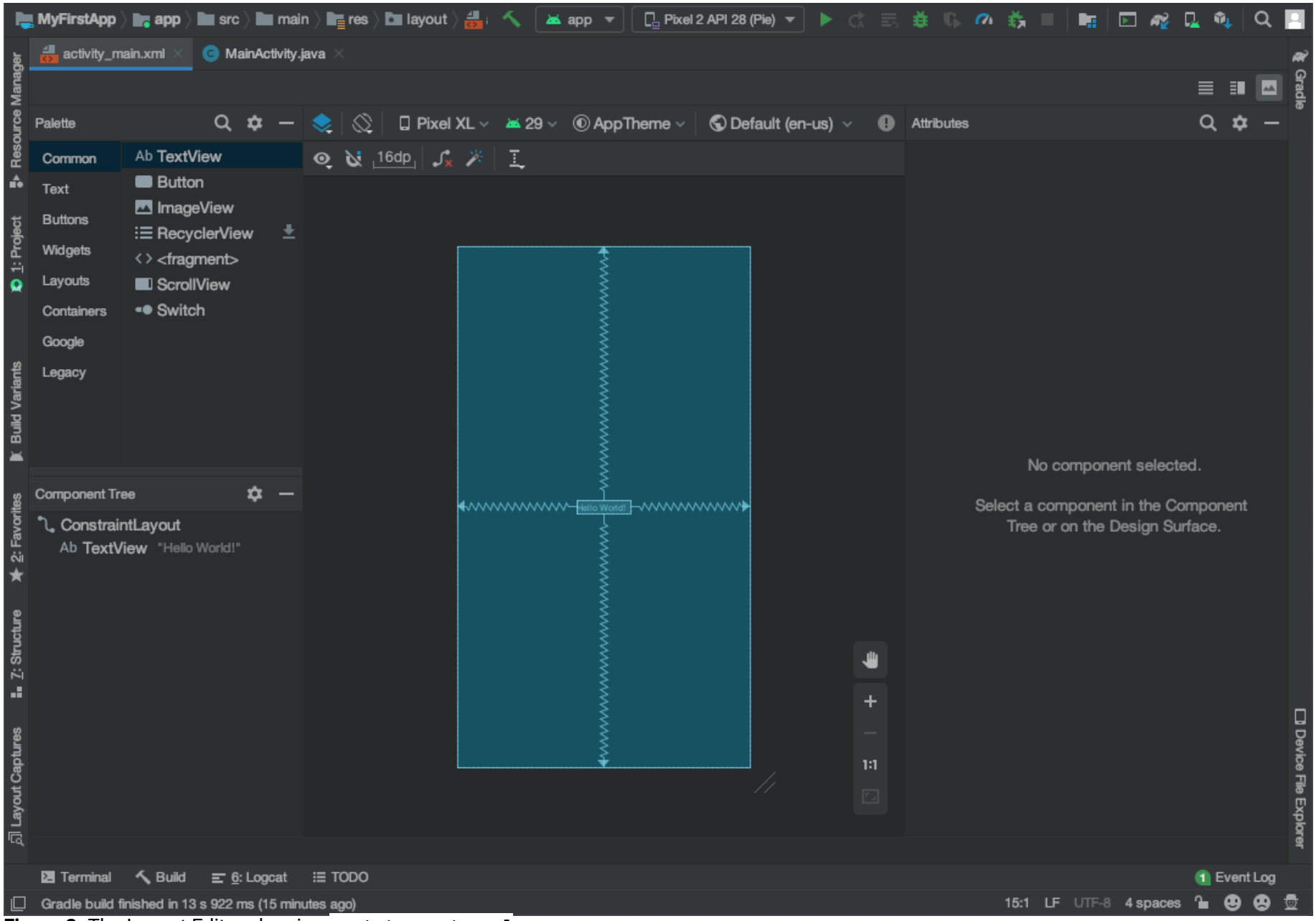

Figure 3. The Layout Editor showing `activity_main.xml`

For additional information, see Introduction to the Layout Editor (/studio/write/layout-editor#intro).

The **Component Tree** panel on the bottom left shows the layout's hierarchy of views. In this case, the root view is a `ConstraintLayout`, which contains just one `TextView` object.

`ConstraintLayout` is a layout that defines the position for each view based on constraints to sibling views and the parent layout. In this way, you can create both simple and complex layouts with a flat view hierarchy. This type of layout avoids the need for nested layouts. A nested layout, which is a layout inside a layout, as shown in figure 2, can increase the time required to draw the UI.

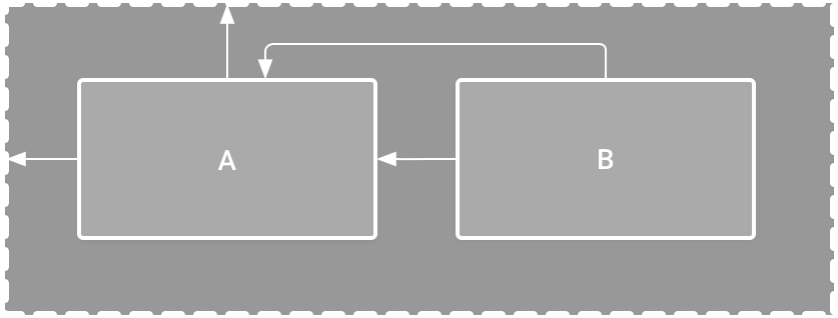

Figure 4. Illustration of two views positioned inside `ConstraintLayout`

For example, you can declare the following layout, which is shown in figure 4:

- View A appears 16 dp from the top of the parent layout.
- View A appears 16 dp from the left of the parent layout.
- View B appears 16 dp to the right of view A.
- View B is aligned to the top of view A.

In the following sections, you'll build a layout similar to the layout in figure 4.

Add a text box

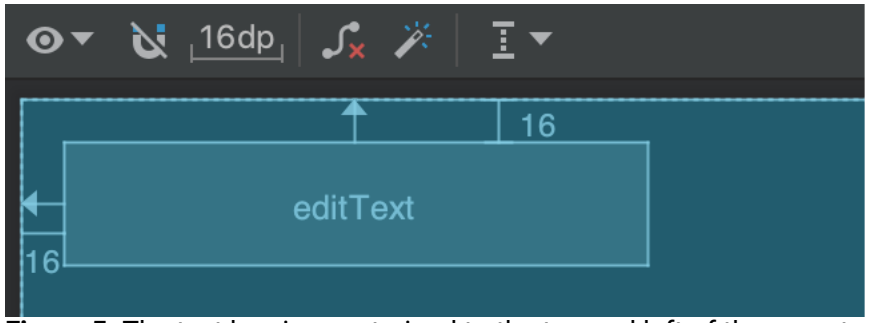

Figure 5. The text box is constrained to the top and left of the parent layout

Follow these steps to add a text box:

1. First, you need to remove what's already in the layout. Click **TextView** in the **Component Tree** panel and then press the `Delete` key.
2. In the **Palette** panel, click **Text** to show the available text controls.
3. Drag the **Plain Text** into the design editor and drop it near the top of the layout. This is an `EditText` (/reference/android/widget/EditText) widget that accepts plain text input.
4. Click the view in the design editor. You can now see the square handles to resize the view on each corner, and the circular constraint anchors on each side. For better control, you might want to zoom in on the editor. To do so, use the **Zoom** buttons in the Layout Editor toolbar.
5. Click and hold the anchor on the top side, drag it up until it snaps to the top of the layout, and then release it. That's a constraint: it constrains the view within the default margin that was set. In this case, you set it to 16 dp from the top of the layout.
6. Use the same process to create a constraint from the left side of the view to the left side of the layout.

The result should look as shown in figure 5.

Add a button

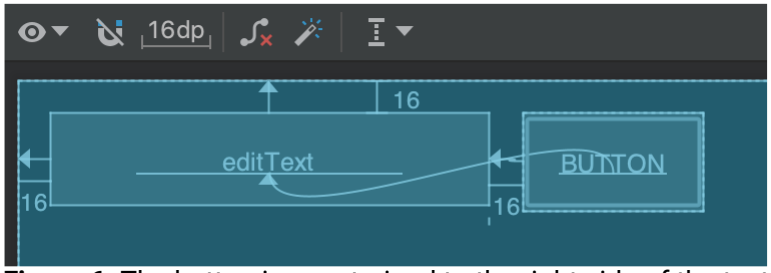

Figure 6. The button is constrained to the right side of the text box and its baseline

1. In the **Palette** panel, click **Buttons**.
2. Drag the **Button** widget into the design editor and drop it near the right side.
3. Create a constraint from the left side of the button to the right side of the text box.
4. To constrain the views in a horizontal alignment, create a constraint between the text baselines. To do so, right-click the button and then select **Show Baseline**
   
   . The baseline anchor appears inside the button. Click and hold this anchor, and then drag it to the baseline anchor that appears in the adjacent text box.

The result should look as shown in figure 6.

Note: You can also use the top or bottom edges to create a horizontal alignment. However, the button image includes padding around it, so the visual alignment is wrong if created that way.

Change the UI strings

To preview the UI, click **Select Design Surface**

in the toolbar and select **Design**. Notice that the text input and button label are set to default values.

Follow these steps to change the UI strings:

1. Open the **Project** window and then open **app > res > values > strings.xml**.

   This is a string resources (/guide/topics/resources/string-resource) file, where you can specify all of your UI strings. It allows you to manage all of your UI strings in a single location, which makes them easier to find, update, and localize.

2. Click **Open editor** at the top of the window. This opens the Translations Editor (/studio/write/translations-editor), which provides a simple interface to add and edit your default strings. It also helps you keep all of your translated strings organized.

3. Click **Add Key**
   
   to create a new string as the "hint text" for the text box. At this point, the window shown in figure 7 opens.

   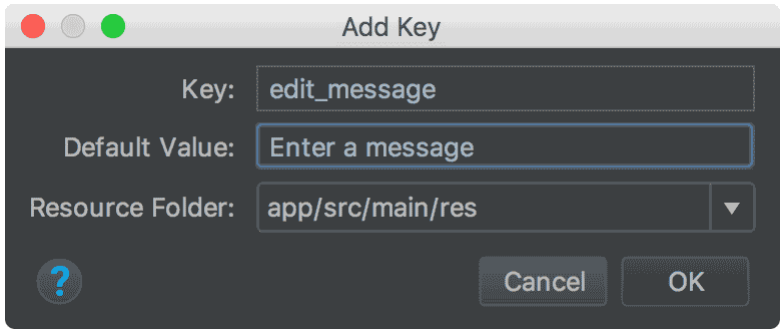

   Figure 7. The dialog to add a new string

   In the **Add Key** dialog box, complete the following steps:

    a. Enter "edit_message" in the **Key** field.

    b. Enter "Enter a message" in the **Default Value** field.

    c. Click **OK**.

4. Add another key named "button_send" with a value of "Send".

Now you can set these strings for each view. To return to the layout file, click **activity_main.xml** in the tab bar. Then, add the strings as follows:

1. Click the text box in the layout. If the **Attributes** window isn't already visible on the right, click **Attributes**
   
   on the right sidebar.
2. Locate the **text** property, which is currently set to "Name," and delete the value.
3. Locate the **hint** property and then click **Pick a Resource**

   , which is to the right of the text box. In the dialog that appears, double-click **edit_message** from the list.
4. Click the button in the layout and locate its **text** property, which is currently set to "Button." Then, click **Pick a Resource**

   and select **button_send**.

Make the text box size flexible

To create a layout that's responsive to different screen sizes, you need to make the text box stretch to fill all the horizontal space that remains after the button and margins are accounted for.

Before you continue, click **Select Design Surface**


in the toolbar and select **Blueprint**.

To make the text box flexible, follow these steps:

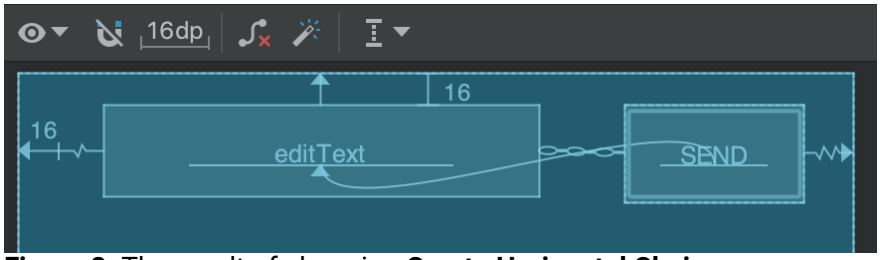

Figure 8. The result of choosing **Create Horizontal Chain**

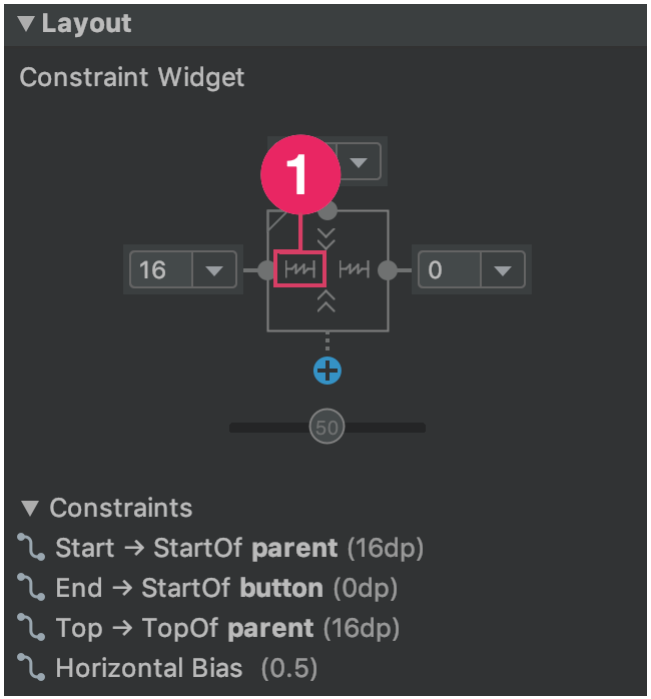

Figure 9. Click to change the width to **Match Constraints**

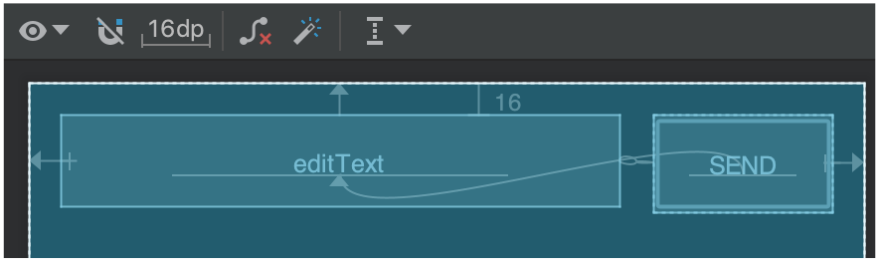

Figure 10. The text box now stretches to fill the remaining space

1. Select both views. To do so, click one, hold `Shift`, then click the other, and then right-click either one and select **Chains > Create Horizontal Chain**. The layout then appears as shown in figure 8.

   A chain (/training/constraint-layout#constrain-chain) is a bidirectional constraint between two or more views that allows you to lay out the chained views in unison.

2. Select the button and open the **Attributes** window. Then, use the view inspector at the top of the **Attributes** window to set the right margin to 16 dp.

3. Click the text box to view its attributes. Then, click the width indicator twice so it's set to **Match Constraints**, as indicated by callout 1 in figure 9.

   Match constraints means that the width expands to meet the definition of the horizontal constraints and margins. Therefore, the text box stretches to fill the horizontal space that remains after the button and all the margins are accounted for.

Now the layout is done, as shown in figure 10.

If your layout didn't turn out as expected, click **See the final layout XML** below to see what your XML should look like. Compare it to what you see in the **Text** tab. If your attributes appear in a different order, that's okay.

See the final layout XML

```
<?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"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.myfirstapp.MainActivity">

    <EditText
        android:id="@+id/editText"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginLeft="16dp"
        android:layout_marginTop="16dp"
        android:ems="10"
        android:hint="@string/edit_message"
        android:inputType="textPersonName"
        app:layout_constraintEnd_toStartOf="@+id/button"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="16dp"
        android:layout_marginStart="16dp"
        android:text="@string/button_send"
        app:layout_constraintBaseline_toBaselineOf="@+id/editText"
```

```
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toEndOf="@+id/editText" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

For more information about chains and all the other things you can do with `ConstraintLayout`, read Build a Responsive UI with ConstraintLayout (/training/constraint-layout).

Run the app

If your app is already installed on the device from the previous lesson (/training/basics/firstapp/running-app), simply click **Apply Changes**

in the toolbar to update the app with the new layout. Or, click **Run**

to install and run the app.

The button still does nothing. To build another activity that starts when the button is tapped, continue to the next lesson (/training/basics/firstapp/starting-activity).

Previous
← Run your app (/training/basics/firstapp/running-app)

Next
Start another activity (/training/basics/firstapp/starting-activity) →

Last updated 2020-08-17 UTC.