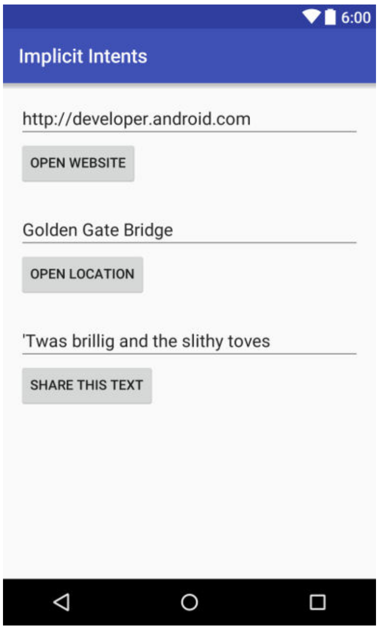
Android Development
Week 05 Practical - Implicit Intents

1 App Overview

In this practical we will create an application with one Activity and three options for actions:

- open a web site
- open a location on a map
- share a snippet of text.

All of the text fields are editable (EditText) but contain default values.



2 Create the project and layout

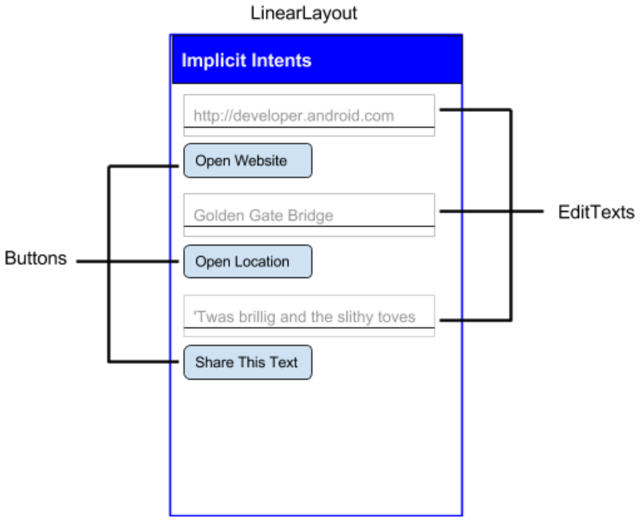
Create a new project and app called Implicit Intents, with a new layout.

2.1 Create the project

1. Start Android Studio and create a new Android Studio project. Name your app **Implicit Intents**.
2. Choose **Empty Views Activity** for the project template.
3. Click **Next.**
4. Accept the default Activity name (MainActivity). Make sure the **Generate Layout file** box is checked.
5. Click **Finish**.

2.2 Create the Layout

Create the layout for the app, use Wizard defaults for min SDK version etc. We are going to use a LinearLayout and create a UI with three Button elements, and three EditText elements, like this:



1. Open app > res > values > strings.xml in the Project > Android pane, and add the following string resources:

```
<string name="edittext_uri">http://developer.android.com</string>
<string name="button_uri">Open Website</string>

<string name="edittext_loc">Golden Gate Bridge</string>
<string name="button_loc">Open Location</string>

<string name="edittext_share">\'Twas brillig and the slithy toves</string>
<string name="button_share">Share This Text</string>
```

2. Open res > layout > activity_main.xml in the Project > Android pane. Right click the UI editor to bring up a context menu and switch to XML code.
3. Change android.support.constraint.ConstraintLayout to LinearLayout, as you learned in a previous practical (cut and paste the following to replace the opening XML

```
<LinearLayout 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"
         android:orientation="vertical"
         android:padding="16dp"
         tools:context="com.example.android.implicitintents.MainActivity">
```

4. You may need to edit the tools context string (to match any explicit package you may have set in your wizard and you will have to modify the closing tag in your layout
5. Remove the TextView that displays "Hello World".
6. Add a set of UI elements to the layout for the Open Website button. You need an EditText element and a Button element. Use these attribute values:

EditText attribute	Value
android:id	"@+id/website_edittext"
android:layout_width	"match_parent"
android:layout_height	"wrap_content"
android:text	"@string/edittext_uri"
Button attribute	**Value**
android:id	"@+id/open_website_button"
android:layout_width	"wrap_content"
android:layout_height	"wrap_content"
android:layout_marginBottom	"24dp"
android:text	"@string/button_uri"
android:onClick	"openWebsite"

The value for the android:onClick attribute will remain underlined in red until you define the callback method later.

7. Add a set of UI elements (EditText and Button) to the layout for the **Open Location** button. Use the same attributes as in the previous step but modify them as shown below. (You can copy the values from the **Open Website** button and modify them.)

EditText attribute	Value
android:id	"@+id/location_edittext"
android:text	"@string/edittext_loc"
Button attribute	**Value**
android:id	"@+id/open_location_button"
android:text	"@string/button_loc"
android:onClick	"openLocation"

The value for the android:onClick attribute will remain underlined in red until you define the callback method later.

8. Add a set of UI elements (EditText and Button) to the layout for the **Share This** button. Use the attributes shown below. (You can copy the values from the **Open Website** button and modify them.)

EditText attribute	Value
android:id	"@+id/share_edittext"
android:text	"@string/edittext_share"

Button attribute	Value
android:id	"@+id/share_text_button"
android:text	"@string/button_share"
android:onClick	"shareText"

Depending on your version of Android Studio, your activity_main.xml code should look something like the following. The values for the android:onClick attributes will remain underlined in red until you define the callback methods.

```
<LinearLayout 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"
    android:orientation="vertical"
    android:padding="16dp"
    tools:context="com.example.android.implicitintents.MainActivity">

    <EditText
        android:id="@+id/website_edittext"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/edittext_uri"/>

    <Button
        android:id="@+id/open_website_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="24dp"
        android:text="@string/button_uri"
        android:onClick="openWebsite"/>

    <EditText
        android:id="@+id/location_edittext"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/edittext_uri"/>

    <Button
        android:id="@+id/open_location_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="24dp"
        android:text="@string/button_loc"
        android:onClick="openLocation"/>

   <EditText
        android:id="@+id/share_edittext"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/edittext_share"/>

    <Button
        android:id="@+id/share_text_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="24dp"
        android:text="@string/button_share"
        android:onClick="shareText"/>

</LinearLayout>
```

3 Manifest changes post SDK 30

On Android 10 and earlier, apps could query the full list of installed apps on the system using methods like queryIntentActivities(). In most cases, this is far broader access than is necessary for an app to implement its functionality. With our ongoing focus on privacy, we're introducing changes on how apps can query and interact with other installed apps on the same device on Android 11. In particular, we're bringing better scoped access to the list of apps installed on a given device.

To provide better accountability for access to installed apps on a device, apps targeting Android 11 (API level 30) will see a filtered list of installed apps by default. In order to access a broader list of installed apps, an app can specify information about apps they need to query and interact with directly. This can be done by adding a <queries> element in the Android manifest.

In order to find the browser using an implicit Intent you need to add a query to your manifest file (above the application tag).

```
<queries>
    <intent>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.BROWSABLE" />
        <data android:scheme="https" />
    </intent>
</queries>
```

4 Implement the 'Open Website' button

4.1 Define openWebsite()

Implement the on-click handler method for the first button in the layout, **Open Website**. This action uses an implicit Intent to send the given URI to an Activity that can handle that implicit Intent (such as a web browser).

1. Click "openWebsite" in the activity_main.xml XML code.
2. Press Alt+Enter (Option+Enter on a Mac) and select **Create 'openWebsite(View)' in 'MainActivity.**

The MainActivity file opens, and Android Studio generates a skeleton method for the openWebsite() handler.

```
public void openWebsite(View view) {
}
```

3. In MainActivity, add a private variable at the top of the class to hold the EditText object for the web site URI.

```
private EditText mWebsiteEditText;
```

4. In the onCreate() method for MainActivity, use findViewById() to get a reference to the EditText instance and assign it to that private variable:

```
mWebsiteEditText = findViewById(R.id.website_edittext);
```

4.2 Add code to openWebsite()

1. Add a statement to the new openWebsite() method that gets the string value of the EditText:

```
String url = mWebsiteEditText.getText().toString();
```

2. Encode and parse that string into a Uri object:

```
Uri webpage = Uri.parse(url);
```

3. Create a new Intent with Intent.ACTION_VIEW as the action and the URI as the data:

```
Intent intent = new Intent(Intent.ACTION_VIEW, webpage);
```

This Intent constructor is different from the one you used to create an explicit Intent. In the previous constructor, you specified the current context and a specific component (Activity class) to send the Intent. In this constructor you specify an action and the data for that action. Actions are defined by the Intent class and can include ACTION_VIEW (to view the given data), ACTION_EDIT (to edit the given data), or ACTION_DIAL (to dial a phone number). In this case the action is ACTION_VIEW because you want to display the web page specified by the URI in the webpage variable.

4. Use the resolveActivity() method and the Android package manager to find an Activity that can handle your implicit Intent. Make sure that the request resolved successfully.

```
if (intent.resolveActivity(getPackageManager()) != null) {
}
```

This request that matches your Intent action and data with the Intent filters for installed apps on the device. You use it to make sure there is at least one Activity that can handle your requests.

5. Inside the if statement, call startActivity() to send the Intent.

```
startActivity(intent);
```

6. Add an else block to print a Log message if the Intent could not be resolved.

```
} else {
    Log.d("ImplicitIntents", "Can't handle this!");
}
```

The openWebsite() method should now look as follows. (Comments added for clarity.)

```
public void openWebsite(View view) {
    // Get the URL text.
    String url = mWebsiteEditText.getText().toString();

    // Parse the URI and create the intent.
    Uri webpage = Uri.parse(url);
    Intent intent = new Intent(Intent.ACTION_VIEW, webpage);

     // Find an activity to hand the intent and start that activity.
     if (intent.resolveActivity(getPackageManager()) != null) {
         startActivity(intent);
     } else {
         Log.d("ImplicitIntents", "Can't handle this intent!");
    }
 }
```

5 Implement the ‘Open Location’ button

Implement the on-click handler method for the second button in the UI, **Open Location**. This method is almost identical to the openWebsite() method. The difference is the use of a geo URI to indicate a map location. You can use a geo URI with latitude and longitude or use a query string for a general location. In this example we've used the latter.

5.1 Define openLocation ()

1. Click "openLocation" in the activity_main.xml XML code.
2. Press Alt+Enter (Option+Enter on a Mac) and select **Create 'openLocation(View)' in MainActivity.**

Android Studio generates a skeleton method in MainActivity for the openLocation() handler.

```
public void openLocation(View view) {
}
```

3. Add a private variable at the top of MainActivity to hold the EditText object for the location URI.

```
private EditText mLocationEditText;
```

4. In the onCreate() method, use findViewByID() to get a reference to the EditText instance and assign it to that private variable:

```
mLocationEditText = findViewById(R.id.location_edittext);
```

5.2 Add code to openLocation()

1. In the new openLocation() method, add a statement to get the string value of the mLocationEditText EditText.

```
String loc = mLocationEditText.getText().toString();
```

2. Parse that string into a Uri object with a geo search query:

```
Uri addressUri = Uri.parse("geo:0,0?q=" + loc);
```

3. Create a new Intent with Intent.ACTION_VIEW as the action and loc as the data.

```
Intent intent = new Intent(Intent.ACTION_VIEW, addressUri);
```

4. Resolve the Intent and check to make sure that the Intent resolved successfully. If so, startActivity(), otherwise log an error message.

```
if (intent.resolveActivity(getPackageManager()) != null) {
    startActivity(intent);
} else {
    Log.d("ImplicitIntents", "Can't handle this intent!");
}
```

The openLocation() method should now look as follows (comments added for clarity):

```
public void openLocation(View view) {
    // Get the string indicating a location. Input is not validated; it is
    // passed to the location handler intact.
    String loc = mLocationEditText.getText().toString();

    // Parse the location and create the intent.
    Uri addressUri = Uri.parse("geo:0,0?q=" + loc);
    Intent intent = new Intent(Intent.ACTION_VIEW, addressUri);

    // Find an activity to handle the intent, and start that activity.
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    } else {
        Log.d("ImplicitIntents", "Can't handle this intent!");
    }
}
```

6 Implement the ‘Share This’ button

A share action is an easy way for users to share items in your app with social networks and other apps. Although you could build a share action in your own app using an implicit Intent, Android provides the ShareCompat.IntentBuilder helper class to make implementing sharing easy. You can use ShareCompat.IntentBuilder to build an Intent and launch a chooser to let the user choose the destination app for sharing.

1. Click "shareText" in the activity_main.xml XML code.
2. Press Alt+Enter (Option+Enter on a Mac) and select Create 'shareText(View)' in MainActivity.

Android Studio generates a skeleton method in MainActivity for the shareText() handler.

```
public void shareText(View view) {
}
```

3. Add a private variable at the top of MainActivity to hold the EditText.

```
private EditText mShareTextEditText;
```

4. In onCreate(), use findViewById() to get a reference to the EditText instance and assign it to that private variable:

```
mShareTextEditText = findViewById(R.id.share_edittext);
```

6.1 Add code to shareText()

1. In the new shareText() method, add a statement to get the string value of the mShareTextEditText EditText.

```
String txt = mShareTextEditText.getText(). toString();
```

2. Define the mime type of the text to share:

```
String mimeType = "text/plain";
```

3. Call ShareCompat.IntentBuilder with these ‘builder pattern’ methods:

```
ShareCompat.IntentBuilder
     .from(this)
     .setType(mimeType)
     .setChooserTitle("Share this text with: ")
     .setText(txt)
     .startChooser();
```

4. Extract the value of .setChooserTitle to a string resource.

The call to ShareCompat.IntentBuilder uses these methods:

Method	Description
from()	The Activity that launches this share Intent (this).

setType()	The MIME type of the item to be shared.
setChooserTitle()	The title that appears on the system app chooser.
setText()	The actual text to be shared
startChooser()	Show the system app chooser and send the Intent.

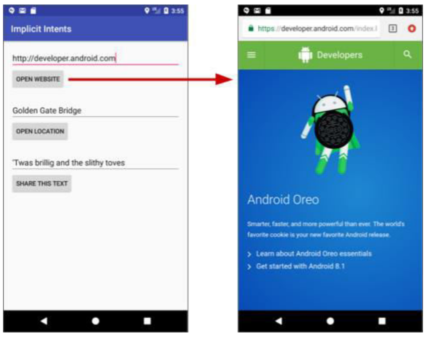
This format, with all the builder's setter methods strung together in one statement, is an easy shorthand way to create and launch the Intent. You can add any of the additional methods to this list.

The shareText() method should now look as follows:

```
public void shareText(View view) {
String txt = mShareTextEditText.getText().toString();
String mimeType = "text/plain";
ShareCompat.IntentBuilder
    .from(this)
    .setType(mimeType)
    .setChooserTitle(R.string.share_text_with)
    .setText(txt)
    .startChooser();
}
```

7 Run the App

1. Run the app.
2. Click the Open Website button to launch a browser with the website URL in the EditText above the Button. The browser and website should appear as shown below.



http://developer.android.com

Golden Gate Bridge

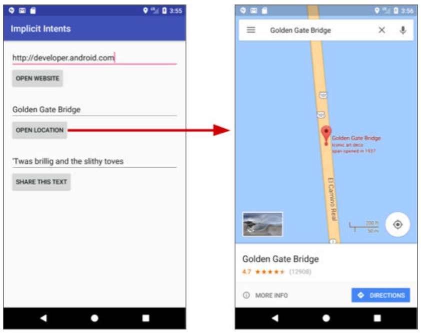
'Twas brillig and the slithy toves

https://developer.android.com/index.l

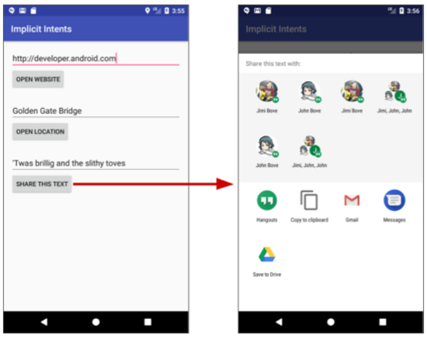
Android Oreo

Smarter, faster, and more powerful than ever. The world's favorite cookie is your new favorite Android release.

3. Click the Open Location button to launch the map with the location in the EditText above the Button. The map with the location should appear as shown below.



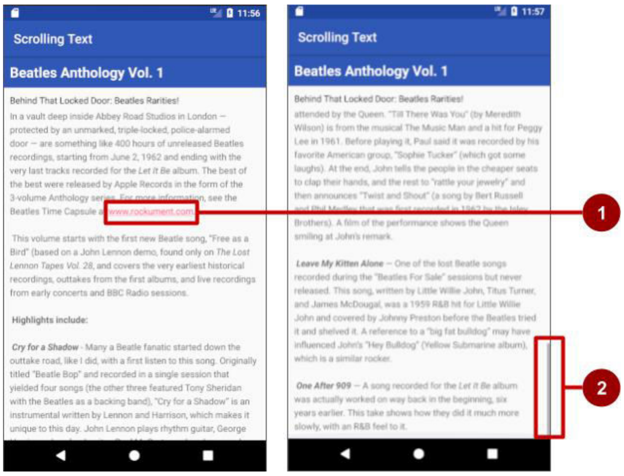
4. Click the Share This Text button to launch a dialog with choices for sharing the text. The dialog with choices should appear as shown below. If you're running on a phone (rather than an emulator) play with it and send some stuff.



Android Development
04 Practical - Text and scrolling View

1 App Overview

The Scrolling Text app demonstrates the ScrollView UI component. ScrollView is a ViewGroup that in this example contains a TextView. It shows a lengthy page of text—in this case, a music album review—that the user can scroll vertically to read by swiping up and down. A scroll bar appears in the right margin. The app shows how you can use text formatted with minimal HTML tags for setting text to bold or italic, and with new-line characters to separate paragraphs. You can also include active web links in the text.



Beatles Anthology Vol. 1

Behind That Locked Door: Beatles Rarities!

In a vault deep inside Abbey Road Studios in London — protected by an unmarked, triple-locked, police-alarmed door — are something like 400 hours of unreleased Beatles recordings, starting from June 2, 1962 and ending with the very last tracks recorded for the *Let It Be* album. The best of the best were released by Apple Records in the form of the 3-volume Anthology series. For more information, see the Beatles Time Capsule at www.rockument.com.

This volume starts with the first new Beatle song, "Free as a Bird" (based on a John Lennon demo, found only on *The Lost Lennon Tapes Vol. 28*, and covers the very earliest historical recordings, outtakes from the first albums, and live recordings from early concerts and BBC Radio sessions.

Highlights include:

Cry for a Shadow - Many a Beatle fanatic started down the outtake road, like I did, with a first listen to this song. Originally titled "Beatle Bop" and recorded in a single session that yielded four songs (the other three featured Tony Sheridan with the Beatles as a backing band), "Cry for a Shadow" is an instrumental written by Lennon and Harrison, which makes it unique to this day. John Lennon plays rhythm guitar, George

Beatles Anthology Vol. 1

Behind That Locked Door: Beatles Rarities!

attended by the Queen. "Till There Was You" (by Meredith Wilson) is from the musical The Music Man and a hit for Peggy Lee in 1961. Before playing it, Paul said it was recorded by his favorite American group, "Sophie Tucker" (which got some laughs). At the end, John tells the people in the cheaper seats to clap their hands, and the rest to "rattle your jewelry" and then announces "Twist and Shout" (a song by Bert Russell and Phil Medley that was first recorded in 1962 by the Isley Brothers). A film of the performance shows the Queen smiling at John's remark.

Leave My Kitten Alone — One of the lost Beatle songs recorded during the "Beatles For Sale" sessions but never released. This song, written by Little Willie John, Titus Turner, and James McDougal, was a 1959 R&B hit for Little Willie John and covered by Johnny Preston before the Beatles tried it and shelved it. A reference to a "big fat bulldog" may have influenced John's "Hey Bulldog" (Yellow Submarine album), which is a similar rocker.

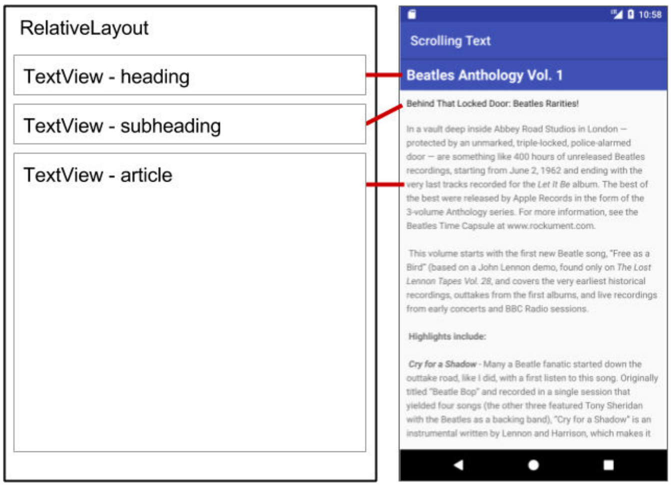
One After 909 — A song recorded for the *Let It Be* album was actually worked on way back in the beginning, six years earlier. This take shows how they did it much more slowly, with an R&B feel to it.

In the above figure, the following appear:

1. An active web link embedded in free-form text
2. The scroll bar that appears when scrolling the text

2 Add and edit TextView elements

You will create an Android project for the ScrollingText app, add TextView elements to the layout for an article title and subtitle, and change the existing "Hello World" TextView element to show a lengthy article. The figure below is a diagram of the layout.



Beatles Anthology Vol. 1

Behind That Locked Door: Beatles Rarities!

In a vault deep inside Abbey Road Studios in London — protected by an unmarked, triple-locked, police-alarmed door — are something like 400 hours of unreleased Beatles recordings, starting from June 2, 1962 and ending with the very last tracks recorded for the *Let It Be* album. The best of the best were released by Apple Records in the form of the 3-volume Anthology series. For more information, see the Beatles Time Capsule at www.rockument.com.

This volume starts with the first new Beatle song, "Free as a Bird" (based on a John Lennon demo, found only on *The Lost Lennon Tapes Vol. 28*, and covers the very earliest historical recordings, outtakes from the first albums, and live recordings from early concerts and BBC Radio sessions.

Highlights include:

Cry for a Shadow - Many a Beatle fanatic started down the outtake road, like I did, with a first listen to this song. Originally titled "Beatle Bop" and recorded in a single session that yielded four songs (the other three featured Tony Sheridan with the Beatles as a backing band), "Cry for a Shadow" is an instrumental written by Lennon and Harrison, which makes it

You will make all these changes in the XML code and in the strings.xml file. You will edit the XML code for the layout in the Text pane, which you show by clicking the **Text** tab, rather than clicking the **Design** tab for the Design pane. Some changes to UI elements and attributes are easier to make directly in the Text pane using XML source code.

2.1 Create the project and the TextView elements

Create the project and the TextView elements and use TextView attributes for styling the text and background.

1. In Android Studio create a new project with the following parameters:

Attribute	Value
Application Name	Scrolling Text
Company Name	android.example.com (or your own domain)
Phone and Tablet Minimum SDK	Use default
Template	Empty Views Activity
Generate Layout File checkbox	Selected
Backwards Compatibility (AppCompat) checkbox	Selected

2. In the **app > res > layout** folder in the **Project > Android** pane, open the **activity_main.xml** file, and click the **Text** tab to see the XML code.

At the top, or root, of the View hierarchy is the ConstraintLayout ViewGroup:

```
android.support.constraint.ConstraintLayout
```

3. Change this ViewGroup to RelativeLayout. The second line of code now looks something like this:

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

RelativeLayout lets you place UI elements relative to each other, or relative to the parent RelativeLayout itself.

The default "Hello World" TextView element created by the Empty Layout template still has constraint attributes (such as app:layout_constraintBottom_toBottomOf="parent"). Don't worry—you will remove them in a subsequent step.

4. Delete the following line of XML code, which is related to ConstraintLayout:

```
xmlns:app="http://schemas.android.com/apk/res-auto"
```

The block of XML code at the top now looks like this:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context="com.example.android.scrollingtext.MainActivity">
```

5. Add a TextView element above the "Hello World" TextView by entering **<TextView**. A TextView block appears that ends with /> and shows the layout_width and layout_height attributes, which are required for the TextView.
6. Enter the following attributes for the TextView. As you enter each attribute and value, suggestions appear to complete the attribute name or value.

TextView #1 attribute	Value
android:layout_width	"match_parent"
android:layout_height	"wrap_content"
android:id	"@+id/article_heading"
android:background	"@color/colorPrimary"
android:textColor	"@android:color/white"
android:padding	"10dp"
android:textAppearance	"@android:style/TextAppearance.DeviceDefault.Large"
android:textStyle	"bold"
android:text	"Article Title"

7. Extract the string resource for the android:text attribute's hardcoded string "Article Title" in the TextView to create an entry for it in **strings.xml**.

   Place the cursor on the hardcoded string, press Alt-Enter (Option-Enter on the Mac) and select **Extract string resource**. Make sure that the **Create the resource in directories** option is selected, and then edit the resource name for the string value to **article_title**.

8. Extract the dimension resource for the android:padding attribute's hardcoded string "10dp" in the TextView to create dimens.xml and add an entry to it.

   Place the cursor on the hardcoded string, press Alt-Enter (Option-Enter on the Mac), and select **Extract dimension resource**. Make sure that the **Create the resource in directories** option is selected, and then edit the Resource name to **padding_regular**.

9. Add another TextView element above the "Hello World" TextView and below the TextView you created in the previous steps. Add the following attributes to the TextView:

TextView #2 Attribute	**Value**
layout_width	"match_parent"
layout_height	"wrap_content"
android:id	"@+id/article_subheading"
android:layout_below	"@id/article_heading"
android:padding	"@dimen/padding_regular"
android:textAppearance	"@android:style/TextAppearance.DeviceDefault"
android:text	"Article Subtitle"

Because you extracted the dimension resource for the "10dp" string to padding_regular in the previously created TextView, you can use "@dimen/padding_regular" for the android:padding attribute in this TextView.

10. Extract the string resource for the android:text attribute's hardcoded string "Article Subtitle" in the TextView to article_subtitle.
11. In the "Hello World" TextView element, delete the layout_constraint attributes:

```
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent"
```

12. Add the following TextView attributes to the "Hello World" TextView element, and change the android:text attribute:

TextView Attribute	**Value**
android:id	"@+id/article"
android:layout_below	"@id/article_subheading"
android:lineSpacingExtra	"5sp"
android:padding	"@dimen/padding_regular"
android:text	Change to "Article text"

13. Extract the string resource for "Article text" to article_text and extract the dimension resource for "5sp"to line_spacing.
14. Reformat and align the code by choosing Code > Reformat Code. It is a good practice to reformat and align your code so that it is easier for you and others to understand.

2.2 Add the text of the article

In a real app that accesses magazine or newspaper articles, the articles that appear would probably come from an online source through a content provider or might be saved in advance in a database on the device.

For this practical, you will create the article as a single long string in the strings.xml resource.

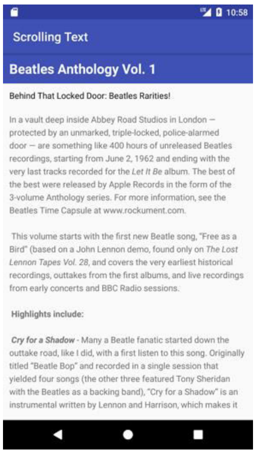
1. In the **app > res > values** folder, open **strings.xml**.
2. Open any text file with a large amount of text.
3. Enter the values for the strings article_title and article_subtitle with either a made-up title and subtitle or use the values in the strings.xml file of the finished ScrollingText app. Make the string values single-line text without HTML tags or multiple lines.
4. Enter or copy and paste text for the article_text string.

You can use the text in your text file, or use the text provided for the article_text string in the strings.xml file of the finished ScrollingText app. The only requirement for this task is that the text must be long enough so that it doesn't fit on the screen.

3 Run the App

Run the app. The article appears, but the user can't scroll the article because you haven't yet included a ScrollView (which you will do next).

Note also that tapping a web link does not currently do anything. You will also fix this next.



Beatles Anthology Vol. 1

Behind That Locked Door: Beatles Rarities!

In a vault deep inside Abbey Road Studios in London — protected by an unmarked, triple-locked, police-alarmed door — are something like 400 hours of unreleased Beatles recordings, starting from June 2, 1962 and ending with the very last tracks recorded for the *Let It Be* album. The best of the best were released by Apple Records in the form of the 3-volume Anthology series. For more information, see the Beatles Time Capsule at www.rockument.com.

This volume starts with the first new Beatle song, "Free as a Bird" (based on a John Lennon demo, found only on *The Lost Lennon Tapes Vol. 28*, and covers the very earliest historical recordings, outtakes from the first albums, and live recordings from early concerts and BBC Radio sessions.

Highlights include:

Cry for a Shadow - Many a Beatle fanatic started down the outtake road, like I did, with a first listen to this song. Originally titled "Beatle Bop" and recorded in a single session that yielded four songs (the other three featured Tony Sheridan with the Beatles as a backing band), "Cry for a Shadow" is an instrumental written by Lennon and Harrison, which makes it

4 Fix Stuff - Add a ScrollView and an active web link

4.1 Add the autoLink attribute for active web links

Add the android:autoLink="web" attribute to the article TextView. The XML code for this TextView now looks like this:

```
<TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/article"
        android:autoLink="web"
        android:layout_below="@id/article_subheading"
        android:lineSpacingExtra="@dimen/line_spacing"
        android:padding="@dimen/padding_regular"
        android:text="@string/article_text" />
```

4.2 Add a ScrollView to the layout

To make a View (such as a TextView) scrollable, embed the View inside a ScrollView.

1. Add a ScrollView between the article_subheading TextView and the article TextView. As you enter **<ScrollView**, Android Studio automatically adds </ScrollView> at the end, and presents the android:layout_width and android:layout_height attributes with suggestions.
2. Choose **wrap_content** from the suggestions for both attributes.

The code for the two TextView elements and the ScrollView now looks like this:

```
<TextView
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:id="@+id/article_subheading"
   android:layout_below="@id/article_heading"
   android:padding="@dimen/padding_regular"
   android:text="@string/article_subtitle"
        android:textAppearance=
                     "@android:style/TextAppearance.DeviceDefault"/>

<ScrollView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"></ScrollView>

<TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/article"
        android:autoLink="web"
        android:layout_below="@id/article_subheading"
        android:lineSpacingExtra="@dimen/line_spacing"
        android:padding="@dimen/padding_regular"
        android:text="@string/article_text" />
```

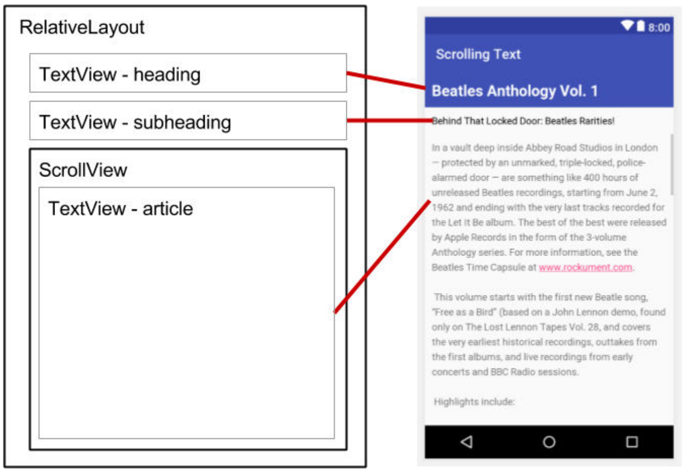
3. Move the ending </ScrollView> code after the article TextView so that the article TextView attributes are entirely inside the ScrollView.
4. Remove the following attribute from the article TextView and add it to the ScrollView:

```
android:layout_below="@id/article_subheading"
```

With the above attribute, the ScrollView element will appear below the article subheading. The article is inside the ScrollView element.

5. Choose **Code > Reformat Code** to reformat the XML code so that the article TextView now appears indented inside the <ScrollView code.
6. Click the **Preview** tab on the right side of the layout editor to see a preview of the layout.

The layout now looks like the right side of the following figure:



Beatles Anthology Vol. 1

Behind That Locked Door: Beatles Rarities!

In a vault deep inside Abbey Road Studios in London — protected by an unmarked, triple-locked, police-alarmed door — are something like 400 hours of unreleased Beatles recordings, starting from June 2, 1962 and ending with the very last tracks recorded for the Let It Be album. The best of the best were released by Apple Records in the form of the 3-volume Anthology series. For more information, see the Beatles Time Capsule at www.rockument.com.

This volume starts with the first new Beatle song, "Free as a Bird" (based on a John Lennon demo, found only on The Lost Lennon Tapes Vol. 28, and covers the very earliest historical recordings, outtakes from the first albums, and live recordings from early concerts and BBC Radio sessions.

Highlights include:

5 Run the App

To examine how the text scrolls:

1. Run the app on a device or emulator.

Swipe up and down to scroll the article. The scroll bar appears in the right margin as you scroll.

Tap the web link to go to the web page. The android:autoLink attribute turns any recognizable URL in the TextView (such as www.rockument.com) into a web link.

2. Rotate your device or emulator while running the app. Notice how the scrolling view widens to use the full display and still scrolls properly.
3. Run the app on a tablet or tablet emulator. Notice how the scrolling view widens to use the full display and still scrolls properly.

Android Development
W04 Practical - Activity Life Cycle

1 App Overview

This is a pretty straightforward practical.

I've included, in this week's resources, a starting code base that extends the TwoActivities App you built last week. It contains new functionality to return a String from the SecondActivity as a result following a call to startActivityForResult() in the MainActivity.

In this session you are going to implement some of the lifecycle callbacks that we've just discussed, and you will explore them by adding logging statements and observing the logs during runtime.

The changes you make will not affect the observed user behaviour.

2 Explore the Project

The project should look familiar to you but look at a number of significant differences (additions)

2.1 Linking the Button Views to onClick handlers is done in code

This is another way to implement UI callbacks that we discussed in last week's sessions.

- In the code below from MainActivity the onClick() listener not a class method (linked from the XML) but a class instance variable containing that method.

```
// Declare an OnClickListener for our button
private OnClickListener buttonListener = new OnClickListener() {
    @Override
    public void onClick(View v) {
        Log.d(LOG_TAG, "Button clicked!");
        Intent intent = new Intent(getApplicationContext(), SecondActivity.class);
        String message = mMessageEditText.getText().toString();
        intent.putExtra(EXTRA_MESSAGE, message);
        startActivityForResult(intent, TEXT_REQUEST);
    }
};
```

- The View is linked to the callback in the Activity onCreate() method as follows:

```
@Override
protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    // Initialize all the view variables.
    mMessageEditText = findViewById(R.id.editText_main);
    mReplyHeadTextView = findViewById(R.id.text_header_reply);
    mReplyTextView = findViewById(R.id.text_message_reply);
    mSendButton = (Button) findViewById(R.id.button_main);

    // Set the 'onClick' listener for the button
    mSendButton.setOnClickListener(buttonListener);

}
```

2.2 MainActivity invokes SecondActivity using startActivityForResult

- The startActivityForResult call takes an intent but additionally an int request id argument so that we can check in the callback that we're being told about the request we sent.

```
// Unique tag required for the intent extra
public static final String EXTRA_MESSAGE = "uk.ac.hope.twohopeactivities.extra.MESSAGE";
// Unique tag for the intent reply
public static final int TEXT_REQUEST = 1;

< CODE >v

Intent intent = new Intent(getApplicationContext(), SecondActivity.class);
intent.putExtra("EXTRA_MESSAGE", messageTextView.getText().toString());
startActivityForResult(intent, SEND_MESSAGE_REQUEST_ID);
```

2.3 MainActivity overrides (implements) onActivityResult

- The main activity implements the onActivityResult callback in order to handle the result of implementing SecondActivity.

```
// onActivityResult implementation
@Override
public void onActivityResult(int requestCode, int resultCode, Intent intent) {

    super.onActivityResult(requestCode, resultCode, intent);
    // Identify activity
    if (requestCode == SEND_MESSAGE_REQUEST_ID) {
        // Activity succeeded
        if (resultCode == RESULT_OK) {
            String reply = intent.getStringExtra(SecondActivity.EXTRA_REPLY);
            replyTextView.setText("Reply: " + reply);
        }
    }
}
```

3 Add lifecycle callbacks

3.1 Implement callbacks in MainActivity

1. Open the TwoActivities project in Android Studio, and open **MainActivity** in the **Project > Android** pane.
2. In the onCreate() method, add the following log statements:

```
Log.d(LOG_TAG, "-------");
Log.d(LOG_TAG, "onCreate");
```

3. Add an override for the onStart() callback, with a statement to the log for that event:

```
@Override
public void onStart(){
    super.onStart();
    Log.d(LOG_TAG, "onStart");
}
```

For a shortcut, select **Code > Override Methods** in Android Studio. A dialog appears with all of the possible methods you can override in your class. Choosing one or more callback methods from the list inserts a complete template for those methods, including the required call to the superclass.

4. Use the onStart() method as a template to implement the onPause(), onRestart(), onResume(), onStop(), and onDestroy() lifecycle callbacks.

All the callback methods have the same signatures (except for the name). If you **Copy** and **Paste** onStart() to create these other callback methods, don't forget to update the contents to call the right method in the superclass, and to log the correct method.

5. Run your app.
6. Click the **Logcat** tab at the bottom of Android Studio to show the **Logcat** pane. You should see three log messages showing the three lifecycle states the Activity has transitioned through as it started:

3.2 Implement lifecycle callbacks in SecondActivity

1. Open SecondActivity.
2. At the top of the class, add a constant for the LOG_TAG variable:

```
private static final String LOG_TAG = SecondActivity.class.getSimpleName();
```

3. Add the lifecycle callbacks and log statements to the second Activity. (You can Copy and Paste the callback methods from MainActivity.)
4. Add a log statement to the returnReply() method just before the finish() method:

```
Log.d(LOG_TAG, "End SecondActivity");
```

3.3 Observe the log as the App runs

Experiment using your app and note that the lifecycle events that occur in response to different actions. In particular, try these things:

- Use the app normally (send a message, reply with another message).
- Use the Back button to go back from the second Activity to the main Activity.
- Use the Up arrow in the app bar to go back from the second Activity to the main Activity.
- Rotate the device on both the main and second Activity at different times in your app and observe what happens in the log and on the screen.
- Press the overview button (the square button to the right of Home) and close the app (tap the **X**).
- Return to the home screen and restart your app.

4 Save and Restore the Activity instance state

Depending on system resources and user behaviour, each Activity in your app may be destroyed and reconstructed far more frequently than you might think.

You may have noticed this behaviour in the last section when you rotated the device or emulator. Rotating the device is one example of a device configuration change. Although rotation is the most common one, all configuration changes result in the current Activity being destroyed and recreated as if it were new. If you don't account for this behaviour in your code, when a configuration change occurs, your Activity layout may revert to its default appearance and initial values, and your users may lose their place, their data, or the state of their progress in your app.

The state of each Activity is stored as a set of key/value pairs in a Bundle object called the Activity instance state. The system saves default state information to instance state Bundle just before the Activity is stopped and passes that Bundle to the new Activity instance to restore.

To keep from losing data in an Activity when it is unexpectedly destroyed and recreated, you need to implement the onSaveInstanceState() method. The system calls this method on your Activity (between onPause() and onStop()) when there is a possibility the Activity may be destroyed and recreated.

The data you save in the instance state is specific to only this instance of this specific Activity during the current app session. When you stop and restart a new app session, the Activity instance state is lost and the Activity reverts to its default appearance. If you need to save user data between app sessions, use shared preferences or a database.

4.1 Save the Activity instance state with onSaveInstanceState()

You may have noticed that rotating the device does not affect the state of the second Activity at all. This is because the second Activity layout and state are generated from the layout and the Intent that activated it. Even if the Activity is recreated, the Intent is still there and the data in that Intent is still used each time the onCreate() method in the second Activity is called.

In addition, you may notice that in each Activity, any text you typed into message or reply EditText elements is retained even when the device is rotated. This is because the state information of some of the View elements in your layout are automatically saved across configuration changes, and the current value of an EditText is one of those cases.

So, the only Activity state you're interested in are the TextView elements for the reply header and the reply text in the main Activity. Both TextView elements are invisible by default; they only appear once you send a message back to the main Activity from the second Activity.

In this task you add code to preserve the instance state of these two TextView elements using **onSaveInstanceState**().

1. Open MainActivity.
2. Add this skeleton implementation of onSaveInstanceState() to the Activity, or use Code > Override Methods to insert a skeleton override.

```
@Override
public void onSaveInstanceState(Bundle outState) {
          super.onSaveInstanceState(outState);
}
```

3. Check to see if the header is currently visible, and if so put that visibility state into the state Bundle with the putBoolean() method and the key "reply_visible".

```
if (mReplyHeadTextView.getVisibility() == View.VISIBLE) {
    outState.putBoolean("reply_visible", true);
}
```

Note that the reply header and text are marked invisible until there is a reply from the second Activity. If the header is visible, then there is reply data that needs to be saved. Note that we're only interested in that visibility state — the actual text of the header doesn't need to be saved, because that text never changes.

4. Inside that same check, add the reply text into the Bundle.

```
outState.putString("reply_text",mReplyTextView.getText().toString());
```

If the header is visible you can assume that the reply message itself is also visible. You don't need to test for or save the current visibility state of the reply message. Only the actual text of the message goes into the state Bundle with the key "reply_text".
You save the state of only those View elements that might change after the Activity is created. The other View elements in your app (the EditText, the Button) can be recreated from the default layout at any time.

Note that the system will save the state of some View elements, such as the contents of the EditText.

4.2 Restore the Activity instance state in onCreate()

Once you've saved the Activity instance state, you also need to restore it when the Activity is recreated. You can do this either in onCreate(), or by implementing the onRestoreInstanceState() callback, which is called after onStart() after the Activity is created.

Most of the time the better place to restore the Activity state is in onCreate(), to ensure that the UI, including the state, is available as soon as possible. It is sometimes convenient to do it in onRestoreInstanceState() after all of the initialization has been done, or to allow subclasses to decide whether to use your default implementation.

1. In the onCreate() method, after the View variables are initialized with findViewById(), add a test to make sure that savedInstanceState is not null.

```
// Initialize all the view variables.
mMessageEditText = findViewById(R.id.editText_main);
mReplyHeadTextView = findViewById(R.id.text_header_reply);
mReplyTextView = findViewById(R.id.text_message_reply);

// Restore the state.
if (savedInstanceState != null) {
}
```

When your Activity is created, the system passes the state Bundle to onCreate() as its only argument. The first time onCreate() is called and your app starts, the Bundle is null—there's no existing state the first time your app starts. Subsequent calls to onCreate() have a bundle populated with the data you stored in onSaveInstanceState().

2. Inside that check, get the current visibility (true or false) out of the Bundle with the key "reply_visible".

```
if (savedInstanceState != null) {
    boolean isVisible = savedInstanceState.getBoolean("reply_visible");
}
```

3. Add a test below that previous line for the isVisible variable.

```
if (isVisible) {
}
```

If there's a reply_visible key in the state Bundle (and isVisible is therefore true), you will need to restore the state.

4. Inside the isVisible test, make the header visible.

```
mReplyHeadTextView.setVisibility(View.VISIBLE);
```

5. Get the text reply message from the Bundle with the key "reply_text" and set the reply TextView to show that string.

```
mReplyTextView.setText(savedInstanceState.getString("reply_text"));
```

5. Make the reply TextView visible as well:

```
mReplyTextView.setVisibility(View.VISIBLE);
```

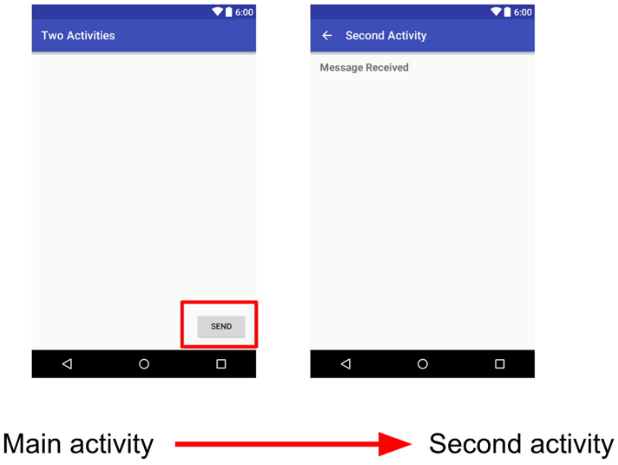
6. Run the app. Try rotating the device or the emulator to ensure that the reply message (if there is one) remains on the screen after the Activity is recreated.

Android Development
05 Practical - Activities

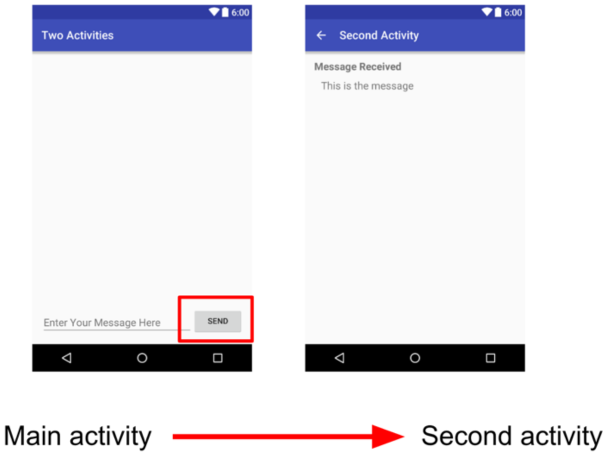
1 App Overview

You will create and build an app called Two Activities that, unsurprisingly, contains two Activity implementations. You build the app in two stages.

In the first stage, you create an app whose main activity contains one button, **Send**. When the user clicks this button, your main activity uses an intent to start the second activity.



In the second stage, you add an EditText view to the main activity. The user enters a message and clicks **Send**. The main activity uses an intent to start the second activity and send the user's message to the second activity. The second activity displays the message it received.



2 Create the TwoActivities project

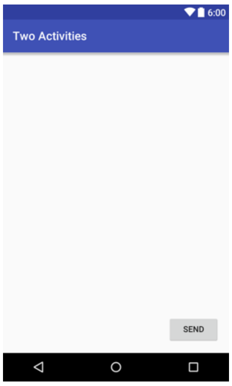
Set up the initial project with a main Activity, define the layout, and define a skeleton method for the onClick button event

2.1 Create the TwoActivities project

1. Start Android Studio and create a new Android Studio project. Name your app **Two Activities** and choose the same **Phone and Tablet** settings that you used in previous practicals. The project folder is automatically named TwoActivities, and the app name that appears in the app bar will be "Two Activities".
2. Choose **Empty Views Activity** for the Activity template. Click **Next.**
3. Change the Name package name if you wish but leave the language as Java and the default SDK as API 24 Android 7.0
4. Click **Finish**.

2.2 Define the layout for the main Activity

1. Open res > layout > activity_main.xml in the Project > Android pane. The layout editor appears.
2. Click the Design tab if it is not already selected and delete the TextView (the one that says "Hello World") in the Component Tree pane.
3. With Autoconnect turned on (the default setting), drag a Button from the Palette pane to the lower right corner of the layout. Autoconnect creates constraints for the Button.
4. In the Attributes pane, set the ID to button_main,
the layout_width and layout_height to wrap_content, and enter Send for the Text field. The layout should now look like this:



5. Click the **Text** tab to edit the XML code. Add the following attribute to the Button:

```
android:onClick="launchSecondActivity"
```

The attribute value is underlined in red because the launchSecondActivity() method has not yet been created. Ignore this error for now; you fix it in the next task.

6. Extract the string resource, as described in a previous practical, for "Send" and use the name button_main for the resource.

The XML code for the Button should look like the following:

```
Button
        android:id="@+id/button_main"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="16dp"
        android:layout_marginRight="16dp"
        android:text="@string/button_main"
        android:onClick="launchSecondActivity"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintRight_toRightOf="parent" />
```

2.3 Define the Button action

Now you implement the launchSecondActivity() method you referred to in the layout for the android:onClick attribute.

1. Click on "launchSecondActivity" in the activity_main.xml XML code.
2. Press Alt+Enter (Option+Enter on a Mac) and select **Create 'launchSecondActivity(View)' in 'MainActivity.**
   The MainActivity file opens, and Android Studio generates a skeleton method for the launchSecondActivity() handler.
3. Inside launchSecondActivity(), add a Log statement that says "Button Clicked!"

```
Log.d(LOG_TAG, "Button clicked!");
```

4. At the top of the MainActivity class, add a constant for the LOG_TAG variable:

```
private static final String LOG_TAG =
                    MainActivity.class.getSimpleName();
```

This constant uses the name of the class itself as the tag.

5. Run your app. When you click the **Send** button you see the "Button Clicked!" message in the **Logcat** pane. If there's too much output in the monitor, type **MainActivity** into the search box, and the **Logcat** pane will only show lines that match that tag.

The code for MainActivity should look as follows:

```
package com.example.android.twoactivities;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;

public class MainActivity extends AppCompatActivity {
   private static final String LOG_TAG =
                              MainActivity.class.getSimpleName();

   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);
   }

   public void launchSecondActivity(View view) {
       Log.d(LOG_TAG, "Button clicked!");
   }
}
```

3 Create and launch a second activity

Each new activity you add to your project has its own layout and Java files, separate from those of the main activity. They also have their own <activity> elements in the AndroidManifest.xml file. As with the main activity, new activity implementations that you create in Android Studio also extend from the AppCompatActivity class.

Each activity in your app is only loosely connected with other activities. However, you can define an activity as a parent of another activity in the AndroidManifest.xml file. This parent-child relationship enables Android to add navigation hints such as left-facing arrows in the title bar for each activity.

3.1 Create a second Activity

1. Click the **app** folder for your project and choose **File > New > Activity > Empty Activity**.
2. Name the new Activity **SecondActivity**. Make sure **Generate Layout File** and **Backwards Compatibility (AppCompat)** are checked. The layout name is filled in as activity_second. Do *not* check the **Launcher Activity** option.
3. Click **Finish**. Android Studio adds both a new Activity layout (activity_second.xml) and a new Java file (SecondActivity.java) to your project for the new Activity. It also updates the AndroidManifest.xml file to include the new Activity.

3.2 Add the new Activity to the manifest

1. Open manifests > AndroidManifest.xml.
2. Find the <activity> element that Android Studio created for the second Activity.

```
<activity android:name=".SecondActivity"></activity>
```

3. Replace the entire <activity> element with the following:

```
<activity android:name=".SecondActivity"
    android:label = "Second Activity"
    android:parentActivityName=".MainActivity">
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=
                    "com.example.android.twoactivities.MainActivity" />
</activity>
```

The label attribute adds the title of the Activity to the app bar.

With the parentActivityName attribute, you indicate that the main activity is the parent of the second activity. This relationship is used for Up navigation in your app: the app bar for the second activity will have a left-facing arrow so the user can navigate "upward" to the main activity.
With the <meta-data> element, you provide additional arbitrary information about the activity in the form of key-value pairs. In this case the metadata attributes do the same thing as the android:parentActivityName attribute—they define a relationship between two activities for upward navigation. These metadata attributes are required for older versions of Android, because the android:parentActivityName attribute is only available for API levels 16 and higher.

4. Extract a string resource for "Second Activity" in the code above and use activity2_name as the resource name.

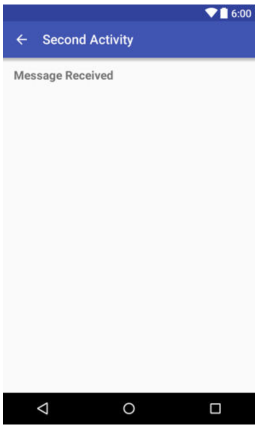
3.3 Define the layout for the second Activity

1. Open **activity_second.xml** and click the **Design** tab if it is not already selected.
2. Drag a **TextView** from the **Palette** pane to the top left corner of the layout and add constraints to the top and left sides of the layout. Set its attributes in the **Attributes** pane as follows:

Attribute	Value
id	text_header
Top margin	16
Left margin	8
layout_width	wrap_content
layout_height	wrap_content

text	Message Received
textAppearance	AppCompat.Medium
textStyle	B (bold)

The value of **textAppearance** is a special Android theme attribute that defines basic font styles. You learn more about themes in a later lesson.
The layout should now look like this:



3. Click the **Text** tab to edit the XML code and extract the "Message Received" string into a resource named text_header.
4. Add the android:layout_marginLeft="8dp" attribute to the TextView to complement the layout_margin Start attribute for older versions of Android.

The XML code for activity_second.xml should be as follows:

```
<android.support.constraint.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.android.twoactivities.SecondActivity">

    <TextView
        android:id="@+id/text_header"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginTop="16dp"
        android:text="@string/text_header"
        android:textAppearance=
                  "@style/TextAppearance.AppCompat.Medium"
        android:textStyle="bold"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
```

3.4 Add an Intent to the main Activity

Now add an explicit Intent to the main Activity. This Intent is used to activate the second Activity when the **Send** button is clicked.

1. Open **MainActivity**.
2. Create a new Intent in the launchSecondActivity() method.

The Intent constructor takes two arguments for an explicit Intent: an application Context and the specific component that will receive that Intent. Here you should use this as the Context, and SecondActivity.class as the specific class:

```
Intent intent = new Intent(this, SecondActivity.class);
```

3. Call the startActivity() method with the new Intent as the argument.
4. Run the app.

When you click the **Send** button, MainActivity sends the Intent and the Android system launches SecondActivity, which appears on the screen. To return to MainActivity, click the **Up** button (the left arrow in the app bar) or the Back button at the bottom of the screen.

4 Send data from the main Activity to the second Activity

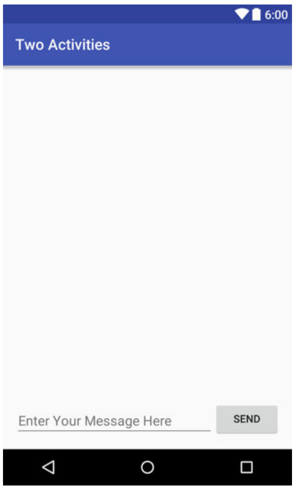
You will modify the explicit intent in MainActivity to include additional data (in this case, a user-entered string) in the intent extra Bundle. You then modify SecondActivity to get that data back out of the intent extra Bundle and display it on the screen.

4.1 Add an EditText to the MainActivity layout

1. Open activity_main.xml.
2. Drag a Plain Text (EditText) element from the Palette pane to the bottom of the layout and add constraints to the left side of the layout, the bottom of the layout, and the left side of the Send Button. Set its attributes in the Attributes pane as follows:

Attribute	Value
id	editText_main
Right margin	8
Left margin	8
Bottom margin	16
layout_width	match_constraint
layout_height	wrap_content
inputType	textLongMessage
hint	Enter Your Message Here
text	(Delete any text in this field)

The new layout in activity_main.xml looks like this:



4.2 Add a string to the Intent extras

The Intent extras are key/value pairs in a Bundle. A Bundle is a collection of data, stored as key/value pairs. To pass information from one Activity to another, you put keys and values into the Intent extra Bundle from the sending Activity, and then get them back out again in the receiving Activity.

1. Open MainActivity.
2. Add a public constant at the top of the class to define the key for the Intent extra:

```
public static final String EXTRA_MESSAGE =
                         "com.example.android.twoactivities.extra.MESSAGE";
```

3. Add a private variable at the top of the class to hold the EditText:

```
private EditText mMessageEditText;
```

4. In the onCreate() method, use findViewById() to get a reference to the EditText and assign it to that private variable:

```
mMessageEditText = findViewById(R.id.editText_main);
```

5. In the launchSecondActivity() method, just under the new Intent, get the text from the EditText as a string:

```
String message = mMessageEditText.getText().toString();
```

6. Add that string to the Intent as an extra with the EXTRA_MESSAGE constant as the key and the string as the value:

```
intent.putExtra(EXTRA_MESSAGE, message);
```

The onCreate() method in MainActivity should now look like the following:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mMessageEditText = findViewById(R.id.editText_main);
}
```

The launchSecondActivity() method in MainActivity should now look like the following:

```
public void launchSecondActivity(View view) {
        Log.d(LOG_TAG, "Button clicked!");
        Intent intent = new Intent(this, SecondActivity.class);
        String message = mMessageEditText.getText().toString();
        intent.putExtra(EXTRA_MESSAGE, message);
        startActivity(intent);
}
```

4.3 Add a TextView to SecondActivity for the message

1. Open activity_second.xml.
2. Drag another TextView to the layout underneath the text_header TextView and add constraints to the left side of the layout and to the bottom of text_header.
3. Set the new TextView attributes in the Attributes pane as follows:

Attribute	Value
id	text_message
Top margin	8
Left margin	8
layout_width	wrap_content
layout_height	wrap_content
text	(Delete any text in this field)
textAppearance	AppCompat.Medium

The new layout looks the same as it did in the previous task, because the new TextView does not (yet) contain any text, and thus does not appear on the screen.
The XML code for the activity_second.xml layout should look something like the following:

```
android.support.constraint.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.android.twoactivities.SecondActivity">

    <TextView
        android:id="@+id/text_header"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="16dp"
        android:text="@string/text_header"
        android:textAppearance=
                          "@style/TextAppearance.AppCompat.Medium"
        android:textStyle="bold"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/text_message"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/text_header" />
</android.support.constraint.ConstraintLayout>
```

4.4 Modify SecondActivity to get the extras and display the message

1. Open **SecondActivity** to add code to the onCreate() method.
2. Get the Intent that activated this Activity:

```
Intent intent = getIntent();
```

3. Get the string containing the message from the Intent extras using the MainActivity.EXTRA_MESSAGE static variable as the key:

```
String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
```

4. Use findViewByID() to get a reference to the TextView for the message from the layout:

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

5. Set the text of the TextView to the string from the Intent extra:

```
textView.setText(message);
```

6. Run the app. When you type a message in MainActivity and click **Send**, SecondActivity launches and displays the message.

The SecondActivity onCreate() method should look as follows:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_second);
    Intent intent = getIntent();
    String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
    TextView textView = findViewById(R.id.text_message);
    textView.setText(message);
}
```

Android Development
02 Practical (b) The layout editor

1 Overview

You just built a simple App using a ConstraintLayout.

ConstraintLayout was designed to make it easy to drag UI elements into the layout editor.

ConstraintLayout is a ViewGroup, which is a special View that can contain other View objects (called *children* or *child views*). This practical shows more features of ConstraintLayout and the layout editor.

This practical also introduces two other ViewGroup subclasses:

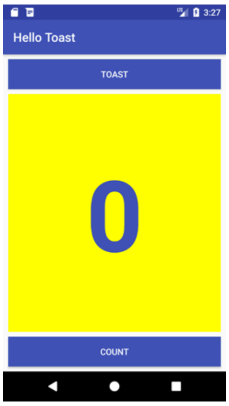
- LinearLayout: A group that aligns child View elements within it horizontally or vertically.
- RelativeLayout: A group of child View elements in which each View element is positioned and aligned relative to other View element within the ViewGroup. Positions of the child View elements are described in relation to each other or to the parent ViewGroup.

1.1 What you'll do

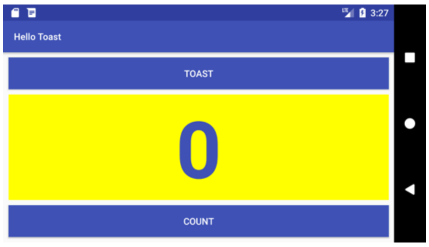
- Create a layout variant for a horizontal display orientation.
- Create a layout variant for tablets and larger displays.
- Modify the layout to add constraints to the UI elements.
- Use ConstraintLayout baseline constraints to align elements with text.
- Use ConstraintLayout pack and align buttons to align elements.

1.2 App Overview

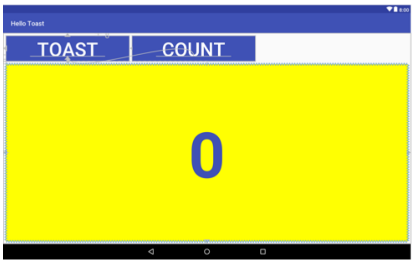
The Hello Toast app in a previous practical uses ConstraintLayout to arrange the UI elements in the Activity layout, as shown in the figure below.



To gain more practice with ConstraintLayout, you will create a variant of this layout for horizontal orientation as shown in the figure below.



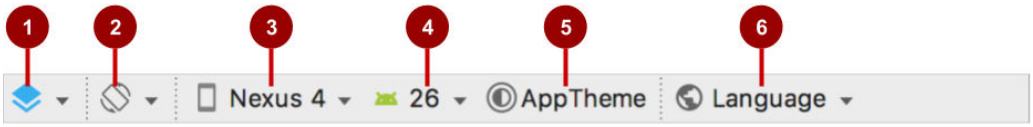
You will also learn how to use baseline constraints and some of the alignment features of ConstraintLayout by creating another layout variant for tablet displays.



You also play with other ViewGroup subclasses such as LinearLayout and RelativeLayout, and change the Hello Toast app layout to use them.

2 Create Layout Variants

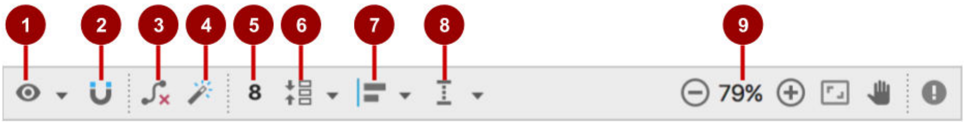
In this task you will create variants of your layout for horizontal (also known as *landscape*) and vertical (also known as *portrait*) orientations for phones, and for larger displays such as tablets.
You will be using some of the buttons in the top two toolbars of the layout editor. The top toolbar lets you configure the appearance of the layout preview in the layout editor:



In the figure above:

1. **Select Design Surface**: Select **Design** to display a color preview of your layout, or **Blueprint** to show only outlines for each UI element. To see *both* panes side by side, select **Design + Blueprint**.
2. **Orientation in Editor**: Select **Portrait** or **Landscape** to show the preview in a vertical or horizontal orientation. This is useful for previewing the layout without having to run the app on an emulator or device. To create alternative layouts, select **Create Landscape Variation** or other variations.
3. **Device in Editor**: Select the device type (phone/tablet, Android TV, or Android Wear).
4. **API Version in Editor**: Select the version of Android to use to show the preview.
5. **Theme in Editor**: Select a theme (such as **AppTheme**) to apply to the preview.
6. **Locale in Editor**: Select the language and locale for the preview. This list displays only the languages available in the string resources (see the lesson on localization for details on how to add languages). You can also choose **Preview as Right To Left** to view the layout as if an RTL language had been chosen.

The second toolbar lets you configure the appearance of UI elements in a ConstraintLayout, and to zoom and pan the preview:



In the figure above:

1. **Show**: Choose **Show Constraints** and **Show Margins** to show them in the preview, or to stop showing them.
2. **Autoconnect**: Enable or disable Autoconnect. With Autoconnect enabled, you can drag any element (such as a Button) to any part of a layout to generate constraints against the parent layout.
3. **Clear All Constraints**: Clear all constraints in the entire layout.
4. **Infer Constraints**: Create constraints by inference.
5. **Default Margins**: Set the default margins.
6. **Pack**: Pack or expand the selected elements.
7. **Align**: Align the selected elements.
8. **Guidelines**: Add vertical or horizontal guidelines.
9. Zoom/pan controls: Zoom in or out.

2.1 Create a layout variant for horizontal orientation

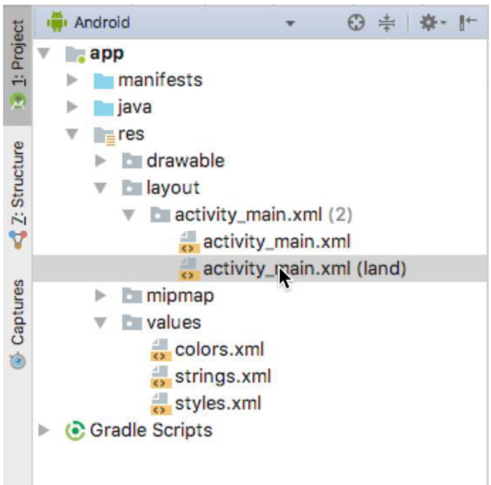
Use the **Orientation in Editor** button to compare the layout in landscape and portrait mode.

The visual difference between vertical and horizontal orientations for this layout is that the digit (0) in the show_count TextView element is too low for the horizontal orientation—too close to the **Count** button. Depending on which device or emulator you use, the TextView element may appear too large or not centred because the text size is fixed to 160sp.
To fix this for horizontal orientations while leaving vertical orientations alone, you can create variant of the Hello Toast app layout that is different for a horizontal orientation. Follow these steps:

1. Click the **Orientation in Editor** button in the top toolbar.
2. Choose **Create Landscape Variation**.

A new editor window opens with the **land/activity_main.xml** tab showing the layout for the landscape (horizontal) orientation. You can change this layout, which is specifically for horizontal orientation, without changing the original portrait (vertical) orientation.

3. In the **Project > Android** pane, look inside the **res > layout** directory, and you will see that Android Studio automatically created the variant for you, called activity_main.xml (land).



2.2 Change the layout for horizontal orientation

You can use the Attributes pane in the **Design** tab to set or change attributes, but it can sometimes be quicker to use the **Text** tab to edit the XML code directly.

To change the layout, follow these steps:

1. The **land/activity_main.xml** tab should still be open in the layout editor; if not, double-click the **activity_main.xml (land)** file in the **layout** directory.
2. Click the **Text** tab and the **Preview** tab (if not already selected).
3. Find the TextView element in the XML code.
4. Change the android:textSize="160sp" attribute to android:textSize="120sp". The layout preview shows the result:
5. Choose different devices in the **Device in Editor** dropdown menu to see how the layout looks on different devices in horizontal orientation.

In the editor pane, the **land/activity_main.xml** tab shows the layout for horizontal orientation. The **activity_main.xml** tab shows the unchanged layout for vertical orientation. You can switch back and forth by clicking the tabs.

6. Run the app on an emulator or device and switch the orientation from vertical to horizontal to see both layouts.

2.3 Create a layout variant for tablets

You can preview the layout for different devices by clicking the **Device in Editor** button ▯ Nexus 5 ▾ in the top toolbar. If you pick a device such as **Nexus 10** (a tablet) from the menu, you can see that the layout is not ideal for a tablet screen—the text of each Button is too small, and the arrangement of the Button elements at the top and bottom is not ideal for a large-screen tablet.

To fix this for tablets while leaving the phone-size horizontal and vertical orientations alone, you can create variant of the layout that is completely different for tablets. Follow these steps:

1. Click the **Design** tab (if not already selected) to show the design and blueprint panes.
2. Click the **Orientation in Editor** button ▾ in the top toolbar.
3. Choose **Create layout x-large Variation**.

A new editor window opens with the **xlarge/activity_main.xml** tab showing the layout for a tablet-sized device. The editor also picks a tablet device, such as the Nexus 9 or Nexus 10, for the preview. You can change this layout, which is specifically for tablets, without changing the other layouts.

2.4 Change the layout for Tablets

You can use the Attributes pane in the **Design** tab to change attributes for this layout.

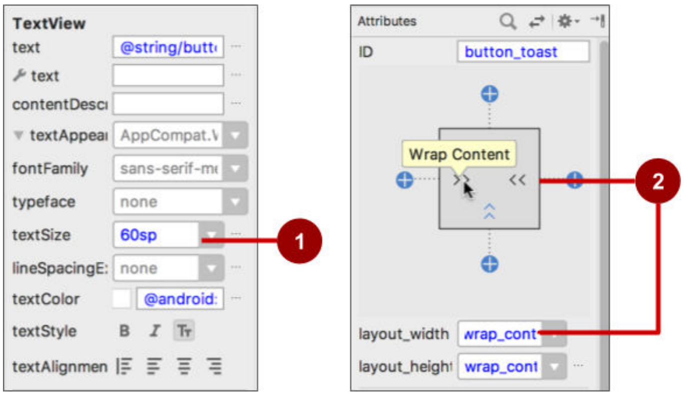
1. Turn off the Autoconnect tool in the toolbar. For this step, ensure that the tool is disabled:
2. Clear all constraints in the layout by clicking the **Clear All Constraints** button in the toolbar.

With constraints removed, you can move and resize the elements on the layout freely.

3. The layout editor offers resizing handles on all four corners of an element to resize it. In the **Component Tree**, select the TextView called show_count. To get the TextView out of the way so that you can freely drag the Button elements, drag a corner of it to resize it.

Resizing an element hardcodes the width and height dimensions. Avoid hardcoding the size dimensions for most elements, because you can't predict how hardcoded dimensions will look on screens of different sizes and densities. You are doing this now just to move the element out of the way, and you will change the dimensions in another step.

4. Select the button_toast Button in the **Component Tree**, click the **Attributes** tab to open the **Attributes** pane, and change the textSize to **60sp** (#1 in the figure below) and the layout_width to wrap_content (#2 in the figure below).



TextView

text @string/butt

text

contentDesc

textAppear AppCompat.\

fontFamily sans-serif-me

typeface none

textSize 60sp

lineSpacingE: none

textColor @android:

textStyle B I Tr

textAlignmen

Attributes

ID button_toast

Wrap Content

layout_width wrap_cont

layout_height wrap_cont

As shown on the right side of the figure above (2), you can click the view inspector's width control, which appears in two segments on the left and right sides of the square, until it shows Wrap Content. As an alternative, you can select **wrap_**content from the layout_width menu.
You use wrap_content so that if the Button text is localized into a different language, the Button will appear wider or thinner to accommodate the word in the different language.

5. Select the button_count Button in the Component Tree, change the textSize to 60sp and the layout_width to wrap_content, and drag the Button above the TextView to an empty space in the layout.

2.5 Use a baseline constraint

You can align one UI element that contains text, such as a TextView or Button, with another UI element that contains text. A *baseline constraint* lets you constrain the elements so that the text baselines match.

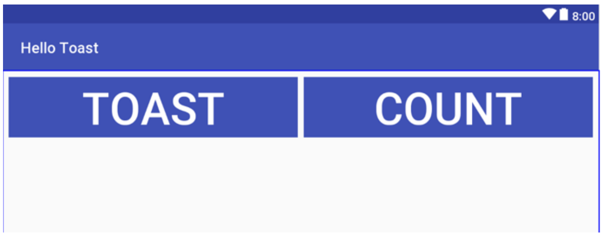
1. Constrain the button_toast Button to the top and left side of the layout, drag the button_count Button to a space near the button_toast Button, and constrain the button_count Button to the left side of the button_toast Button.
2. Using a *baseline constraint*, you can constrain the button_count Button so that its text baseline matches the text baseline of the button_toast Button. Select the button_count element, and then hover your pointer over the element until the baseline constraint button ab appears underneath the element. (you may have to right click the element and select 'Show Baseline' – depends on the exact IDE installation)
3. Click the baseline constraint button. The baseline handle appears, blinking in green. Click and drag a baseline constraint line to the baseline of the button_toast element.

2.6 Expand the buttons horizontally

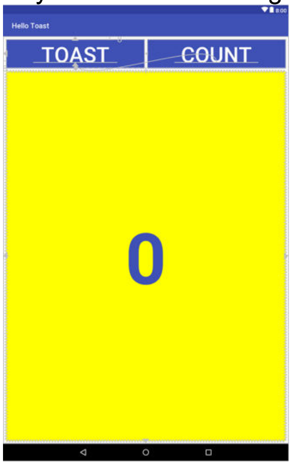
The pack button in the toolbar provides options for packing or expanding selected UI elements. You can use it to equally arrange the Button elements horizontally across the layout.

1. Select the button_count Button in the **Component Tree**, and Shift-select the button_toast Button so that both are selected.
2. Click the pack button in the toolbar, and choose **Expand Horizontally** as shown in the figure below.

The Button elements expand horizontally to fill the layout as shown below.



3. To finish the layout, constraint the show_count TextView to the bottom of the button_toast Button and to the sides and bottom of the layout.
4. The final steps are to change the show_count TextView layout_width and layout_height to Match **Constraints** and the textSize to 200sp. The final layout looks like the figure below.

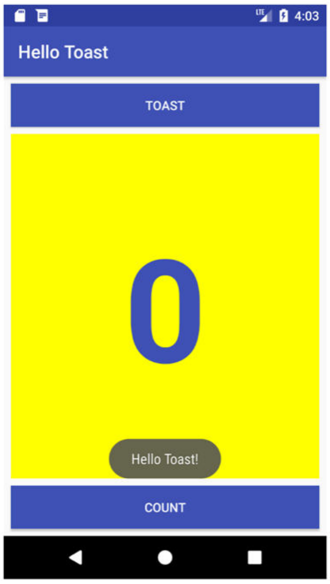


5. Click the **Orientation in Editor** button in the top toolbar and choose **Switch to Landscape**.
6. Run the app on different emulators, and change the orientation after running the app, to see how it looks on different types of devices. You have successfully created an app that can run with a proper UI on phones and tablets that have different screen sizes and densities.

Android Development
02 Practical (a) Interactive UI

1 App Overview

The HelloToast app consists of two Button elements and one TextView. When the user taps the first Button, it displays a short message (a Toast) on the screen. Tapping the second Button increases a "click" counter displayed in the TextView, which starts at zero. Here's what the finished app looks like:



2 Create the Project

A few things to consider here.

When you open the IDE and select New project you will be presented with an application wizard presenting a number of possible starter templates:

- No Activity
- Empty Activity
- Basix Views Activity
- Bottom Navigation Views Activity
- Empty Views Activity
- Navigation Drawer Views Activity
- Responsive Views Activity
- Game Activity (C++)
- Native C++

For our initial coding sessions we will always use the Empty Views Activity. This builds a single Activity ‘Hello World’ Application coded (default) in Java. You can easily expand this template to include all of the functionality present in the other starter templates but, initially, we want to keep things simple.

The other templates include things that we will cover later in the course but which initially add confusion to understanding the Android basics.

- Fragments
- Floating action buttons
- Graph based navigation
- Navigation drawers
- Bottom navigation tabs
- UI data binding
- …

Initially, therefore, you can follow the instructions in section 2.1 or simply open the starter project I've zipped up in Moodle.

2.1 Start Android Studio and create a new project with the following parameters:

Attribute	Value
Application Name	Hello Toast
Company Name	com.example.android (or your own domain)
Phone and Tablet Minimum SDK	Use default
Template	Empty Views Activity
Generate Layout file box	Selected
Backwards Compatibility box	Selected

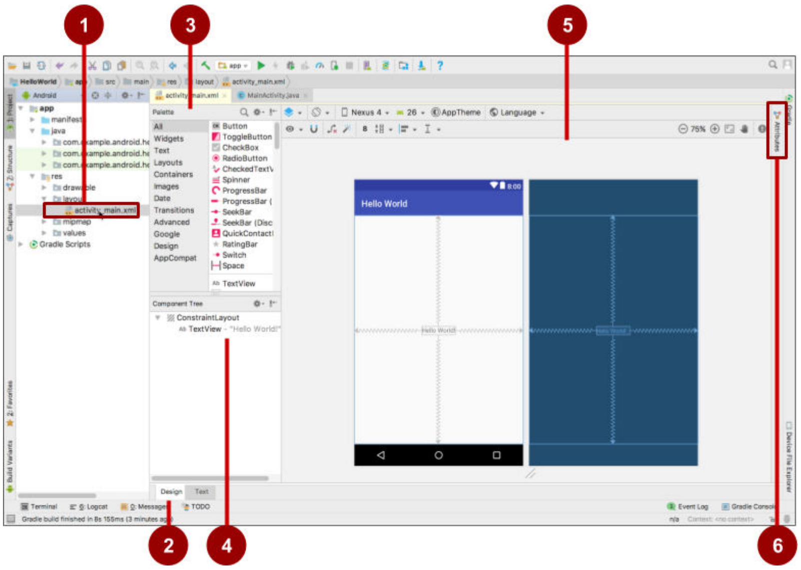
2.2 Run App in AVD

Select **Run > Run app** or click the **Run icon** ▶ in the toolbar to build and execute the app on the emulator or your device.

3 Explore the layout editor

Android Studio provides the layout editor for quickly building an app's layout of user interface (UI) elements. It lets you drag elements to a visual design and blueprint view, position them in the layout, add constraints, and set attributes. *Constraints* determine the position of a UI element within the layout. A constraint represents a connection or alignment to another view, the parent layout, or an invisible guideline.

Explore the layout editor, and refer to the figure below as you follow the numbered steps:



1. In the **app > res > layout** folder in the **Project > Android** pane, double-click the **activity_main.xml** file to open it, if it is not already open.
2. Click the **Design** tab if it is not already selected. You use the **Design** tab to manipulate elements and the layout, and the **Text** tab to edit the XML code for the layout.
3. The **Palettes** pane shows UI elements that you can use in your app's layout.
4. The **Component tree** pane shows the view hierarchy of UI elements. View elements are organized into a tree hierarchy of parents and children, in which a child inherits the attributes of its parent. In the figure above, the TextView is a child of the ConstraintLayout. You will learn about these elements later in this lesson.
5. The design and blueprint panes of the layout editor showing the UI elements in the layout. In the figure above, the layout shows only one element: a TextView that displays "Hello World".
6. The **Attributes** tab displays the **Attributes** pane for setting properties for a UI element.

NOTE: The GUI for Android Studio has changed slightly here. XML/Visual editors used to be tabbed at the bottom of the layout editor. Now there is a selection control top right of the main window when you are using layout editor that swaps between visual, XML or both.

4 Add View elements in the layout editor

Here you create the UI layout for the HelloToast app in the layout editor using the ConstraintLayout features. You can create the constraints manually, as shown later, or automatically using the **Autoconnect** tool.

4.1 Examine the element constraints

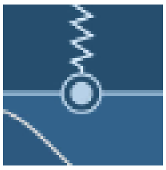
Follow these steps:

1. Open activity_main.xml from the **Project > Android** pane if it is not already open. If the **Design** tab is not already selected, click it.

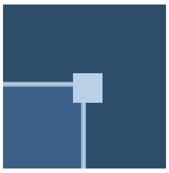
If there is no blueprint, click the **Select Design Surface** button in the toolbar and choose **Design + Blueprint**.

2. The **Autoconnect** tool is also located in the toolbar. It is enabled by default. For this step, ensure that the tool is not disabled.
3. Click the zoom in ⊖ 90% ⊕ button to zoom into the design and blueprint panes for a close-up look.
4. Select **TextView** in the Component Tree pane. The "Hello World" TextView is highlighted in the design and blueprint panes and the constraints for the element are visible.
5. Refer to the animated figure below for this step. Click the circular handle on the right side of the TextView to delete the horizontal constraint that binds the view to the right side of the layout. The TextView jumps to the left side because it is no longer constrained to the right side. To add back the horizontal constraint, click the same handle and drag a line to the right side of the layout.

In the blueprint or design panes, the following handles appear on the TextView element:
Constraint handle: To create a constraint as shown in the animated figure above, click a constraint handle, shown as a circle on the side of an element. Then drag the handle to another constraint handle, or to a parent boundary. A zigzag line represents the constraint.



Resizing handle: To resize the element, drag the square resizing handles. The handle changes to an angled corner while you are dragging it.

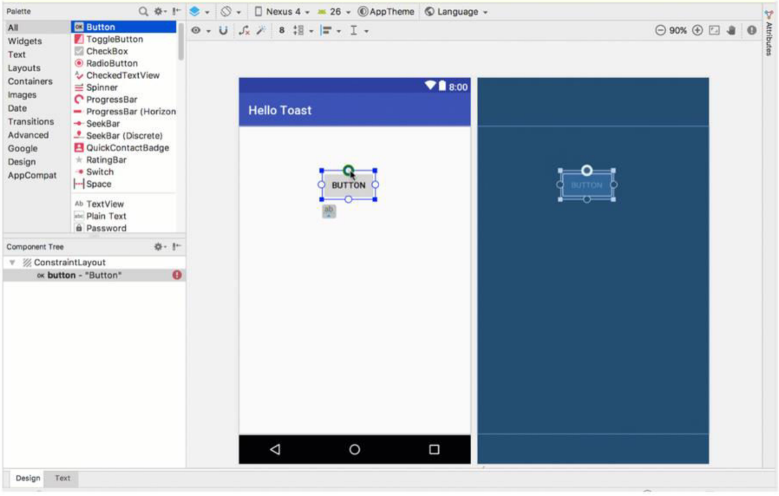


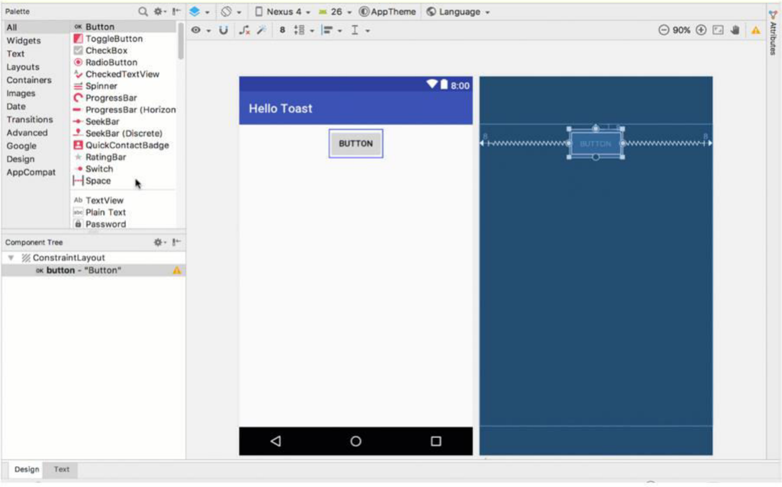
4.2 Add a button to the layout

When enabled, the **Autoconnect** tool automatically creates two or more constraints for a UI element to the parent layout. After you drag the element to the layout, it creates constraints based on the element's position.

Follow these steps to add a Button:

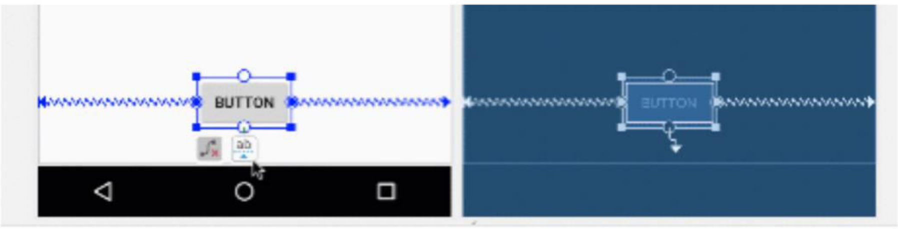
1. Start with a clean slate. The TextView element is not needed, so while it is still selected, press the **Delete** key or choose **Edit > Delete**. You now have a completely blank layout.
2. Drag a **Button** from the **Palette** pane to any position in the layout. If you drop the Button in the top middle area of the layout, constraints may automatically appear. If not, you can drag constraints to the top, left side, and right side of the layout.





4.3 Add a second button to the bottom of the layout

1. Drag another **Button** from the **Palette** pane to the middle of the layout as shown in the animated figure below. Autoconnect may provide the horizontal constraints for you (if not, you can drag them yourself).
2. Drag a vertical constraint to the bottom of the layout (refer to the figure below).



You can remove constraints from an element by selecting the element and hovering your pointer over it to show the Clear Constraints button. Click this button to remove *all* constraints on the selected element. To clear a single constraint, click the specific handle that sets the constraint.
To clear all constraints in the entire layout, click the **Clear All Constraints** tool in the toolbar. This tool is useful if you want to redo all the constraints in your layout.

5 Change the UI element attributes

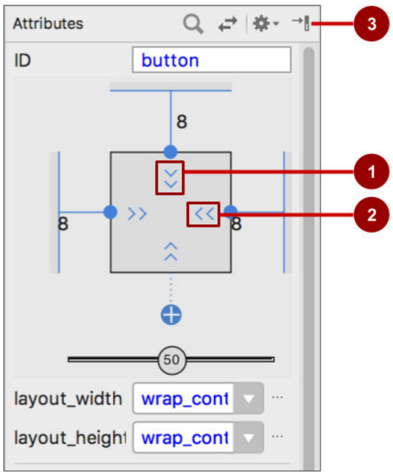
The **Attributes** pane offers access to all of the XML attributes you can assign to a UI element. You can find the attributes (known as *properties*) common to all views in the View class documentation.

Here you enter new values and change values for important Button attributes, which are applicable to most View types.

5.1 Change the Button size

The layout editor offers resizing handles on all four corners of a View so you can resize the View quickly. You can drag the handles on each corner of the View to resize it but doing so hardcodes the width and height dimensions. Avoid hardcoding sizes for most View elements, because hardcoded dimensions can't adapt to different content and screen sizes.

Instead, use the **Attributes** pane on the right side of the layout editor to select a sizing mode that doesn't use hardcoded dimensions. The **Attributes** pane includes a square sizing panel called the *view inspector* at the top. The symbols inside the square represent the height and width settings as follows:



ID: button

layout_width: wrap_cont

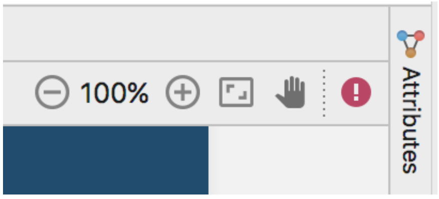
layout_height: wrap_cont

In the figure above:

1. **Height control**. This control specifies the layout_height attribute and appears in two segments on the top and bottom sides of the square. The angles indicate that this control is set to wrap_content, which means the View will expand vertically as needed to fit its contents. The "8" indicates a standard margin set to 8dp.
2. **Width control**. This control specifies the layout_width and appears in two segments on the left and right sides of the square. The angles indicate that this control is set to wrap_content, which means the View will expand horizontally as needed to fit its contents, up to a margin of 8dp.
3. **Attributes** pane close button. Click to close the pane.

Follow these steps:

1. Select the top Button in the **Component Tree** pane.
2. Click the **Attributes** tab on the right side of the layout editor window.



3. Click the width control twice—the first click changes it to **Fixed** with straight lines, and the second click changes it to **Match Constraints** with spring coils, as shown in the animated figure below.

As a result of changing the width control, the layout_width attribute in the **Attributes** pane shows the value match_constraint and the Button element stretches horizontally to fill the space between the left and right sides of the layout.

4. Select the second Button, and make the same changes to the layout_width

As shown in the previous steps, the layout_width and layout_height attributes in the **Attributes** pane change as you change the height and width controls in the inspector. These attributes can take one of three values for the layout, which is a ConstraintLayout:

- The match_constraint setting expands the View element to fill its parent by width or height—up to a margin, if one is set. The parent in this case is the ConstraintLayout. You learn more about ConstraintLayout in the next task.
- The wrap_content setting shrinks the View element's dimensions so it is just big enough to enclose its content. If there is no content, the View element becomes invisible.
- To specify a fixed size that adjusts for the screen size of the device, use a fixed number of density-independent pixels (dp units). For example, 16dp means 16 density-independent pixels.

5.2 Change the button attributes

To identify each View uniquely within an Activity layout, each View or View subclass (such as Button) needs a unique ID. And to be of any use, the Button elements need text. View elements can also have backgrounds that can be colours or images.

The **Attributes** pane offers access to all of the attributes you can assign to a View element. You can enter values for each attribute, such as the android:id, background, textColor, and text attributes.

1. After selecting the first Button, edit the ID field at the top of the **Attributes** pane to **button_toast** for the android:id attribute, which is used to identify the element in the layout.
2. Set the background attribute to **@color/colorPrimary**. (As you enter **@c**, choices appear for easy selection.)
3. Set the textColor attribute to **@android:color/white**.
4. Edit the text attribute to **Toast**.
5. Perform the same attribute changes for the second Button, using **button_count** as the ID, **Count** for the text attribute, and the same colours for the background and text as the previous steps.

The colorPrimary is the primary color of the theme, one of the predefined theme base colours defined in the colors.xml resource file. It is used for the app bar. Using the base colours for other UI elements creates a uniform UI.

6 Add a TextEdit and set its attributes

One of the benefits of ConstraintLayout is the ability to align or otherwise constrain elements relative to other elements. In this task you will add a TextView in the middle of the layout and constrain it horizontally to the margins and vertically to the two Button elements. You will then change the attributes for the TextView in the **Attributes** pane.

6.1 Add a TextView and constraints

1. Drag a TextView from the **Palette** pane to the upper part of the layout and drag a constraint from the top of the TextView to the handle on the bottom of the **Toast** Button. This constrains the TextView to be underneath the Button.
2. Drag a constraint from the bottom of the TextView to the handle on the top of the **Count** Button, and from the sides of the TextView to the sides of the layout. This constrains the TextView to be in the middle of the layout between the two Button elements.

6.2 Set the TextView attributes

With the TextView selected, open the **Attributes** pane, if it is not already open. Set attributes for the TextView:

1. Set the ID to **show_count**.
2. Set the text to **0**.
3. Set the textSize to **160sp**.
4. Set the textStyle to **B** (bold) and the textAlignment to ALIGNCENTER (center the paragraph).
5. Change the horizontal and vertical view size controls (layout_width and layout_height) to **match_constraint**.
6. Set the textColor to **@color/colorPrimary**.
7. Scroll down the pane and click **View all attributes**, scroll down the second page of attributes to background, and then enter **#FFFF00** for a shade of yellow.
8. Scroll down to gravity, expand gravity, and select **center_ver** (for center-vertical).

Note*

- textSize: The text size of the TextView. For this lesson, the size is set to 160sp. The sp stands for *scale-independent pixel*, and like dp, is a unit that scales with the screen density and user's font size preference. Use dp units when you specify font sizes so that the sizes are adjusted for both the screen density and the user's preference.
- textStyle and textAlignment: The text style, set to **B** (bold) in this lesson, and the text alignment, set to ALIGNCENTER (center the paragraph).
- gravity: The gravity attribute specifies how a View is aligned within its *parent* View or ViewGroup. In this step, you center the TextView to be centred vertically within the parent ConstraintLayout.

7 Edit the layout in XML

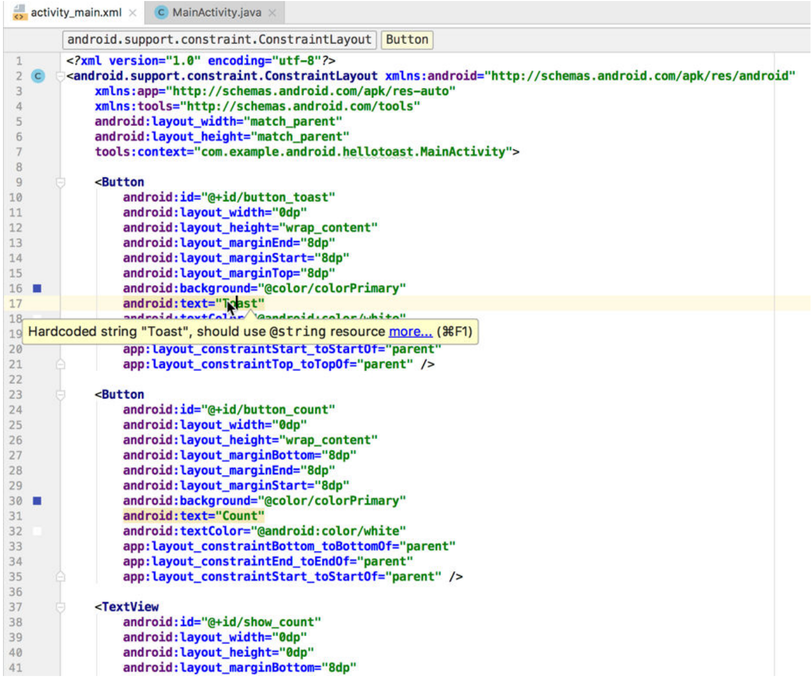
The Hello Toast app layout is nearly finished! However, an exclamation point appears next to each UI element in the Component Tree. Hover your pointer over these exclamation points to see warning messages, as shown below. The same warning appears for all three elements: hardcoded strings should use resources.

The easiest way to fix layout problems is to edit the layout in XML. While the layout editor is a powerful tool, some changes are easier to make directly in the XML source code.

7.1 Open the XML code for the layout

Open the activity_main.xml file if it is not already open, and click the **Text** tab Design Text at the bottom of the layout editor.

The XML editor appears, replacing the design and blueprint panes. As you can see in the figure below, which shows part of the XML code for the layout, the warnings are highlighted—the hardcoded strings "Toast" and "Count". (The hardcoded "0" is also highlighted but not shown in the figure.) Hover your pointer over the hardcoded string "Toast" to see the warning message.



```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.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.android.hellotoast.MainActivity">

    <Button
        android:id="@+id/button_toast"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:background="@color/colorPrimary"
        android:text="Toast"
        android:textColor="@android:color/white"

        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/button_count"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:background="@color/colorPrimary"
        android:text="Count"
        android:textColor="@android:color/white"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />

    <TextView
        android:id="@+id/show_count"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:layout_marginBottom="8dp"
```

Hardcoded string "Toast", should use @string resource more... (⌘F1)

7.2 Extract string resources

Instead of hard-coding strings, it is a best practice to use string resources, which represent the strings. Having the strings in a separate file makes it easier to manage them, especially if you use these strings more than once. Also, string resources are mandatory for translating and localizing your app, because you need to create a string resource file for each language.

1. Click once on the word "Toast" (the first highlighted warning).
2. Press **Alt-Enter** in Windows or **Option-Enter** in macOS and choose **Extract string resource** from the popup menu.
3. Enter **button_label_toast** for the **Resource name**.
4. Click **OK**. A string resource is created in the values/res/string.xml file, and the string in your code is replaced with a reference to the resource: @string/button_label_toast
5. Extract the remaining strings: button_label_count for "Count", and count_initial_value for "0".
6. In the Project > Android pane, expand values within res, and then double-click strings.xml to see your string resources in the strings.xml file:

```
<resources>
    <string name="app_name">Hello Toast</string>
    <string name="button_label_toast">Toast</string>
    <string name="button_label_count">Count</string>
    <string name="count_initial_value">0</string>
</resources>
```

7. You need another string to use in a subsequent task that displays a message. Add to the strings.xml file another string resource named toast_message for the phrase "Hello Toast!":

```
<resources>
    <string name="app_name">Hello Toast</string>
    <string name="button_label_toast">Toast</string>
    <string name="button_label_count">Count</string>
    <string name="count_initial_value">0</string>
    <string name="toast_message">Hello Toast!</string>
</resources>
```

8 Add onClick handlers for the buttons

Now add a Java method for each Button in MainActivity that executes when the user taps the Button.

8.1 Add the onClick attribute and handler to each Button

A *click handler* is a method that is invoked when the user clicks or taps on a clickable UI element. In Android Studio you can specify the name of the method in the onClick field in the **Design** tab's **Attributes** pane. You can also specify the name of the handler method in the XML editor by adding the android:onClick property to the Button. You will use the latter method because you haven't yet created the handler methods, and the XML editor provides an automatic way to create those methods.

1. With the XML editor open (the Text tab), find the Button with the android:id set to button_toast:

```
<Button
        android:id="@+id/button_toast"
        android:layout_width="0dp"
        ...
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
```

2. Add the android:onClick attribute to the end of the button_toast element after the last attribute and before the /> end indicator:

```
android:onClick="showToast" />
```

3. Click the red bulb icon that appears next to attribute. Select **Create click handler**, choose **MainActivity**, and click **OK**.

If the red bulb icon doesn't appear, click the method name ("showToast"). Press **Alt-Enter** (**Option-Enter** on the Mac), select **Create 'showToast(view)' in MainActivity**, and click **OK**.
This action creates a placeholder method stub for the showToast() method in MainActivity, as shown at the end of these steps.

4. Repeat the last two steps with the button_count Button: Add the android:onClick attribute to the end, and add the click handler:

```
android:onClick="countUp" />
```

The XML code for the UI elements within the ConstraintLayout now looks like this:

```
<Button
        android:id="@+id/button_toast"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
```

```
    android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:background="@color/colorPrimary"
        android:text="@string/button_label_toast"
        android:textColor="@android:color/white"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        android:onClick="showToast"/>

    <Button
        android:id="@+id/button_count"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:background="@color/colorPrimary"
        android:text="@string/button_label_count"
        android:textColor="@android:color/white"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        android:onClick="countUp" />

    <TextView
        android:id="@+id/show_count"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:background="#FFFF00"
        android:gravity="center_vertical"
        android:text="@string/count_initial_value"
        android:textAlignment="center"
        android:textColor="@color/colorPrimary"
        android:textSize="160sp"
        android:textStyle="bold"
        app:layout_constraintBottom_toTopOf="@+id/button_count"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/button_toast" />
```

5. If MainActivity.java is not already open, expand **java** in the Project > Android view, expand **com.example.android.hellotoast**, and then double-click **MainActivity**. The code editor appears with the code in MainActivity:

```
package com.example.android.hellotoast;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void showToast(View view) {
    }

    public void countUp(View view) {
    }
}
```

8.2 Edit the Toast Button handler

Now edit the showToast() method—the Toast Button click handler in MainActivity—so that it shows a message. A Toast provides a way to show a simple message in a small popup window. It fills only the amount of space required for the message. The current activity remains visible and interactive.
A Toast can be useful for testing interactivity in your app—add a Toast message to show the result of tapping a Button or performing an action.

Follow these steps to edit the Toast Button click handler:

1. Locate the newly created showToast() method.

```
public void showToast(View view) {
}
```

2. To create an instance of a Toast, call the makeText() factory method on the Toast class.

This statement is incomplete until you finish all of the steps.

3. Supply the context of the app Activity. Because a Toast displays on top of the Activity UI, the system needs information about the current Activity. When you are already within the context of the Activity whose context you need, use this as a shortcut.

```
Toast toast = Toast.makeText(this,
```

4. Supply the message to display, such as a string resource (the toast_message you created in a previous step). The string resource toast_message is identified by R.string.

```
Toast toast = Toast.makeText(this, R.string.toast_message,
```

5. Supply a duration for the display. For example, Toast.LENGTH_SHORT displays the toast for a relatively short time.

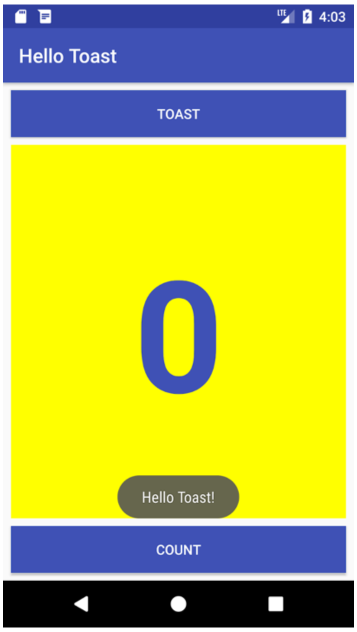
```
Toast toast = Toast.makeText(this, R.string.toast_message,
                                          Toast.LENGTH_SHORT);
```

The duration of a Toast display can be either Toast.LENGTH_LONG or Toast.LENGTH_SHORT. The actual lengths are about 3.5 seconds for the long Toast and 2 seconds for the short Toast.

6. Show the Toast by calling show(). The following is the entire showToast() method:

```
public void showToast(View view) {
   Toast toast = Toast.makeText(this, R.string.toast_message,
                                          Toast.LENGTH_SHORT);
   toast.show();
}
```

Run the app and verify that the Toast message appears when the **Toast** button is tapped.



Hello Toast!

8.3 Edit the Count Button handler

Now edit the countUp() method—the Count Button click handler in MainActivity—so that it displays the current count after **Count** is tapped. Each tap increases the count by one.
The code for the handler must:

- Keep track of the count as it changes.
- Send the updated count to the TextView to display it.

Follow these steps to edit the Count Button click handler:

Follow these steps to edit the **Count** Button click handler:

1. Locate the newly created countUp() method.

```
public void countUp(View view) {
}
```

2. To keep track of the count, you need a private member variable. Each tap of the Count button increases the value of this variable. Enter the following, which will be highlighted in red and show a red bulb icon:

```
public void countUp(View view) {
    mCount++;
}
```

If the red bulb icon doesn't appear, select the mCount++ expression. The red bulb eventually appears.

3. Click the red bulb icon and choose **Create field 'mCount'** from the popup menu. This creates a private member variable at the top of MainActivity, and Android Studio assumes that you want it to be an integer (int):

```
public class MainActivity extends AppCompatActivity {
    private int mCount;
```

4. Change the private member variable statement to initialize the variable to zero:

```
public class MainActivity extends AppCompatActivity {
    private int mCount = 0;
```

5. Along with the variable above, you also need a private member variable for the reference of the show_count TextView, which you will add to the click handler. Call this variable mShowCount:

```
public class MainActivity extends AppCompatActivity {
    private int mCount = 0;
    private TextView mShowCount;
```

6. Now that you have mShowCount, you can get a reference to the TextView using the ID you set in the layout file. In order to get this reference only once, specify it in the onCreate() method. As you learn in another lesson, the onCreate() method is used to *inflate the layout*, which means to set the content view of the screen to the XML layout. You can also use it to get references to other UI elements in the layout, such as the TextView. Locate the onCreate() method in MainActivity:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}
```

7. Add the findViewById statement to the end of the method:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    mShowCount = (TextView) findViewById(R.id.show_count);
}
```

A View, like a string, is a resource that can have an id. The findViewById call takes the ID of a view as its parameter and returns the View. Because the method returns a View, you have to cast the result to the view type you expect, in this case (TextView).

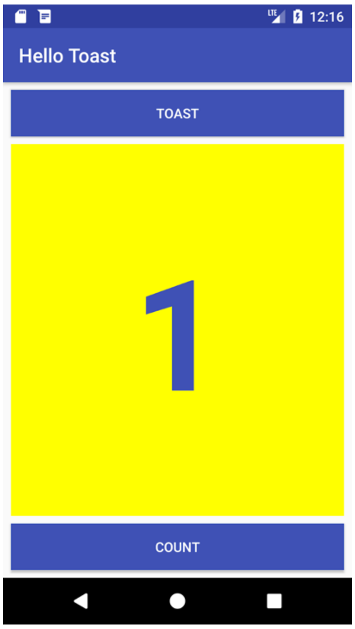
8. Now that you have assigned to mShowCount the TextView, you can use the variable to set the text in the TextView to the value of the mCount variable. Add the following to the countUp() method:

```
if (mShowCount != null)
            mShowCount.setText(Integer.toString(mCount));
```

The entire countUp() method now looks like this:

```
public void countUp(View view) {
    ++mCount;
    if (mShowCount != null)
        mShowCount.setText(Integer.toString(mCount));
}
```

9. Run the app to verify that the count increases when you tap the **Count** button.


12:16
Hello Toast
TOAST
1
COUNT