

Android Developer Fundamentals V2

Build your first app

Lesson 1



1.2 Layouts and resources for the UI

Contents

- Views, view groups, and view hierarchy
- The layout editor and ConstraintLayout
- Event handling
- Resources and measurements

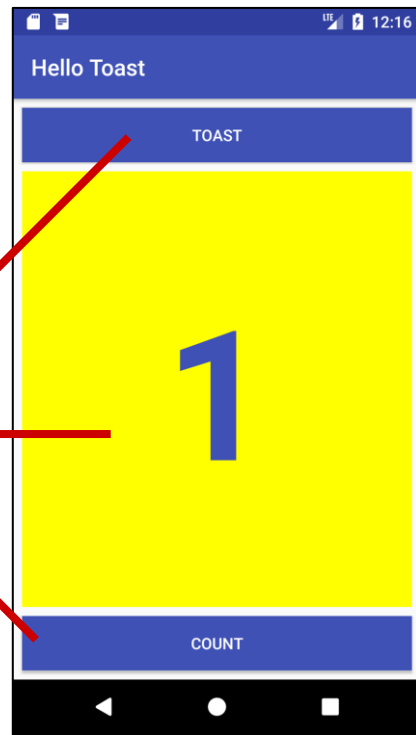


Views

Everything you see is a view

If you look at your mobile device, every user interface element that you see is a **View**.

Views



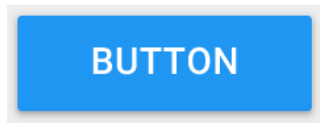
What is a view?

View subclasses are basic user interface building blocks

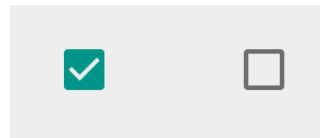
- Display text (TextView class), edit text (EditText class)
- Buttons (Button class), menus, other controls
- Scrollable (ScrollView, RecyclerView)
- Show images (ImageView)
- Group views (ConstraintLayout and LinearLayout)

Examples of view subclasses

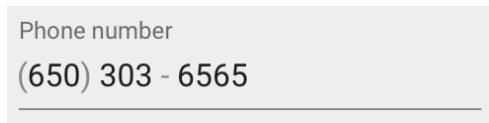
Button



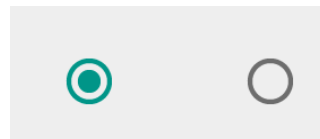
CheckBox



EditText



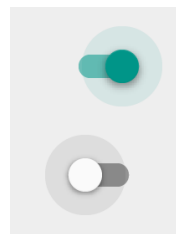
RadioButton



Slider



Switch



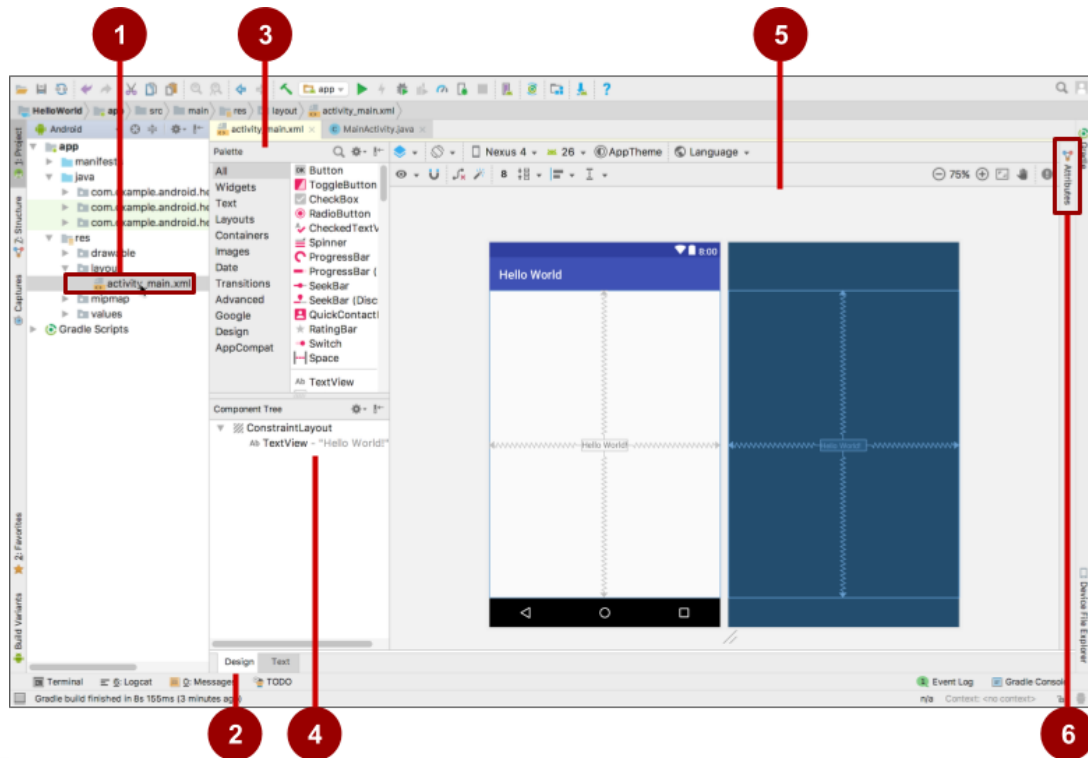
View attributes

- Color, dimensions, positioning
- May have focus (e.g., selected to receive user input)
- May be interactive (respond to user clicks)
- May be visible or not
- Relationships to other views

Create views and layouts

- Android Studio layout editor: visual representation of XML
- XML editor
- Java code

Android Studio layout editor



1. XML layout file
2. **Design** and **Text** tabs
3. **Palette** pane
4. **Component Tree**
5. Design and blueprint panes
6. **Attributes** tab

View defined in XML

<TextView

```
    android:id="@+id/show_count"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:background="@color/myBackgroundColor"  
    android:text="@string/count_initial_value"  
    android:textColor="@color/colorPrimary"  
    android:textSize="@dimen/count_text_size"  
    android:textStyle="bold"
```

```
/>
```

View defined in XML

<TextView

```
    android:id            = "@+id/show_count"
    android:layout_width  = "match_parent"
    android:layout_height = "wrap_content"
    android:background    = "@color/myBackgroundColor"
    android:text           = "@string/count_initial_value"
    android:textColor     = "@color/colorPrimary"
    android:textSize      = "@dimen/count_text_size"
    android:textStyle     = "bold"
```

```
/>
```

View attributes in XML

android:<property_name>=<property_value>

Example: android:layout_width="match_parent"

android:<property_name>="@<resource_type>/resource_id"

Example: android:text="@string/button_label_next"

android:<property_name>="@+id/view_id"

Example: android:id="@+id/show_count"

Create View in Java code

context



In an Activity:

```
TextView myText = new TextView(this);  
myText.setText("Display this text!");
```

What is the context?

- [Context](#) is an interface to global information about an application environment

- Get the context:

```
Context context = getApplicationContext();
```

- An Activity is its own context:

```
TextView myText = new TextView(this);
```

Custom views

- Over 100 (!) different types of views available from the Android system, all children of the [View](#) class
- If necessary, [create custom views](#) by subclassing existing views or the View class

ViewGroup and View hierarchy

ViewGroup contains "child" views

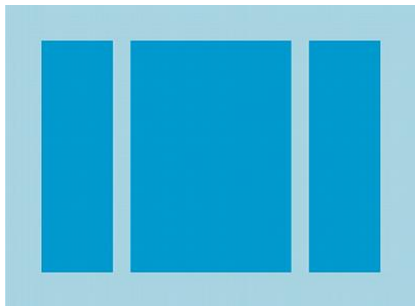
- [ConstraintLayout](#): Positions UI elements using constraint connections to other elements and to the layout edges
- [ScrollView](#): Contains one element and enables scrolling
- [RecyclerView](#): Contains a list of elements and enables scrolling by adding and removing elements dynamically

ViewGroups for layouts

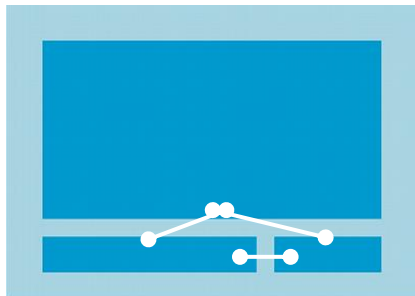
Layouts

- are specific types of ViewGroups (subclasses of [ViewGroup](#))
- contain child views
- can be in a row, column, grid, table, absolute

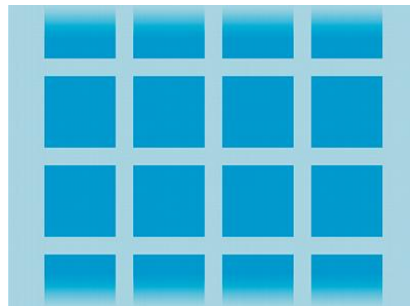
Common Layout Classes



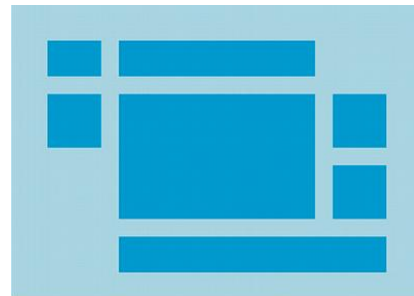
LinearLayout



ConstraintLayout



GridLayout



TableLayout

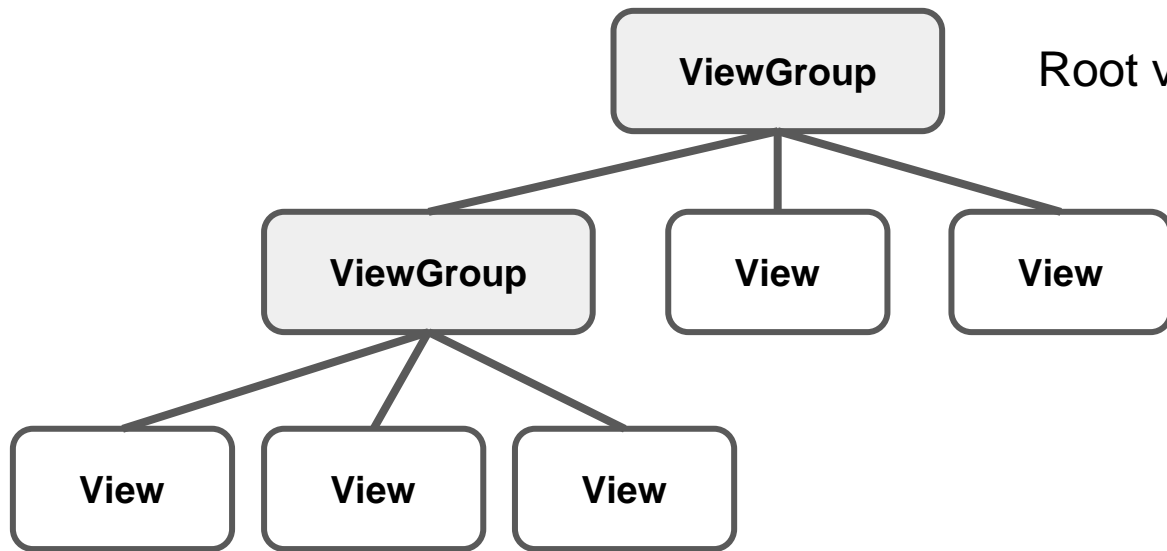
Common Layout Classes

- `ConstraintLayout`: Connect views with constraints
- `LinearLayout`: Horizontal or vertical row
- `RelativeLayout`: Child views relative to each other
- `TableLayout`: Rows and columns
- `FrameLayout`: Shows one child of a stack of children

Class hierarchy vs. layout hierarchy

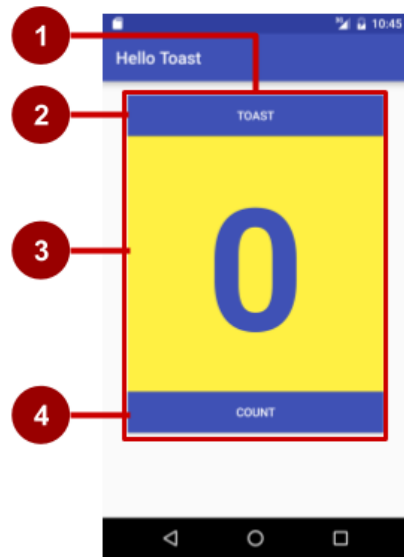
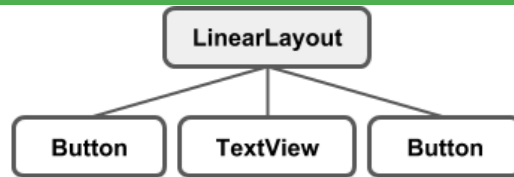
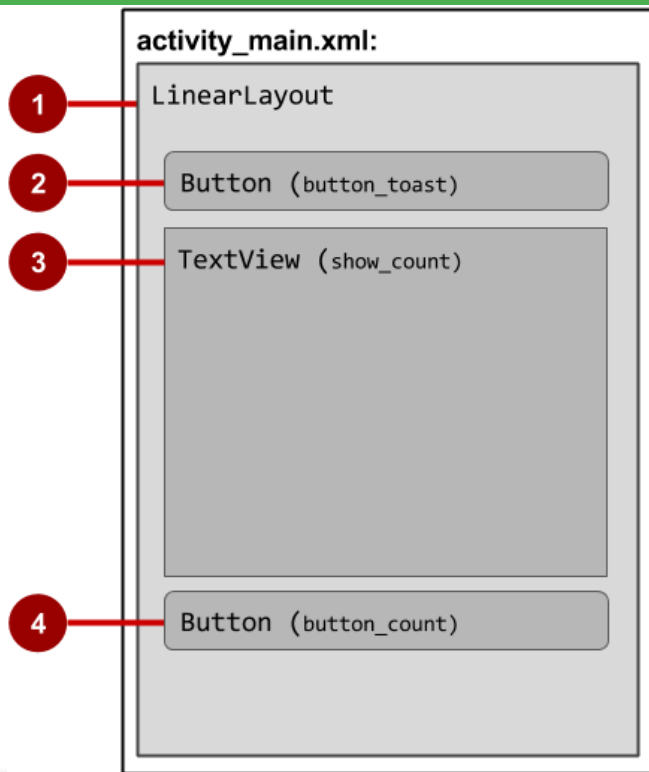
- View class-hierarchy is standard object-oriented class inheritance
 - For example, Button is-a TextView is-a View is-an Object
 - Superclass-subclass relationship
- Layout hierarchy is how views are visually arranged
 - For example, LinearLayout can contain Buttons arranged in a row
 - Parent-child relationship

Hierarchy of viewgroups and views

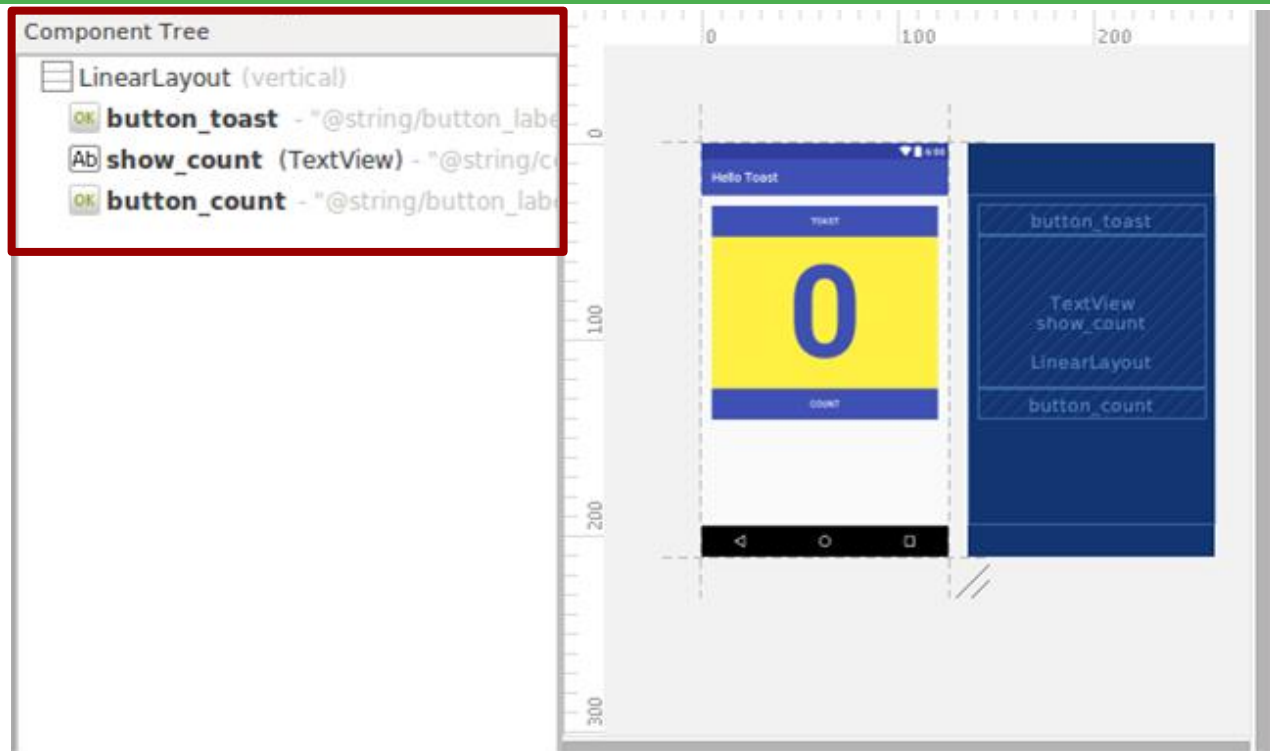


Root view is always a ViewGroup

View hierarchy and screen layout



View hierarchy in the layout editor



Layout created in XML

```
<LinearLayout
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        ... />
    <TextView
        ... />
    <Button
        ... />
</LinearLayout>
```

Layout created in Java Activity code

```
LinearLayout linearL = new LinearLayout(this);  
linearL.setOrientation(LinearLayout.VERTICAL);  
  
TextView myText = new TextView(this);  
myText.setText("Display this text!");  
  
linearL.addView(myText);  
setContentView(linearL);
```

Set width and height in Java code

Set the width and height of a view:

```
LinearLayout.LayoutParams layoutParams =  
    new LinearLayout.LayoutParams(  
        layoutParams.MATCH_PARENT,  
        layoutParams.MATCH_CONTENT);  
myView.setLayoutParams(layoutParams);
```

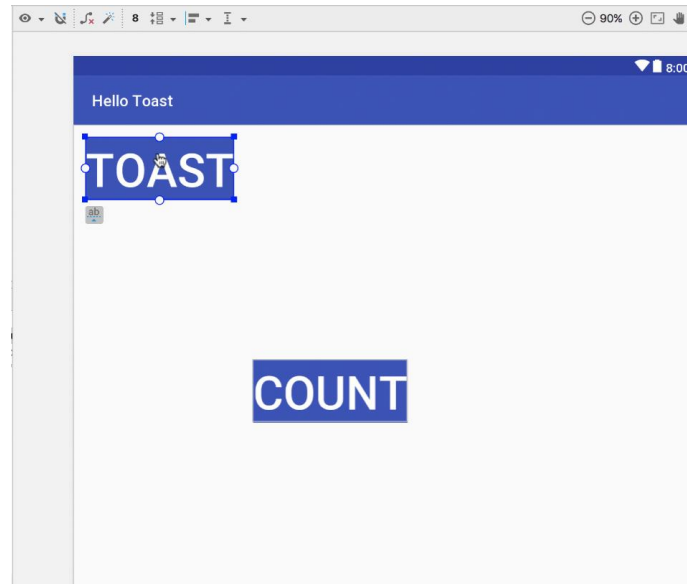
Best practices for view hierarchies

- Arrangement of view hierarchy affects app performance
- Use smallest number of simplest views possible
- Keep the hierarchy flat—limit nesting of views and view groups

The layout editor and Constraint Layout

The layout editor with ConstraintLayout

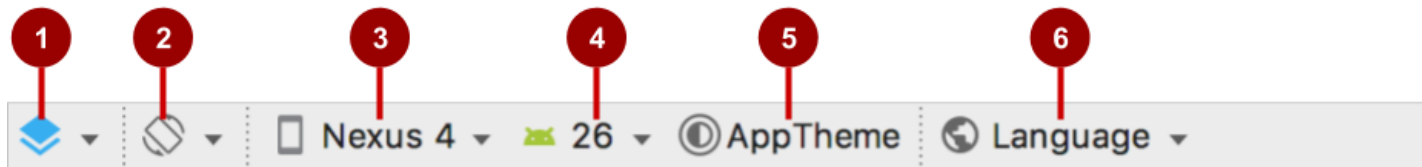
- Connect UI elements to parent layout
- Resize and position elements
- Align elements to others
- Adjust margins and dimensions
- Change attributes



What is ConstraintLayout?

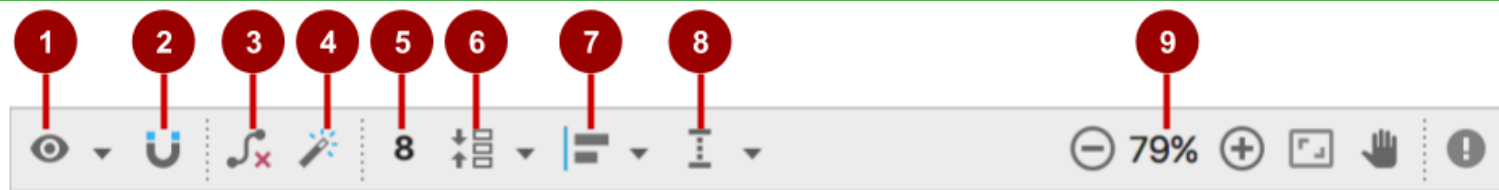
- Default layout for new Android Studio project
- ViewGroup that offers flexibility for layout design
- Provides constraints to determine positions and alignment of UI elements
- Constraint is a connection to another view, parent layout, or invisible guideline

Layout editor main toolbar




1. Select Design Surface: Design and Blueprint panes
2. Orientation in Editor: Portrait and Landscape
3. Device in Editor: Choose device for preview
4. API Version in Editor: Choose API for preview
5. Theme in Editor: Choose theme for preview
6. Locale in Editor: Choose language/locale for preview

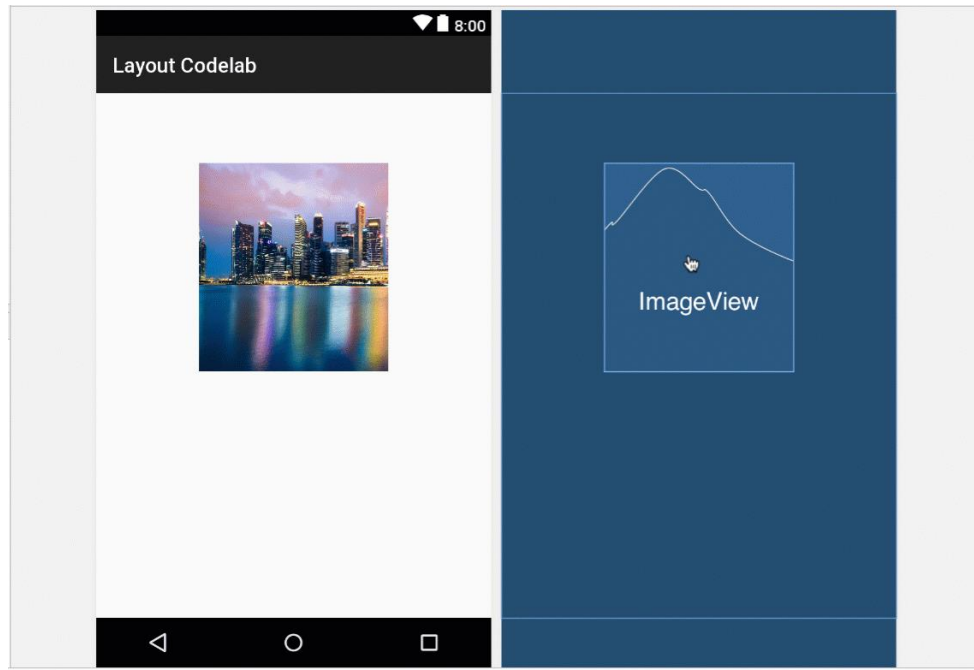
ConstraintLayout toolbar in layout editor



1. Show: Show Constraints and Show Margins
2. Autoconnect: Enable or disable
3. Clear All Constraints: Clear all constraints in layout
4. Infer Constraints: Create constraints by inference
5. Default Margins: Set default margins
6. Pack: Pack or expand selected elements
7. Align: Align selected elements
8. Guidelines: Add vertical or horizontal guidelines
9. Zoom controls: Zoom in or out

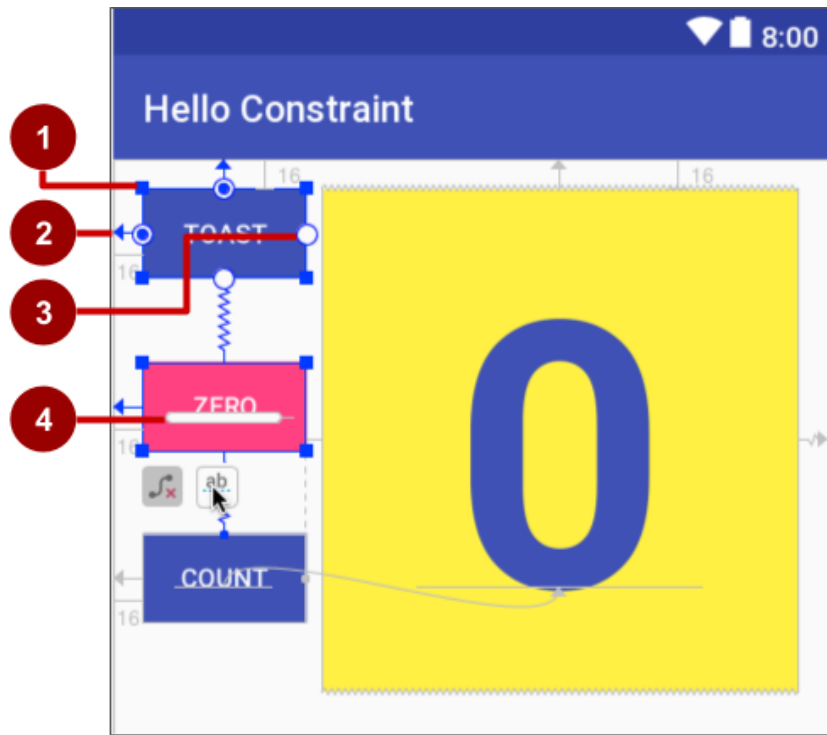
Autoconnect

- Enable Autoconnect  in toolbar if disabled
- Drag element to any part of a layout
- Autoconnect generates constraints against parent layout




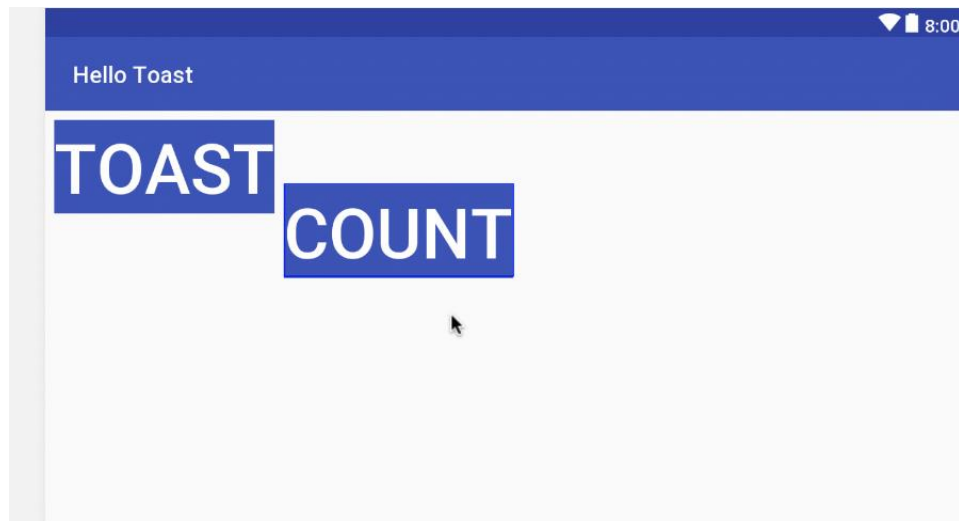
ConstraintLayout handles

1. Resizing handle
2. Constraint line and handle
3. Constraint handle
4. Baseline handle



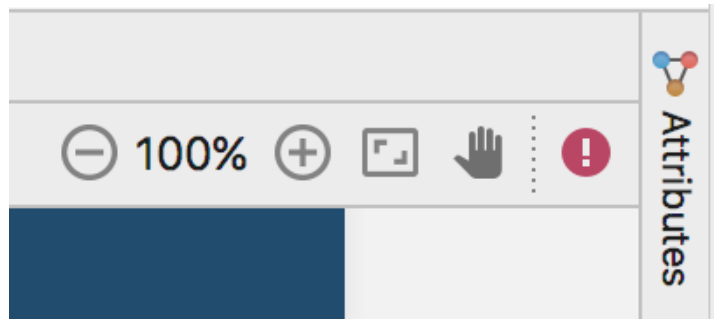
Align elements by baseline

1. Click the  baseline constraint button
2. Drag from baseline to other element's baseline



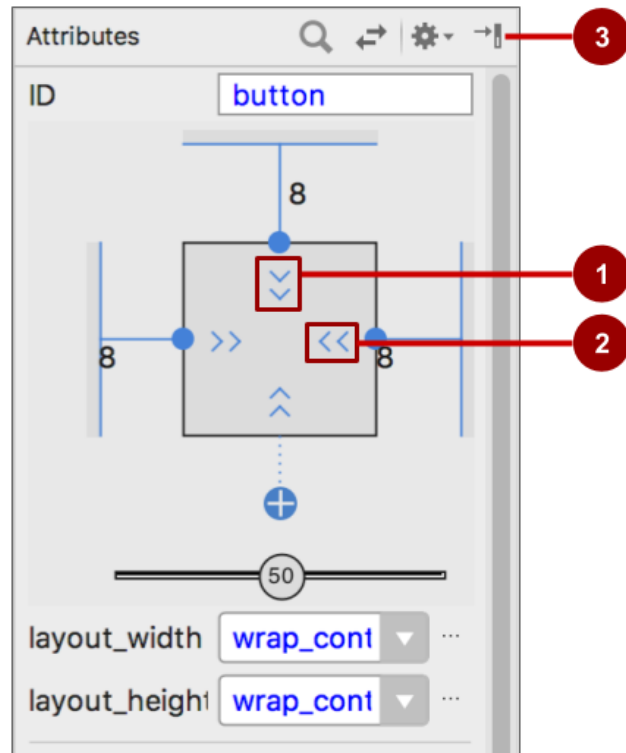
Attributes pane

- Click the Attributes tab
- Attributes pane includes:
 - Margin controls for positioning
 - Attributes such as `layout_width`






Attributes pane view inspector

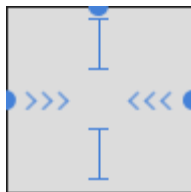
1. Vertical view size control specifies `layout_height`
2. Horizontal view size control specifies `layout_width`
3. Attributes pane close button



Layout_width and layout_height


layout_width and layout_height change with size controls

-  match_constraint: Expands element to fill its parent
-  wrap_content: Shrinks element to enclose content
-  Fixed number of dp (density-independent pixels)

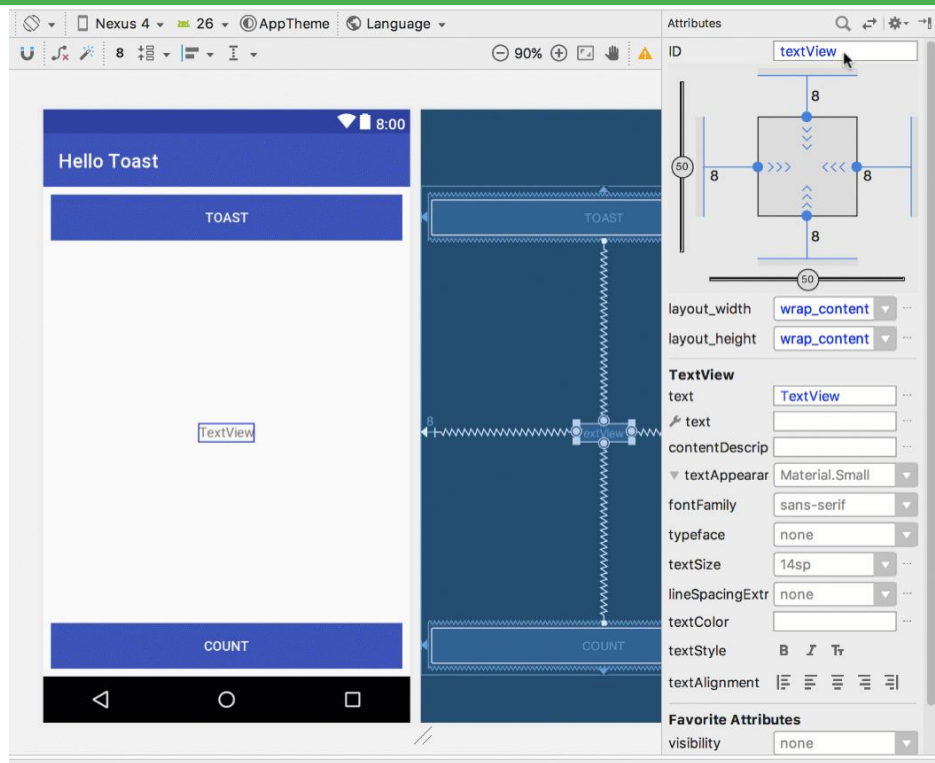


Set attributes

To view and edit all attributes for element:


1. Click **Attributes** tab
2. Select element in design, blueprint, or Component Tree
3. Change most-used attributes
4. Click  at top or **View more attributes** at bottom to see and change more attributes

Set attributes example: TextView



Preview layouts


Preview layout with horizontal/vertical orientation:

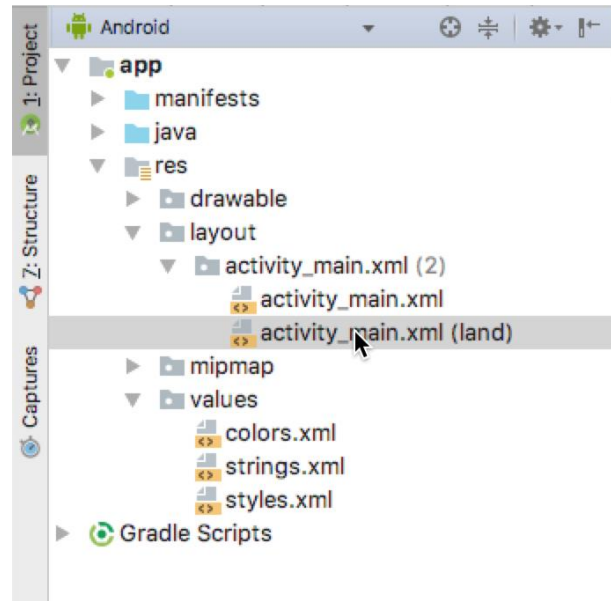
1. Click Orientation in Editor button 
2. Choose **Switch to Landscape** or **Switch to Portrait**

Preview layout with different devices:


1. Click Device in Editor button  Nexus 5 ▾
2. Choose device

Create layout variant for landscape

1. Click Orientation in Editor button 
2. Choose **Create Landscape Variation**
3. Layout variant created:
activity_main.xml (land)
4. Edit the layout variant as needed



Create layout variant for tablet

1. Click Orientation in Layout Editor 
2. Choose **Create layout x-large Variation**
3. Layout variant created: `activity_main.xml` (xlarge)
4. Edit the layout variant as needed

Event Handling

Events

Something that happens

- In UI: Click, tap, drag
- Device: [DetectedActivity](#) such as walking, driving, tilting
- Events are "noticed" by the Android system

Event Handlers

Methods that do something in response to a click

- A method, called an **event handler**, is triggered by a specific event and does something in response to the event

Attach in XML and implement in Java

Attach handler to view in XML layout:

```
android:onClick="showToast"
```

Implement handler in Java activity:

```
public void showToast(View view) {  
    String msg = "Hello Toast!";  
    Toast toast = Toast.makeText(  
        this, msg, duration);  
    toast.show();  
}  
}
```

Alternative: Set click handler in Java

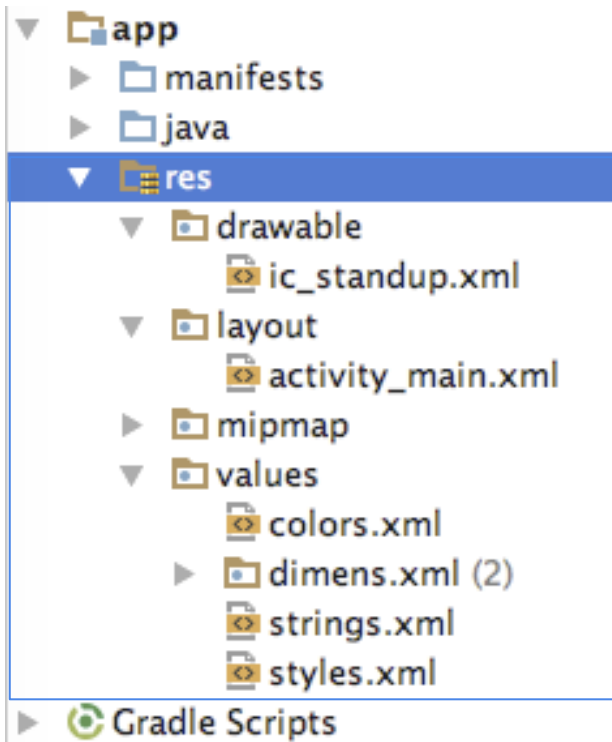
```
final Button button = (Button) findViewById(R.id.button_id);
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        String msg = "Hello Toast!";
        Toast toast = Toast.makeText(this, msg, duration);
        toast.show();
    }
});
```

Resources and measurements

Resources

- Separate static data from code in your layouts.
- Strings, dimensions, images, menu text, colors, styles
- Useful for localization

Where are the resources in your project?



← resources and resource files stored in **res** folder

Refer to resources in code

- Layout:

```
R.layout.activity_main  
setContentView(R.layout.activity_main);
```

- View:

```
R.id.recyclerview  
rv = (RecyclerView) findViewById(R.id.recyclerview);
```

- String:

In Java: `R.string.title`

In XML: `android:text="@string/title"`

Measurements

- Density-independent Pixels (dp): for Views
- Scale-independent Pixels (sp): for text

Don't use device-dependent or density-dependent units:

- ~~• Actual Pixels (px)~~
- ~~• Actual Measurement (in, mm)~~
- ~~• Points - typography 1/72 inch (pt)~~

Learn more

Learn more

Views:

- [View class documentation](#)
- [device independent pixels](#)
- [Button class documentation](#)
- [TextView class documentation](#)

Layouts:

- [developer.android.com Layouts](#)
- [Common Layout Objects](#)

Learn even more

Resources:

- [Android resources](#)
- [Color](#) class definition
- [R.color resources](#)
- [Supporting Different Densities](#)
- [Color Hex Color Codes](#)

Other:

- [Android Studio documentation](#)
- [Image Asset Studio](#)
- [UI Overview](#)
- [Vocabulary words and concepts glossary](#)
- [Model-View-Presenter](#)
(MVP) architecture pattern
- [Architectural patterns](#)

What's Next?

- Concept Chapter: [1.2 Layouts and resources for the UI](#)
- Practicals:
 - [1.2A : Your first interactive UI](#)
 - [1.2B : The layout editor](#)

END