

ATLS 4120/5120: Mobile Application Development

Week 10: Android Layouts

Layouts

A layout defines the visual structure for a user interface and acts as a container for your UI widgets.

<https://developer.android.com/guide/topics/ui/declaring-layout#CommonLayouts>

Before Android added constraint layouts developers used a combination of other layouts, often nesting them to achieve the layout they were aiming for.

- Linear layouts organize its widgets into a single horizontal or vertical row.
<https://developer.android.com/guide/topics/ui/layout/linear.html>
 - android:orientation determines which direction you want to arrange views in (required)
 - vertical: views are displayed in a single column
 - horizontal: views are displayed in a single row
- Relative layouts display views relative to each other or to their parent.
<https://developer.android.com/guide/topics/ui/layout/relative.html>

You can also build layouts dynamically using adapters. This is useful when your layout is not pre-determined and you want to populate it at run time. We'll get into adapters next semester.

- List Views display a scrolling list of data
- Grid views display items in a two-dimensional, scrollable grid.
<https://developer.android.com/guide/topics/ui/layout/gridview.html>

Constraint Layout


Constraint Layout was added in Android Studio 2.2 to allow developers to create adaptive layouts. In constraint layouts views are laid out according to relationships between sibling views and the parent layout. Constraint layouts have a flat view hierarchy, so there's no nesting of other layouts. This is especially useful for large and complex layouts and helps with performance.

Constraint Layout is compatible with Android 2.3 (API level 9) and higher.

The new layout editor to create ConstraintLayout was added to Android Studio 2.2 and became the default layout in Android Studio 3.0.

Constraint Rules

Every view **MUST** have at least two constraints: one horizontal and one vertical.

- if a view has no constraints when you run your layout, it will be drawn at position [0,0] (the top-left corner).
 - A missing vertical constraint will cause it to be drawn at the top of the screen
 - A missing horizontal constraint will cause it to be drawn at the left of the screen
- Missing a constraint won't cause a compile error
- The Layout Editor indicates missing constraints as an error in the toolbar. To view the errors and other warnings, click Show Warnings and Errors 

Creating Constraints


Each constraint represents a connection or alignment to another view, the parent layout, or an invisible guideline.

Show Constraints (eye icon) in the toolbar will let you see the constraints in the design mode.


There are 3 ways to create constraints: Auto-connect, Inference, and manually.

Autoconnect:


Autoconnect to parent automatically creates constraints for each view as you add them to the layout. Autoconnect will constrain the view to the parent layout when appropriate but does not create constraints to other views in the layout.

- You can enable Autoconnect by clicking Turn on Autoconnect in the Layout Editor toolbar . Auto connect is turned off by default.
- Autoconnect does not create constraints for existing views in the layout, only as you add views

Inference:

Infer Constraints  is a one-time action that creates a set of constraints for either the view you have selected or all views in your layout if none are selected.

Manual:

If you disable auto connect or delete a views' constraints you can create constraints manually. When you drop a view into the Layout Editor, it stays where you leave it even if it has no constraints. But the layout editor will show a missing constraint error, as the view will be placed at the top left if it has no constraints. Show Warnings and Errors  There are also some constraints that can only be created manually.

Adding a Constraint

A constraint describes how a view should be positioned relative to other views in a layout or its parent. A constraint is typically defined for one or more sides by connecting the view to:

- An anchor point or another view
- An edge of the layout
- An invisible guideline

When you add a view in a ConstraintLayout, it displays a bounding box with square resizing handles on each corner and circular constraint handles on each side.

To create a constraint, click the view to select it then click-and-hold one of the constraint handles. Drag the line to an anchor point (the edge of another view, the edge of the layout, or a guideline). When you release, the constraint is made, with a default margin separating the two views. You can adjust the margins by moving the view in the design preview or change the value in the Attributes window.

You can create constraints only between a constraint handle and an anchor point that share the same plane. So a vertical plane (the left and right sides) of a view can be constrained only to another vertical plane; and baselines can constrain only to other baselines.

Each constraint handle can be used for just one constraint, but you can create multiple constraints (from different views) to the same anchor point.

You can also add a constraint by clicking the + in the layout section of the Attributes window. In the Attributes window there's a Constraints section which lists the constraints relationships of the currently selected UI component.

You can also create some constraints through the right-click menu on a view.

Constraint Types

Parent position:


Connect the side of a view to the corresponding edge of the layout.

Order position:

Define the order of appearance for two views, either vertically or horizontally.

Alignment:

Align the edge of a view to the same edge of another view.

You can align multiple views by selecting them and clicking Align  in the toolbar to select the alignment type.


Baseline alignment:

Align the text baseline of a view to the text baseline of another view.

- Right click on the textview you want to constrain and then click Show Baseline. Then click the text baseline and drag the line to another baseline.

Constrain to a guideline:


You can add a vertical or horizontal guideline to which you can attach constraints. You can position the guideline within the layout based on either dp units or percent, relative to the layout's edge.

To create a guideline, click Guidelines  in the toolbar, and then click either Add Vertical Guideline or Add Horizontal Guideline. Drag the dotted line to reposition it and click the circle at the edge of the guideline to toggle the measurement mode.

Constrain to a barrier:

A barrier is an invisible line that you can constrain views to. A barrier does not define its own position, its position moves based on the position of views contained within it. This is useful when you want to constrain a view to a set of views rather than to one specific view.

To create a barrier:

1. Click Guidelines  in the toolbar, and then click Add Vertical Barrier or Add Horizontal Barrier.
2. In the Component Tree window, select the views you want inside the barrier and drag them into the barrier component.
3. Select the barrier from the Component Tree, open the Attributes window, and then set the barrierDirection.


Now you can create a constraint from another view to the barrier.

You can also constrain views that are inside the barrier to the barrier. This way, you can ensure that all views in the barrier always align to each other, even if you don't know which view will be the longest or tallest.

Constraint Size Modes


To select from one of the dynamic sizing modes, click a view and open the Attributes window to adjust the constraints. You can select the mode in the Declared Attributes section or just click on the size mode symbols inside the square to toggle the size mode.

- `android:layout_width` and `android:layout_height` are required attributes for almost all widgets


 Fixed: You specify the dimension in the text box below or by resizing the view in the editor.

- `android:layout_width` and `android:layout_height` will have the fixed value

You can use the handles on each corner of the view to resize it but doing so will set the width and height values to fixed values, which you should avoid for most views because hard-coded view sizes cannot adapt to different content and screen sizes.

 Wrap Content: The view expands only as much as needed to fit its contents.

- `android:layout_width` and `android:layout_height` will be “wrap_content”

 Match constraints: The view expands as much as possible to meet the constraints on each side after accounting for the views' margins. You can modify the following attributes:

- `layout_constraintWidth_default`
 - spread: Expands the view as much as possible to meet the constraints on each side. This is the default behavior.
 - wrap: Expands the view only as much as needed to fit its contents, but still allows the view to be smaller than that if the constraints require it.
- `layout_width` and `layout_height` will be 0dp because the view has no desired dimensions, but it resizes as needed to meet the constraints.
- You can also set the constraint width to have a `layout_constraintWidth_min` or `layout_constraintWidth_max`.

You can't use `match_parent` for any view in a `ConstraintLayout`, use match constraints instead.

Adjusting constraints

Bias:

If you add constraints to both sides of a view (and the view size for the same dimension is either "fixed" or "wrap content"), the constraint lines become squiggly like a spring to indicate the opposing forces and the view becomes centered between the two constraints with a bias that's a percentage between 1 (left or top) and 100 (right or bottom). Centered, 50% is default. You can

adjust the horizontal or vertical bias using the slider in the Attributes window or by dragging the view. 1 and 100 are swapped in languages that are read right to left.


If you want the view to stretch its size to meet the constraints, switch the size to “match constraints”. In XML the percentages are represented as decimals from 0 to 1.

- Vertical bias allows you to position a view along the vertical axis using a bias value, this will be relative to its constrained position.
- Horizontal bias allows you to position a view along the horizontal axis using a bias value, this will be relative to its constrained position.

Size ratio:

You can set the view size to a width:height ratio if at least one of the view dimensions is set to "match constraints" (0dp). To set up a ratio click Toggle Aspect Ratio Constraint in the Attributes window by clicking on the triangle in the top left corner of the square with the constraints (only visible if at least one of the view dimensions is set to match constraints)

View Margins:

To ensure that all your views are evenly spaced, click Margin  in the toolbar to select the default margin for each view that you add to the layout. The button changes to show your current margin selection. Any change you make to the default margin applies only to the views you add from then on.

You can control the margin for each view in the Attributes window by clicking the number on the line that represents each constraint.

All margins offered by the tool are factors of 8dp to help your views align to Material Design's 8dp square grid recommendations.

Building Interfaces with Constraint Layout

<https://developer.android.com/training/constraint-layout/index.html> 1:05 - end (4 mins)

Chains

A chain is a group of views that are linked to each other with bi-directional position constraints. The views within a chain can be distributed horizontally or vertically.

To create a chain of views quickly, select them all, right-click one of the views, and then select either Center Horizontally or Center Vertically, to create either a horizontal or vertical chain.

Chain styles:

Once you've created a chain you can cycle through the chain styles by clicking on the chain icon on the views in the chain.

Spread

- The views are evenly distributed after margins are accounted for
- This is the default

Spread inside


- The first and last view are affixed to the constraints on each end of the chain and the rest are evenly distributed.

Weighted

- When the chain is set to either spread or spread inside, you can fill the remaining space by setting one or more views to "match constraints" (0dp). By default, the space is evenly distributed between each view that's set to "match constraints," but you can assign a weight of importance to each view using the `layout_constraintHorizontal_weight` and `layout_constraintVertical_weight` attributes.
 - Weight is a proportion used to determine how views are distributed
 - For example if one view has a weight of 1 and a second view has a weight of 7, the first would get 1/8 the space, the second 7/8 of the space
 - the view with the highest weight value gets the most amount of space
 - views that have the same weight get the same amount of space.
 - If only 1 view has a weight of 1 it will get all the extra space

Packed

- The views are packed together after margins are accounted for
- You can adjust the whole chain's bias by changing the chain's head view bias.

The chain's "head" view is the left-most view in a horizontal chain and the top-most view in a vertical chain and defines the chain's style in XML. You can toggle between spread, spread inside, and packed by selecting any view in the chain and then clicking the chain button  that appears below the view.

Chain rules:

- A view can be a part of both a horizontal and a vertical chain, which makes it easy to build flexible grid layouts.
- Each end of the chain must be constrained to another object on the same axis
- Using a vertical or horizontal chain does not align the views in that direction. Use other constraints to position each view in the chain, such as alignment constraints.

Deleting Constraints

You can delete a single constraint by clicking on it in the design view and pressing Delete. You can also press `Control` (Command on macOS), and then hover on a constraint anchor. The constraint turns red to indicate that you can click to delete it.

You can also select a constraint in the layout section in the Attributes window and delete it from there.

Right click on a view gives you the option to clear all constraints on that selection.

In the toolbar Clear All Constraints will clear the constraints on all views in your layout.

Attributes

Constraint Layout reference

<https://developer.android.com/reference/android/support/constraint/ConstraintLayout.html>

Halloween

Let's go through all the constraints our views currently have and understand them.

All have `android:layout_width` and `android:layout_height` set to `wrap_content` which means they're sized based on their content. Change the button text to Happy Halloween and you'll see it'll get larger.

Notice that if you drag a view it will update the margins and bias according to its new position.

In a constraint layout you can create constraints using Autoconnect, Infer Constraints, manually.

Manually Creating Constraints

Make a little room above the edittext to add a textview.

Turn Autoconnect off to see what happens when you add a textview.

Since autoconnect was off no constraints were added so even though it looked right in design mode, when we run the app it will automatically be placed at (0,0)

We could use Infer Constraints but instead let's look at how we can create the constraints manually.

Click on the text view and you'll see that when a view is added to a ConstraintLayout it displays a bounding box with circular constraint handles on each side and square resizing handles on each corner.

Click-and-drag the top constraint handle to an available anchor point (the edge of another view, the edge of the layout, or a guideline) such as the top of the layout. When you release, the constraint is made, with a default margin(set in toolbar). You can see the constraint in the Attributes window layout section and you can change the spacing there or just drag the text view. You can also create constraints in the Attributes window using the +. Wherever there's a + it means you could add a constraint. It will add it with the margin the view currently has in the layout.

If you click the + for a bottom constraint it will create one to the view below it, the edit text. If wanted it to the bottom layout you can change it to be to "parent" and then either change the margin or the vertical bias. Or create it by dragging the constraint handle to the bottom layout and it will create a bottom constraint to parent. But you don't need a bottom constraint because the top one handles the vertical placement.

Click and drag the left circle to the left margin and then release. Do the same thing with the circle on the right. You should now see it added two constraints. Since it is now horizontally and vertically positioned the constraint error went away and if you run it the text view is now centered. You can move the slider to change the bias.

You can also use Align in the toolbar (or right click) and chose horizontal.

Since we're not going to keep this textview, let's look at how you can delete its constraints.

You can delete a single constraint by clicking on it in the design view and pressing Delete. You can also press `Control` (Command on macOS), and then hover on a constraint anchor. The constraint turns red to indicate that you can click to delete it.

You can also select a constraint in the Constraints Section in the Attributes window and delete it from there.

Right click on a view also provides all the autoconnect options including Clear Constraints of Selection to clear the constraints for just that view.

In the toolbar you can chose Clear All Constraints and that will clear the constraints on ALL the views in your layout. (Luckily you can undo this if you do it by mistake)

Practice creating constraints manually by clearing all the constraints and then creating them manually.

Landscape

If you run this app and rotate to landscape you'll see it doesn't look too good. (On the Pixel you need to click the icon on the bottom right to allow rotation)

(Note: When you rotate it seems to "reset". It is working properly. When rotation occurs on Android the activity is killed and restarted. We'll talk about why next week when we discuss the Android lifecycle).

Android categorizes device screens using two general properties: size and density. The screen orientation (landscape or portrait) is considered a variation of screen size. Android automatically scales your layout in order to properly fit the screen. So your layouts for different screen sizes don't need to worry about the absolute size of UI elements but instead focus on the layout structure that affects the user experience (such as the size or position of important views relative to sibling views). If you can't define one layout that works on all of these you need to define different layouts for each screen size.

Let's see if we can change the constraints so that this layout can work in both portrait and landscape.

Starting from the top I'm going to update the edittext to have a top constraint instead of a bottom constraint so I'm not wasting space at the top.

I deleted the bottom constraint and created a top constraint to the parent.

I did the same for the button, delete the bottom constraint and added a top constraint to the edittext.

Same for the textview, delete the bottom constraint. I also deleted/changed the top constraint to go to the bottom of the button (instead of the parent). It might be helpful to put in some temporary text while you're working on the layout.

The imageview required the most work as I needed the image to scale. First I deleted/changed the top constraint to go to the bottom of the textview(id textView). I also updated the bottom constraint to be 8dp.

Then I changed all the imageview constraints to match_constraint so that I could set up a ratio.

You can do this either by clicking on the constraint arrows or in the select list in Declared Attribute change layout_height to 0dp (match constraint).

Once you change all 4 constraints to match_constraint you should have

android:layout_width="0dp" and **android:layout_height="0dp"** in Declared Attributes.

Then click on the triangle in the top left of the square in the Attributes window and set up a 1:1 ratio. This will allow the image to scale, keeping its aspect ratio, to match the constraints.

For this layout I standardized on a 8dp margin to the parent and 16dp margin between the views in the layout.

Run it and now see what it looks like in landscape.