

ATLS 4120/5120: Mobile Application Development

Week 11: Android Constraint Layout

Constraint Layout is a new layout in Android Studio 2.2 that allows you to create layouts with a flat view hierarchy which helps with performance. This is especially useful for large and complex layouts.

ConstraintLayout is similar to RelativeLayout, but uses various handles (or anchors) for the constraints. There is no nesting of other layouts in a ConstraintLayout.

Constraint Layouts

<https://developer.android.com/training/constraint-layout/index.html>

A constraint is the description of how a view should be positioned relative to other items in a layout. 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 guide line

A Constraint is essentially a defined rule for a view which declares its positioning and alignment on screen, the ConstraintLayout class supports several different types of handles.

Using Constraints

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

The new layout editor to create ConstraintLayout is available in Android Studio 2.2 and higher.

Creating Constraints

You can either create a new layout and select the root element to be a ConstraintLayout or convert an existing layout into a ConstraintLayout.

You can use the design or text editor for constraint layouts

Each constraint defines the view's position along either the vertical or horizontal axis; so each view must have a minimum of one constraint for each axis, but often more are necessary.


Constraint Rules

- Every view must have at least two constraints: one horizontal and one vertical.
 - Each constraint represents a connection or alignment to another view, the parent layout, or an invisible guideline.
- 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.


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

Auto-connect

Autoconnect automatically creates two or more constraints for each view as you add them to the layout.

- creates constraints to only the nearest elements.
- disabled by default
- You can enable it by clicking Turn on Autoconnect in the Layout Editor toolbar 
- 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 scans the entire layout to create a set of constraints for all views in your layout. 

Manual

You can create constraints manually by disabling auto connect.

When you drop a view into the Layout Editor, it stays where you leave it even if it has no constraints. However, this is only to make editing easier; if a view has no constraints when you run your layout on a device, it is drawn at position [0,0] (the top-left corner).

Although a missing constraint won't cause a compilation 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.

Constraint Types

Parent constraint

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

Position constraint

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

Alignment constraint

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

Baseline alignment constraint

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

- Baseline constraint handles can only be constrained 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.

If you add opposing constraints on a view, the constraint lines become squiggly like a spring to indicate the opposing forces

Adjusting Constraints

Resize

You can use the handles on each corner of the view to resize it, but doing so hard codes the width and height values, which you should avoid for most views because hard-coded view sizes cannot adapt to different content and screen sizes. To select from one of the dynamic sizing modes or to define more specific dimensions, click a view and open the Properties window and use the Inspector Pane to adjust the constraints.

Inspector Pane:



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



Any Size: The view expands as needed to match the constraints. The actual value is 0dp because the view has no desired dimensions, but it resizes as needed to meet the constraints. Another way to think of it is "match constraints" (instead of match_parent) because it expands the view as much as possible after accounting for the limits of each constraint and its margins.

You should not use match_parent for any view in a ConstraintLayout. Instead use "Any Size" (0dp).



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


Bias

When you add a constraint to both sides of a view (and the view size for the same dimension is either "fixed" or "wrap content"), the view becomes centered between the two anchor points by default. When a view is centered, the bias is 50%. You can adjust the bias by dragging the bias slider in the Properties window or by dragging the view

If you instead want the view to stretch its size to meet the constraints, switch the size to "any size".

- 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.

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 Properties window by clicking the number on the line that represents each constraint (in figure 10, the margins are each set to 16dp).

Deleting Constraints

- remove a single constraint by simply clicking on the anchor point of the constraint that you wish to delete.
- When a view is selected, a little icon will appear to the bottom left hand side - clicking this icon will remove **all** of the constraints that have been set on that view.

Attributes

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

Welcome/Halloween

Make a duplicate of your Welcome/Halloween app in the Finder and so we can convert it to a Constraint Layout. (Halloween constraint)

Setup needed in Android Studio

To use ConstraintLayout in your project:

Ensure you have the latest Constraint Layout library:

1. Click Tools > Android > SDK Manager.
2. Click the SDK Tools tab.
3. Expand Support Repository and then check ConstraintLayout for Android and Solver for ConstraintLayout. Check Show Package Details and take note of the version you're downloading (you'll need this below).
4. Click OK.
5. Under Gradle Scripts open build.gradle(Module:app)
6. Add the ConstraintLayout library as a dependency in your module-level build.gradle file:

```
dependencies {  
    compile 'com.android.support.constraint:constraint-layout:1.0.0-beta1'  
}
```

The library version you download may be higher, so be sure the value you specify here matches the version from step 3.

7. In the toolbar or sync notification, click Sync Project with Gradle Files.

Now we can convert our layout with ConstraintLayout.

Convert a layout to ConstraintLayout

To convert an existing layout to a constraint layout, follow these steps:

1. Open your layout in Android Studio and click the Design tab at the bottom of the editor window.
2. In the Component Tree window, right-click the layout and click Convert layout to ConstraintLayout.

It will prompt you that it requires the constraint-layout library, chose OK.
You might need to build your project.

Adjusting Constraints

Well, it tried! Before fixing this, make sure you have Show Constraints checked(eye icon)
I can't see my image so I went into the component tree and selected imageView.
In the properties I added the src @drawable/ghost just so I could see the image.
It's down past the bottom of the screen. If you hover over the image you'll see an X on the bottom left to delete all the constraints on this view. Click that and then drag the imageView where you want it.
Then click Infer Constraints(light bulb) to add constraints only to the imageView.
Once you get it where you want it remove the src so it's only added when the button is clicked.
Now let's fix the others. All of mine seems to be too far to the right. When I look at the editText I see that it has an end constraint of 53dp, but that's not what I want, so delete it.

Error: This view is not constrained horizontally: at runtime it will jump to the left unless you add a horizontal constraint

Instead of using Infer Constraints(which we could use), let's add the constraints manually.
You'll see a circle on the left side of name. Click and drag it 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 and the constraint error went away. If you run it this is now centered.

The button is constrained to the right and left of the editText so that looks fine. Play with the slider to see how you can change the bias.

Lastly, the textview does not look centered. If you select it you'll see it has a start constraint of 6 so I deleted it. I put in some text to make it bigger so I can see it.

I dragged it to the center and then used the handles the same as before to the right for a right constraint and then to the left for a left constraint. This added the following to the layout:

```
android:layout_marginEnd="16dp"  
app:layout_constraintRight_toRightOf="parent"  
android:layout_marginRight="16dp"  
android:layout_marginStart="16dp"  
app:layout_constraintLeft_toLeftOf="parent"  
android:layout_marginLeft="16dp"
```

Adding a Widget

Make a little room between the editText and button. Notice when you do this the constraint is automatically changed.

When you add widgets to a view, Autoconnect will automatically create your constraints.

If you don't have Autoconnect on then it will automatically be placed at (0,0)

[Show this]

So when you use a constraint layout you either should have Autoconnect on or create constraints manually.

Turn Autoconnect on by clicking on the U in the toolbar.

Then drag a widget from the palette into the editor below the edittext and above the button. Notice how the constraints are added automatically. Try to have it evenly spaced (you can change the values in the properties pane to any value).

Start by dragging a view from the Palette window into the editor. 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.

Click the view to select it. Then click-and-hold one of the constraint handles and drag the line to an available 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.