Constraint Layout

ConstraintLayout allows you to create large and complex layouts with a flat view hierarchy (no nested view groups). It's similar to RelativeLayout in that all views are laid out according to relationships between sibling views and the parent layout, but it's more flexible than RelativeLayout and easier to use with Android Studio's Layout Editor.

Compatibility:MIN API 9

Ref: https://developer.android.com/training/constraint-layout

Constraints overview

To define a view's position in ConstraintLayout,
you must add at least one horizontal and one vertical constraint for the view.
Each constraint represents a connection or alignment to another view, the parent layout, or an invisible guideline.
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.

Constraints overview

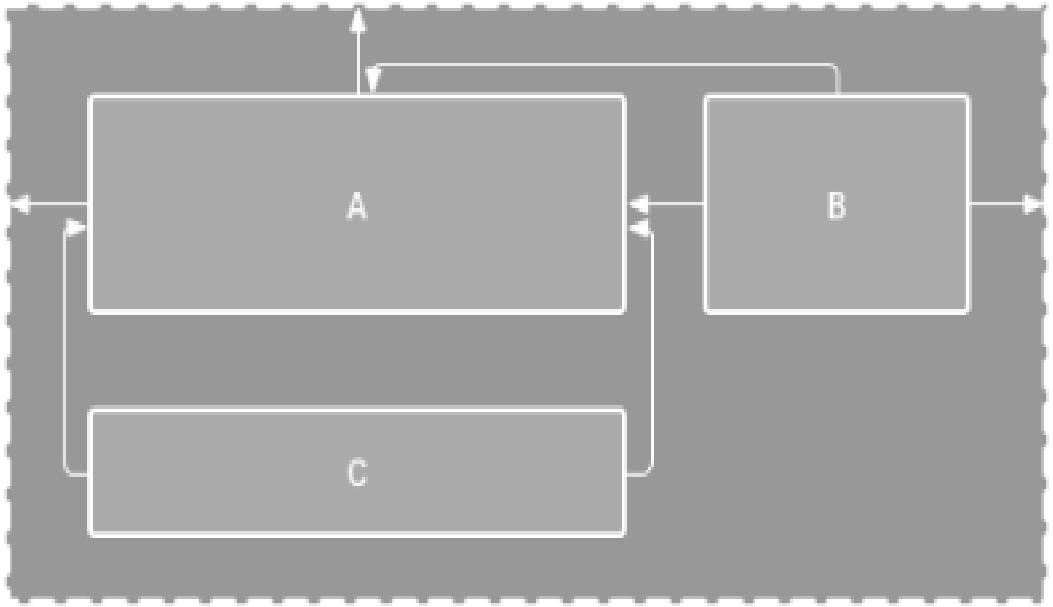

Figure 1. The editor shows view C below A, but it has no vertical constraint

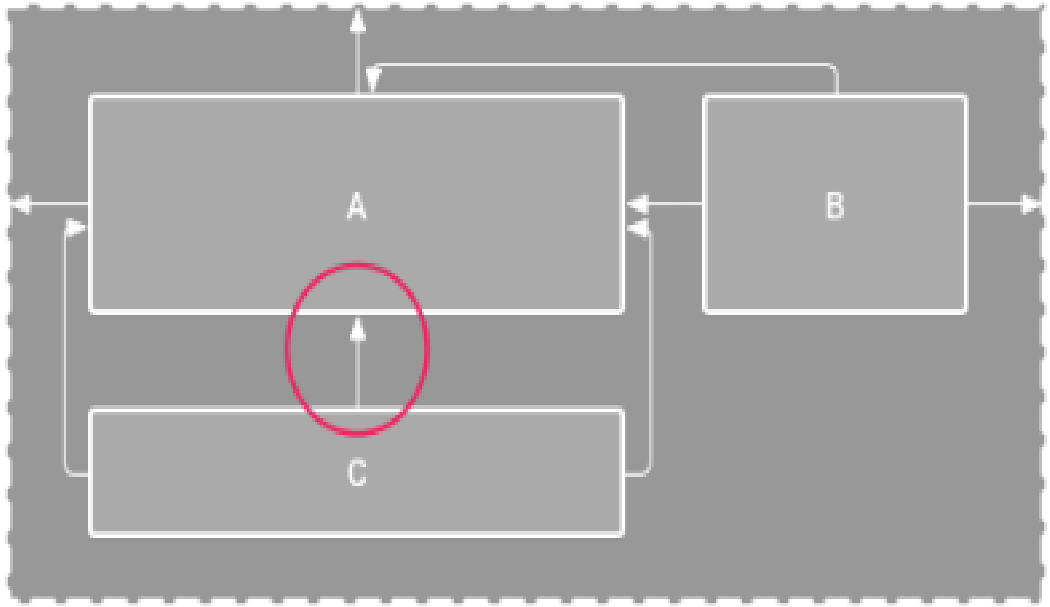

Figure 2. View C is now vertically constrained below view A

Add ConstraintLayout to your project

To use ConstraintLayout in your project, proceed as follows:
1.Ensure you have the maven.google.com repository declared in your module-level build.gradle file:

```
repositories {
        google()
 }
```

2.Add the library as a dependency in the same build.gradle file,as shown in the example below. Note that the latest version might be different than what is shown in the example:

```
dependencies {
 implementation 'com.android.support.constraint:constraint-layout:1.1.2'
 }
```

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

View Sizes

Fixed: You specify a specific dimension in the text box below or by resizing the view in the editor.
Wrap Content: The view expands only as much as needed to fit its contents.
Match Constraints: The view expands as much as possible to meet the constraints on each side

- **layout_constraintWidth_min**

This takes a dp dimension for the view's minimum width.

- **layout_constraintWidth_max**

This takes a dp dimension for the view's maximum width.

•**layout_constraintWidth_defaultspread**: 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. So the difference between this and using **Wrap Content** (above), is that setting the width to **Wrap Content** forces the width to always exactly match the content width; whereas using **Match Constraints** with **layout_constraintWidth_default** set to **wrap** also allows the view to be smaller than the content width.