

LINK 3

Support different screen sizes

- Android devices come in all shapes and sizes, so your app's layout needs to be flexible. That is, instead of defining your layout with rigid dimensions that assume a certain screen size and aspect ratio, your layout should gracefully respond to different screen sizes and orientations.
- By supporting as many screens as possible, your app can be made available to the greatest number of users with different devices, using a single APK. Additionally, making your app flexible for different screen sizes ensures that your app can handle window configuration changes on the device, such as when the user enables multi-window mode.

Multi-Window Support:

Android 7.0 adds support for displaying more than one app at the same time. On handheld devices, two apps can run side-by-side or one-above-the-other in *split-screen* mode. On TV devices, apps can use *picture-in-picture* mode to continue video playback while users are interacting with another app.

Different screen sizes can be supported using following techniques:

- Use view dimensions that allow the layout to resize
- Create alternative UI layouts according to the screen configuration
- Provide bitmaps that can stretch with the views

Create a flexible layout

No matter what hardware profile you want to support first, you need to create a layout that is responsive to even small variations in screen size.

Use `ConstraintLayout`:

- The best way to create a responsive layout for different screen sizes is to use `ConstraintLayout` as the base layout in your UI.
- `ConstraintLayout` allows you to specify the position and size for each view according to spatial relationships with other views in the layout. This way, all the views can move and stretch together as the screen size changes.
- The easiest way to build a layout with `ConstraintLayout` is to use the Layout Editor in Android Studio. It allows you to drag new views to the layout, attach their constraints to the parent view and other sibling views, and edit the view's properties, all without editing any XML by hand
- A `ConstraintLayout` is a `android.view.ViewGroup` which allows you to position and size widgets in a flexible way.

Refer this video =>

<https://www.youtube.com/watch?v=4N4bCdyGcUc&list=PLmPJCHvNZuA80INWNCLICR3qYzhw3iPI>

Avoid hard-coded layout sizes:

1. To ensure that your layout is flexible and adapts to different screen sizes, you should use `"wrap_content"` and `"match_parent"` for the width and height of most view components, instead of hard-coded sizes.
2. `"wrap_content"` tells the view to set its size to whatever is necessary to fit the content within that view.
3. `"match_parent"` makes the view expand to as much as possible within the parent view.

For example:

```

<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/lorem_ipsum" />

```

This `TextView` intends to set its width to fill all available space (`match_parent`) and set its height to exactly as much space is required by the length of the text (`wrap_content`). This allows the view to adapt to different screen sizes and different lengths of text.

Following figure shows how the width of the text view using `"match_parent"` adjusts as the screen width changes with device orientation.



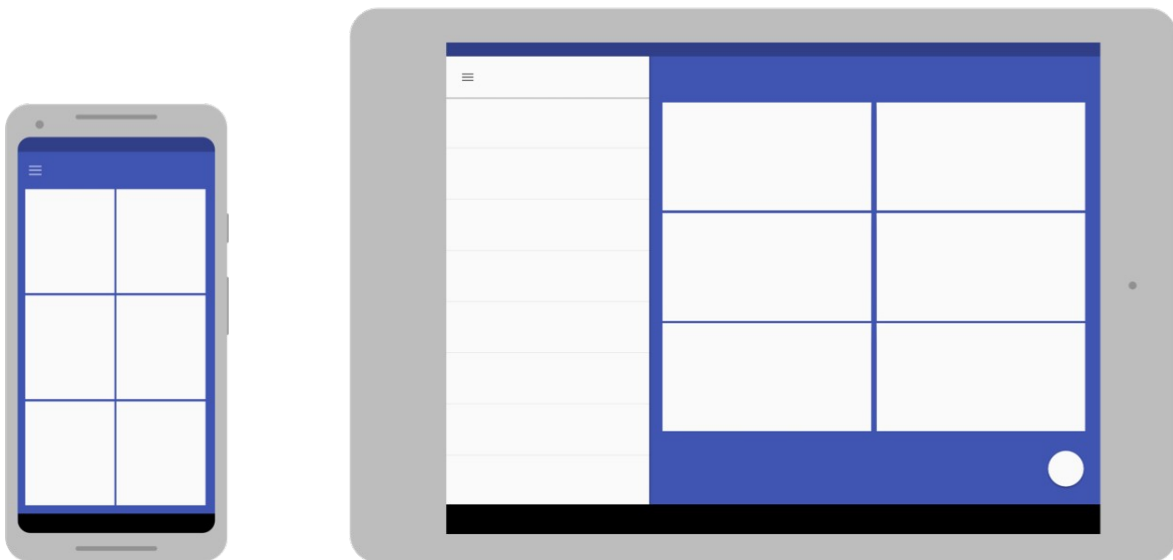
A flexible text view

- If you're using a `LinearLayout`, you can also expand the child views with layout weight so that each view fills the remaining space proportional to their weight value
- `LinearLayout` : A layout that arranges other views either horizontally in a single column or vertically in a single row.
- `Layout Weight`: `LinearLayout` also supports assigning a *weight* to individual children with the `android:layout_weight` attribute. This attribute assigns an "importance" value to a view in terms of how much space it should occupy on the screen. A larger weight value allows it to expand to

fill any remaining space in the parent view. Child views can specify a weight value, and then any remaining space in the view group is assigned to children in the proportion of their declared weight. Default weight is zero.

Create alternative layouts

Although your layout should always respond to different screen sizes by stretching the space within and around its views, that might not provide the best user experience for every screen size. For example, the UI you designed for a phone, probably doesn't offer a good experience on a tablet. Therefore, your app should also provide alternative layout resources to optimize the UI design for certain screen sizes.



Example Figure: The same app on different screen sizes uses a different layout for each

NOTE: You can provide screen-specific layouts by creating additional `res/layout/` directories—one for each screen configuration that requires a different layout—and then append a screen configuration qualifier to the `layout` directory name (such as `layout-w600dp` for screens that have 600dp of available width).

These configuration qualifiers represent the visible screen space available for your app UI. The system takes into account any system decorations (such as the navigation bar) and window configuration changes (such as when the user enables multi-window mode) when selecting the layout from your app.

For any doubt contact on Whatsapp number 9873961590