

## LINK 3 continued..

Some videos referred here are short videos related to the specific topics covered in this pdf. Watch the video and study the corresponding text before moving to next video.

## Use the available width qualifier

- Instead of changing the layout based on the smallest width of the screen, you might want to change your layout based on how much width or height is *currently available*.
- For example, if you have a two-pane layout, you might want to use that whenever the screen provides at least 600dp of width, which might change depending on whether the device is in landscape or portrait orientation.
- In this case, you should use the "available width" qualifier as follows:

```
res/layout/main_activity.xml          # For handsets (smaller than 600dp
available width)
res/layout-w600dp/main_activity.xml  # For 7" tablets or any screen with
600dp
                                       # available width (possibly
landscape handsets)
```

- If the available height is a concern for you, then you can do the same using the "available height" qualifier.
- For example, `layout-h600dp` for screens with at least 600dp of screen height.

## Add orientation qualifiers

- To change the user experience when the user switches between portrait and landscape orientations.

- For that you can add the `port` or `land` qualifiers to your resource directory names. Just be sure these come *after* the other size qualifiers.
- For example:

```
res/layout/main_activity.xml           # For handsets
res/layout-land/main_activity.xml    # For handsets in landscape
res/layout-sw600dp/main_activity.xml # For 7" tablets
res/layout-sw600dp-land/main_activity.xml # For 7" tablets in landscape
```

Refer → <https://www.youtube.com/watch?v=15jPM79isbg>

## Modularize UI components with fragments

- When designing your app for multiple screen sizes you want to make sure you aren't needlessly duplicating your UI behavior across your activities. So you should use fragments to extract your UI logic into separate components.
- Combine fragments to create multi-pane layouts when running on a large screen or place in separate activities when running on a handset.
- A `Fragment` represents a behavior or a portion of user interface in a `FragmentActivity`.
- Combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities. You can think of a fragment as a modular section of an activity, which has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running .
- For example, a news app on a tablet might show a list of articles on the left side and a full article on the right side—selecting an article on the left updates the article view on the right. On a handset, however, these two components should appear on separate screens—selecting an article from a list changes the entire screen to show that article.

Refer →

<https://www.youtube.com/watch?v=mcF28h9WiGQ>

<https://www.youtube.com/watch?v=FF-e6CnBwYY>

## Support Android 3.1 with legacy size qualifiers

- If your app supports Android 3.1 (API level 12) or lower, you need to use the legacy size qualifiers in addition to the smallest/available width qualifiers from above.
- From the example above, if you want a two pane layout on larger devices you need to use the "large" configuration qualifier to support version 3.1 and lower. So, to implement these layouts on those older versions, you might have the following files:

```
res/layout/main_activity.xml          # For handsets (smaller than 640dp
x 480dp)
res/layout-large/main_activity.xml    # For small tablets (640dp x 480dp
and bigger)
res/layout-xlarge/main_activity.xml  # For large tablets (960dp x 720dp
and bigger)
```

## Use layout aliases

When supporting both pre- and post-3.2 Android versions you have to use both the smallest-width and large qualifiers for your layouts. So, you would have a file named `res/layout-large/main.xml` which might be identical to `res/layout-sw600dp/main.xml`.

To avoid this duplication of the same file, you can use alias files. For example, you can define the following layouts:

```
res/layout/main.xml          # single-pane layout
res/layout/main_twopanes.xml # two-pane layout
```

And add these two files:

- `res/values-large/layout.xml`:

```
<resources>
  <item name="main" type="layout">@layout/main_twopanes</item>
</resources>
```

- `res/values-sw600dp/layout.xml`:

```
<resources>
  <item name="main" type="layout">@layout/main_twopanes</item>
</resources>
```

These two files have identical content, but they don't actually define the layout. They merely set up `main` to be an alias to `main_twopanels`. Since these files have `large` and `sw600dp` selectors, they are applied to large screens regardless of Android version (pre-3.2 tablets and TVs match `large`, and post-3.2 will match `sw600dp`).

For any doubt contact 9873961590