Configuration	Qualifier Values	Description

MCC and MNC	Examples:
	`mcc310`
	`mcc310-mnc004`
	`mcc208-mnc00`
	etc.
		The mobile country code (MCC), optionally followed by mobile network code (MNC) from the SIM card in the device. For example, `mcc310` is U.S. on any carrier, `mcc310-mnc004` is U.S. on Verizon, and `mcc208-mnc00` is France on Orange.
		
		If the device uses a radio connection (GSM phone), the MCC and MNC values come from the SIM card.
		
		You can also use the MCC alone (for example, to include country-specific legal resources in your application). If you need to specify based on the language only, then use the *language and region* qualifier instead (discussed next). If you decide to use the MCC and MNC qualifier, you should do so with care and test that it works as expected.
		
		Also see the configuration fields `mcc`, and `mnc`, which indicate the current mobile country code and mobile network code, respectively.

Language and region	Examples:
	`en`
	`fr`
	`en-rUS`
	`fr-rFR`
	`fr-rCA`
	etc.
		The language is defined by a two-letter ISO 639-1 language code, optionally followed by a two letter ISO 3166-1-alpha-2 region code (preceded by lowercase "`r`").
		
		The codes are *not* case-sensitive; the `r` prefix is used to distinguish the region portion. You cannot specify a region alone.
		
		This can change during the life of your application if the user changes his or her language in the system settings. See Handling Runtime Changes for information about how this can affect your application during runtime.
		
		See Localization for a complete guide to localizing your application for other languages.
		
		Also see the `locale` configuration field, which indicates the current locale.

Layout Direction	`ldrtl` `ldltr`	The layout direction of your application. `ldrtl` means "layout-direction-right-to-left". `ldltr` means "layout-direction-left-to-right" and is the default implicit value.

This can apply to any resource such as layouts, drawables, or values.

For example, if you want to provide some specific layout for the Arabic language and some generic layout for any other "right-to-left" language (like Persian or Hebrew) then you would have:

```
res/
    layout/
        main.xml  (Default layout)
    layout-ar/
        main.xml  (Specific layout for Arabic)
    layout-ldrtl/
        main.xml  (Any "right-to-left" language, except
                   for Arabic, because the "ar" language qualifier
                   has a higher precedence.)
```

Note: To enable right-to-left layout features for your app, you must set `supportsRtl` to `"true"` and set `targetSdkVersion` to 17 or higher.

Added in API level 17.

smallestWidth	`sw<N>dp`
	
	Examples:
	`sw320dp`
	`sw600dp`
	`sw720dp`
	etc.
		The fundamental size of a screen, as indicated by the shortest dimension of the available screen area. Specifically, the device's smallestWidth is the shortest of the screen's available height and width (you may also think of it as the "smallest possible width" for the screen). You can use this qualifier to ensure that, regardless of the screen's current orientation, your application has at least `<N>` dps of width available for its UI.
		
		For example, if your layout requires that its smallest dimension of screen area be at least 600 dp at all times, then you can use this qualifer to create the layout resources, `res/layout-sw600dp/`. The system will use these resources only when the smallest dimension of available screen is at least 600dp, regardless of whether the 600dp side is the user-perceived height or width. The smallestWidth is a fixed screen size characteristic of the device; **the device's smallestWidth does not change when the screen's orientation changes**.
		
		The smallestWidth of a device takes into account screen decorations and system UI. For example, if the device has some persistent UI elements on the screen that account for space along the axis of the smallestWidth, the system declares the smallestWidth to be smaller than the actual screen size, because those are screen pixels not available for your UI. Thus, the value you use should be the actual smallest dimension *required by your layout* (usually, this

value is the "smallest width" that your layout supports, regardless of the screen's current orientation).

Some values you might use here for common screen sizes:

- 320, for devices with screen configurations such as:
  - 240x320 ldpi (QVGA handset)
  - 320x480 mdpi (handset)
  - 480x800 hdpi (high-density handset)
- 480, for screens such as 480x800 mdpi (tablet/handset).
- 600, for screens such as 600x1024 mdpi (7" tablet).
- 720, for screens such as 720x1280 mdpi (10" tablet).

When your application provides multiple resource directories with different values for the smallestWidth qualifier, the system uses the one closest to (without exceeding) the device's smallestWidth.

Added in API level 13.

Also see the `android:requiresSmallestWidthDp` attribute, which declares the minimum smallestWidth with which your application is compatible, and the `smallestScreenWidthDp` configuration field, which holds the device's smallestWidth value.

For more information about designing for different screens and using this qualifier, see the Supporting Multiple Screens developer guide.

Available width	`w<N>dp`
	
	Examples:
	`w720dp`
	`w1024dp`
	etc.
		Specifies a minimum available screen width, in `dp` units at which the resource should be used—defined by the `<N>` value. This configuration value will change when the orientation changes between landscape and portrait to match the current actual width.
		
		When your application provides multiple resource directories with different values for this configuration, the system uses the one closest to (without exceeding) the device's current screen width. The value here takes into account screen decorations, so if the device has some persistent UI elements on the left or right edge of the display, it uses a value for the width that is smaller than the real screen size, accounting for these UI elements and reducing

		the application's available space. *Added in API level 13.* Also see the `screenWidthDp` configuration field, which holds the current screen width. For more information about designing for different screens and using this qualifier, see the Supporting Multiple Screens developer guide.

Available height	`h<N>dp`
	
	Examples:
	`h720dp`
	`h1024dp`
	etc.
		Specifies a minimum available screen height, in "dp" units at which the resource should be used—defined by the `<N>` value. This configuration value will change when the orientation changes between landscape and portrait to match the current actual height.
		
		When your application provides multiple resource directories with different values for this configuration, the system uses the one closest to (without exceeding) the device's current screen height. The value here takes into account screen decorations, so if the device has some persistent UI elements on the top or bottom edge of the display, it uses a value for the height that is smaller than the real screen size, accounting for these UI elements and reducing the application's available space. Screen decorations that are not fixed (such as a phone status bar that can be hidden when full screen) are *not* accounted for here, nor are window decorations like the title bar or action bar, so applications must be prepared to deal with a somewhat smaller space than they specify.
		
		Added in API level 13.
		
		Also see the `screenHeightDp` configuration field, which holds the current screen width.
		
		For more information about designing for different screens and using this qualifier, see the Supporting Multiple Screens developer guide.

Screen size	`small`
	`normal`
	`large`
	`xlarge`
		• `small`: Screens that are of similar size to a low-density QVGA screen. The minimum layout size for a small screen is approximately 320x426 dp units. Examples are QVGA low-density and VGA high density.
		• `normal`: Screens that are of similar size to a medium-density HVGA screen. The minimum layout size for a normal screen is approximately 320x470 dp units. Examples of such screens a WQVGA low-density, HVGA medium-density, WVGA high-density.

		• `large`: Screens that are of similar size to a medium-density VGA screen. The minimum layout size for a large screen is approximately 480x640 dp units. Examples are VGA and WVGA medium-density screens. • `xlarge`: Screens that are considerably larger than the traditional medium-density HVGA screen. The minimum layout size for an xlarge screen is approximately 720x960 dp units. In most cases, devices with extra-large screens would be too large to carry in a pocket and would most likely be tablet-style devices. *Added in API level 9.* **Note:** Using a size qualifier does not imply that the resources are *only* for screens of that size. If you do not provide alternative resources with qualifiers that better match the current device configuration, the system may use whichever resources are the best match. **Caution:** If all your resources use a size qualifier that is *larger* than the current screen, the system will **not** use them and your application will crash at runtime (for example, if all layout resources are tagged with the `xlarge` qualifier, but the device is a normal-size screen). *Added in API level 4.* See Supporting Multiple Screens for more information. Also see the `screenLayout` configuration field, which indicates whether the screen is small, normal, or large.
		• `long`: Long screens, such as WQVGA, WVGA, FWVGA • `notlong`: Not long screens, such as QVGA, HVGA, and VGA *Added in API level 4.*
Screen aspect	`long` `notlong`	This is based purely on the aspect ratio of the screen (a "long" screen is wider). This is not related to the screen orientation. Also see the `screenLayout` configuration field, which indicates whether the screen is long.
Round screen	`round` `notround`	• `round`: Round screens, such as a round wearable device

		• `notround`: Rectangular screens, such as phones or tablets *Added in API level 23.* Also see the `isScreenRound()` configuration method, which indicates whether the screen is round.
Screen orientation	`port` `land`	• `port`: Device is in portrait orientation (vertical) • `land`: Device is in landscape orientation (horizontal) This can change during the life of your application if the user rotates the screen. See Handling Runtime Changes for information about how this affects your application during runtime. Also see the `orientation` configuration field, which indicates the current device orientation.

UI mode	`car`
	`desk`
	`television`
	`appliance`
	`watch`
		• `car`: Device is displaying in a car dock
		• `desk`: Device is displaying in a desk dock
		• `television`: Device is displaying on a television, providing a "ten foot" experience where its UI is on a large screen that the user is far away from, primarily oriented around DPAD or other non-pointer interaction
		• `appliance`: Device is serving as an appliance, with no display
		• `watch`: Device has a display and is worn on the wrist
		
		Added in API level 8, television added in API 13, watch added in API 20.
		
		For information about how your app can respond when the device is inserted into or removed from a dock, read Determining and Monitoring the Docking State and Type.
		
		This can change during the life of your application if the user places the device in a dock. You can enable or disable some of these modes using `UiModeManager`. See Handling Runtime Changes for information about how this affects your application during runtime.

Night mode	`night` `notnight`	• `night`: Night time • `notnight`: Day time

Added in API level 8.

This can change during the life of your application if night mode is left in auto mode (default), in which case the mode changes based on the time of day. You can enable or disable this mode using `UiModeManager`. See Handling Runtime Changes for information about how this affects your application during runtime.

Screen pixel density (dpi)	`ldpi`
	`mdpi`
	`hdpi`
	`xhdpi`
	`xxhdpi`
	`xxxhdpi`
	`nodpi`
	`tvdpi`
		• `ldpi`: Low-density screens; approximately 120dpi.
		• `mdpi`: Medium-density (on traditional HVGA) screens; approximately 160dpi.
		• `hdpi`: High-density screens; approximately 240dpi.
		• `xhdpi`: Extra-high-density screens; approximately 320dpi. *Added in API Level 8*
		• `xxhdpi`: Extra-extra-high-density screens; approximately 480dpi. *Added in API Level 16*
		• `xxxhdpi`: Extra-extra-extra-high-density uses (launcher icon only, see the note in *Supporting Multiple Screens*); approximately 640dpi. *Added in API Level 18*
		• `nodpi`: This can be used for bitmap resources that you do not want to be scaled to match the device density.
		• `tvdpi`: Screens somewhere between mdpi and hdpi; approximately 213dpi. This is not considered a "primary" density group. It is mostly intended for televisions and most apps shouldn't need it—providing mdpi and hdpi resources is sufficient for most apps and the system will scale them as appropriate. This qualifier was introduced with API level 13.

There is a 3:4:6:8:12:16 scaling ratio between the six primary densities (ignoring the tvdpi density). So, a 9x9 bitmap in ldpi is 12x12 in mdpi, 18x18 in hdpi, 24x24 in xhdpi and so on.

If you decide that your image resources don't look good enough on a television or other certain devices and want to try tvdpi resources, the scaling factor is 1.33*mdpi. For example, a 100px x 100px image for mdpi screens should be 133px x 133px for tvdpi.

Note: Using a density qualifier does not imply that the resources are *only* for screens of that density. If you do not provide alternative resources with qualifiers that better match the current device configuration, the system may use whichever resources are the best match.

See Supporting Multiple Screens for more information about how to handle different screen densities and how

		Android might scale your bitmaps to fit the current density.
Touchscreen type	`notouch` `finger`	• `notouch`: Device does not have a touchscreen. • `finger`: Device has a touchscreen that is intended to be used through direction interaction of the user's finger. Also see the `touchscreen` configuration field, which indicates the type of touchscreen on the device.

Keyboard availability	`keysexposed`
	`keyshidden`
	`keyssoft`
		• `keysexposed`: Device has a keyboard available. If the device has a software keyboard enabled (which is likely), this may be used even when the hardware keyboard is *not* exposed to the user, even if the device has no hardware keyboard. If no software keyboard is provided or it's disabled, then this is only used when a hardware keyboard is exposed.
		• `keyshidden`: Device has a hardware keyboard available but it is hidden *and* the device does *not* have a software keyboard enabled.
		• `keyssoft`: Device has a software keyboard enabled, whether it's visible or not.
		
		If you provide `keysexposed` resources, but not `keyssoft` resources, the system uses the `keysexposed` resources regardless of whether a keyboard is visible, as long as the system has a software keyboard enabled.
		
		This can change during the life of your application if the user opens a hardware keyboard. See Handling Runtime Changes for information about how this affects your application during runtime.
		
		Also see the configuration fields `hardKeyboardHidden` and `keyboardHidden`, which indicate the visibility of a hardware keyboard and and the visibility of any kind of keyboard (including software), respectively.

Primary text input method	`nokeys`
	`qwerty`
	`12key`
		• `nokeys`: Device has no hardware keys for text input.
		• `qwerty`: Device has a hardware qwerty keyboard, whether it's visible to the user or not.
		• `12key`: Device has a hardware 12-key keyboard, whether it's visible to the user or not.
		
		Also see the `keyboard` configuration field, which indicates the primary text input method available.

Navigation key	`navexposed`	• `navexposed`: Navigation keys are available to the user.

availability	`navhidden`	• `navhidden`: Navigation keys are not available (such as behind a closed lid). This can change during the life of your application if the user reveals the navigation keys. See Handling Runtime Changes for information about how this affects your application during runtime. Also see the `navigationHidden` configuration field, which indicates whether navigation keys are hidden.

Primary non-touch navigation method	`nonav`
	`dpad`
	`trackball`
	`wheel`
		• `nonav`: Device has no navigation facility other than using the touchscreen.
		• `dpad`: Device has a directional-pad (d-pad) for navigation.
		• `trackball`: Device has a trackball for navigation.
		• `wheel`: Device has a directional wheel(s) for navigation (uncommon).
		
		Also see the `navigation` configuration field, which indicates the type of navigation method available.

Platform Version (API level)	Examples:
	`v3`
	`v4`
	`v7`
	etc.
		The API level supported by the device. For example, `v1` for API level 1 (devices with Android 1.0 or higher) and `v4` for API level 4 (devices with Android 1.6 or higher). See the Android API levels document for more information about these values.