Providing Resources

You should always externalize application resources such as images and strings from your code, so that you can maintain them independently. You should also provide alternative resources for specific device configurations, by grouping them in specially-named resource directories. At runtime, Android uses the appropriate resource based on the current configuration. For example, you might want to provide a different UI layout depending on the screen size or different strings depending on the language setting.

Once you externalize your application resources, you can access them using resource IDs that are generated in your project's `R` class. How to use resources in your application is discussed in Accessing Resources (https://developer.android.com/guide/topics/resources/accessing-resources.html). This document shows you how to group your resources in your Android project and provide alternative resources for specific device configurations.

Grouping Resource Types

You should place each type of resource in a specific subdirectory of your project's `res/` directory. For example, here's the file hierarchy for a simple project:

```
MyProject/
    src/
        MyActivity.java
    res/
        drawable/
            graphic.png
        layout/
            main.xml
            info.xml
        mipmap/
            icon.png
        values/
            strings.xml
```

As you can see in this example, the `res/` directory contains all the resources (in subdirectories): an image resource, two layout resources, `mipmap/` directories for launcher icons, and a string resource file. The resource directory names are important and are described in table 1.

> **Note:** For more information about using the mipmap folders, see Managing Projects Overview (https://developer.android.com/tools/projects/index.html#mipmap).

Table 1. Resource directories supported inside project `res/` directory.

Directory	Resource Type
`animator/`	XML files that define property animations (https://developer.android.com/guide/topics/graphics/prop-

	animation.html).
`anim/`	XML files that define tween animations (https://developer.android.com/guide/topics/graphics/view-animation.html#tween-animation). (Property animations can also be saved in this directory, but the `animator/` directory is preferred for property animations to distinguish between the two types.)
`color/`	XML files that define a state list of colors. See Color State List Resource (https://developer.android.com/guide/topics/resources/color-list-resource.html)
`drawable/`	Bitmap files (`.png`, `.9.png`, `.jpg`, `.gif`) or XML files that are compiled into the following drawable resource subtypes: • Bitmap files • Nine-Patches (re-sizable bitmaps) • State lists • Shapes • Animation drawables • Other drawables See Drawable Resources (https://developer.android.com/guide/topics/resources/drawable-resource.html).
`mipmap/`	Drawable files for different launcher icon densities. For more information on managing launcher icons with `mipmap/` folders, see Managing Projects Overview (https://developer.android.com/tools/projects/index.html#mipmap).
`layout/`	XML files that define a user interface layout. See Layout Resource (https://developer.android.com/guide/topics/resources/layout-resource.html).
`menu/`	XML files that define application menus, such as an Options Menu, Context Menu, or Sub Menu. See Menu Resource (https://developer.android.com/guide/topics/resources/menu-resource.html).
`raw/`	Arbitrary files to save in their raw form. To open these resources with a raw `InputStream` (https://developer.android.com/reference/java/io/InputStream.html), call `Resources.openRawResource()` (https://developer.android.com/reference/android/content/res/Resources.html#openRawResource(int)) with the resource ID, which is `R.raw.filename`. However, if you need access to original file names and file hierarchy, you might consider saving some resources in the `assets/` directory (instead of `res/raw/`). Files in `assets/` are not given a resource ID, so you can read them only using `AssetManager` (https://developer.android.com/reference/android/content/res/AssetManager.html).
`values/`	XML files that contain simple values, such as strings, integers, and colors. Whereas XML resource files in other `res/` subdirectories define a single resource based on the XML filename, files in the `values/` directory describe multiple resources. For a file in this directory, each child of the `<resources>` element defines a single resource. For example, a

	`<string>` element creates an `R.string` resource and a `<color>` element creates an `R.color` resource. Because each resource is defined with its own XML element, you can name the file whatever you want and place different resource types in one file. However, for clarity, you might want to place unique resource types in different files. For example, here are some filename conventions for resources you can create in this directory: • arrays.xml for resource arrays (typed arrays (https://developer.android.com/guide/topics/resources/more-resources.html#TypedArray)). • colors.xml for color values (https://developer.android.com/guide/topics/resources/more-resources.html#Color) • dimens.xml for dimension values (https://developer.android.com/guide/topics/resources/more-resources.html#Dimension). • strings.xml for string values (https://developer.android.com/guide/topics/resources/string-resource.html). • styles.xml for styles (https://developer.android.com/guide/topics/resources/style-resource.html). See String Resources (https://developer.android.com/guide/topics/resources/string-resource.html), Style Resource (https://developer.android.com/guide/topics/resources/style-resource.html), and More Resource Types (https://developer.android.com/guide/topics/resources/more-resources.html).
`xml/`	Arbitrary XML files that can be read at runtime by calling `Resources.getXML()` (`https://developer.android.com/reference/android/content/res/Resources.html#getXml(int)`). Various XML configuration files must be saved here, such as a searchable configuration (https://developer.android.com/guide/topics/search/searchable-config.html).

> **Caution:** Never save resource files directly inside the `res/` directory—it will cause a compiler error.

For more information about certain types of resources, see the Resource Types (https://developer.android.com/guide/topics/resources/available-resources.html) documentation.

The resources that you save in the subdirectories defined in table 1 are your "default" resources. That is, these resources define the default design and content for your application. However, different types of Android-powered devices might call for different types of resources. For example, if a device has a larger than normal screen, then you should provide different layout resources that take advantage of the extra screen space. Or, if a device has a different language setting, then you should provide different string resources that translate the text in your user interface. To provide these different resources for different device configurations, you need to provide alternative resources, in addition to your default resources.

Providing Alternative Resources

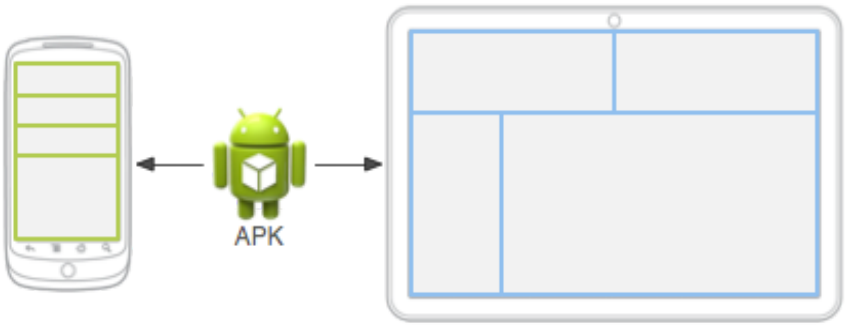

Figure 1. Two different devices, each using different layout resources.

Almost every application should provide alternative resources to support specific device configurations. For instance, you should include alternative drawable resources for different screen densities and alternative string resources for different languages. At runtime, Android detects the current device configuration and loads the appropriate resources for your application.

To specify configuration-specific alternatives for a set of resources:

1. Create a new directory in `res/` named in the form `<resources_name>-<config_qualifier>`.
   - `<resources_name>` is the directory name of the corresponding default resources (defined in table 1).
   - `<qualifier>` is a name that specifies an individual configuration for which these resources are to be used (defined in table 2).

   You can append more than one `<qualifier>`. Separate each one with a dash.

   > **Caution:** When appending multiple qualifiers, you must place them in the same order in which they are listed in table 2. If the qualifiers are ordered wrong, the resources are ignored.

2. Save the respective alternative resources in this new directory. The resource files must be named exactly the same as the default resource files.

For example, here are some default and alternative resources:

```
res/
    drawable/
        icon.png
        background.png
    drawable-hdpi/
        icon.png
        background.png
```

The `hdpi` qualifier indicates that the resources in that directory are for devices with a high-density screen. The images in each of these drawable directories are sized for a specific screen density, but the filenames are exactly the same. This way, the resource ID that you use to reference the `icon.png` or `background.png` image is always the same, but Android selects the version of each resource that best matches the current device, by comparing the device configuration information with the qualifiers in the resource directory name.

Android supports several configuration qualifiers and you can add multiple qualifiers to one directory name, by separating each qualifier with a dash. Table 2 lists the valid configuration qualifiers, in order of precedence—if you

use multiple qualifiers for a resource directory, you must add them to the directory name in the order they are listed in the table.

Table 2. Configuration qualifier names.

Configuration	Qualifier Values	Description

MCC and MNC	Examples:
	`mcc310`
	`mcc310-mnc004`
	`mcc208-mnc00`
	etc.
		The mobile country code (MCC), optionally followed by mobile network cod in the device. For example, `mcc310` is U.S. on any carrier, `mcc310-mnc004` is `mcc208-mnc00` is France on Orange.
		
		If the device uses a radio connection (GSM phone), the MCC and MNC valu
		
		You can also use the MCC alone (for example, to include country-specific le application). If you need to specify based on the language only, then use the qualifier instead (discussed next). If you decide to use the MCC and MNC q with care and test that it works as expected.
		
		Also see the configuration fields `mcc` (https://developer.android.com/reference/android/content/res/Configuration (https://developer.android.com/reference/android/content/res/Configuration 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 (http://www.loc.gov/standards/ language code, optionally followed by a two letter ISO 3166-1-alpha-2 (http:// services/iso3166ma/02iso-3166-code-lists/list-en1.html) region code (preceded by lov
		
		The codes are *not* case-sensitive; the `r` prefix is used to distinguish the regi specify a region alone.
		
		This can change during the life of your application if the user changes his o system settings. See Handling Runtime Changes (https://developer.android.com/ changes.html) for information about how this can affect your application durir
		
		See Localization (https://developer.android.com/guide/topics/resources/localization.htm localizing your application for other languages.
		
		Also see the `locale` (https://developer.android.com/reference/android/content/res/Configuration field, which indicates the current locale.

Layout Direction	`ldrtl`
	`ldltr`
		The layout direction of your application. `ldrtl` means "layout-direction-righ "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 lang layout for any other "right-to-left" language (like Persian or Hebrew) then yo
		
		`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 qua has a higher precedence.)`
		
		> **Note:** To enable right-to-left layout features for your app, you must set s (https://developer.android.com/guide/topics/manifest/application-element.html#suppor `targetSdkVersion` (https://developer.android.com/guide/topics/manifest/uses-sd 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 o Specifically, the device's smallestWidth is the shortest of the screen's availa may also think of it as the "smallest possible width" for the screen). You ca that, regardless of the screen's current orientation, your application has at l available for its UI.
		
		For example, if your layout requires that its smallest dimension of screen a times, then you can use this qualifer to create the layout resources, `res/la` system will use these resources only when the smallest dimension of availa 600dp, regardless of whether the 600dp side is the user-perceived height o is a fixed screen size characteristic of the device; **the device's smallestWidt the screen's orientation changes**.
		
		The smallestWidth of a device takes into account screen decorations and s the device has some persistent UI elements on the screen that account for smallestWidth, the system declares the smallestWidth to be smaller than th because those are screen pixels not available for your UI. Thus, the value yo smallest dimension *required by your layout* (usually, this value is the "smalle 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 smallestWidth qualifier, the system uses the one closest to (without exceed smallestWidth. *Added in API level 13.* Also see the `android:requiresSmallestWidthDp` (https://developer.android.com/guide/topics/manifest/supports-screens-element.html#requi declares the minimum smallestWidth with which your application is compa `smallestScreenWidthDp` (https://developer.android.com/reference/android/content/res/Configuration configuration field, which holds the device's smallestWidth value. For more information about designing for different screens and using this c Multiple Screens (https://developer.android.com/guide/practices/screens_support.html)

Available width	`w<N>dp`
	
	Examples:
	`w720dp`
	`w1024dp`
	etc.
		Specifies a minimum available screen width, in `dp` units at which the resour by the `<N>` value. This configuration value will change when the orientation landscape and portrait to match the current actual width.
		
		When your application provides multiple resource directories with different configuration, the system uses the one closest to (without exceeding) the d width. The value here takes into account screen decorations, so if the devic elements on the left or right edge of the display, it uses a value for the widt real screen size, accounting for these UI elements and reducing the applica
		
		Added in API level 13.
		
		Also see the `screenWidthDp` (https://developer.android.com/reference/android/content/res/Configuration configuration field, which holds the current screen width.
		
		For more information about designing for different screens and using this c Multiple Screens (https://developer.android.com/guide/practices/screens_support.html)

Available height	`h<N>dp`
	
	Examples:
	`h720dp`
	`h1024dp`
	etc.
		Specifies a minimum available screen height, in "dp" units at which the reso defined by the `<N>` value. This configuration value will change when the orie landscape and portrait to match the current actual height.
		
		When your application provides multiple resource directories with different configuration, the system uses the one closest to (without exceeding) the d height. The value here takes into account screen decorations, so if the devi elements on the top or bottom edge of the display, it uses a value for the he real screen size, accounting for these UI elements and reducing the applica Screen decorations that are not fixed (such as a phone status bar that can l are *not* accounted for here, nor are window decorations like the title bar or a must be prepared to deal with a somewhat smaller space than they specify
		
		Added in API level 13.

		Also see the `screenHeightDp` (https://developer.android.com/reference/android/content/res/Configuration configuration field, which holds the current screen width. For more information about designing for different screens and using this c Multiple Screens (https://developer.android.com/guide/practices/screens_support.html)

Screen size	`small`
	`normal`
	`large`
	`xlarge`
		`small`: Screens that are of similar size to a low-density QVGA screen. The r small screen is approximately 320x426 dp units. Examples are QVGA low-d density.
		
		`normal`: Screens that are of similar size to a medium-density HVGA screen. for a normal screen is approximately 320x470 dp units. Examples of such s density, HVGA medium-density, WVGA high-density.
		
		`large`: Screens that are of similar size to a medium-density VGA screen. Th large screen is approximately 480x640 dp units. Examples are VGA and WV screens.
		
		`xlarge`: Screens that are considerably larger than the traditional medium-d minimum layout size for an xlarge screen is approximately 720x960 dp unit with extra-large screens would be too large to carry in a pocket and would r devices. *Added in API level 9.*
		
		> **Note:** Using a size qualifier does not imply that the resources are *only* fo do not provide alternative resources with qualifiers that better match th configuration, the system may use whichever resources are the best ma
		
		> **Caution:** If all your resources use a size qualifier that is *larger* than the c will **not** use them and your application will crash at runtime (for exampl tagged with the `xlarge` qualifier, but the device is a normal-size screen)
		
		Added in API level 4.
		
		See Supporting Multiple Screens (https://developer.android.com/guide/practices/scre information.
		
		Also see the `screenLayout` (https://developer.android.com/reference/android/content/res/Configuration configuration field, which indicates whether the screen is small, normal, or l

Screen aspect	`long` `notlong`	`long`: Long screens, such as WQVGA, WVGA, FWVGA `notlong`: Not long screens, such as QVGA, HVGA, and VGA *Added in API level 4.* This is based purely on the aspect ratio of the screen (a "long" screen is wic the screen orientation.

		Also see the `screenLayout` (https://developer.android.com/reference/android/content/res/Configuration 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()` (https://developer.android.com/reference/android/content/res/Configuration 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 sc Changes (https://developer.android.com/guide/topics/resources/runtime-changes.html) fc this affects your application during runtime. Also see the `orientation` (https://developer.android.com/reference/android/content/res/Configuration 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" exp large screen that the user is far away from, primarily oriented around DPAD 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 inserte dock, read Determining and Monitoring the Docking State and Type (https://developer.android.com/training/monitoring-device-state/docking-monitoring.html).
		
		This can change during the life of your application if the user places the de enable or disable some of these modes using `UiModeManager` (https://developer.android.com/reference/android/app/UiModeManager.html). S Changes (https://developer.android.com/guide/topics/resources/runtime-changes.html) fc this affects your application during runtime.

Night mode	`night`	`night`: Night time

	`notnight`	`notnight`: Day time *Added in API level 8.* This can change during the life of your application if night mode is left in au case the mode changes based on the time of day. You can enable or disabl `UiModeManager` (https://developer.android.com/reference/android/app/UiMo Handling Runtime Changes (https://developer.android.com/guide/topics/resources/ru information about how this affects your application during runtime.

Screen pixel density (dpi)	`ldpi`
	`mdpi`
	`hdpi`
	`xhdpi`
	`xxhdpi`
	`xxxhdpi`
	`nodpi`
	`tvdpi`
	`anydpi`
		`ldpi`: Low-density screens; approximately 120dpi.
		
		`mdpi`: Medium-density (on traditional HVGA) screens; approximately 160dp
		
		`hdpi`: High-density screens; approximately 240dpi.
		
		`xhdpi`: Extra-high-density screens; approximately 320dpi. *Added in API Leve*
		
		`xxhdpi`: Extra-extra-high-density screens; approximately 480dpi. *Added in A*
		
		`xxxhdpi`: Extra-extra-extra-high-density uses (launcher icon only, see the nc (https://developer.android.com/guide/practices/screens_support.html#xxxhdpi-note) in *Sup* approximately 640dpi. *Added in API Level 18*
		
		`nodpi`: This can be used for bitmap resources that you do not want to be sc density.
		
		`tvdpi`: Screens somewhere between mdpi and hdpi; approximately 213dpi. "primary" density group. It is mostly intended for televisions and most apps mdpi and hdpi resources is sufficient for most apps and the system will sc *Added in API Level 13*
		
		`anydpi`: This qualifier matches all screen densities and takes precedence c useful for vector drawables (https://developer.android.com/training/material/drawable *in API Level 21*
		
		There is a 3:4:6:8:12:16 scaling ratio between the six primary densities (ign a 9x9 bitmap in ldpi is 12x12 in mdpi, 18x18 in hdpi, 24x24 in xhdpi and so
		
		If you decide that your image resources don't look good enough on a televis and want to try tvdpi resources, the scaling factor is 1.33*mdpi. For exampl for mdpi screens should be 133px x 133px for tvdpi.
		
		> **Note:** Using a density qualifier does not imply that the resources are *onl* density. If you do not provide alternative resources with qualifiers that b device configuration, the system may use whichever resources are the b
		
		See Supporting Multiple Screens (https://developer.android.com/guide/practices/scre information about how to handle different screen densities and how Androi

		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 direc finger. Also see the `touchscreen` (https://developer.android.com/reference/android/content/res/Configuration 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 softwar is likely), this may be used even when the hardware keyboard is *not* expose device has no hardware keyboard. If no software keyboard is provided or it' used when a hardware keyboard is exposed.
		
		`keyshidden`: Device has a hardware keyboard available but it is hidden *and* software keyboard enabled.
		
		`keyssoft`: Device has a software keyboard enabled, whether it's visible or r
		
		If you provide `keysexposed` resources, but not `keyssoft` resources, the sy `keysexposed` resources regardless of whether a keyboard is visible, as lon software keyboard enabled.
		
		This can change during the life of your application if the user opens a hardv Runtime Changes (https://developer.android.com/guide/topics/resources/runtime-chang about how this affects your application during runtime.
		
		Also see the configuration fields `hardKeyboardHidden` (https://developer.android.com/reference/android/content/res/Configuration and `keyboardHidden` (https://developer.android.com/reference/android/content/res/Configuration indicate the visibility of a hardware keyboard and and the visibility of any ki 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
		
		`12key`: Device has a hardware 12-key keyboard, whether it's visible to the us
		
		Also see the `keyboard` (https://developer.android.com/reference/android/content/res/Configuration configuration field, which indicates the primary text input method available.

Navigation key availability	`navexposed` `navhidden`	`navexposed`: Navigation keys are available to the user. `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 na Runtime Changes (https://developer.android.com/guide/topics/resources/runtime-chang about how this affects your application during runtime. Also see the `navigationHidden` (https://developer.android.com/reference/android/content/res/Configuration 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` (https://developer.android.com/reference/android/content/res/Configuration
		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 (devi
		higher) and `v4` for API level 4 (devices with Android 1.6 or higher). See the A
		(https://developer.android.com/guide/topics/manifest/uses-sdk-element.html#ApiLevels) do
		information about these values.

> **Note:** Some configuration qualifiers have been added since Android 1.0, so not all versions of Android support all the qualifiers. Using a new qualifier implicitly adds the platform version qualifier so that older devices are sure to ignore it. For example, using a `w600dp` qualifier will automatically include the `v13` qualifier, because the available-width qualifier was new in API level 13. To avoid any issues, always include a set of default resources (a set of resources with *no qualifiers*). For more information, see the section about Providing the Best Device Compatibility with Resources (#Compatibility).

Qualifier name rules

Here are some rules about using configuration qualifier names:

- You can specify multiple qualifiers for a single set of resources, separated by dashes. For example, `drawable-en-rUS-land` applies to US-English devices in landscape orientation.
- The qualifiers must be in the order listed in table 2 (#table2). For example:
  - Wrong: `drawable-hdpi-port/`
  - Correct: `drawable-port-hdpi/`
- Alternative resource directories cannot be nested. For example, you cannot have `res/drawable/drawable-en/`.

- Values are case-insensitive. The resource compiler converts directory names to lower case before processing to avoid problems on case-insensitive file systems. Any capitalization in the names is only to benefit readability.
- Only one value for each qualifier type is supported. For example, if you want to use the same drawable files for Spain and France, you *cannot* have a directory named `drawable-rES-rFR/`. Instead you need two resource directories, such as `drawable-rES/` and `drawable-rFR/`, which contain the appropriate files. However, you are not required to actually duplicate the same files in both locations. Instead, you can create an alias to a resource. See Creating alias resources (#AliasResources) below.

After you save alternative resources into directories named with these qualifiers, Android automatically applies the resources in your application based on the current device configuration. Each time a resource is requested, Android checks for alternative resource directories that contain the requested resource file, then finds the best-matching resource (#BestMatch) (discussed below). If there are no alternative resources that match a particular device configuration, then Android uses the corresponding default resources (the set of resources for a particular resource type that does not include a configuration qualifier).

Creating alias resources

When you have a resource that you'd like to use for more than one device configuration (but do not want to provide as a default resource), you do not need to put the same resource in more than one alternative resource directory. Instead, you can (in some cases) create an alternative resource that acts as an alias for a resource saved in your default resource directory.

> **Note:** Not all resources offer a mechanism by which you can create an alias to another resource. In particular, animation, menu, raw, and other unspecified resources in the `xml/` directory do not offer this feature.

For example, imagine you have an application icon, `icon.png`, and need unique version of it for different locales. However, two locales, English-Canadian and French-Canadian, need to use the same version. You might assume that you need to copy the same image into the resource directory for both English-Canadian and French-Canadian, but it's not true. Instead, you can save the image that's used for both as `icon_ca.png` (any name other than `icon.png`) and put it in the default `res/drawable/` directory. Then create an `icon.xml` file in `res/drawable-en-rCA/` and `res/drawable-fr-rCA/` that refers to the `icon_ca.png` resource using the `<bitmap>` element. This allows you to store just one version of the PNG file and two small XML files that point to it. (An example XML file is shown below.)

Drawable

To create an alias to an existing drawable, use the `<bitmap>` element. For example:

```
<?xml version="1.0" encoding="utf-8"?>
<bitmap xmlns:android="http://schemas.android.com/apk/res/android"
    android:src="@drawable/icon_ca" />
```

If you save this file as `icon.xml` (in an alternative resource directory, such as `res/drawable-en-rCA/`), it is compiled into a resource that you can reference as `R.drawable.icon`, but is actually an alias for the `R.drawable.icon_ca` resource (which is saved in `res/drawable/`).

Layout

To create an alias to an existing layout, use the `<include>` element, wrapped in a `<merge>`. For example:

```
<?xml version="1.0" encoding="utf-8"?>
<merge>
    <include layout="@layout/main_ltr"/>
</merge>
```

If you save this file as `main.xml`, it is compiled into a resource you can reference as `R.layout.main`, but is actually an alias for the `R.layout.main_ltr` resource.

Strings and other simple values

To create an alias to an existing string, simply use the resource ID of the desired string as the value for the new string. For example:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello</string>
    <string name="hi">@string/hello</string>
</resources>
```

The `R.string.hi` resource is now an alias for the `R.string.hello`.

Other simple values (https://developer.android.com/guide/topics/resources/more-resources.html) work the same way. For example, a color:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="red">#f00</color>
    <color name="highlight">@color/red</color>
</resources>
```

Providing the Best Device Compatibility with Resources

In order for your application to support multiple device configurations, it's very important that you always provide default resources for each type of resource that your application uses.

For example, if your application supports several languages, always include a `values/` directory (in which your strings are saved) *without* a language and region qualifier (#LocaleQualifier). If you instead put all your string files in directories that have a language and region qualifier, then your application will crash when run on a device set to a language that your strings do not support. But, as long as you provide default `values/` resources, then your application will run properly (even if the user doesn't understand that language—it's better than crashing).

Likewise, if you provide different layout resources based on the screen orientation, you should pick one orientation as your default. For example, instead of providing layout resources in `layout-land/` for landscape and `layout-`

`port/` for portrait, leave one as the default, such as `layout/` for landscape and `layout-port/` for portrait.

Providing default resources is important not only because your application might run on a configuration you had not anticipated, but also because new versions of Android sometimes add configuration qualifiers that older versions do not support. If you use a new resource qualifier, but maintain code compatibility with older versions of Android, then when an older version of Android runs your application, it will crash if you do not provide default resources, because it cannot use the resources named with the new qualifier. For example, if your `minSdkVersion` (https://developer.android.com/guide/topics/manifest/uses-sdk-element.html#min) is set to 4, and you qualify all of your drawable resources using night mode (#NightQualifier) (`night` or `notnight`, which were added in API Level 8), then an API level 4 device cannot access your drawable resources and will crash. In this case, you probably want `notnight` to be your default resources, so you should exclude that qualifier so your drawable resources are in either `drawable/` or `drawable-night/`.

So, in order to provide the best device compatibility, always provide default resources for the resources your application needs to perform properly. Then create alternative resources for specific device configurations using the configuration qualifiers.

There is one exception to this rule: If your application's `minSdkVersion` (https://developer.android.com/guide/topics/manifest/uses-sdk-element.html#min) is 4 or greater, you *do not* need default drawable resources when you provide alternative drawable resources with the screen density (#DensityQualifier) qualifier. Even without default drawable resources, Android can find the best match among the alternative screen densities and scale the bitmaps as necessary. However, for the best experience on all types of devices, you should provide alternative drawables for all three types of density.

How Android Finds the Best-matching Resource

When you request a resource for which you provide alternatives, Android selects which alternative resource to use at runtime, depending on the current device configuration. To demonstrate how Android selects an alternative resource, assume the following drawable directories each contain different versions of the same images:

```
drawable/
drawable-en/
drawable-fr-rCA/
drawable-en-port/
drawable-en-notouch-12key/
drawable-port-ldpi/
drawable-port-notouch-12key/
```

And assume the following is the device configuration:

- Locale = `en-GB`
- Screen orientation = `port`
- Screen pixel density = `hdpi`
- Touchscreen type = `notouch`
- Primary text input method = `12key`

By comparing the device configuration to the available alternative resources, Android selects drawables from `drawable-en-port`.

The system arrives at its decision for which resources to use with the following logic:

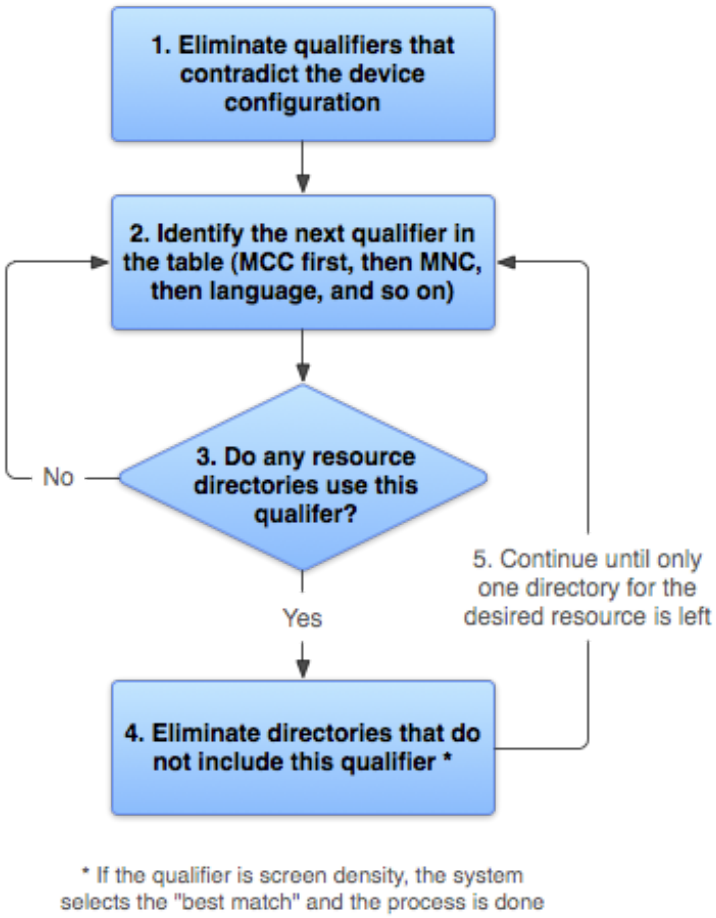

Figure 2. Flowchart of how Android finds the best-matching resource.

1. Eliminate resource files that contradict the device configuration.

   The `drawable-fr-rCA/` directory is eliminated, because it contradicts the `en-GB` locale.

   ```
   drawable/
   drawable-en/
   ~~drawable-fr-rCA/~~
   drawable-en-port/
   drawable-en-notouch-12key/
   drawable-port-ldpi/
   drawable-port-notouch-12key/
   ```

   > **Exception:** Screen pixel density is the one qualifier that is not eliminated due to a contradiction. Even though the screen density of the device is hdpi, `drawable-port-ldpi/` is not eliminated because every screen density is considered to be a match at this point. More information is available in the Supporting Multiple Screens (https://developer.android.com/guide/practices/screens_support.html) document.

2. Pick the (next) highest-precedence qualifier in the list (table 2 (#table2)). (Start with MCC, then move down.)

3. Do any of the resource directories include this qualifier?

- If No, return to step 2 and look at the next qualifier. (In the example, the answer is "no" until the language qualifier is reached.)
- If Yes, continue to step 4.

4. Eliminate resource directories that do not include this qualifier. In the example, the system eliminates all the directories that do not include a language qualifier:

   ~~`drawable/`~~
   `drawable-en/`
   `drawable-en-port/`
   `drawable-en-notouch-12key/`
   ~~`drawable-port-ldpi/`~~
   ~~`drawable-port-notouch-12key/`~~

   > **Exception:** If the qualifier in question is screen pixel density, Android selects the option that most closely matches the device screen density. In general, Android prefers scaling down a larger original image to scaling up a smaller original image. See Supporting Multiple Screens (https://developer.android.com/guide/practices/screens_support.html).

5. Go back and repeat steps 2, 3, and 4 until only one directory remains. In the example, screen orientation is the next qualifier for which there are any matches. So, resources that do not specify a screen orientation are eliminated:

   ~~`drawable-en/`~~
   `drawable-en-port/`
   ~~`drawable-en-notouch-12key/`~~

   The remaining directory is `drawable-en-port`.

Though this procedure is executed for each resource requested, the system further optimizes some aspects. One such optimization is that once the device configuration is known, it might eliminate alternative resources that can never match. For example, if the configuration language is English ("en"), then any resource directory that has a language qualifier set to something other than English is never included in the pool of resources checked (though a resource directory *without* the language qualifier is still included).

When selecting resources based on the screen size qualifiers, the system will use resources designed for a screen smaller than the current screen if there are no resources that better match (for example, a large-size screen will use normal-size screen resources if necessary). However, if the only available resources are *larger* than the current screen, the system will **not** use them and your application will crash if no other resources match the device configuration (for example, if all layout resources are tagged with the `xlarge` qualifier, but the device is a normal-size screen).

> **Note:** The *precedence* of the qualifier (in table 2 (#table2)) is more important than the number of qualifiers that exactly match the device. For example, in step 4 above, the last choice on the list includes three qualifiers that exactly match the device (orientation, touchscreen type, and input method), while `drawable-en` has only one parameter that matches (language). However, language has a higher precedence than these other qualifiers, so `drawable-port-notouch-12key` is out.

To learn more about how to use resources in your application, continue to Accessing Resources (https://developer.android.com/guide/topics/resources/accessing-resources.html).