Lecture 4
Resources

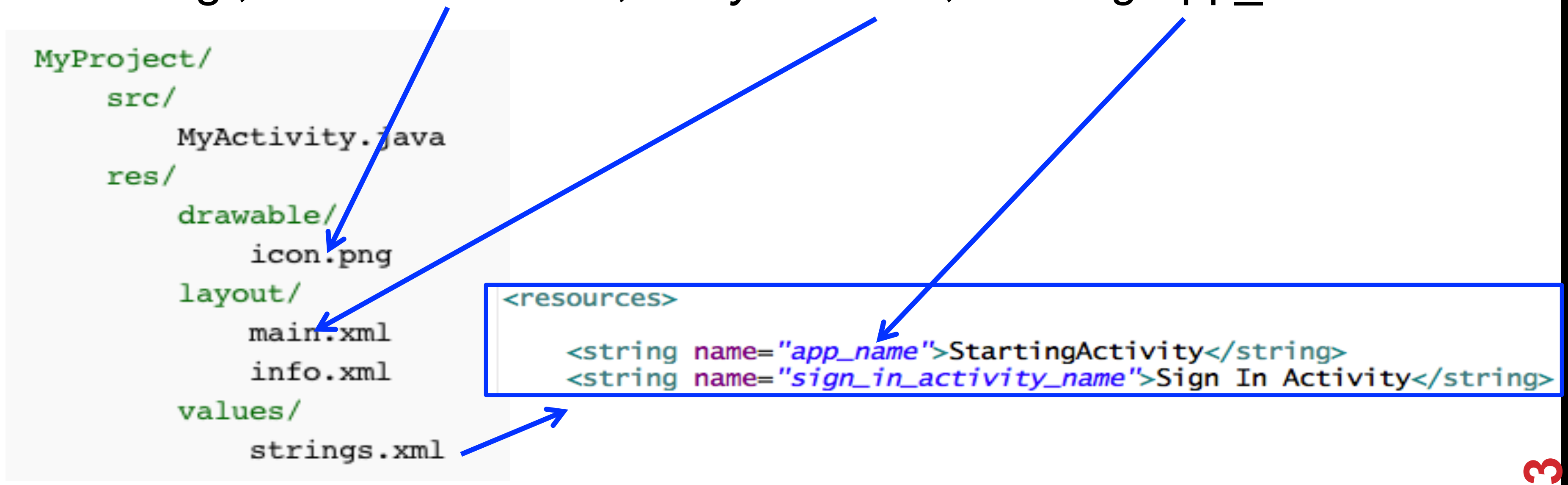
CMSC 4303/5303 Mobile Apps Programming
Hong K. Sung, Ph.D.
University of Central Oklahoma

App Resources Overview

- Always externalize resources (images, strings, etc) from the application code.
  - This will make it easy to support variety of device configurations; languages, screen sizes, orientation
- **res/** directory to store externalized resources
- For any type of resource, you can specify default and multiple alternative resources
  - default: used regardless of the device configuration or when there is no alternative.
    - e.g., res/layout
  - alternative: designed for a specific configuration, using **appended qualifier**
    - e.g., res/layout**-land**

Grouping Resource Types

- Resources are grouped by specially-named resource directories.
- You can access resources using resource IDs that are generated automatically in R class
  - e.g., R.*drawable*.icon, R.*layout*.main, R.*string*.app_name



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

```
<resources>

    <string name="app_name">StartingActivity</string>
    <string name="sign_in_activity_name">Sign In Activity</string>
```

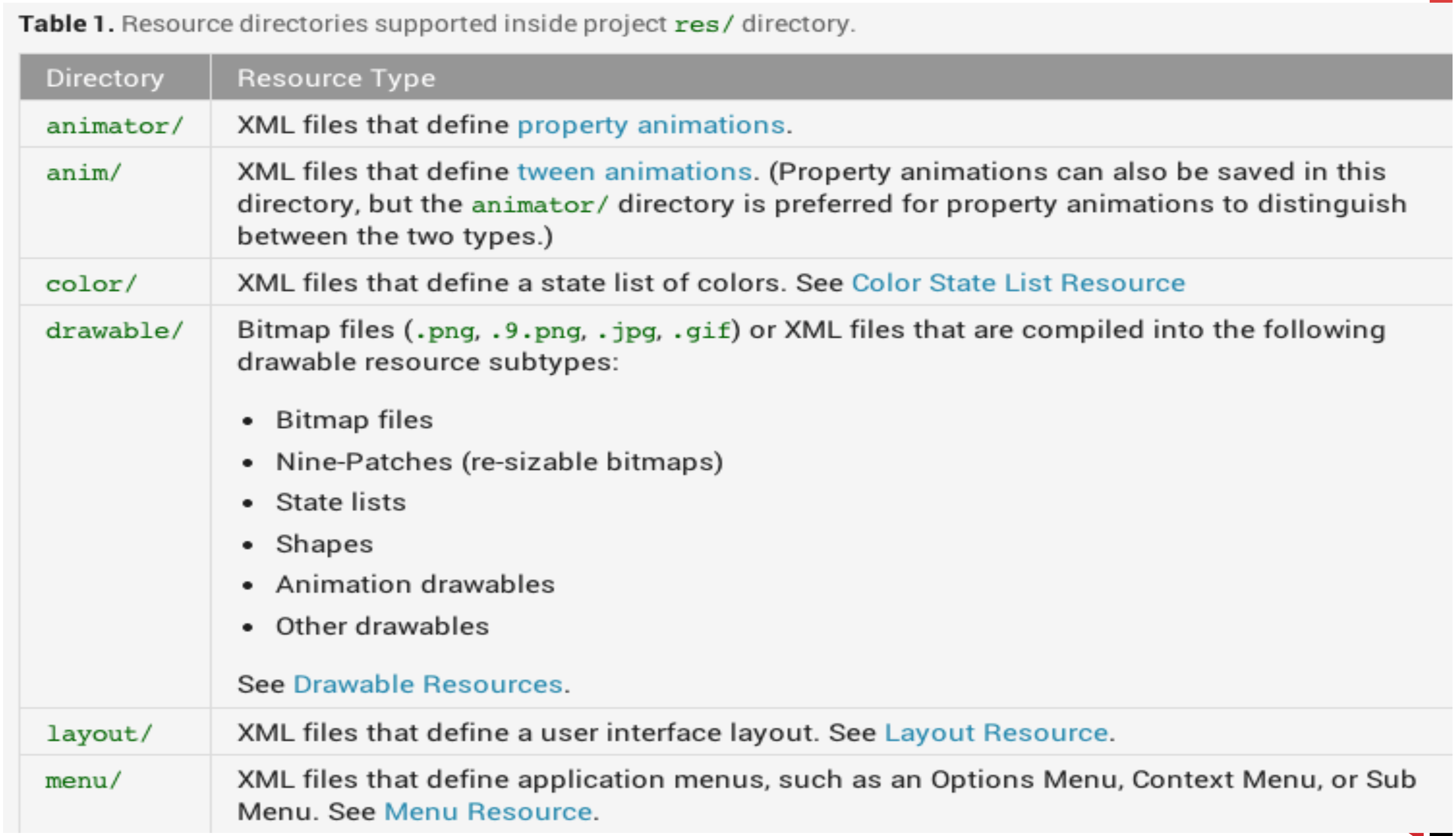
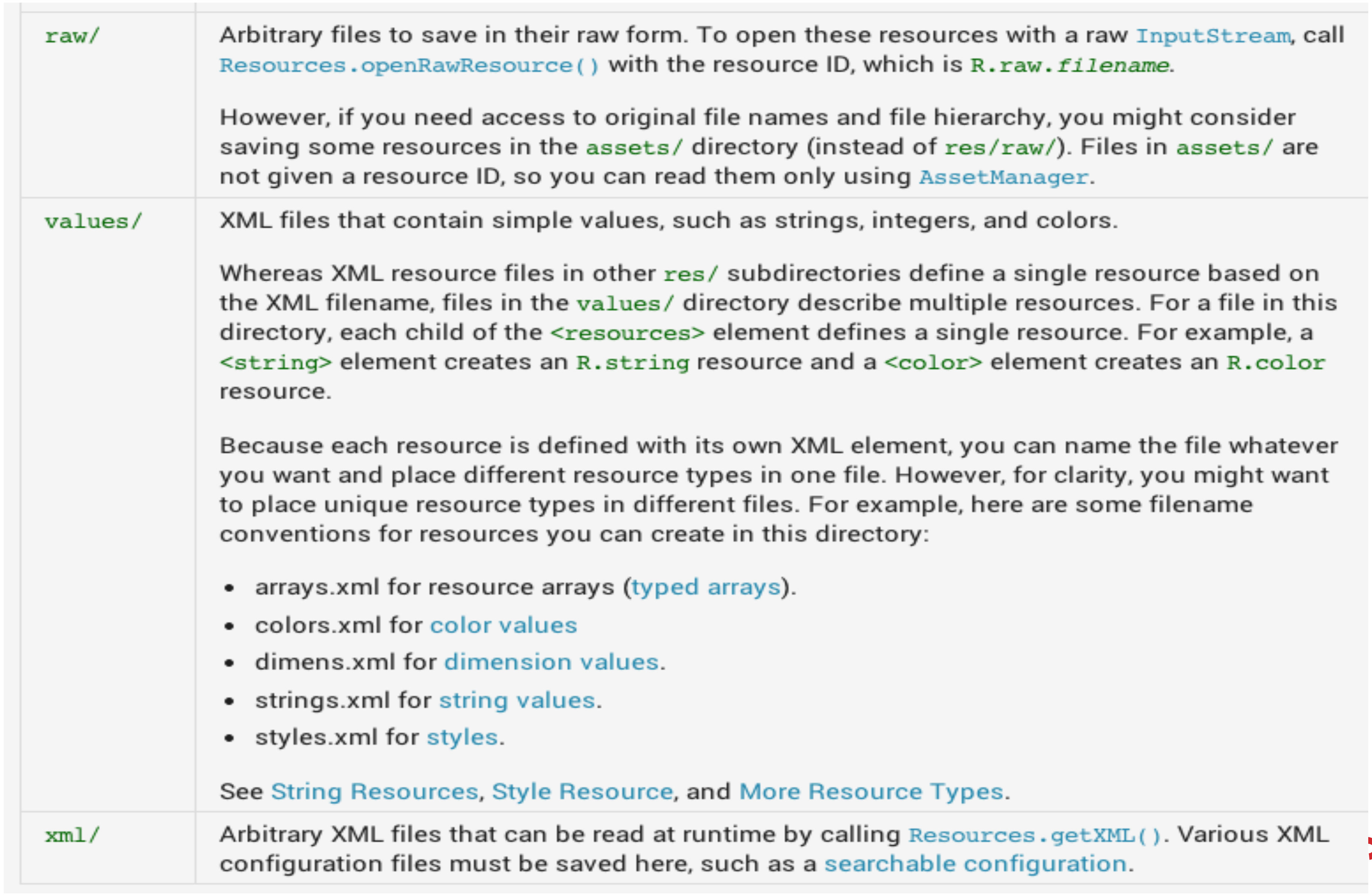


Table 1. Resource directories supported inside project `res/` directory.

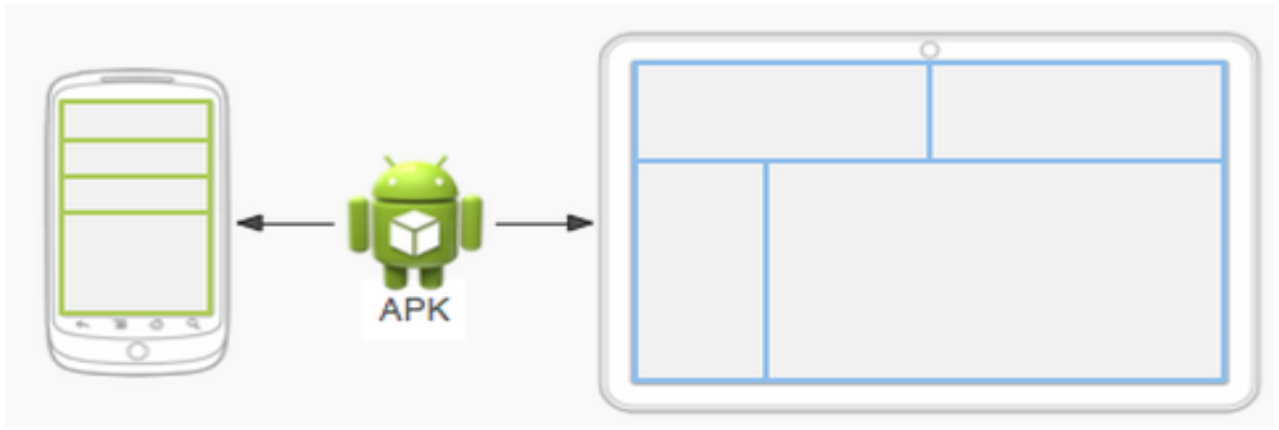
Directory	Resource Type
`animator/`	XML files that define property animations.
`anim/`	XML files that define tween animations. (Property animations can also be saved in this directory, but the `animator/` directory is preferred for property animations to distinguish between the two types.)
`color/`	XML files that define a state list of colors. See Color State List Resource
`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.
`layout/`	XML files that define a user interface layout. See Layout Resource.
`menu/`	XML files that define application menus, such as an Options Menu, Context Menu, or Sub Menu. See Menu Resource.



`raw/`	Arbitrary files to save in their raw form. To open these resources with a raw `InputStream`, call `Resources.openRawResource()` 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`.
`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). • colors.xml for color values • dimens.xml for dimension values. • strings.xml for string values. • styles.xml for styles. See String Resources, Style Resource, and More Resource Types.
`xml/`	Arbitrary XML files that can be read at runtime by calling `Resources.getXML()`. Various XML configuration files must be saved here, such as a searchable configuration.

Providing Alternative Resources

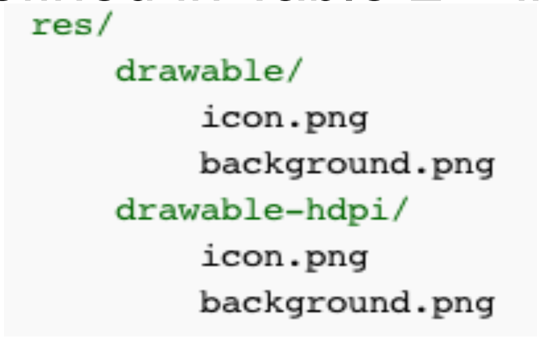
- Alternative resources are used to support specific device configurations. e.g.,
  - alternative drawables for different screen densities
  - alternative strings for different languages
  - device orientation: portrait, landscape
- At runtime, Android detects the current device configuration and loads the appropriate resources for the app.



alternative resource specifier

format: <*resource_name*>**-**<*config_qualifier*>

- **<resource_name>**: the directory name of the corresponding default resources (defined in Table 1 – in the previous slides>
- **<config_qualifier>**: a name that specifies an individual configuration for which these resources are to be used (defined in Table 2 – in the following slides)



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

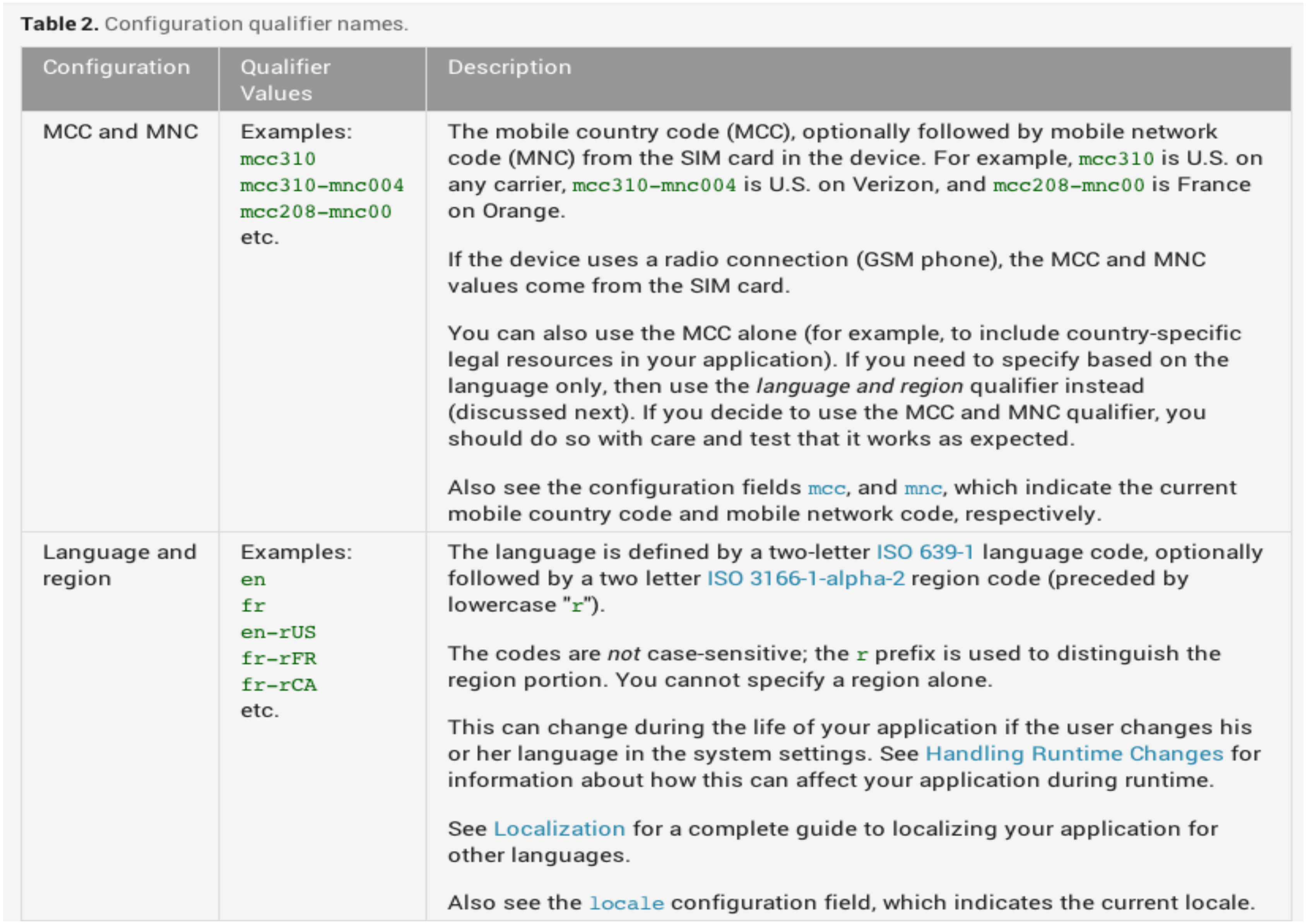
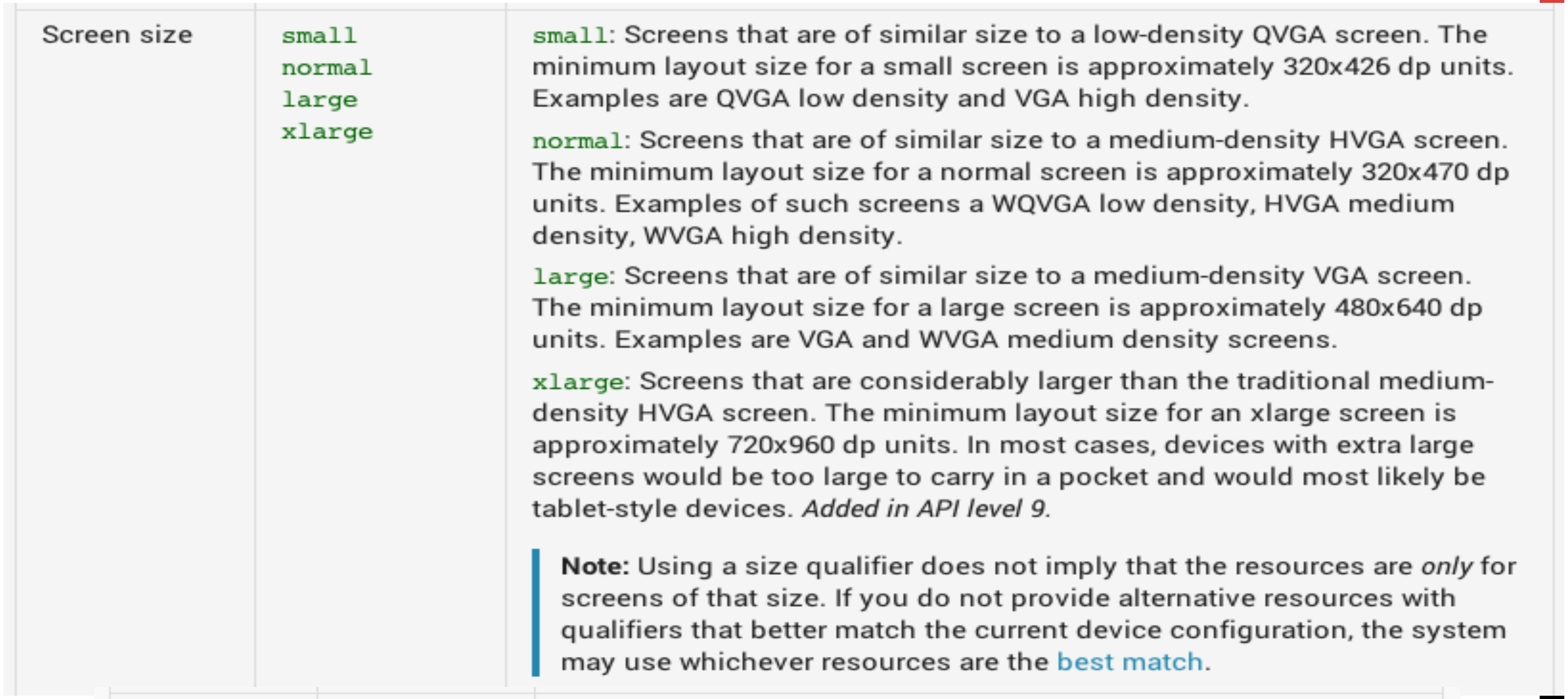
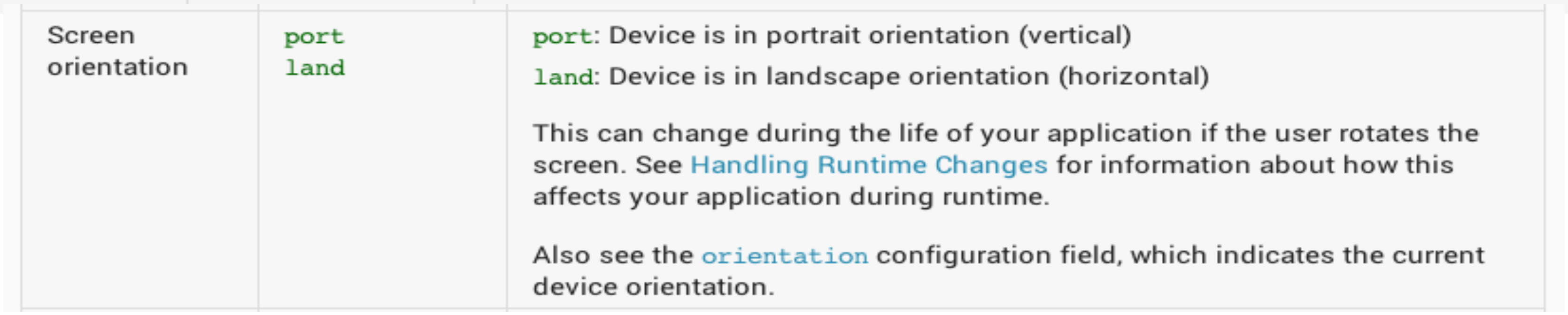


Table 2. Configuration qualifier names.

Configuration	Qualifier Values	Description

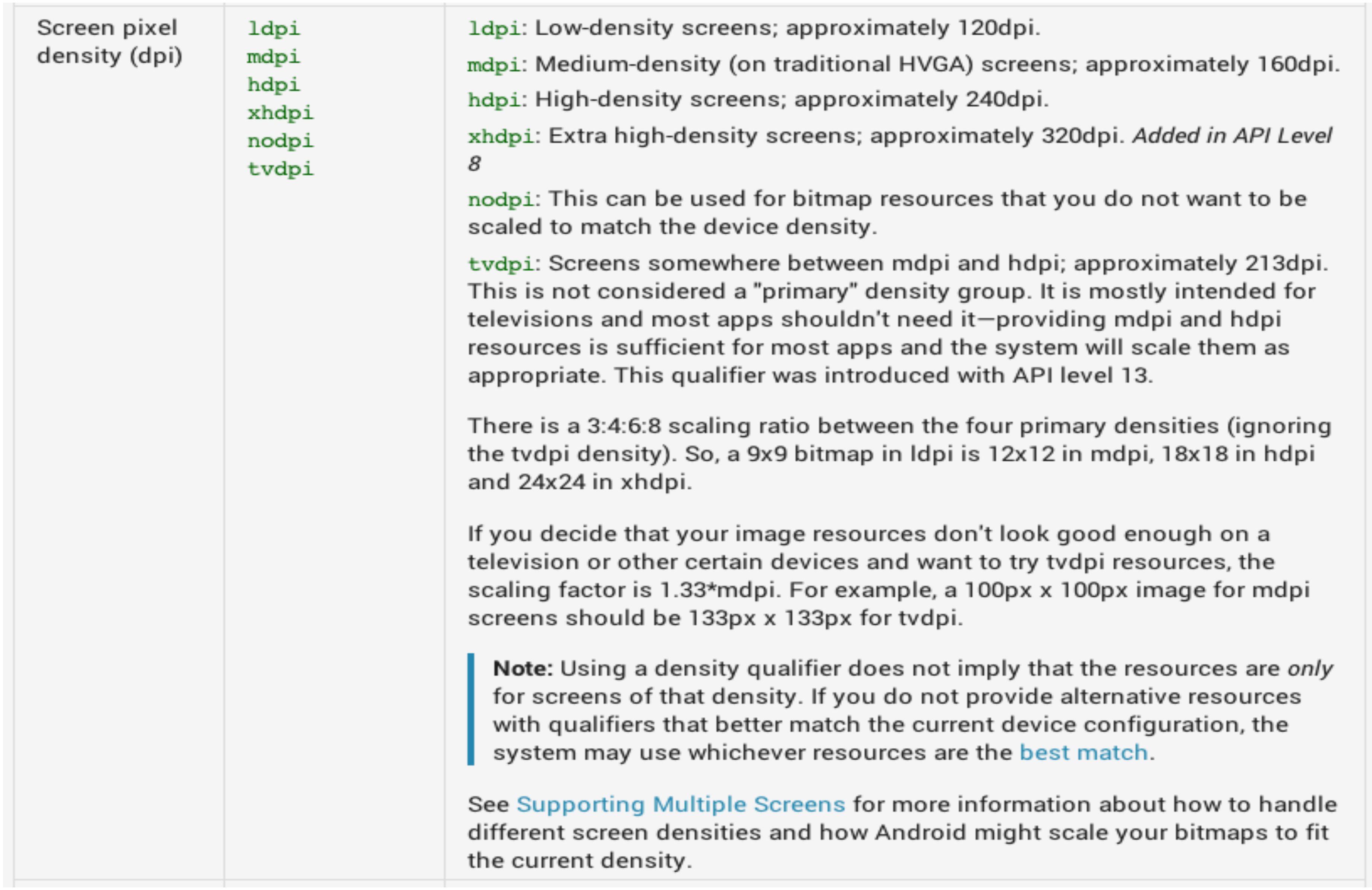
MCC and MNC	Examples:
	`mcc310`
	`mcc310-mnc004`
	`mcc208-mnc00`
	etc.
		The mobile country code (MCC), optionally followed by mobile network code (MNC) from the SIM card in the device. For example, `mcc310` is U.S. on any carrier, `mcc310-mnc004` is U.S. on Verizon, and `mcc208-mnc00` is France on Orange.
		
		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.

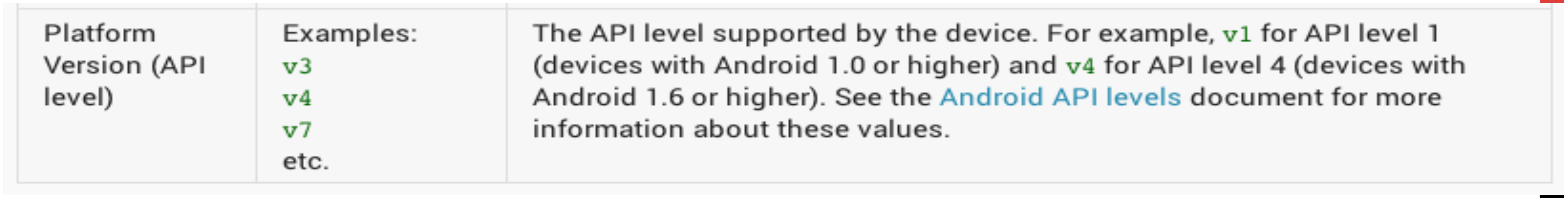



Screen size	`small`
	`normal`
	`large`
	`xlarge`
		`small`: Screens that are of similar size to a low-density QVGA screen. The minimum layout size for a small screen is approximately 320x426 dp units. Examples are QVGA low density and VGA high density.
		`normal`: Screens that are of similar size to 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.

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.



Screen pixel density (dpi)	`ldpi`
	`mdpi`
	`hdpi`
	`xhdpi`
	`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*
		
		`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 scaling ratio between the four primary densities (ignoring the tvdpi density). So, a 9x9 bitmap in ldpi is 12x12 in mdpi, 18x18 in hdpi and 24x24 in xhdpi.
		
		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.



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.

Qualifier name rules

- You can specify multiple qualifiers for a single set of resources, separated by dashes, however, the qualifiers must be in the order listed in Table 2 above.
  - wrong: drawable-hdpi-port/
  - correct: drawable-port-hdpi/
- Qualifier values are case-insensitive

Accessing Resources

```
ImageView imageView = (ImageView) findViewById(R.id.myimageview);
imageView.setImageResource(R.drawable.myimage);
```

```
<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/submit" />
```

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <color name="opaque_red">#f00</color>
   <string name="hello">Hello!</string>
</resources>
```

```
<?xml version="1.0" encoding="utf-8"?>
<EditText xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:textColor="@color/opaque_red"
    android:text="@string/hello" />
```

Device Configurations

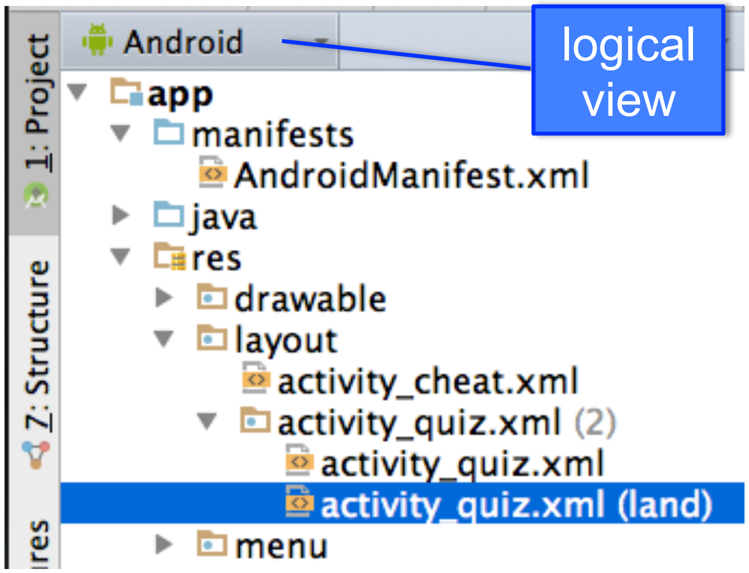
- Rotating the device changes the device configuration.
- Device configuration is a set of characteristics that describe the current state of an individual device.
  - screen orientation, screen density, screen size, keyboard type, dock mode, language, etc
- Applications may provide alternative resources to match different device configurations.
- When runtime configuration change occurs, there may be better matching resources for new configuration.

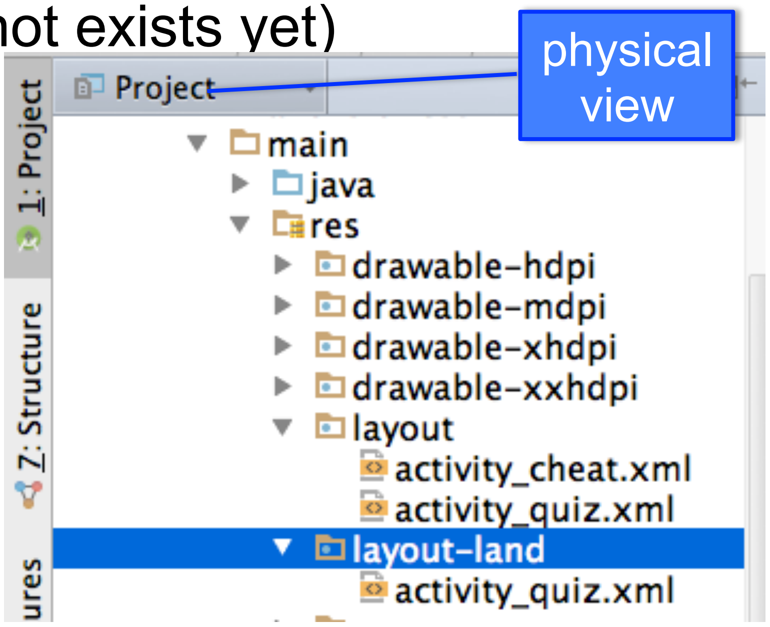
Handling Runtime Changes

- When a runtime change (e.g., screen orientation) occurs, Android restarts the running Activity
  - **onDestroy( )** is called, followed by **onCreate( )**
- To properly handle a restart, the Activity must restore its previous state through the normal Activity lifecycle.
  - Android calls **onSaveInstanceState( )** before it destroys the Activity.
  - You can restore the state during **onCreate( )**

Example: GeoQuiz5_StartActivity

- Screen orientation changes can be handled at runtime by providing layouts accordingly
- Creating a landscape layout
  - On `res` directory: New → Android resource directory
  - Move “orientation” to the Chosen qualifiers section
  - Choose “landscape” as screen orientation
- With the above, Android Studio will create `res/layout-land` folder (if not exists yet)

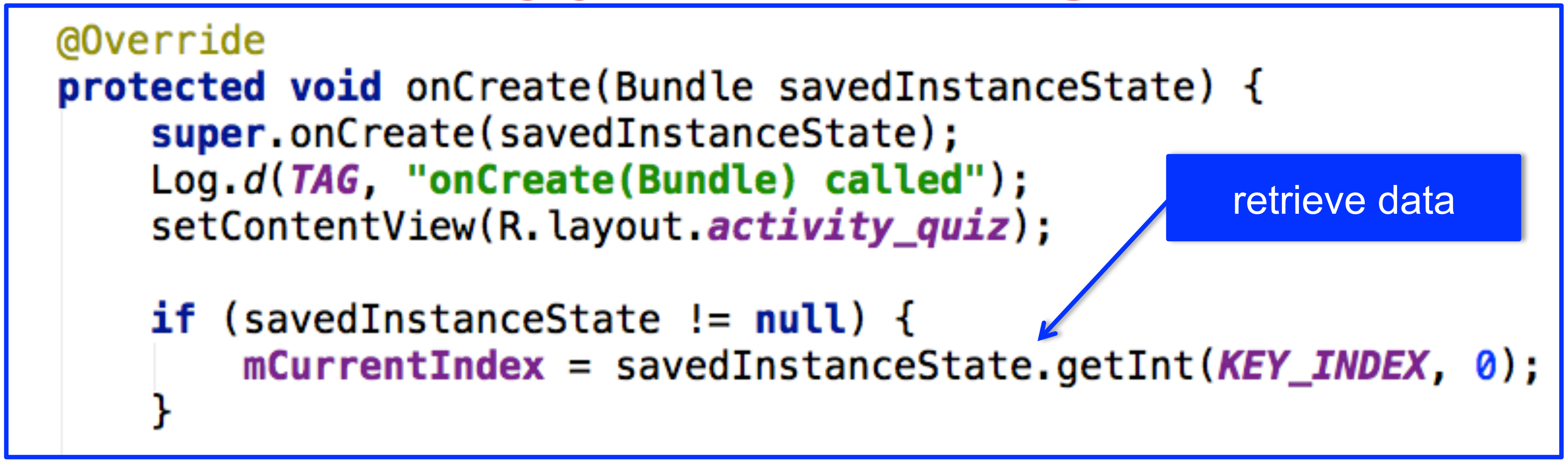




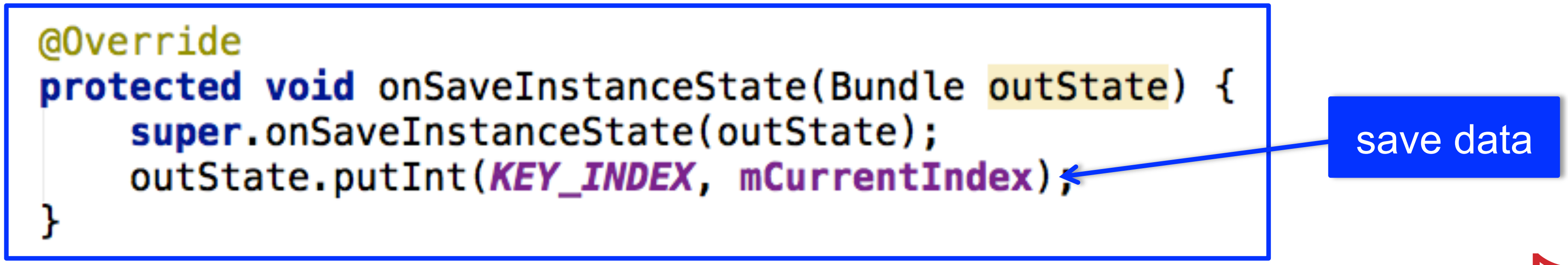
Saving data across rotation

- On screen rotation, activities are destroyed and re-created.
- Thus, after rotation, `QuizActivity` needs to know the old value of `mCurrentIndex`
- protected void `onSaveInstanceState(Bundle outState)` is called by the system before `onPause()`, `onStop()` and `onDestroy()`
  - This method directs all of activity's views to save their state as data in the `Bundle` object.
  - A `Bundle` is a structure that maps string keys to values of certain type

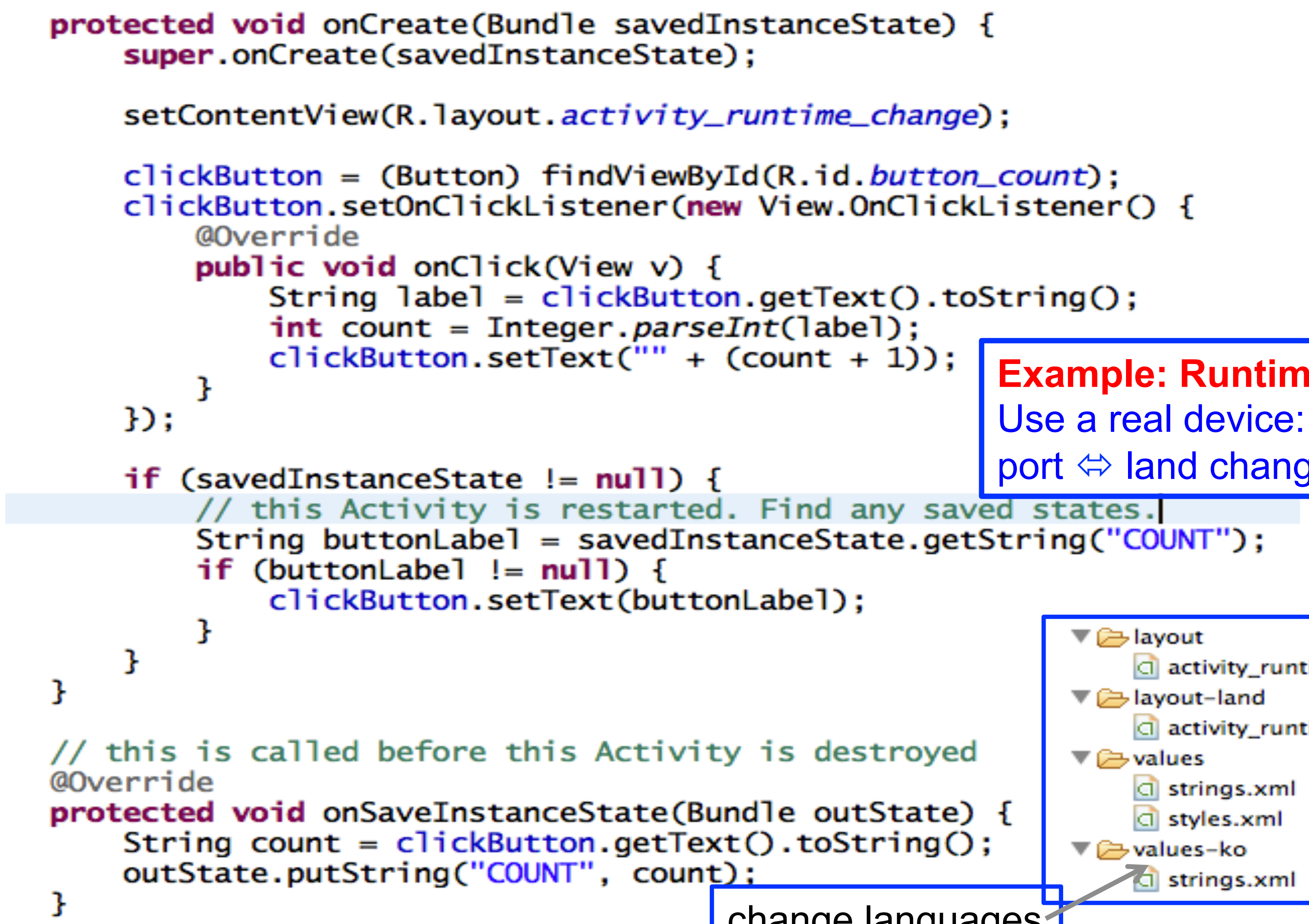
MainActivity.java: handling rotation



```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d(TAG, "onCreate(Bundle) called");
    setContentView(R.layout.activity_quiz);

    if (savedInstanceState != null) {
        mCurrentIndex = savedInstanceState.getInt(KEY_INDEX, 0);
    }
```

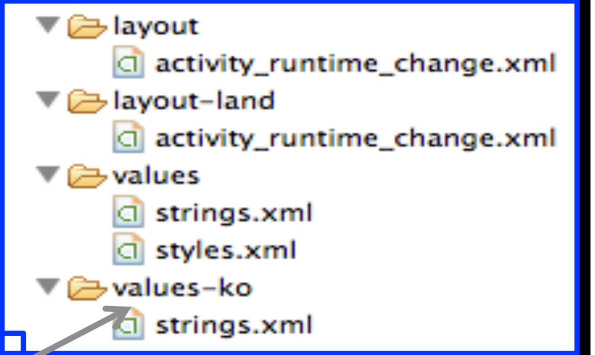


```java
@Override
protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    outState.putInt(KEY_INDEX, mCurrentIndex);
}
```



```java
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity_runtime_change);

    clickButton = (Button) findViewById(R.id.button_count);
    clickButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            String label = clickButton.getText().toString();
            int count = Integer.parseInt(label);
            clickButton.setText("" + (count + 1));
        }
    });

    if (savedInstanceState != null) {
        // this Activity is restarted. Find any saved states.
        String buttonLabel = savedInstanceState.getString("COUNT");
        if (buttonLabel != null) {
            clickButton.setText(buttonLabel);
        }
    }
}

// this is called before this Activity is destroyed
@Override
protected void onSaveInstanceState(Bundle outState) {
    String count = clickButton.getText().toString();
    outState.putString("COUNT", count);
}
```

Example: RuntimeChanges
Use a real device:
port ⇔ land change



```
▼ layout
    activity_runtime_change.xml
▼ layout-land
    activity_runtime_change.xml
▼ values
    strings.xml
    styles.xml
▼ values-ko
    strings.xml
```

change languages

Resource Types

- tween animation: **res/anim**
- drawable: **res/drawable**
- layout: **res/layout**
- menu: **res/menu**
- string: **res/values**
- style: **res/values**
- color: **res/values**