

# CHAPTER 4. **APPLICATION RESOURCES**



What is a **resource**?

**Declaration** of a resource

Resource **type**: *integer, string, array*

Resource **type**: *color, dimension, style*

Resource **type**: *drawable, raw, xml*

OUTLINE **Defining Configuration-specific** resources

Providing the **Best resources** for a device

- An Application is composed of: **code** and **resources**.

**DEF. Resources are everything that is not code** (including: XML layout files, language packs, images, audio/video files, etc)

Utilization of Resources... why?

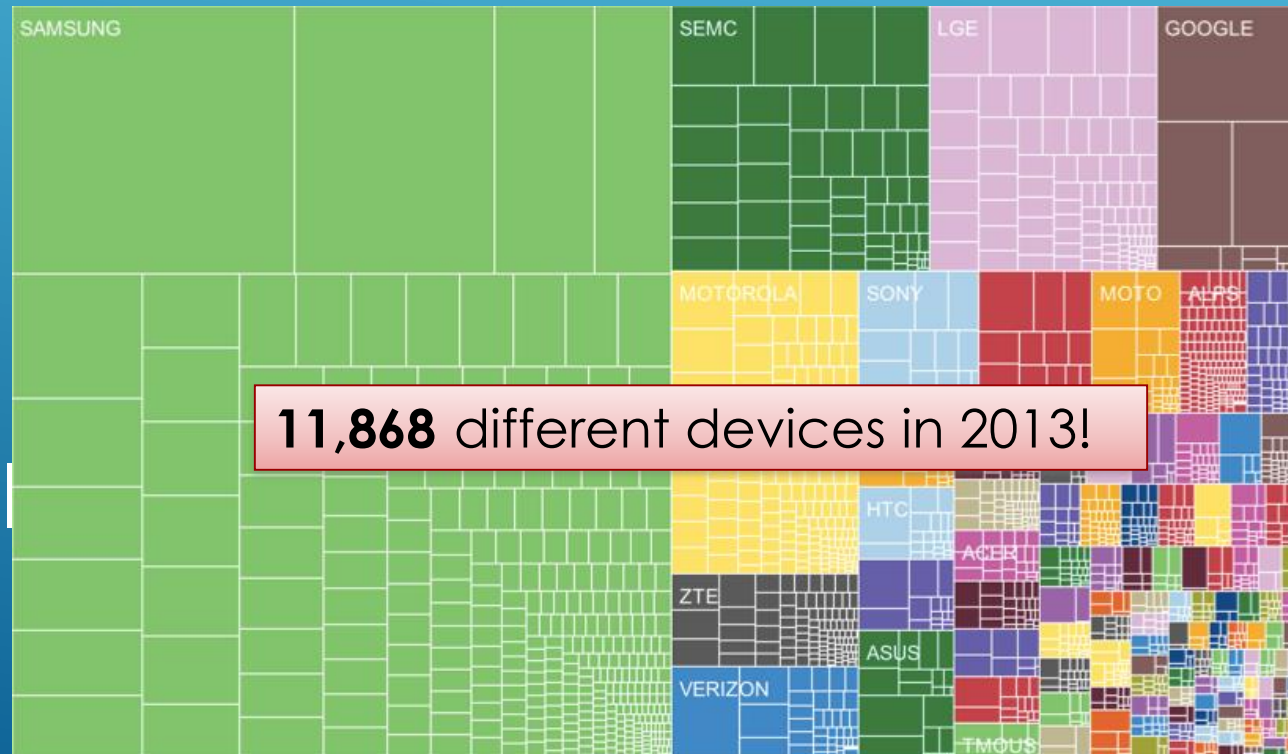
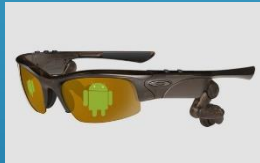
- **Separate** data presentation (layout) from data management

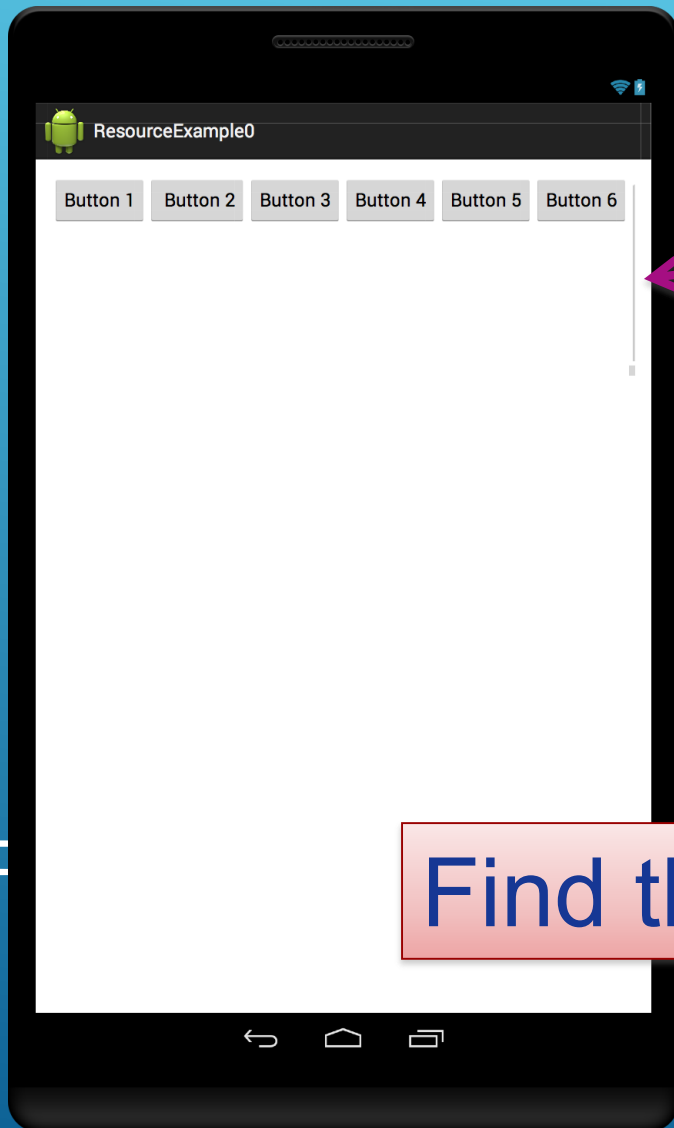
**APPLICATION RESOURCES DEFINITION**

- **Provide** alternative resources to support specific device configurations (e.g. different language packs)

- **Re-compile** only when strictly needed!

**PROBLEM.** An Android application might run on heterogenous devices with different characteristics (e.g. screen size, language support, keyboard type, input devices, etc).





The same **application layout** with 8 buttons, on a **tablet** and on a **smartphone** (Nexus 7) device.



Find the differences ...

AF

**PROBLEM.** An Android application might run on heterogenous devices with different characteristics (e.g. screen size, language support, keyboard type, input devices, etc).

TRADITIONAL SOLUTION. Foresee all the alternatives in Java code

- The code is full of **if-else** cases
- Recompile when need to change layout or add a new language package.

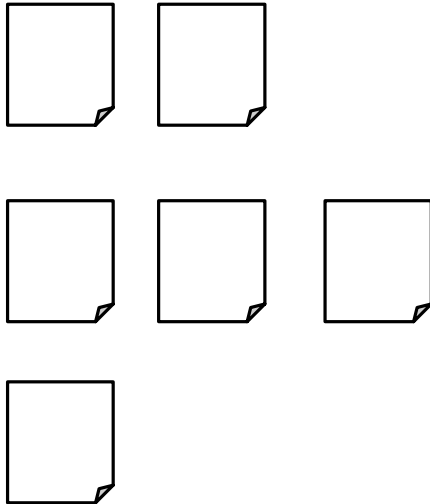
**ANDROID SOLUTION.** Separate code from application resources

- Use declative XML-based approach to define resources (images, files, layout, text, etc)

A



Java App Code



RESOURCES DE

Resources

➤ Use XML files to define  
**(declarative approach):**

- Application Layout
- Text used in the applications
- Application Menu
- Animations
- ...
- Foresee different **resources alternatives** for different device configurations (e.g. screen resolution, language, input devices. etc)

## EXAMPLE



**Device 1**  
HIGH screen pixel density



**Device 2**  
LOW screen pixel density

Java App Code

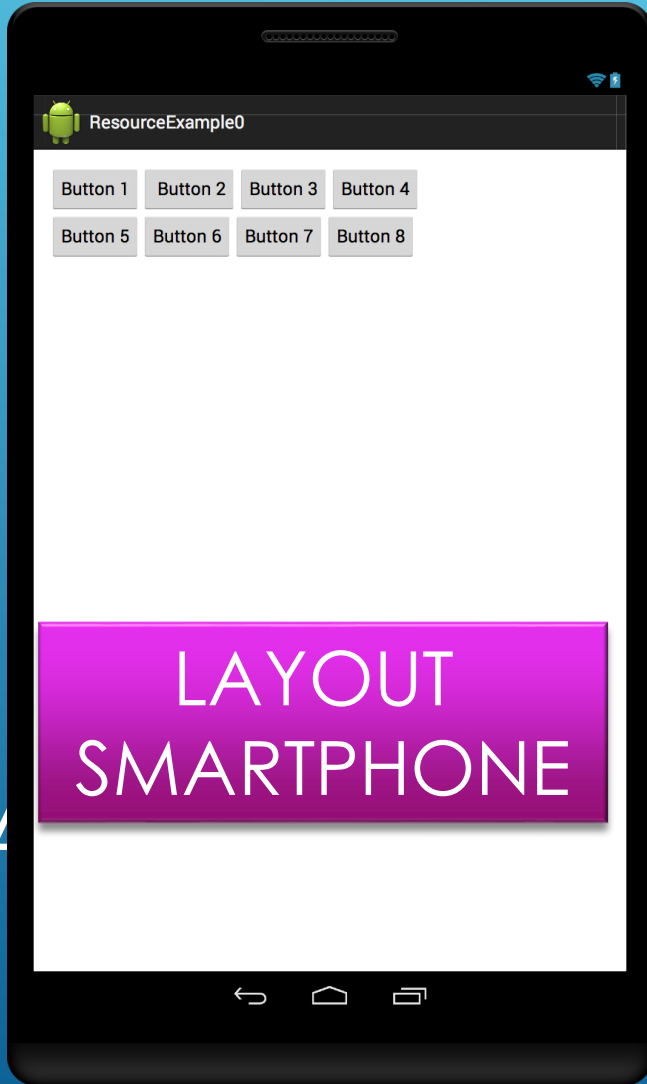
APPLICATION RESOURCES

XML Layout File  
Device 1

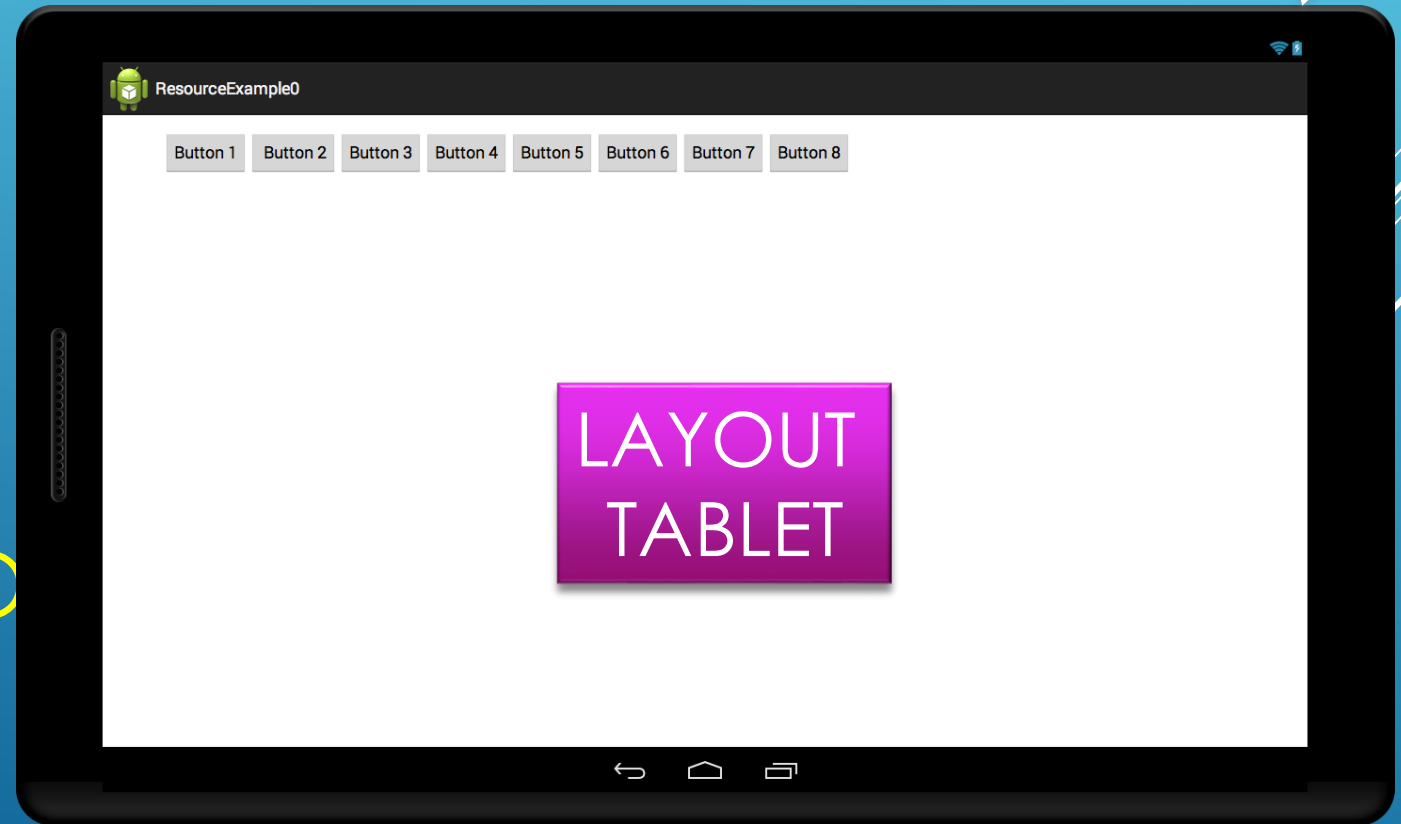
XML Layout File  
Device 2

- Build the **application layout** through XML files (like HTML)
- Define **two** different XML **layouts** for two different devices
- At **runtime**, Android detects the current device configuration and loads the appropriate resources for the application
- **No need to recompile!**
- Just add a new XML file if you need to support a new device

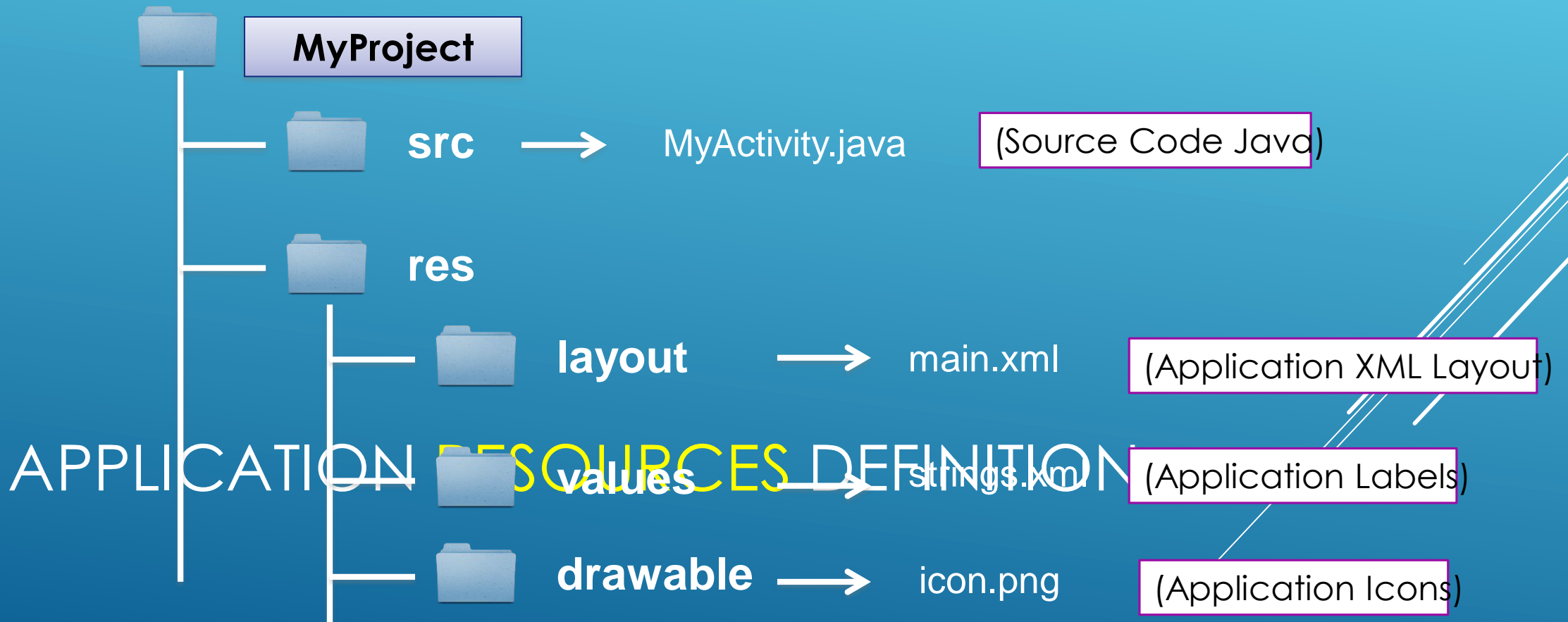




The same **application layout** with 8 buttons, on a **tablet** and on a **smartphone** (Nexus 7) device.



- ❖ Resources are defined in the **res/** folder of the project.



Resource Type	Resource contained
<b>res/animator</b>	<i>XML files that define property animations.</i>
<b>res/anim</b>	<i>XML files that define tween animations.</i>
<b>res/color</b>	<i>XML files that define a state list of colors.</i>
<b>res/drawable</b>	<i>Bitmap files (.png, .9.png, .jpg, .gif) or XML files that are compiled into other resources.</i>
<b>res/layout</b>	<i>XML files that define a user interface layout.</i>
<b>res/menu</b>	<i>XML files that define application menus.</i>
<b>res/raw</b>	<i>Arbitrary files to save in their raw form.</i>
<b>res/values</b>	<i>XML files that contain simple values, such as strings, integers, array.</i>
<b>res/xml</b>	<i>Arbitrary XML files.</i>

# APPLICATION RESOURCES DEFINITION

- Resources are defined in a **declarative** way through **XML**.
- Each resource has a name/identifier (see details later).

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

  <string name="hello"> Hello world! </string>
  <string name="labelButton"> Insert your username </string>

</resources>
```

**Resource type**  
(string)

- Resource can be accessed in the **Java** code through the **R class**, that works as a **glue** between the world of java and the world of resources.
- **Automatically generated** file, no need to modify it.
- **Recreated** in case of changes in the **res/** directory.

## APPLICATION RESOURCES DEFINITION

```
public final class R {  
    public static final class string {  
        public static final int hello=0x7f040001;  
        public static final int label1=0x7f040005;  
    }  
}
```

**R** contains **resource IDs** for all the resources in the **res/** directory.

- Resources can be accessed from Java code by using the **R** class and methods of the **Activity** class (details later).
- We just need to know the **resource Identifier** (ID) ... how to know it? (see next slides)

## APPLICATION RESOURCES DEFINITION

```
...  
final String hello=getResources().getString(R.string.hello);  
final String label=getResources().getString(R.string.labelButton);  
Log.i(STRING_TAG," String1 " + hello);  
Log.i(STRING_TAG," String2 " + label);  
...  
...
```

# APPLICATION RESOURCES DEFINITION

**STEP0: Declare** resources in res/

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

  <string name="hello"> Hello </string>
  <string name="label1"> Label </string>
</resources>
```

**XML-Based, Declarative Approach**

**STEP2: Access** resources through R class

```
public final class R {

    public static final class string {
        public static final int hello=0x7f040001;
        public static final int label1=0x7f040005;
    }

}
```

**Java Code, Programmatic Approach**

**STEP1: Compile** the project

# ACCESS TO APPLICATION RESOURCES

- Each Resource is associated with an **Identifier** (ID), that is composed of two parts:
  - The resource **type**: Each resource is grouped into a "type," (e.g. string, color, menu, drawable, layout, etc)
  - The resource **name**, which is either: the filename, excluding the extension; or the value in the XML `<android:name>` attribute.
  - Identifiers must be unique!!
- Two ways to access resources:
  - From the **Java** Code
  - From the **XML** files



# ACCESS TO APPLICATION RESOURCES: XML

**@[<package\_name>:]<resource\_type>/<resource\_name>**  
**>**

- **<package\_name>** is the name of the package in which the resource is located (not required when referencing resources from the same package)
- **<resource\_type>** is the the name of the resource type
- **<resource\_name>** is either the resource filename without the extension or the android:name attribute value in the XML element.

# ACCESS TO APPLICATION RESOURCES: XML

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

STRING.XML

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <Textview android:id="@+id/label1" android:text="@string/labelText"
android:textcolor="@color/opaque_red">
    </Textview>
  <Button android:id="@+id/button1" android:text="@string/labelButton">
    </Button>
</resources>
```

MAIN.XML

# ACCESS TO APPLICATION RESOURCES: JAVA

**[<package\_name>.]**R**.<resource\_type>.<resource\_name>**

- **<package\_name>** is the name of the package in which the resource is located (not required when referencing resources from the same package)
- **<resource\_type>** is the **R** subclass for the resource type
- **<resource\_name>** is either the resource filename without the extension or the android:name attribute value in the XML element.

# ACCESS TO APPLICATION RESOURCES: JAVA

```
// Get a string resource from the string.xml file  
final String hello=getResources().getString(R.string.hello);  
  
// Get a color resource from the string.xml file  
final int color=getResources().getColor(R.color.opaque_red);  
  
// Load a custom layout for the current screen  
setContentView(R.layout.main_screen);  
  
// Set the text on a TextView object using a resource ID  
TextView msgTextView = (TextView) findViewById(R.id.label1);  
msgTextView.setText(R.string.labelText);
```

# RESOURCES TYPES: STRING AND ARRAY

Resource Type	File	Java constant	XML tag	Description
<b>string</b>	Any file in the res/values/	R.string.<key>	<string>	<b>String</b> value associated to a key.
<b>integer</b>	Any file in the res/values/	R.integer.<key>	<integer>	<b>Integer</b> value associated to a key.
<b>array</b>	Any file in the res/values/	R.array.<key>	<string-array> <item> <item> </string-array>	<b>Array of strings.</b> Each element is a described by an <item>
<b>array</b>	Any file in the res/values/	R.array.<key>	<integer-array> <item> <item> </integer-array>	<b>Array of integers.</b> Each element is a described by an <item>

# RESOURCES TYPES: STRING AND ARRAY

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<resources>
```

MYVALUES.XML

```
    <string name="app_title"> Example Application </string>
```

```
    <string name="label" > Hello world!  </string>
```

```
    <integer name="val" > 53 </integer>
```

```
    <string-array name="nameArray">
```

```
        <item> John </item>
```

```
        <item> Michael </item>
```

```
    </string-array>
```

```
    <integer-array name="valArray">
```

```
        <item> 1 </item>
```

```
        <item> 2 </item>
```

```
    </integer-array>
```

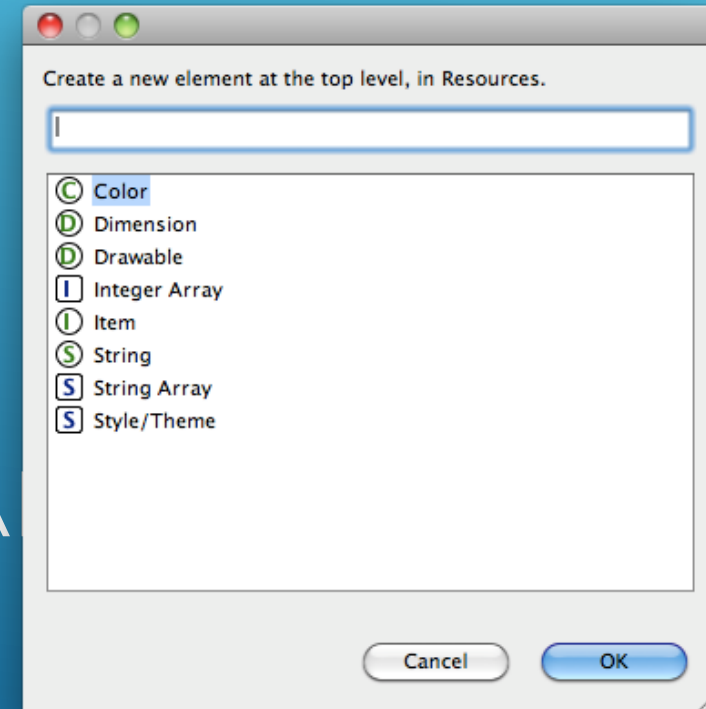
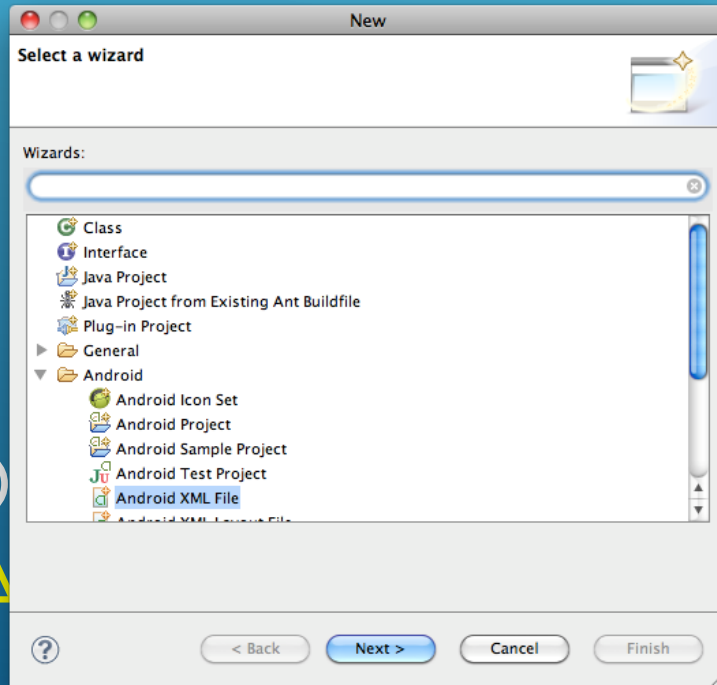
```
</resources>
```

```
// Access the string value
final String hello=getResources().getString(R.string.app_title);

// Access the string-array values
final String[] nameS=getResources().getStringArray
                    (R.array.nameArray);

// Access the integer-array values
final int[] val=getResources().getIntArray(R.array.valArray);
```

- Resources can be defined in the **res/string.xml** or in any **other file** defined by the users (File → New → Android XML File)

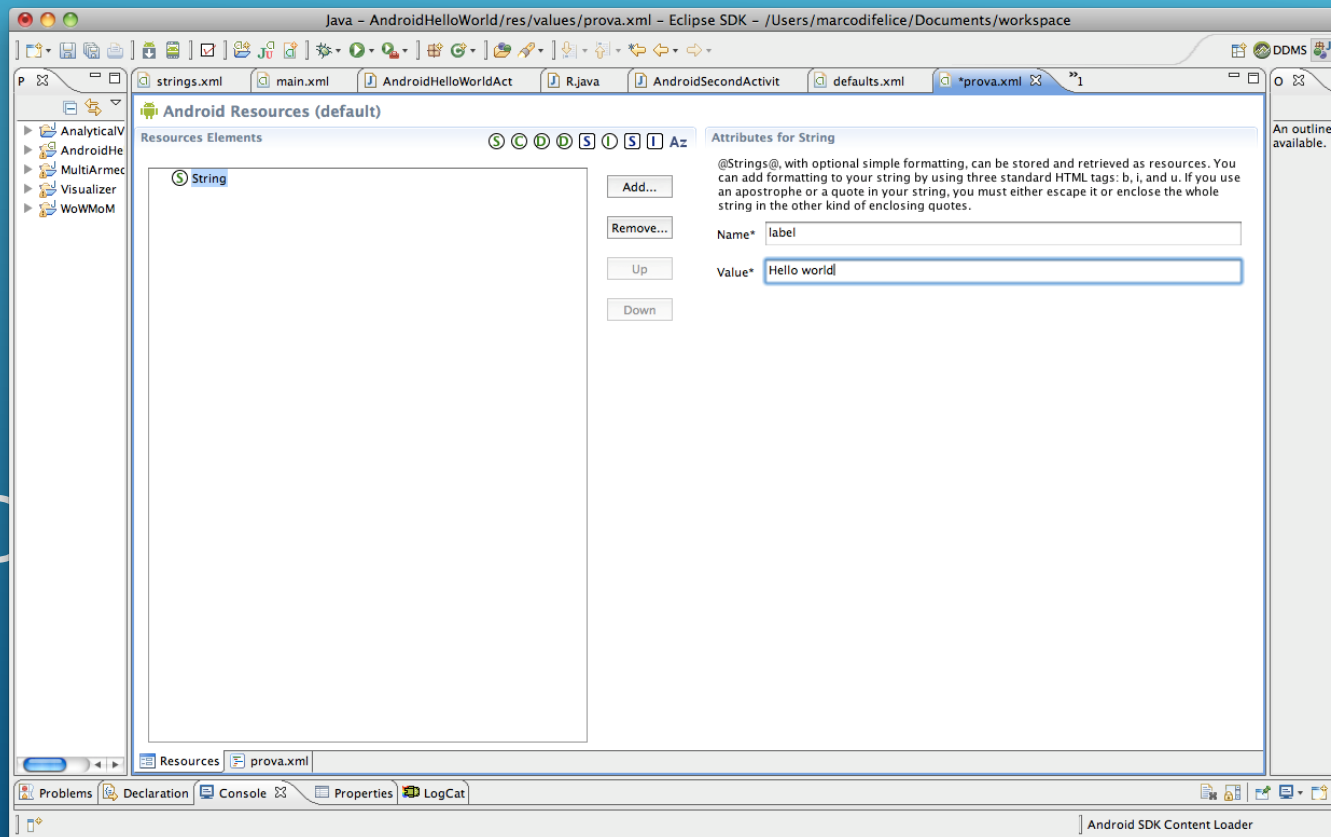


RESO  
ARRA

RING A



- Android XML Files can be edited by hand or through the **Eclipse plugin** (recommended).



RESOURCE  
ARRAY

# OTHER RESOURCES TYPES

➤ Some other resources types (we will meet later ...)

Resource Type	File	Java constant	XML tag	Description
<b>layout</b>	Any file in the res/layout/	R.layout.<key>	<layout>	Defines a layout of the screen
<b>animation</b>	Any file in the res/animator/	R.animator.<key>	<animator>	Defines a property animation (not the only method!)
<b>menu</b>	Any file in the res/menu/	R.menu.<key>	<menu>	User-defined menus with multiple options

# RESOURCES TYPES: COLOR, DIMENSION, STYLE

Resource Type	File	Java constant	XML tag	Description
<b>color</b>	Any file in the res/values/	R.color.<key>	<color>	Definition of <b>colors</b> used in the GUI
<b>dimension</b>	Any file in the res/values/	R.dimen.<key>	<dimen>	Dimension <b>units</b> of the GUI components
<b>style/theme</b>	Any file in the res/values/	R.style.<key>	<style>	<b>Themes</b> and <b>styles</b> used by applications or by components

# RESOURCES TYPES: COLOR, DIMENSION, STYLE

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<resources>
```

```
    <color name="red"> #FF0000 </color>
```

```
    <color name="red_transparent" > #66DDCCDD</color>
```

```
</resources>
```

STYLES.XML

- Color values can be defined based on one of these syntax rules: **#RGB**, **#ARGB**, **#RRGGBB**, **#AARRGGBB** (R=*red*, G=*green*, B=*blue*, A=*transparency*).

- From Java code:

```
int redTransparent=getResources.getColor(R.color.red_transparent)
```

# RESOURCES TYPES: COLOR, DIMENSION, STYLE

Code	Description
<b>px</b>	Pixel units
<b>in</b>	Inch units
<b>mm</b>	Millimeter units
<b>pt</b>	Points of 1/72 inch
<b>dp</b>	Abstract unit, independent from pixel density of a display
<b>sp</b>	Abstract unit, independent from pixel density of a display (font)

These units are relative to a 160 dpi (dots per inch) screen, on which 1dp is roughly equal to 1px. When running on a higher density screen, the number of pixels used to draw 1dp is scaled up by a factor appropriate for the screen's dpi. Likewise, when on a lower density screen, the number of pixels used for 1dp is scaled down

# RESOURCES TYPES: COLOR, DIMENSION, STYLE

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
```

MYVALUES.XML

```
    <dimen name="textview_height">25dp</dimen>
    <dimen name="textview_width">150dp</dimen>
    <dimen name="font_size">16sp</dimen>
```

```
</resources>
```

➤ **Applying dimensions** to attributes in the XML layout:

```
<TextView
    android:layout_height="@dimen/textview_height"
    android:layout_width="@dimen/textview_width"
    android:textSize="@dimen/font_size"/>
```

MAIN.XML

# RESOURCES TYPES: COLOR, DIMENSION, STYLE

```
<?xml version="1.0" encoding="utf-8"?>  
<resources>
```

MYVALUES.XML

```
    <dimen name="textview_height">25dp</dimen>  
    <dimen name="textview_width">150dp</dimen>  
    <dimen name="font_size">16sp</dimen>
```

```
</resources>
```

➤ **Applying dimensions** to attributes in the XML layout:

```
<TextView  
    android:layout_height="@dimen/textview_height"  
    android:layout_width="@dimen/textview_width"  
    android:textSize="@dimen/font_size"/>
```

MAIN.XML

# RESOURCES TYPES: COLOR, DIMENSION, STYLE

- A **Style** is a set of **attributes** that can be applied to a specific component of the GUI (View) or to the whole screen or application (in this case, it is also referred as “theme”).
- A style is an XML resource that is referenced using the value provided in the **name** attribute.
- Styles can be organized in a **hierarchical** structure. A style can inherit properties from another style, through the **parent** attribute.
- Use **<style></style>** tags to define a style in the **res/** folder. Use **<item>** to define the attributes of the style.



# RESOURCES TYPES: COLOR, DIMENSION, STYLE

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="CustomText" parent="@style/Text">
        <item name="android:textSize">20sp</item>
        <item name="android:textColor">#008</item>
    </style>
</resources>
```

MYVALUES.XML

➤ Applying a style to a View in the XML layout:

```
<EditText style="@style/CustomText"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Hello, World!" />
```

MAIN.XML

# RESOURCES TYPES: DRAWABLE

Resource Type	File	Java constant	XML tag	Description
<b>drawable</b>	Any file in the res/drawable/	R.drawable.<key>	<drawable>	Images and everything that can be drawn

A **Drawable** resource is a general concept for a graphic that can be drawn on the screen:

- Images
- XML resources with attributes such as **android:drawable** and **android:icon** (e.g. a Button can have a drawable resource as background)

Complete list of drawable resource type can be found here:

<http://developer.android.com/guide/topics/resources/drawable-resource.html>

## RESOURCES TYPES: DRAWABLE

- A **Bitmap** file is a **.png**, **.jpg** or a **.gif** file.
- Android creates a **Bitmap** resource for any of these files saved in the **res/drawable** directory.

This layout XML applies the file `myimage.png` saved in `res/drawable` to a **View**.

```
<ImageView  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:src="drawable/myimage" />
```

Retrieve the image as a Drawable from Java:

```
Drawable draw=res.getDrawable(R.drawable.myimage);
```

# RESOURCES TYPES: DRAWABLE

- An **XMLBitmap** is an XML resource that **points to a bitmap file**.
- Usage: (i) **Alias** to the raw bitmap file, (ii) Specify additional properties such as **dithering** and **tiling**.

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

Some properties of an XMLBitmap:

**android:src, android:antialias, android:dither, android:filter, android:gravity**

# RESOURCES TYPES: DRAWABLE

Drawable type	Description
<b>BitMap File</b>	A bitMap Graphic file (.png, .gif, .jpeg)
<b>Nine-Patch File</b>	A PNG file with stretchable regions to allow resizing
<b>Layer List</b>	A Drawable managing an array of other drawable
<b>State List</b>	A Drawable that references different graphics based on the states
<b>Level List</b>	An XML managing alternate Drawables. Each assigned a value
<b>Transition</b>	A Drawable that can cross-fade between two Drawable
<b>Inset</b>	A Drawable that insets another Drawable by a specific distance
<b>Clip</b>	A Drawable that clips another Drawable based on its current level
<b>Scale</b>	A Drawable that changes the size of another Drawable
<b>Shape</b>	An XML file that defines a geometric shape, colors and gradients

Complete list of drawable resource type can be found here:

<http://developer.android.com/guide/topics/resources/drawable-resource.html>

# RESOURCES TYPES: XML AND RAW

Resource Type	File	Java constant	XML tag	Description
xml	Any file in the res/xml/	R.xml.<key>	<xml>	User-specific XML file with name equal to key
raw	Any file in the res/raw/	R.raw.<key>	<raw>	Raw resources, accessible through the R class but not optimized

Used to define resources for which no run-time optimization must be performed (e.g. audio/video files). They can be accessed as a stream of bytes, by using Java **InputStream** objects:

```
InputStream is= getResources().openRawResource(R.raw.videoFile)
```

## RESOURCES TYPES: XML AND RAW

- The res/xml folder might contain arbitrary XML files that can be read at runtime through the `R.xml.<filename>` constant.
- It is possible to parse the XML file through a **XMLResourceParser** object, that implements an XML parser:

```
XMLResourceParser parser=getResources().getXML(R.xml.myfile)
```

```
<?xml version="1.0" encoding="utf-8"?>
<names>
    <name code="1234">Marco Di Felice </item>
    <name code="4324">Luca Bedogni </item>
</names>
```

# RESOURCES **ALTERNATIVES**

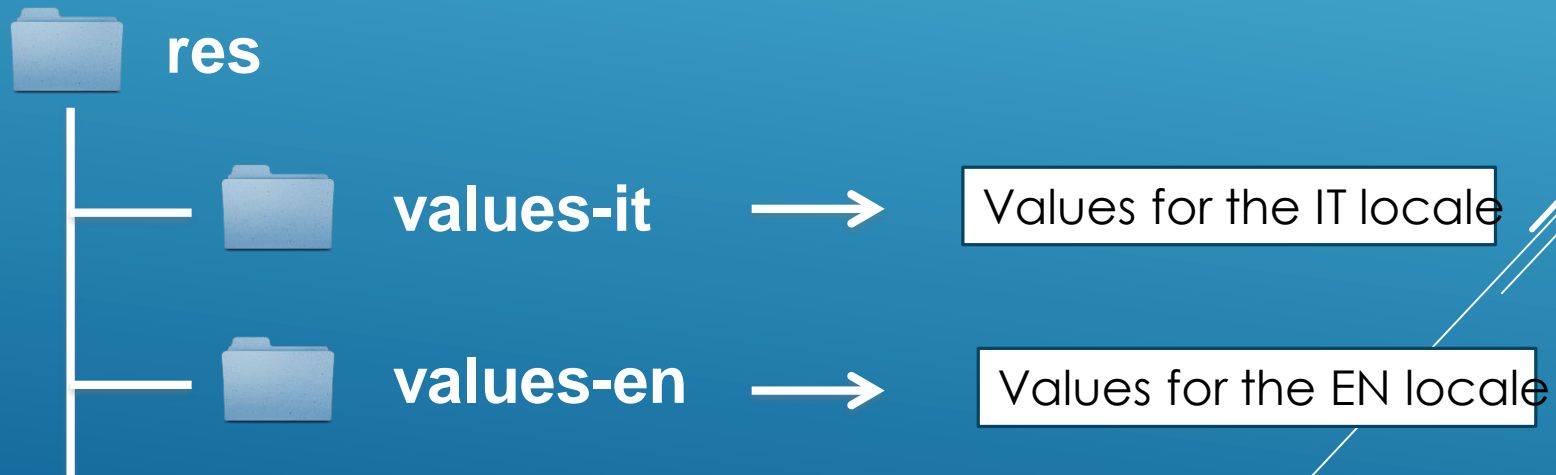
- Android applications might provide **alternative resources** to support specific device configurations (e.g. different languages).
- At runtime, Android **detects** the current device configuration and **loads** the appropriate resources for the application.
- To specify configuration-specific alternatives:
  1. Create a new directory in **res/** named in the form **<resources\_name>-<config\_qualifier>**
  2. Save the respective alternative resources in this new directory



# RESOURCES **ALTERNATIVES**

Name of the folder: **<resources\_name>-<config\_qualifier>**.

- *<resources\_name>* is the directory name of the corresponding default resources (see previous slides).
- *<qualifier>* is a name that specifies an individual configuration for which these resources are to be used (see next slide).



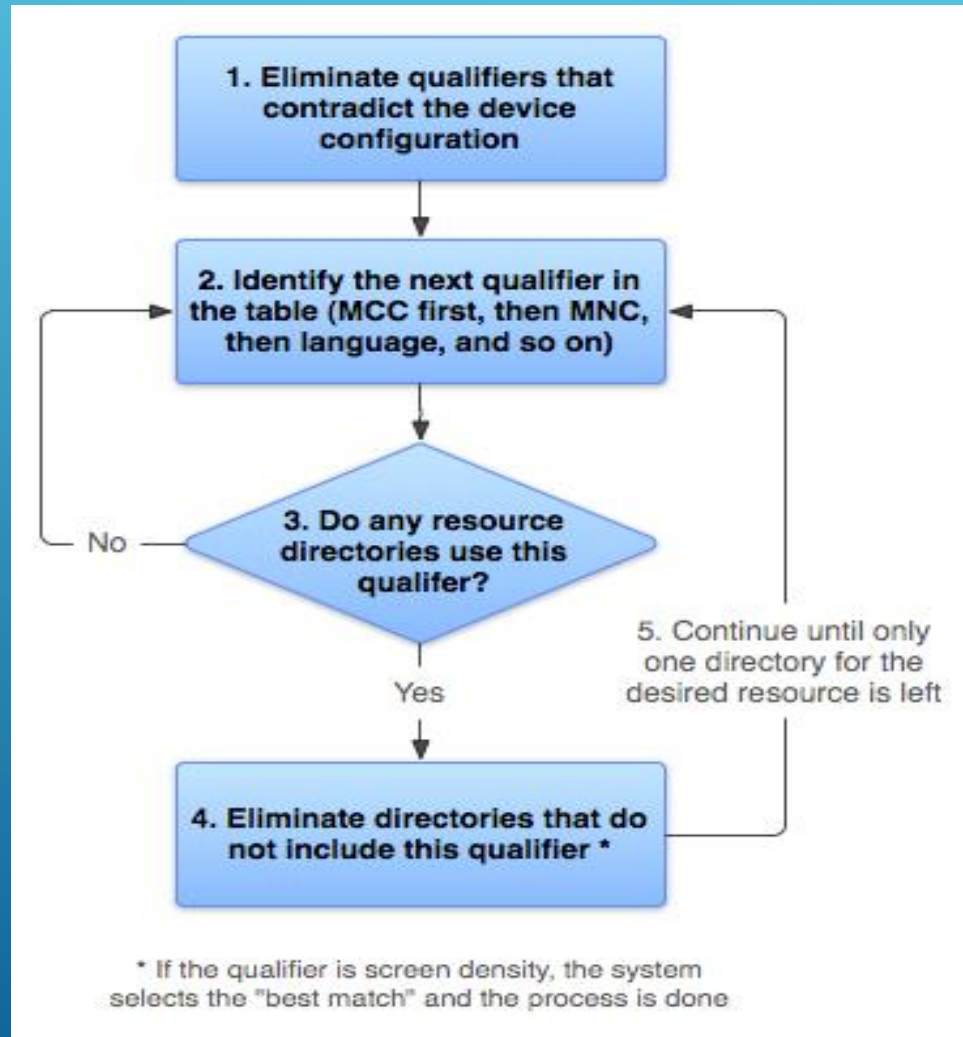
# RESOURCES **ALTERNATIVES:** QUALIFIERS

Configuration	Values Example	Description
<b>MCC and MNC</b>	mcc310, mcc208, etc	mobile country code (MCC)
<b>Language and region</b>	en, fr, en-rUS, etc	ISO 639-1 language code
<b>smallestWidth</b>	sw320dp, etc	shortest dimension of screen
<b>Available width</b>	w720dp, w320dp, etc	minimum available screen width
<b>Available height</b>	h720dp, etc	minimum available screen height
<b>Screen size</b>	small, normal, large	screen size expressed in dp
<b>Screen aspect</b>	long, notlong	aspect ratio of the screen
<b>Screen orientation</b>	port, land	screen orientation (can change!)
<b>Screen pixel density (dpi)</b>	ldpi, mdpi, hdpi	screen pixel density
<b>Keyboard availability</b>	keysexposed, etc	type of keyboard
<b>Primary text input method</b>	nokeys, qwerty	availability of qwerty keyboard
<b>Navigation key availability</b>	navexposed, etc	navigation keys of the application
<b>Platform Version (API level)</b>	v3, v4, v7, etc	API supported by the device

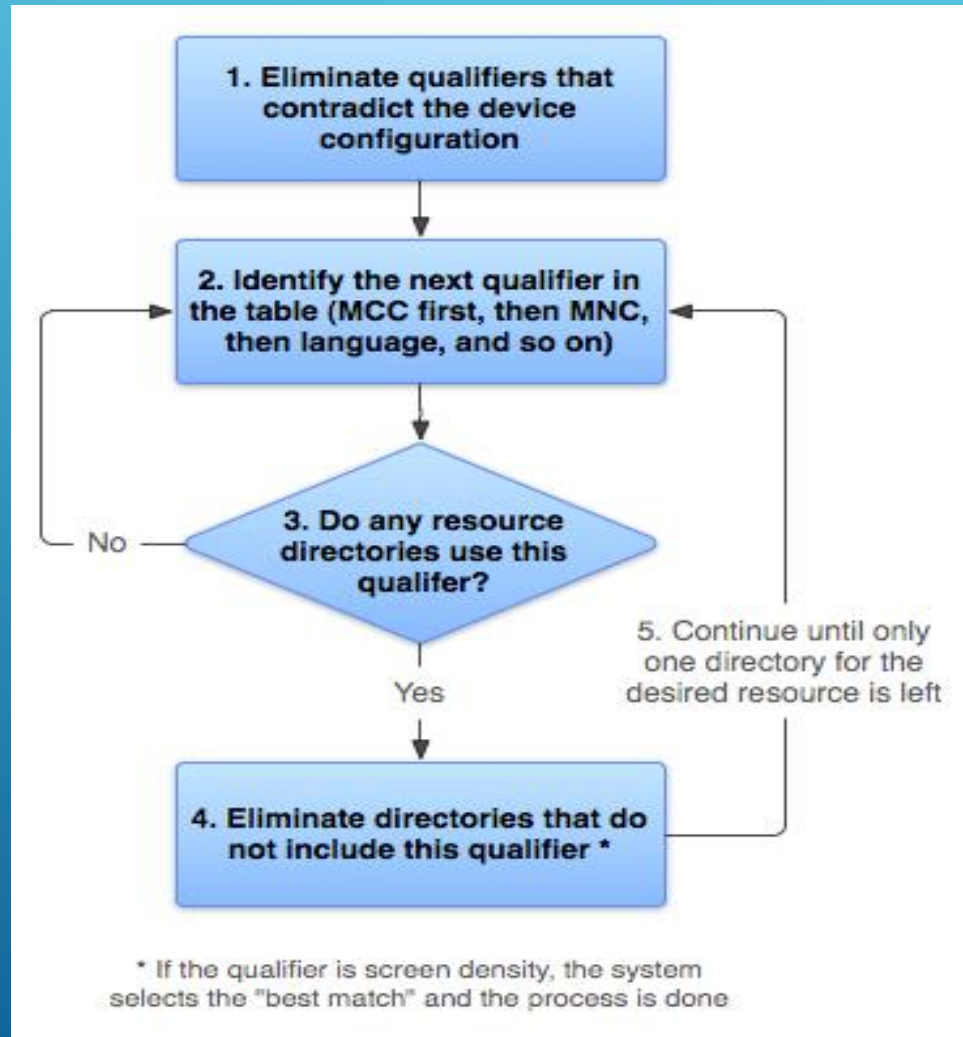
# RESOURCES **ALTERNATIVES**

- Android applications might provide **alternative resources** to support specific device configurations (e.g. different languages).
- At runtime, Android **detects** the current device configuration and **loads** the appropriate resources for the application.
- To specify configuration-specific alternatives:
  1. Create a new directory in **res/** named in the form **<resources\_name>-<config\_qualifier>**
  2. Save the respective alternative resources in this new directory

# RESOURCES ALTERNATIVES MATCHING



# RESOURCES ALTERNATIVES MATCHING

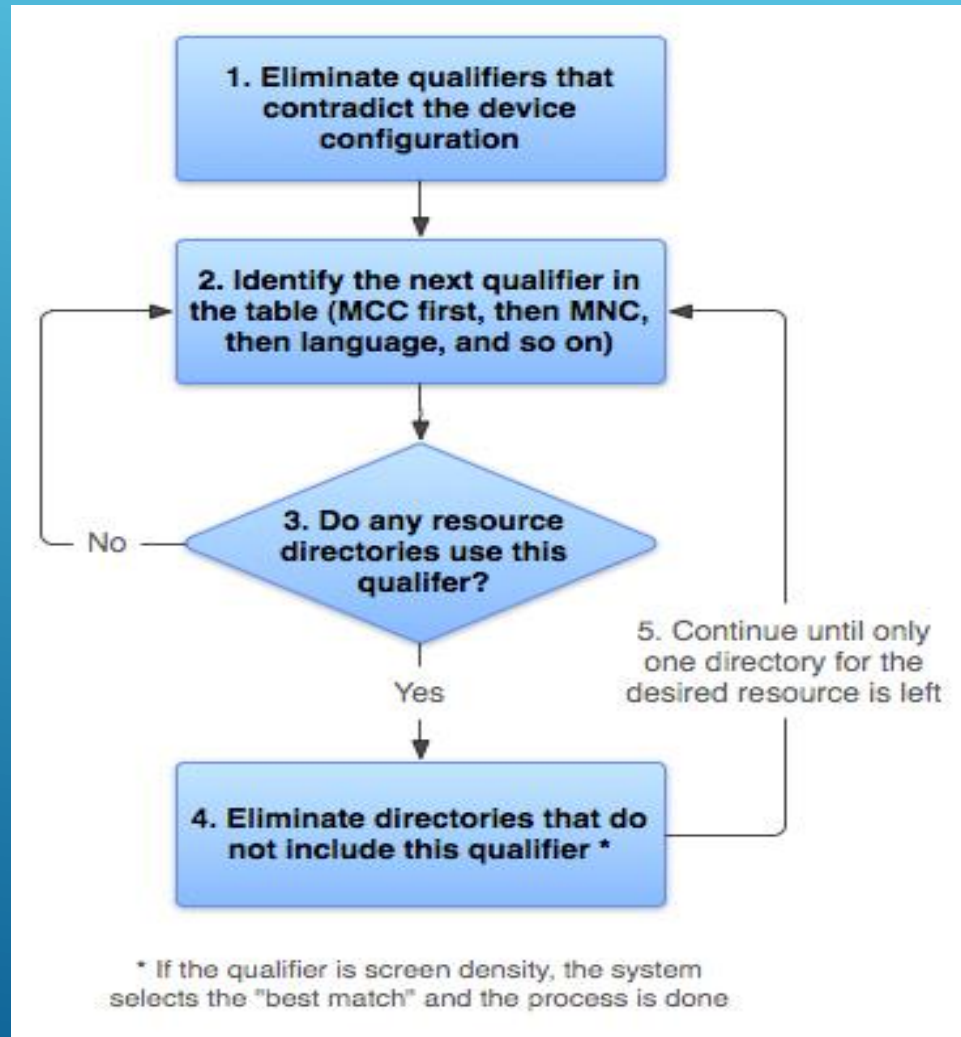


## DEVICE CONFIGURATION

Locale = it<sup>[SEP]</sup>Screen orientation = port<sup>[SEP]</sup>  
Screen pixel density = hdpi<sup>[SEP]</sup>  
Touchscreen type = notouch<sup>[SEP]</sup>  
Primary text input method = 12key<sup>[SEP]</sup>

drawable/  
drawable-it/  
drawable-fr-rCA/  
drawable-it-port/  
drawable-it-notouch-12key/  
drawable-port-ldpi/  
drawable-port-notouch-12key/

# RESOURCES ALTERNATIVES MATCHING



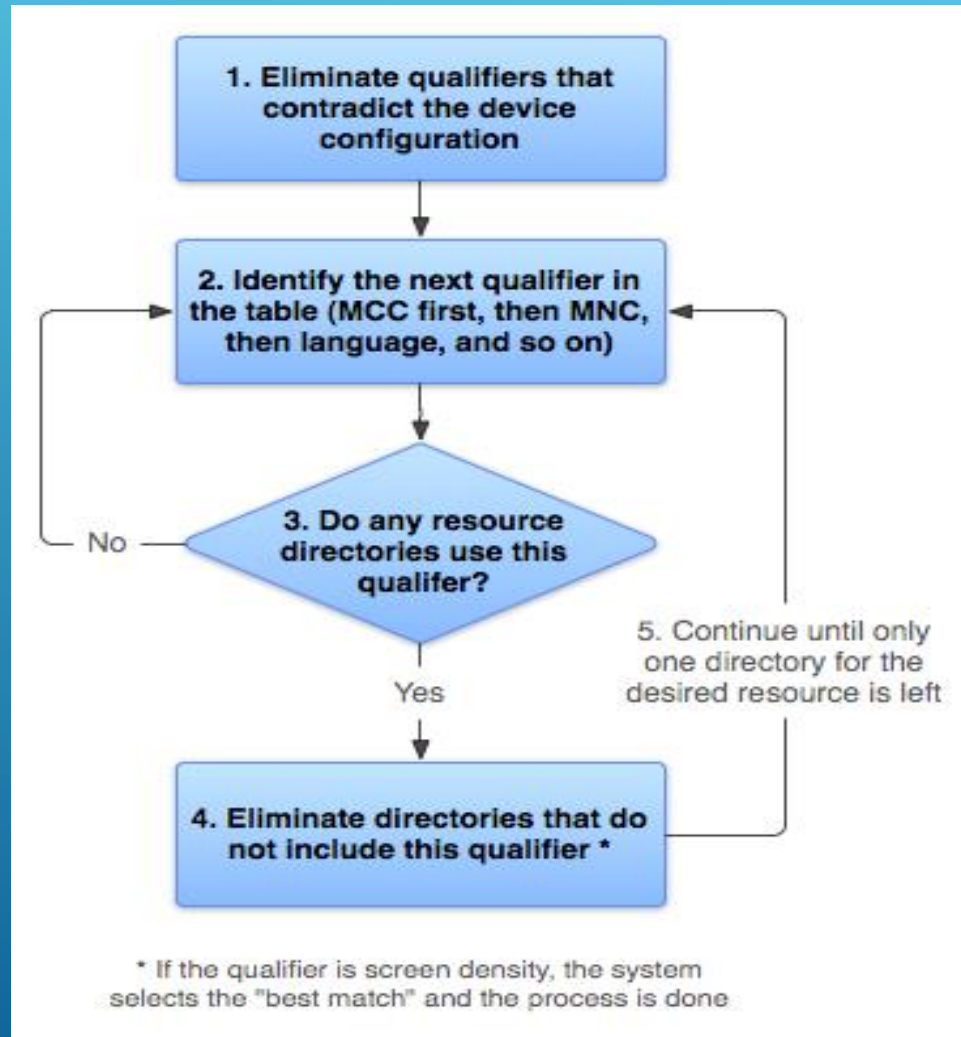
## DEVICE CONFIGURATION

Locale = it<sup>[SEP]</sup>Screen orientation = port<sup>[SEP]</sup>  
Screen pixel density = hdpi<sup>[SEP]</sup>  
Touchscreen type = notouch<sup>[SEP]</sup>  
Primary text input method = 12key<sup>[SEP]</sup>

~~drawable/~~  
~~drawable-it/~~  
~~drawable-fr-rCA/~~  
~~drawable-it-port/~~  
~~drawable-it-notouch-12key/~~  
~~drawable-port-ldpi/~~  
~~drawable-port-notouch-12key/~~



# RESOURCES ALTERNATIVES MATCHING



## DEVICE CONFIGURATION

Locale = it<sup>[SEP]</sup>Screen orientation = port<sup>[SEP]</sup>  
Screen pixel density = hdpi<sup>[SEP]</sup>  
Touchscreen type = notouch<sup>[SEP]</sup>  
Primary text input method = 12key<sup>[SEP]</sup>

~~drawable/~~  
~~drawable-it/~~  
~~drawable-fr-rCA/~~  
**drawable-it-port/**  
~~drawable-it-notouch-12key/~~  
~~drawable-port-ldpi/~~  
~~drawable-port-notouch-12key/~~

## **BEST PRACTICE**

- Provide **default** resources for your application.
- Provide **alternative resources** based on the target market of your application.
- Avoid **unnecessary or unused** resources alternatives.
- Use **alias** to reduce the duplicated resources.