Laboratorio di Applicazioni Mobili
Bachelor in Computer Science &
Computer Science for Management

University of Bologna

Application Resources

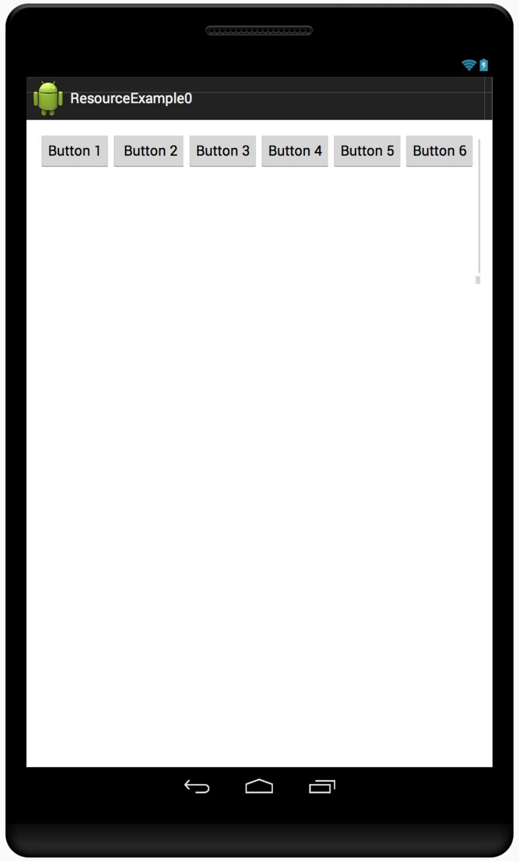
Federico Montori
federico.montori2@unibo.it

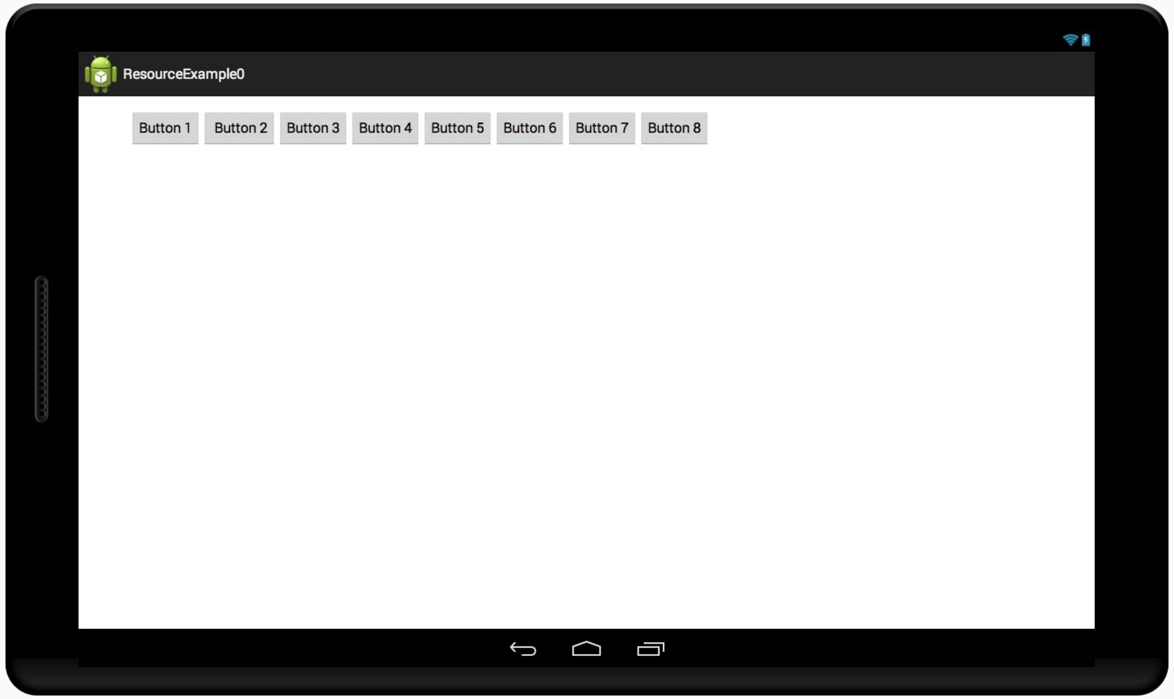
Table of Contents

Overview on Resources

PROBLEM: Android is designed to run on many different devices, such as phones, tablets, and televisions. The range of devices provides a huge potential audience for your app. For your app to be successful on all devices, it must tolerate feature variability and provide a flexible user interface that adapts to different screen configurations.

Overview on Resources





The same application layout with 8 buttons, on a smartphone and on a tablet

Overview on Resources

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

TRADITIONAL SOLUTION: Foresee all the alternatives in the 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 declarative XML-based approach to define resources (images, files, layout, text, etc)

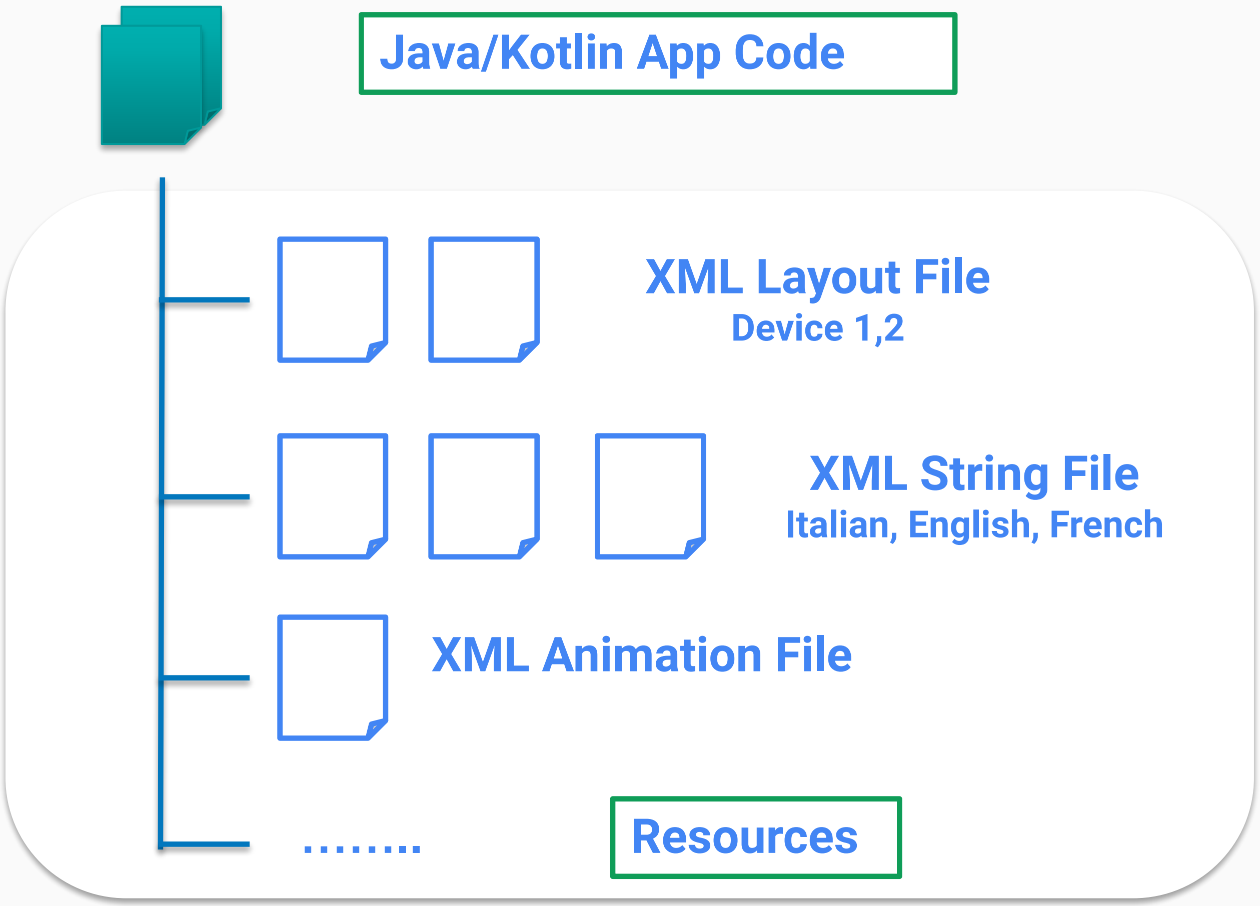
Overview on Resources

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)

- **Separate** data presentation (layout) from data management
- **Provide** alternative resources to support specific device configurations (e.g. different language packs)
- **Re-compile** only when strictly needed!

Overview on Resources

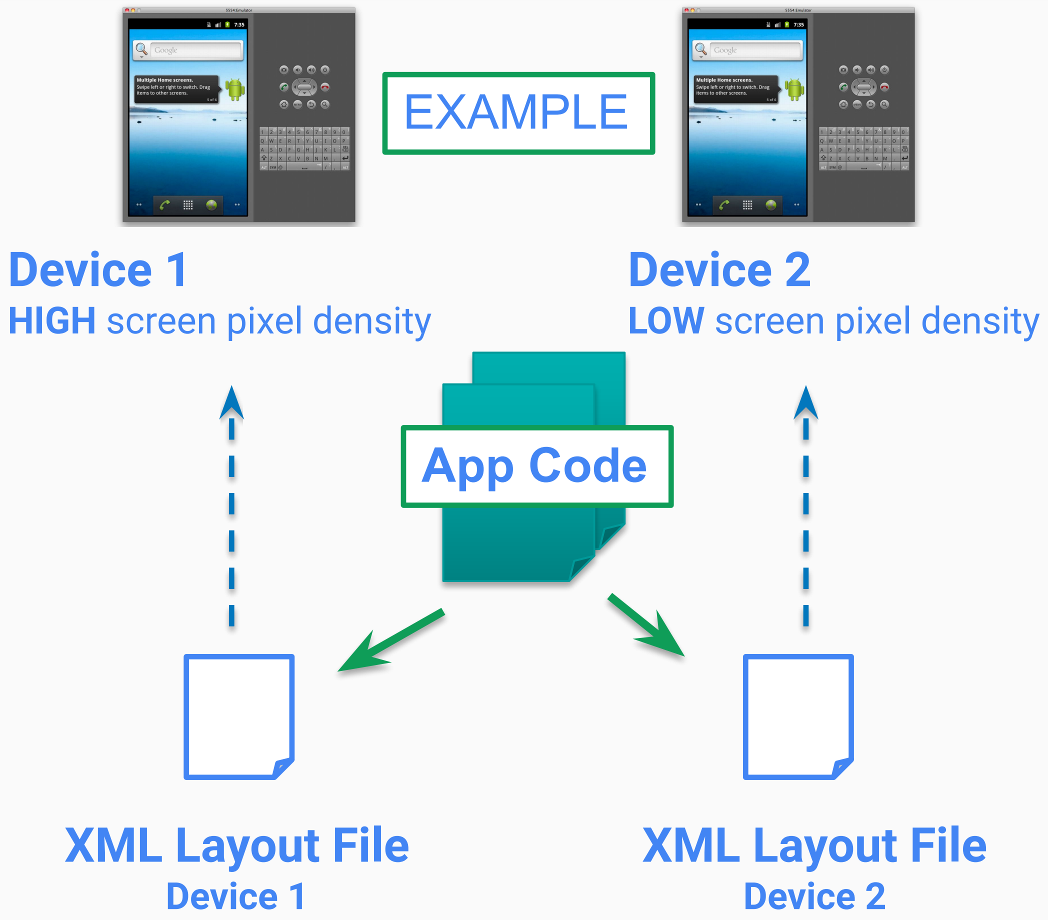


Use XML files to define (declarative approach):

- Application Layout
- Text labels
- Application Menu
- ...

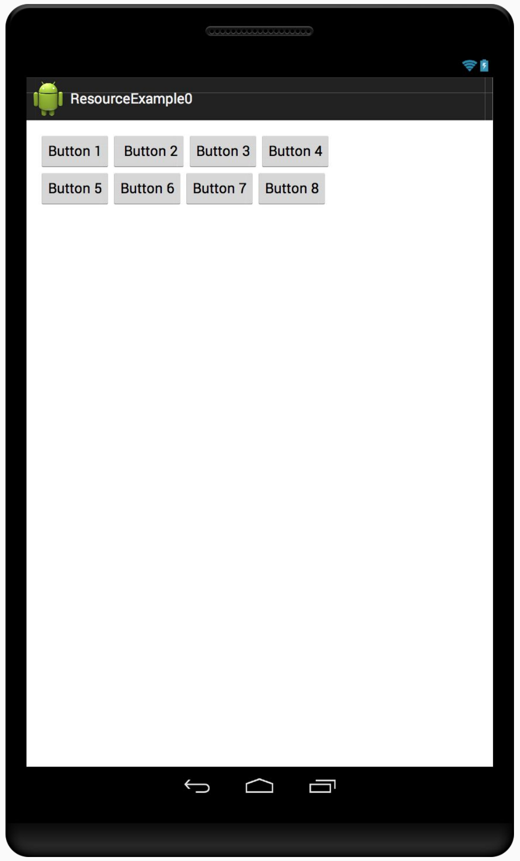
Foresee different resources alternatives for different device configurations (e.g. screen resolution, language, input devices, etc.

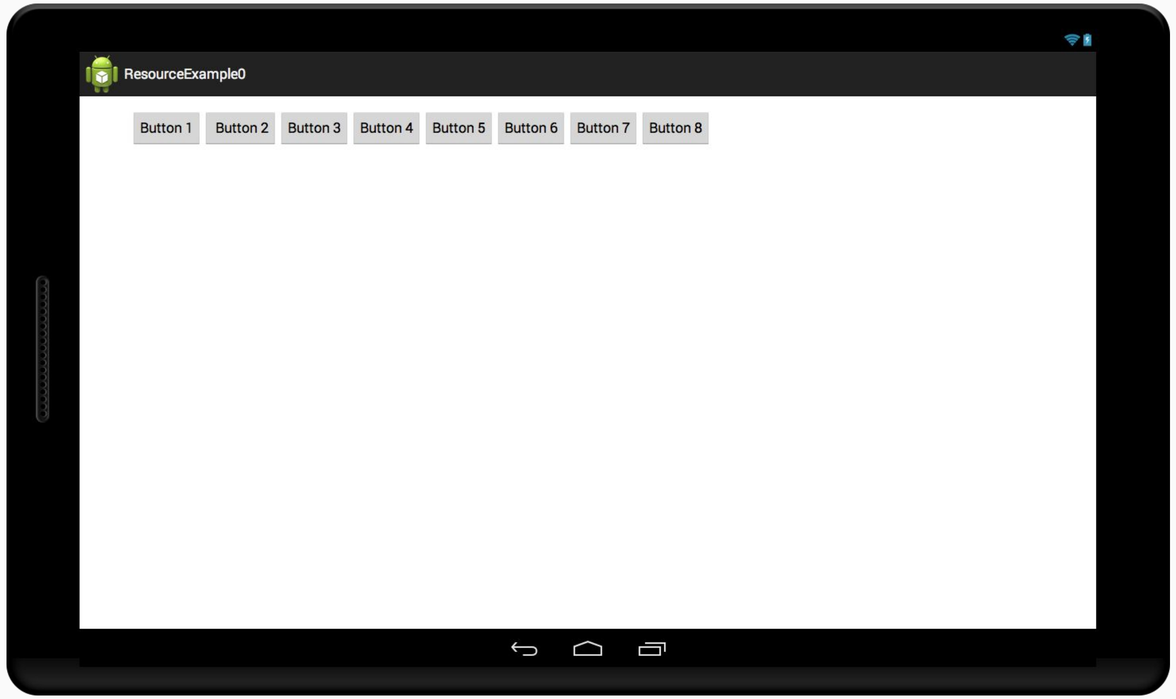
Overview on Resources



- 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
- <u>No need to recompile!</u>
- Just add a new XML file if you need to support a new device

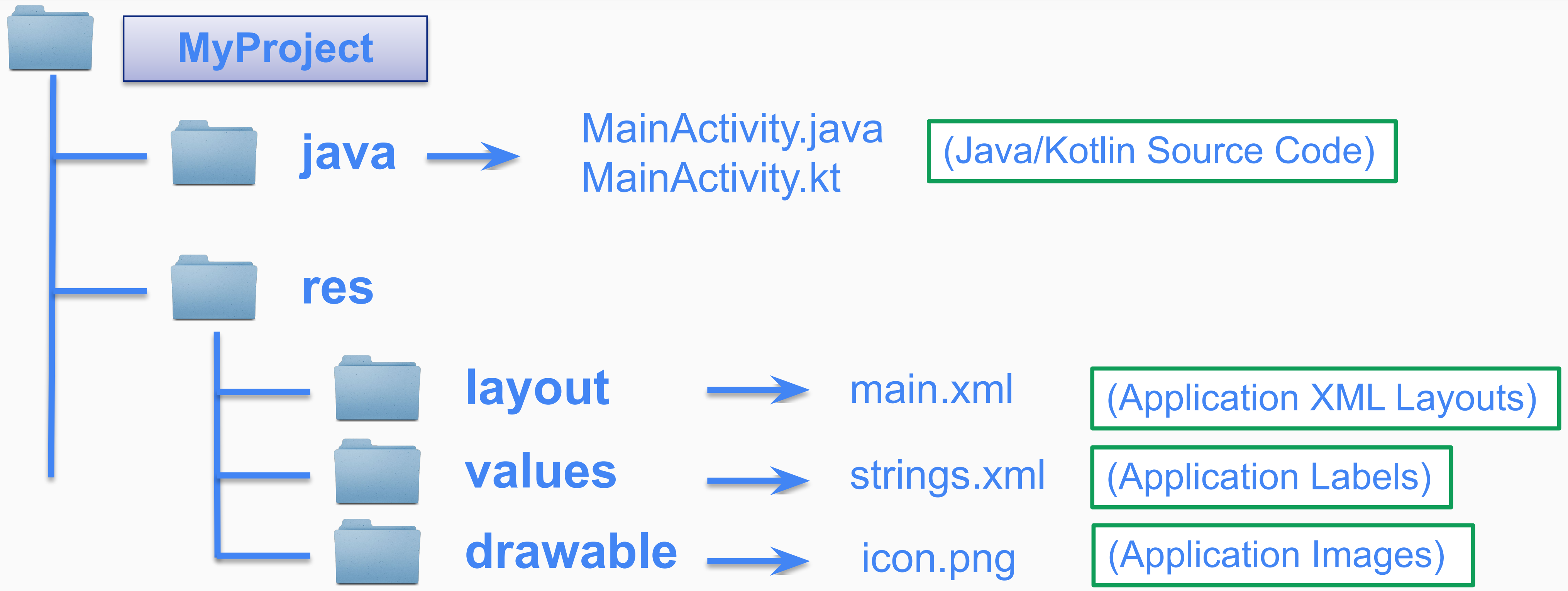
Overview on Resources





The same application layout with 8 buttons, on a smartphone and on a tablet

Resources Definition



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

Resources Definition

Resource Type	Resource contained
- **res/animator**	*XML for property animations - old framework to change properties over time.*
- **res/anim**	*XML for tween animations - newer framework to change properties in a bulk.*
- **res/color**	*XML files that define a state list of colors (simple colors are in **values**).*
- **res/drawable**	*Bitmap files (.png, .jpg, .gif) or XML files compiled into other resources.*
- **res/layout**	*XML files that define a user interface layout.*
- **res/mipmap**	*Drawable files for different launcher icon densities*
- **res/menu**	*XML files that define application menus.*
- **res/raw**	*Arbitrary files to save in their raw form.*
- **res/values**	*XML files that contain simple values, such as strings, integers, arrays.*
- **res/xml**	*Arbitrary XML files.*

Resources Definition

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

Example: **values/string.xml** contains all the text that the application uses. For example, the name of buttons, labels. default text, etc

```
<?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)

Resources Definition

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

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

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

Resources Definition

- Each Resource is associated with a *<u>unique</u>* **Identifier** (ID), that allows its access, which is composed of two parts:
    - The resource **type**: e.g. string, color, menu, drawable, layout, etc.
    - The resource **name**, which is either:
        - the filename, excluding the extension
        - the value in the XML ***android:name*** attribute.

- *<u>Two ways to access resources</u>*:
    - From the **Java/Kotlin** Code
    - From the **XML** files

Resources Definition

When the resource is a **View** the ID must be specified explicitly and does not use the **type + name** scheme:

```
android:id="@+id/button1"
```

@ means: “parse and expand the rest of the string as an id resource.
+ means: “this is going to be added as a new id in **R.java**”

This means that the View will be seen as an **id** resource.

Resources Access

Access from 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.

Resources Access

res/values/string.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <color name="my_red"> #FF3333 </color>
  <string name="labelButton"> Press Here! </string>
  <string name="labelText"> Hello world! </string>
</resources>
```

res/layout/layout_main.xml

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

Resources Access

Access from Java/Kotlin Code:

```
[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 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.

Resources Access

MainActivity.kt

```
// Get a string resource from the string.xml file
// when assigning to a variable use context.getResources()
val hello: String = this.getResources().getString(R.string.hello)

// Get a color resource from the string.xml file
val myRed: Color = getResources().getColor(R.color.my_red)

// Load a custom layout for the current screen
setContentView(R.layout.layout_main)

// Set the text on a TextView object using a resource ID
// keyword as is equivalent to explicit cast
val msgTextView = findViewById(R.id.label1) as TextView
msgTextView.setText(R.string.labelText)
```

Values: string, numbers and arrays

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

Values: string, numbers and arrays

values/my_values.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_title"> Example Application </string>
    <string name="label"> Hello world! </string>
    <integer name="value"> 53 </integer>
    <string-array name="nameArray">
        <item> John Bonham </item>
        <item> Frank Zappa </item>
    </string-array>
    <integer-array name="valArray">
        <item> 1 </item>
        <item> 2 </item>
    </integer-array>
</resources>
```

Values: string, numbers and arrays

MainActivity.kt

```
// Here we use the property access syntax for resources

// Access the string value
var label: String = resources.getString(R.string.label)

// Access the integer value
var value: Int = resources.getInteger(R.integer.value)

// Access the string-array values
var nameArray: Array<String> = resources.getStringArray(R.array.nameArray)

// Access the integer-array values
var valArray: IntArray = resources.getIntArray(R.array.valArray)
```

Values: color, dimension and style

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

Values: color, dimension and style

values/colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="coin_yellow">#FF9800</color>
    <color name="hound_grey">#4E4E4E</color>
    <color name="coin_yellow_transparent">#40FF9800</color>
</resources>
```

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

From Kotlin code:

```
val coinColor: Int = resources.getColor(R.color.coin_yellow, null)
    // the second parameter is the theme, which is nullable
```

Values: color, dimension and style

values/dimen.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="textview_height">25dp</dimen>
    <dimen name="textview_width">150dp</dimen>
    <dimen name="font_size">20sp</dimen>
</resources>
```

Applying dimensions to attributes in the XML layout:

layout/layout_main.xml

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

Values: color, dimension and style

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

Values: color, dimension and 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.

Values: color, dimension and style

values/themes.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="MyTheme" parent="Theme.Material3.DayNight.NoActionBar">
    <item name="colorPrimary">@color/coin_yellow</item>
    <item name="colorSecondary">@color/hound_grey</item>
  </style>
</resources>
```

Applying a style to a view:

layout/layout_main.xml

```
<Button style="@style/MyTheme"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="Push me!" />
```

Values: color, dimension and style

values/themes.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="MyTheme" parent="Theme.Material3.DayNight.NoActionBar">
    <item name="colorPrimary">@color/coin_yellow</item>
    <item name="colorSecondary">@color/hound_grey</item>
  </style>
</resources>
```

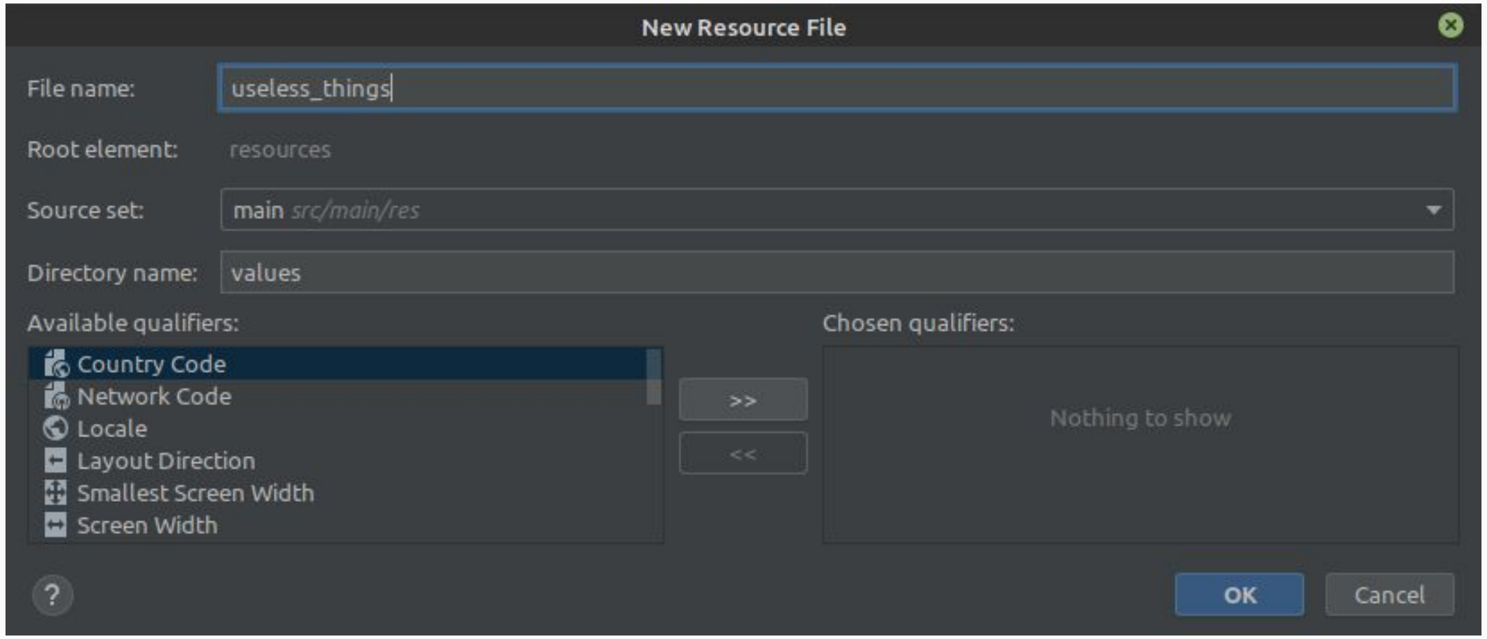
Styles can also be applied to the whole application in the Manifest

AndroidManifest.xml

```
<application
    …
    android:theme="@style/MyTheme"
> … </ application>
```

Values

Resources can be defined in **any other file** defined by the users
(File →New →Android resource file)



New Resource File

File name: useless_things

Root element: resources

Source set: main src/main/res

Directory name: values

Available qualifiers:

Chosen qualifiers:

Nothing to show

Drawables

Resource Type	File	Java/Kotlin constant	XML tag	Description
drawable	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:

- Images (literally put raw images in the drawable folder)
- 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

Drawables

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

Drawables

A **BitMap** file is a .png, .jpg or a .gif file.

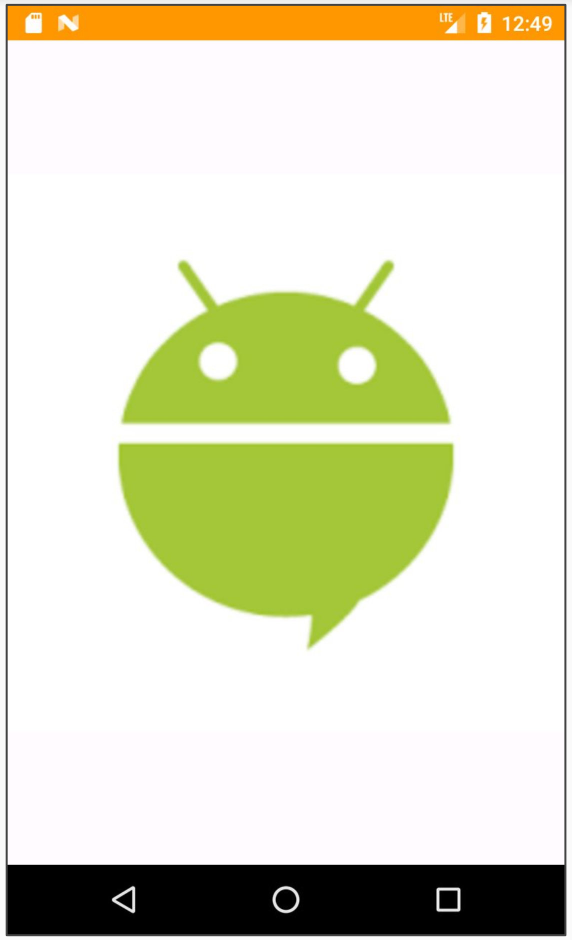
Android creates a BitMap resource for any of these files saved in the res/drawable directory.

Retrieve the image as a Drawable from Kotlin:

```
val drawing: Drawable = theme.resources.getDrawable(R.drawable.AndroidQuestion)
    // theme is the property access syntax for getTheme()
    // theme.resources uses the resources for the theme associated with the context
    // alternative syntax for getDrawable(id, theme), similar to getColor
```

The resource is **AndroidQuestion.png** in **drawable/**

Drawables



The most common view that displays images is the **ImageView** with XML tag: **<ImageView>**

```
<ImageView
    android:id="@+id/imgMain"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:src="@drawable/AndroidQuestion"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintDimensionRatio="1:1"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Mipmap

Resource Type	File	Java/Kotlin constant	XML tag	Description
mipmap	Any file in the res/mipmap/	R.mipmap.<key>	<mipmap>	Images to be used as icons

The mipmap directory is dedicated to all images that are used as icons for:

- App launcher
- App notifications
- App bar

Icons are retrieved by the manifest file:

AndroidManifest.xml

```
<application ...
     android:icon="@mipmap/ic_launcher"
     android:roundIcon="@mipmap/ic_launcher_round"
> ... </ application>
```

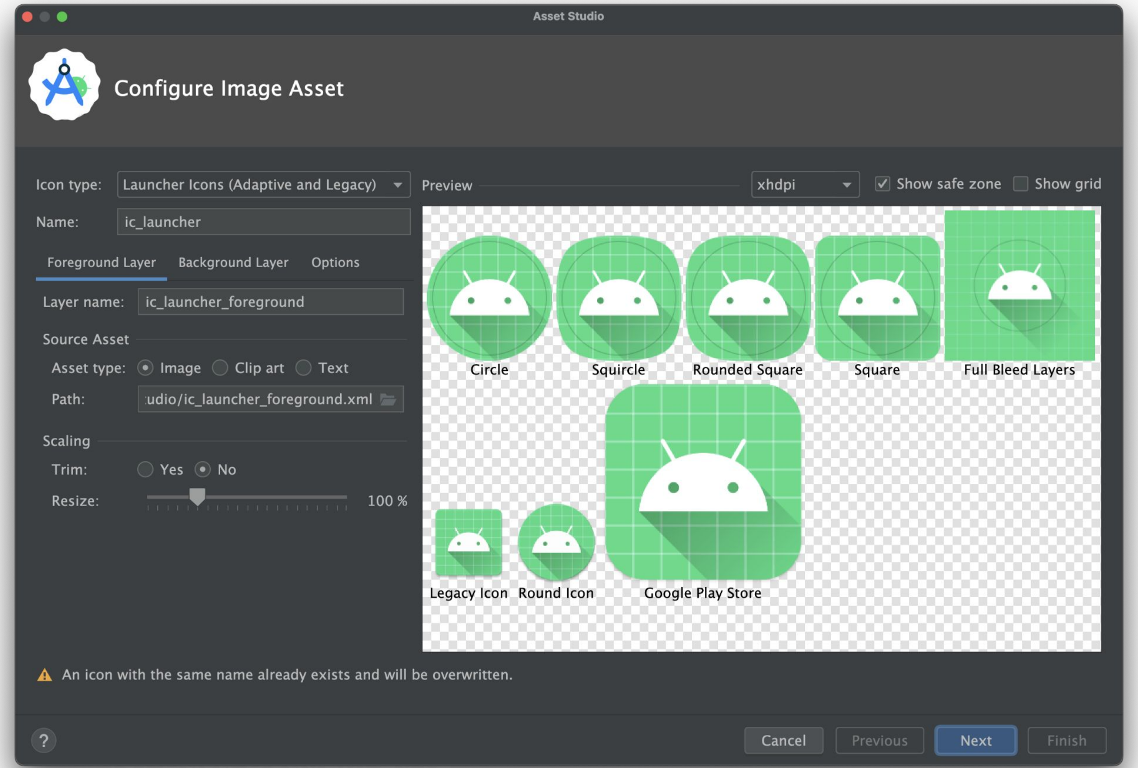
Mipmap

Image Asset Studio is a tool for creating such icons

Go to the **Project** tab and select the **Android View**

Right-click on the **res** directory and select "**new**→ **Image Asset**"

This will populate the Mipmap directory with icon versions for different usages (shape, resolution, etc.)



Asset Studio

Configure Image Asset

Icon type: Launcher Icons (Adaptive and Legacy)

Name: ic_launcher

Layer name: ic_launcher_foreground

Asset type: [✓ Image] Clip art Text

Path: udio/ic_launcher_foreground.xml

Trim: Yes [✓ No]

Resize: 100 %

Preview: xhdpi [✓ Show safe zone] Show grid

An icon with the same name already exists and will be overwritten.

Other Resources

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

We have seen it

We will see it

Other Resources

Resource Type	File	Java constant	XML tag	Description
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 **InputStream** objects:

```
val inputStream: InputStream = resources.openRawResource(R.raw.videoFile)
```

Other Resources

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 Also used by Preferences (we will see it)

- The **res/xml** folder contains arbitrary XML files that can be read at runtime through the R.xml.<filename> constant.
- It is possible to parse the XML file through an XML Parser.

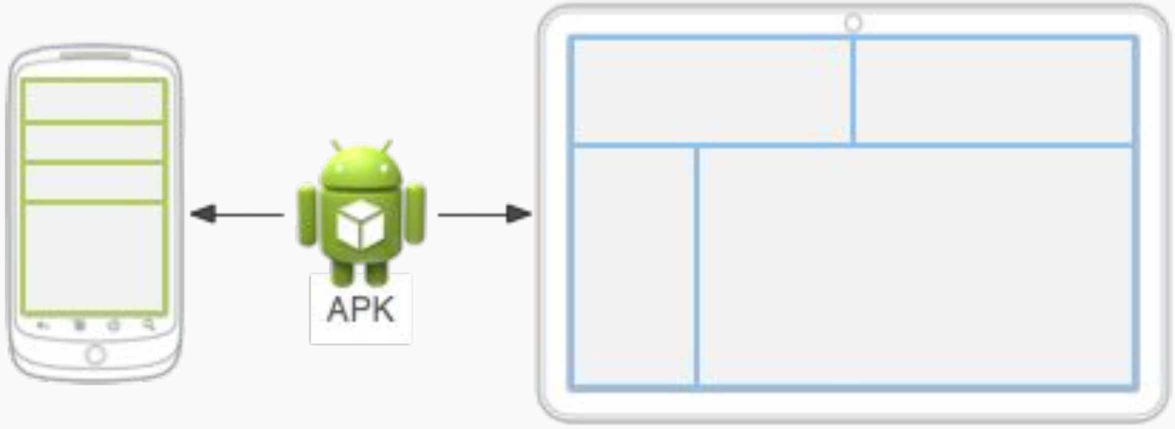
```
<?xml version="1.0" encoding="utf-8"?>
<names
xmlns:android="http://schemas.android.com/apk/res/android">
   <name code="1"> Federico Montori </name>
   <name code="2"> Nicholas Lazzari </name>
</names>
```

```
val xmlParser: XmlResourceParser = resources.getXml(R.xml.my_repository)
```

Resource Alternatives

Android applications should provide alternative resources to support specific device configurations (e.g. different languages, screen orientations, etc…).

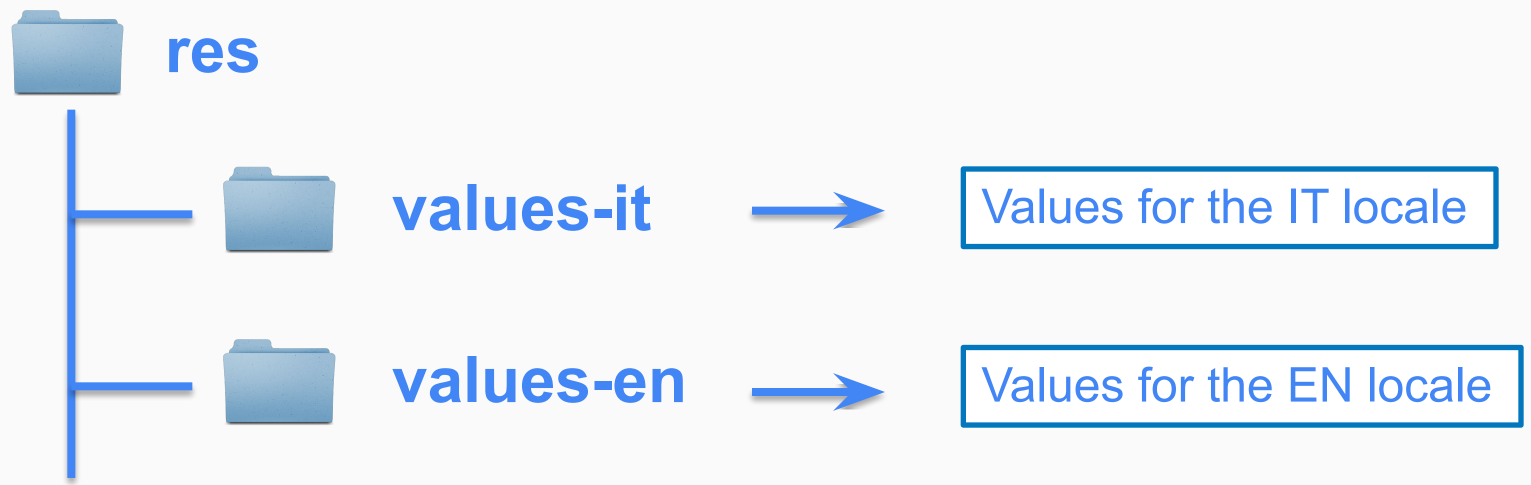
At runtime, Android detects the current device configuration and loads the appropriate resources for the application.



Resource Alternatives

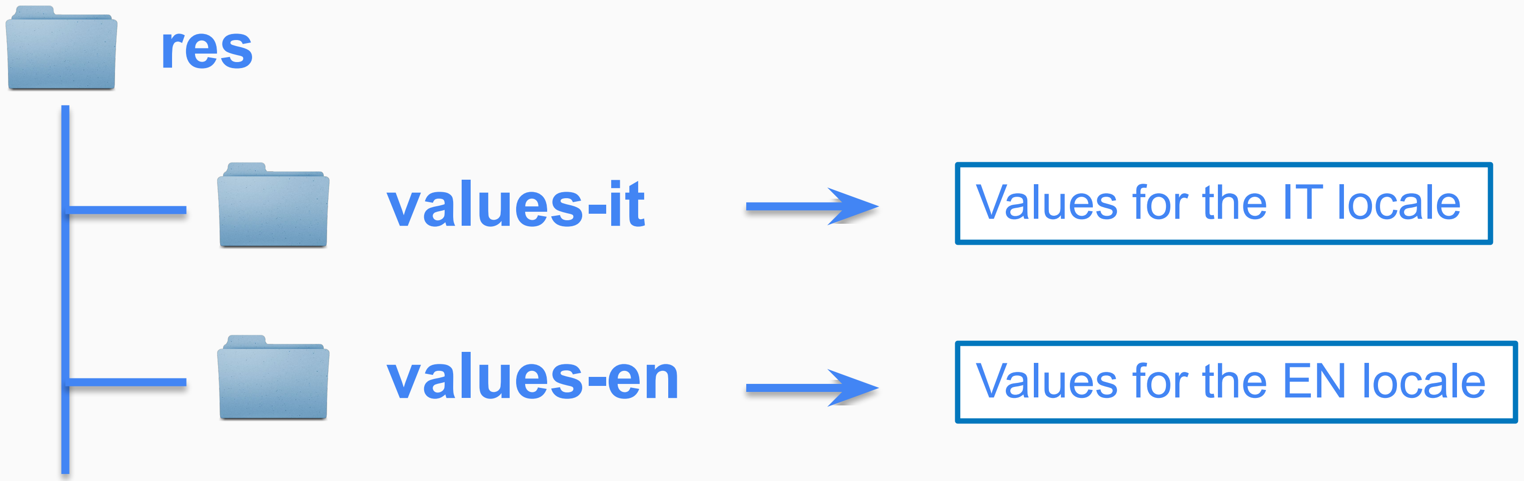
To specify configuration-specific alternatives:

- Create a new directory in **res/** named in the form **<resources_name>-<qualifier>**
- Save the respective alternative resources in this new directory



Resource Alternatives

- **<resources_name>** is the directory name of the corresponding default resources (see previous slides, e.g.: values, layouts, etc.).
- **<qualifier>** is a name that specifies an individual configuration for which these resources are to be used (see next slide).



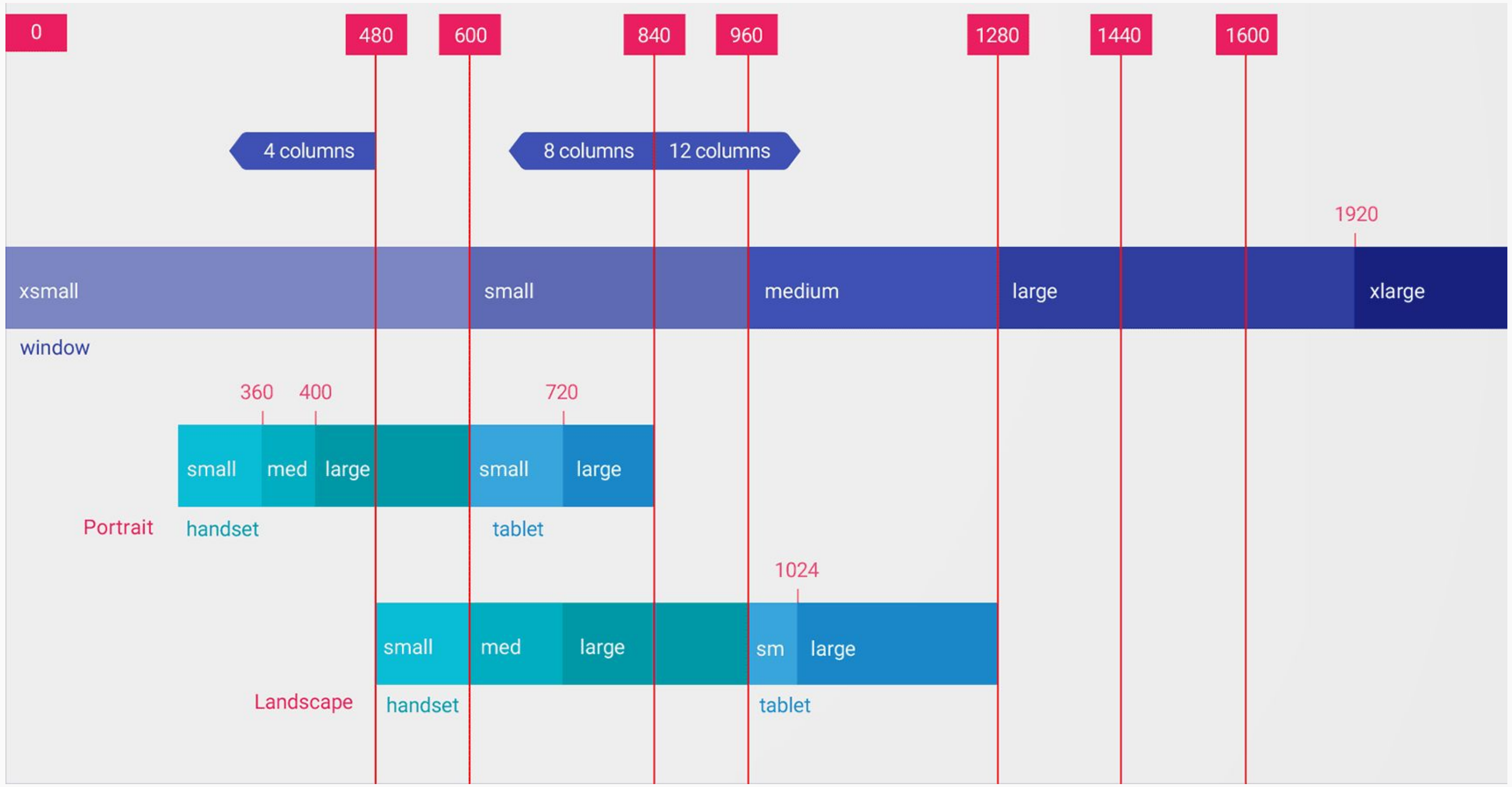
Resource Alternatives

Configuration	Values Example	Description
MCC and MNC	mcc310, mcc208, etc.	mobile country code (MCC)
Language and region	en, fr, en-rUS, etc.	ISO 639-1 language code
Layout direction	ldrtl, ldltr	e.g. Arab goes right-to-left
Smallest width	sw320dp, etc.	shortest dimension of screen
Available width	w720dp, w320dp, etc.	minimum available module width
Available height	h720dp, etc	minimum available module height
Screen size	small, normal, large, etc.	screen size
Screen aspect	long, notlong	aspect ratio of the screen
Round screen	round, notround	whether the screen is round
Wide color gamut	widecg, nowidecg	whether a WCG is available
High dynamic range (HDR)	highdr, lowdr	whether the screen is HDR

Resource Alternatives

Configuration	Values Example	Description
Screen orientation	port, land	screen orientation (can change!)
UI mode	car, desk, television, etc.	what kind of appliance is used
Night mode	night, nonight	whether the night mode is active
Screen pixel density (dpi)	ldpi, mdpi, hdpi	screen pixel density
Touchscreen type	notouch, finger	type of touch
Keyboard availability	keysexposed, etc.	type of keyword
Primary text input method	nokeys, qwerty	availability of qwerty keyboard
Navigation key availability	navexposed, etc	navigation keys of the application
Primary non-touch navigation method	dpad, trackball, etc.	Navigation means
Platform version (API level)	v3, v4, v7, etc	API supported by the device

Resource Alternatives

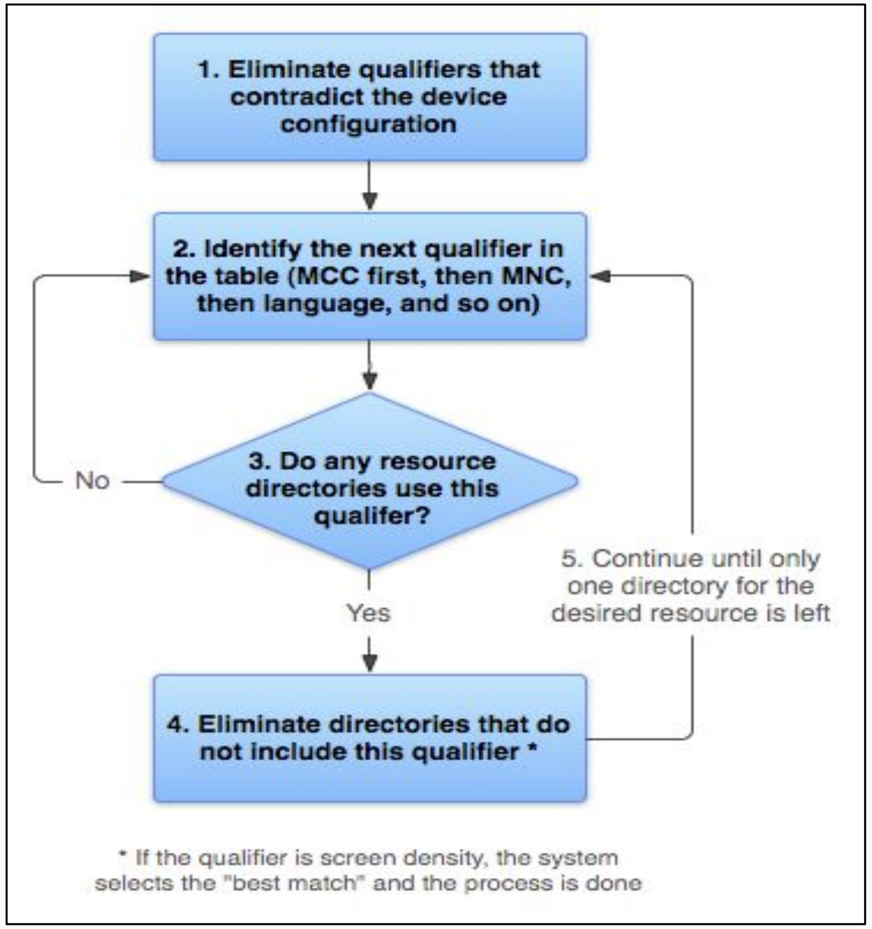


Legenda for size-oriented properties.

All values in **dp** (density-independent pixels)

dp != dpi

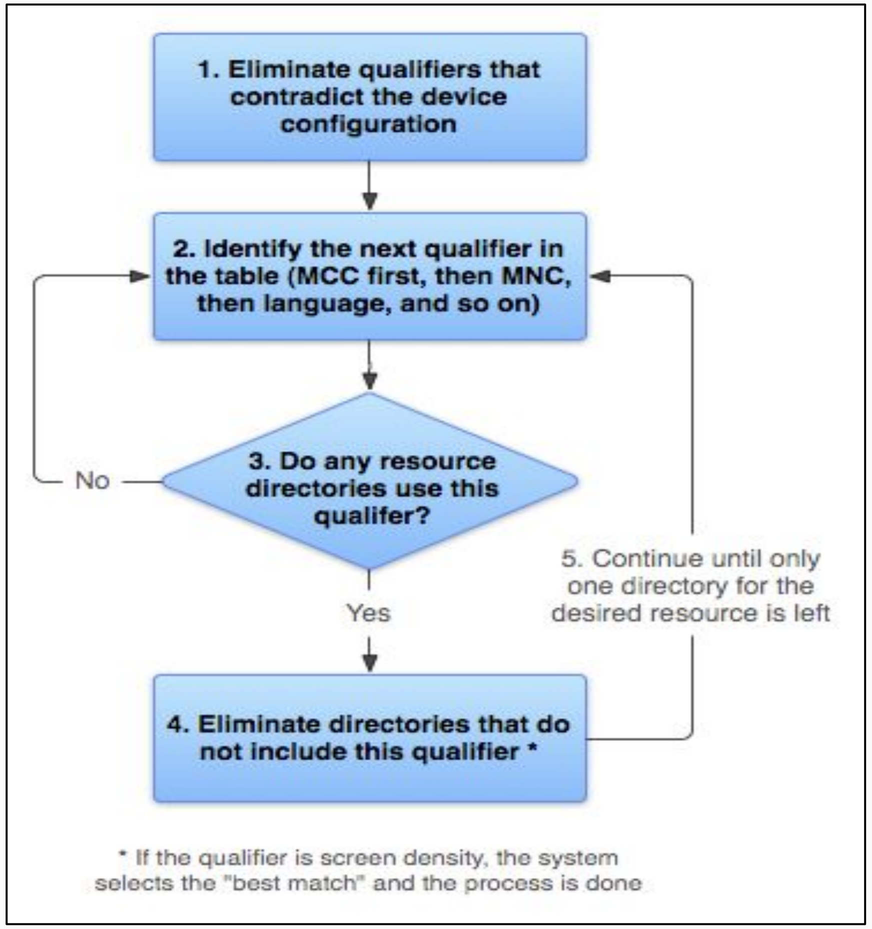
Resource Alternatives


1. Eliminate qualifiers that contradict the device configuration
2. Identify the next qualifier in the table (MCC first, then MNC, then language, and so on)
3. Do any resource directories use this qualifer?
No
Yes
4. Eliminate directories that do not include this qualifier *
5. Continue until only one directory for the desired resource is left
* If the qualifier is screen density, the system selects the "best match" and the process is done

When the application requests a resource for which there are multiple alternatives, **Android selects which alternative resource to use** at runtime, depending on the current device configuration, **through the algorithm shown in the Figure**.

Qualifiers are considered ordered as per the previous table.

Resource Alternatives


1. Eliminate qualifiers that contradict the device configuration
2. Identify the next qualifier in the table (MCC first, then MNC, then language, and so on)
3. Do any resource directories use this qualifer?
No
Yes
4. Eliminate directories that do not include this qualifier *
5. Continue until only one directory for the desired resource is left
* If the qualifier is screen density, the system selects the "best match" and the process is done

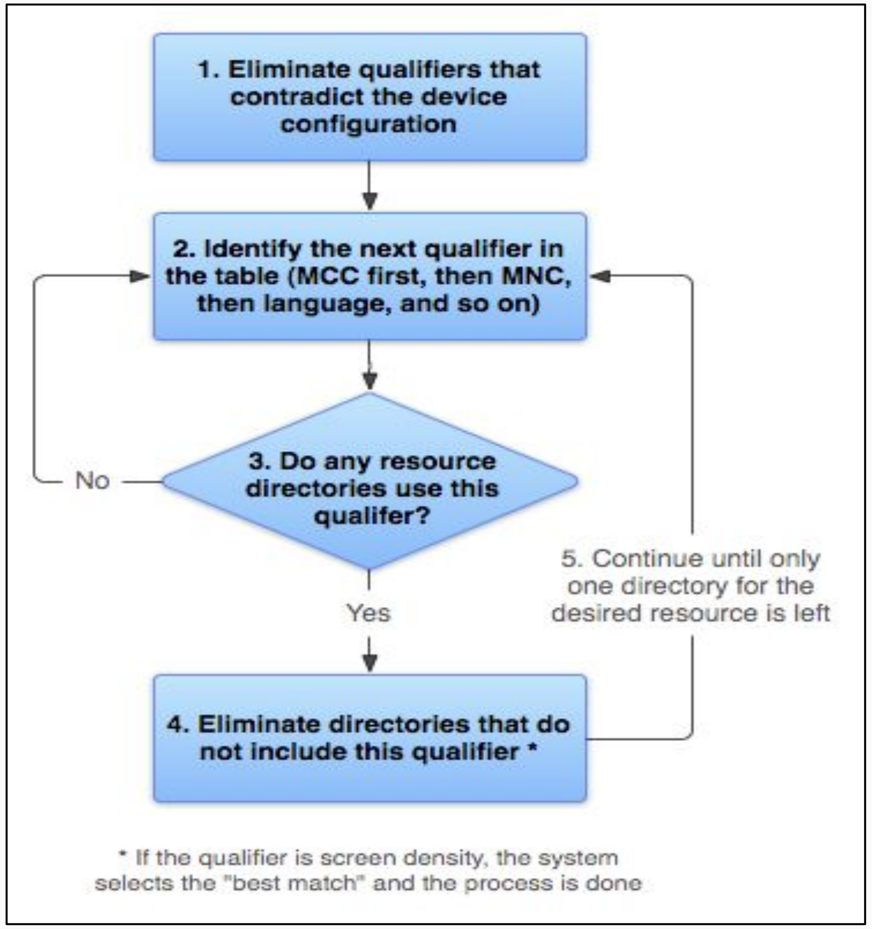
Device configuration:

Locale = it Screen orientation = port
Screen pixel density = hdpi
Touchscreen type = notouch
Primary text input method = 12key

Application resources:

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

Resource Alternatives



1. Eliminate qualifiers that contradict the device configuration
2. Identify the next qualifier in the table (MCC first, then MNC, then language, and so on)
3. Do any resource directories use this qualifer?
No
Yes
4. Eliminate directories that do not include this qualifier *
5. Continue until only one directory for the desired resource is left
* If the qualifier is screen density, the system selects the "best match" and the process is done

1

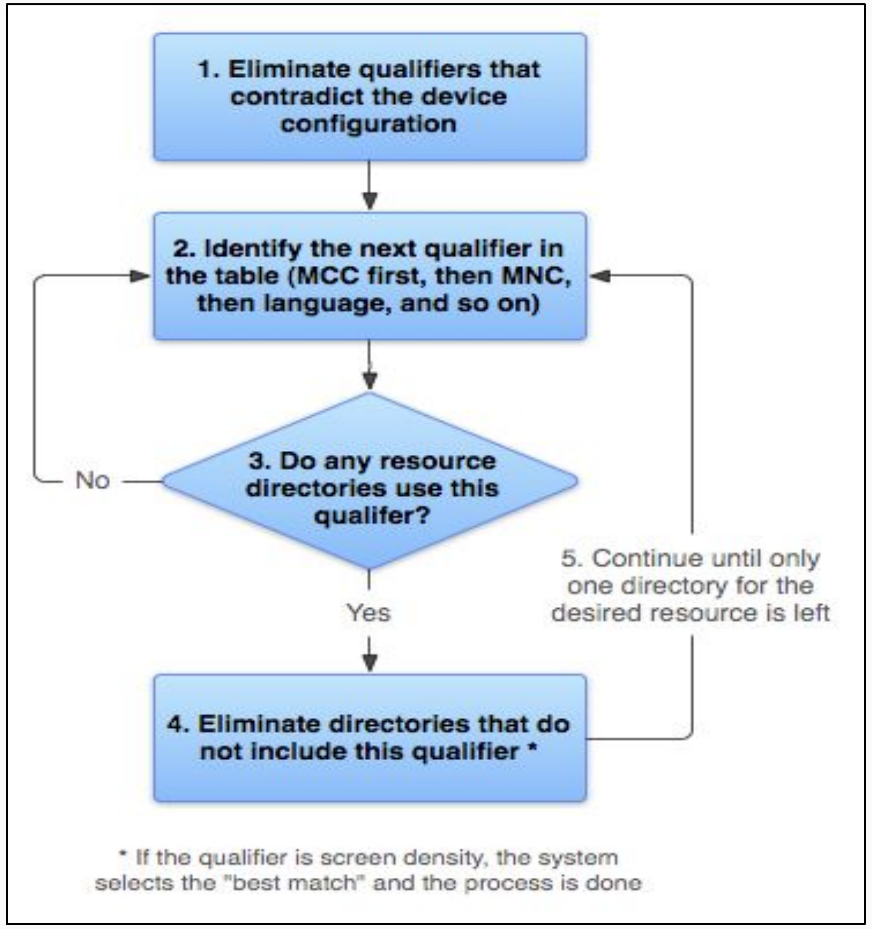
Device configuration:

Locale = it Screen orientation = port
Screen pixel density = hdpi
Touchscreen type = notouch
Primary text input method = 12key

Application resources:

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

Resource Alternatives


1. Eliminate qualifiers that contradict the device configuration
2. Identify the next qualifier in the table (MCC first, then MNC, then language, and so on)
3. Do any resource directories use this qualifer?
No
Yes
4. Eliminate directories that do not include this qualifier *
5. Continue until only one directory for the desired resource is left
* If the qualifier is screen density, the system selects the "best match" and the process is done

2
3
4

Device configuration:

Locale = it Screen orientation = port
Screen pixel density = hdpi
Touchscreen type = notouch
Primary text input method = 12key

Application resources:

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

Resource Alternatives

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.

Resource Alternatives

Location / Language	US	France	Canada	Italy	Germany	Rest of the world
English	Hello		Hello			Hello
French		Bonjour	Bonjour			Bonjour
Italian				Ciao		Ciao
German					Hallo	Hallo
Rest of the languages						Hello

Resource Alternatives

How to change the splash screen depending on the language and location

res/

values	Hello
values-it	Ciao
values-fr	Bonjour
values-de	Hallo
drawable	
drawable-en-rUS	
drawable-en-rCA	
drawable-it-rIT	
drawable-de-rDE	
drawable-fr-rFR	
drawable-fr-rCA	

Questions?

federico.montori2@unibo.it