Localization and Resources

Originals of Slides and Source Code for Examples:
http://www.coreservlets.com/android-tutorial/

Topics in This Section

- **Localization overview**
- **Localization options**
  - Language
  - Country/region language variations
  - Screen orientation
  - Display resolution
  - Others
- **Configuration qualifier precedence**

Overview

Big Idea

- **Good news**
  - Android's use of resource files (res/layout/main.xml, res/values/strings.xml, etc.) simplifies GUI development
- **Bad news**
  - Descriptions in English won't work in German
  - What fits in portrait orientation won't fit in landscape
  - Images for high-density screens are too large for low-density ones
- **Solution**
  - Make multiple layout and resource files
    - For different languages, orientations, etc.
  - Have Android automatically switch among or combine them
- **Notes**
  - Localization sometimes called L10N (L, 10 letters, N)
  - Also sometimes called Internationalization (I18N)

Process

- **Make qualified versions of resource files**
  - Find the settings that affect your application
    - Language, orientation, touchscreen type, dock mode, etc.
  - Find qualifier names that correspond to each setting
    - Language: en, en-rUS, es, es-rMX, etc.
    - Screen orientation: port, land
    - Display density: xhdpi, hdpi, mdpi, ldpi
    - Dock mode: car, desk
    - Etc.
  - Append qualifier names to folder names
    - res/values/strings.xml, res/values-es/strings.xml, res/values-es-rMX/main.xml
    - res/layout/main.xml, res/layout-land/main.xml
- **Load the resource normally**
  - R.string.title, R.layout.main, etc.
  - Android will switch among layout files automatically
  - Android will combine values files automatically
    - More specific values will replace earlier ones

Language

Overview

- **Idea**
  - Change the display based on the user's language
- **Resources that typically change**
  - Strings (in res/values, e.g., in res/values/strings.xml)
  - Images (in res/drawable – image file or XML file)
  - Colors (in res/values, e.g., in res/values/colors.xml)
  - Audio and video (in res/raw)
  - Dimensions, arrays, and styles (in res/values, e.g., …/dimens.xml, …/arrays.xml, …/styles.xml)
- **Resources that do not usually change**
  - Layout files (in res/layout)
    - Changing layout based on language makes for hard-to-maintain apps. See best-practices slide.

Steps

- **Make multiple folders with language codes**
  - res/values, res/values-es, res/values-ja, etc.
    - Language codes are specified by ISO 639-1
      - http://en.wikipedia.org/wiki/ISO_639-1
- **Define *all* strings in default folder**
  - In res/values, define *all* names
    - Use the most common language
  - E.g., res/values/strings.xml (or other name in res/values)

    ```
    <string name="company_name">Apple</string>
    <string name="welcome_message">Welcome!</string>
    ```
- **Use similar approach for colors, images, etc.**
  - Use res/values/ for *all* colors, dimensions, arrays, etc.
  - Use res/drawable for *all* image files
  - Use res/raw for *all* audio and video files

Steps (Continued)

- **Put language-specific strings in language-specific folders**
  - In res/values-es/strings.xml (or res/values-ja, etc), redefine *only* the names that change based on language
  - E.g., in res/values-es/strings.xml

    ```
    <string name="welcome_message">¡Bienvenidos!</string>
    ```
    - No entry for company_name, since the company name does not change (in Spanish, it is still Apple, not Manzana)
  - E.g., in res/values-ja/strings.xml

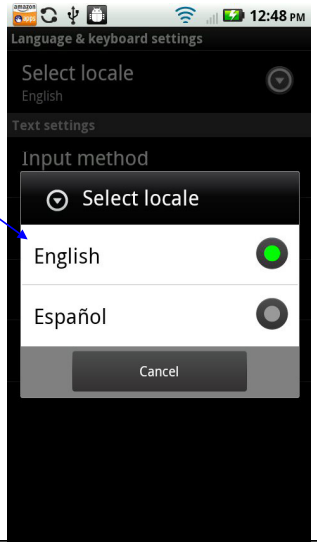
    ```
    <string name="welcome_message">ようこそ！</string>
    ```
    - No entry for company_name, since the company name does not change (in Japanese, it is still Apple, not アップル)
- **Use similar approach for other resources**
  - res/values-es/colors.xml, res/drawable-es/flag.png, etc.
    - *Only* redefine the ones that change based on language

Steps (Continued)

- **In XML, refer to base string name**
  - someAttribute="@string/company_name"
  - someAttribute="@ string/welcome_message"
    - No reference to folder or language.
    - Android will provide the proper version automatically. It *first* loads values from res/values/strings.xml, *then* loads values from res/values-es/strings.xml. Any names in second file that are common to first file are replaced.
- **In Java, refer to base string name**
  - getString(R.string.company_name)
  - getString(R.string.welcome_message)
    - No reference to folder or language. Same process as above.
- **Use similar approach for other resources**
  - XML: @drawable/flag, @color/default_foreground, etc.
  - Java: R.drawable.flag, R.color.default_foreground, etc.

How User Changes Device Language

- **On phone (or other physical Android device)**
  - Go to home screen, press Menu button, select Settings
    - (Most people also have the Settings app on desktop)
  - Choose Language and Keyboard
  - Choose Select locale at the top
    - Most phones will have a very limited number of choices, based on what device manufacturer supports
      - Android cannot (easily) use localization within apps unless entire OS supports that language. Major Android failing.
      - But, see upcoming slides on programmatic Locale changes.



How User Changes Device Language

- **On emulator**
  - Option 1: use same approach as above
    - You will have a limited number of choices, based on what was installed in your Android SDK image. But, it will almost certainly be a bigger set than the real phone.
  - Option 2: use Custom Locale app on app screen
    - If Locale you want is not there, add it (even regional ones)
    - Long-press the Locale to choose it

Example: The Android Resort

- **Idea**
  - Make an app that advertises a luxury resort where visitors can sit inside all day and play with their smart phones
- **Approach**
  - res/values/strings.xml defines
    - resort_name, welcome, our, pool, reserve, confirmed
  - res/values-es/strings.xml defines
    - welcome, our, pool, reserve, confirmed
      - Does not redefine resort_name
  - Also uses dimensions, colors, images, and layout files
    - But these do not change based on language, so localized versions are not shown until a later example

Strings File: English/Other (res/values/strings.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="resort_name">AndroidResort.com</string>
    <string name="welcome">Welcome to </string>
    <string name="our">Our </string>
    <string name="pool">swimming pool</string>
    <string name="reserve">Reserve Now!</string>
    <string name="confirmed">Registration Confirmed</string>
</resources>
```

Reminder from intents lecture: is a non-breaking space. Android does not preserve whitespace at the beginning and end of strings in resource files.

Strings File: Spanish (res/values-es/strings.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="welcome">Bienvenido a </string>
    <string name="our">Nuestra </string>
    <string name="pool">piscina</string>
    <string name="reserve">¡Reserva Ahora!</string>
    <string name="confirmed">Registro Confirmado</string>
</resources>
```

Note that there is not an entry for resort_name. The value from res/values/strings.xml carries over.

Layout File res/layout/main.xml (No Language-Based Versions)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://..."
              android:orientation="vertical" ...>
    <TextView android:text="@string/welcome" ... />
    <TextView android:text="@string/resort_name" ... />
    <ImageView android:src="@drawable/android_resort_pool"
               android:layout_height="wrap_content"
               android:layout_width="wrap_content"
               android:adjustViewBounds="true"
               android:scaleType="fitXY"/>
    <LinearLayout android:gravity="center_horizontal"
                  android:layout_height="wrap_content"
                  android:layout_width="match_parent">
        <TextView android:text="@string/our" ... />
        <TextView android:text="@string/pool" .../>
    </LinearLayout>
    <Button android:text="@string/reserve" .../>
</LinearLayout>
```


Manifest File (No Language-Based Versions)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.coreservlets.localization"
      android:versionCode="1"
      android:versionName="1.0">
    <uses-sdk android:minSdkVersion="8" />
    <application android:icon="@drawable/icon"
                 android:label="@string/resort_name">
        <activity android:name=".AndroidResortActivity"
                  android:label="@string/resort_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

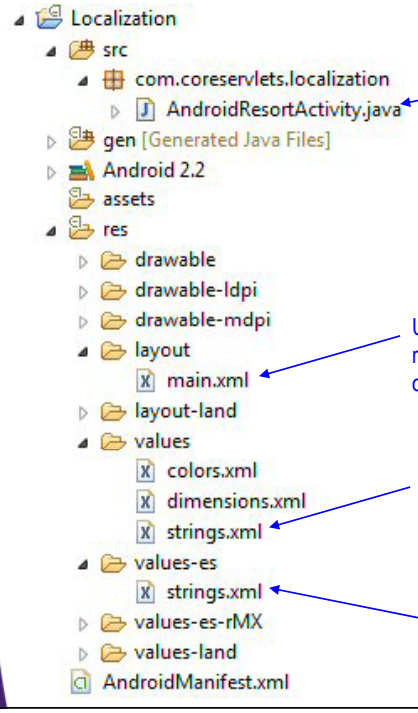
Java Code

```
public class AndroidResortActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void confirmRegistration(View clickedButton) {
        String message = getString(R.string.confirmed);
        showToast(message);
    }

    private void showToast(String text) {
        Toast.makeText(this, text, Toast.LENGTH_LONG).show();
    }
}
```


Project Layout



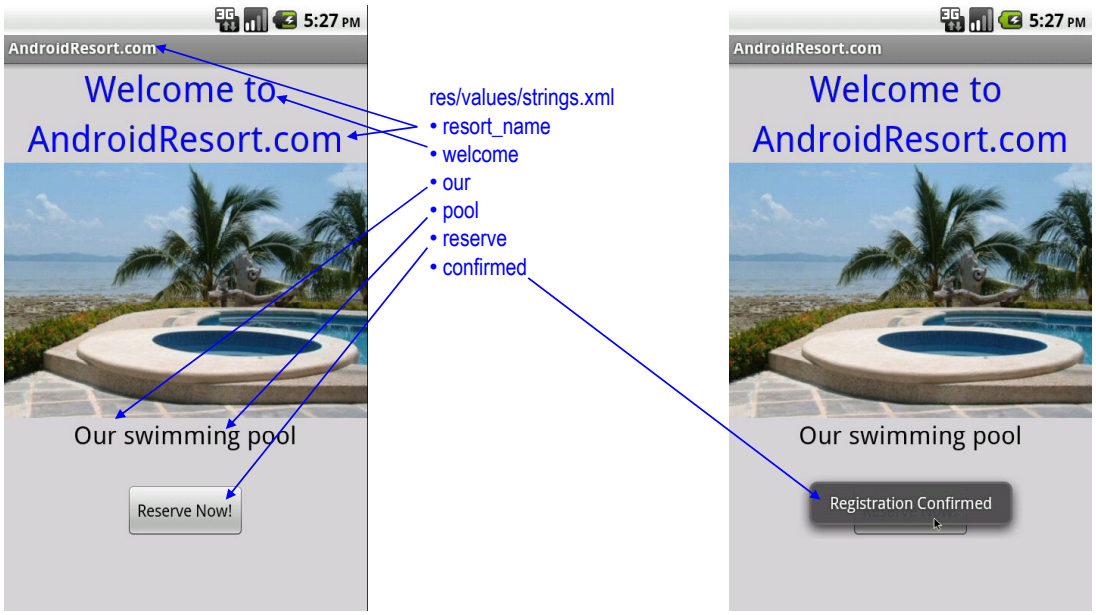
Uses the named strings via getString(R.string.string_name). Uses same syntax regardless of whether the string came from values/strings.xml or was overridden in values-es/strings.xml

Uses the named strings via @string/string_name. Uses same syntax regardless of whether the string came from values/strings.xml or was overridden in values-es/strings.xml

Defines *all* named strings. Applies to all languages except Spanish, and even one of the names applies to Spanish, since the Spanish version does not override resort_name.

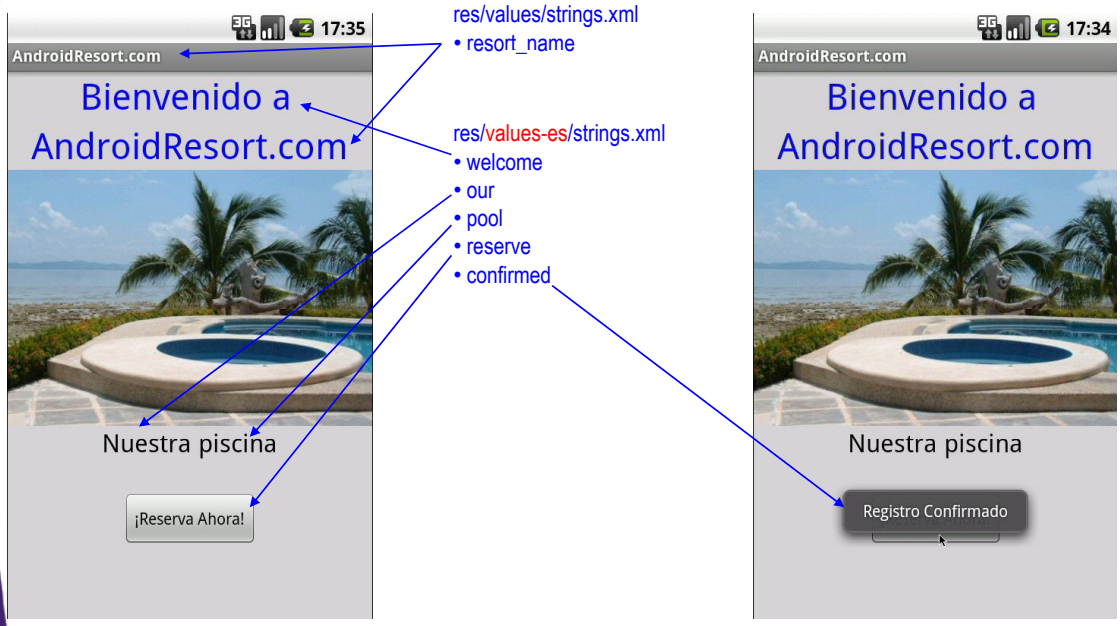
Overrides the named strings that are different in Spanish.

Results: English



This is the result not just for English, but for any language other than Spanish.

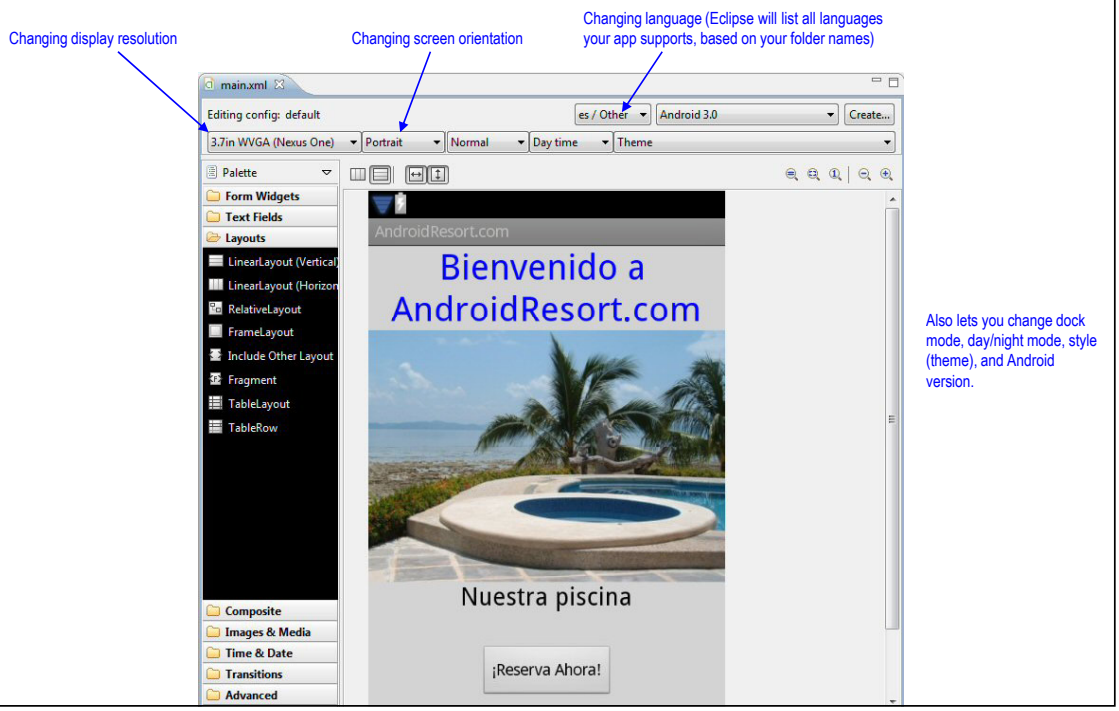
Results: Spanish



Best Practices

- **Provide unlocalized defaults for all values**
  - So if Locale is unexpected, it displays in default language
    - E.g., if English is main language, use res/values, not res/values-en. Also, test your app in unsupported Locale.
- **Use graphical layout editor for testing**
  - For layout file, it lets you interactively change language, screen orientation, display resolution, and more
- **Avoid changing layouts based on language**
  - Might be necessary in some cases (e.g., US English asks for given name first and family name second, whereas Indian English asks for them in opposite order). However, makes for hard-to-maintain code.
    - Consider putting the logic in Java code instead

Graphical Layout Editor in Eclipse (Editing main.xml)



Changing the Language Programmatically

- **Letting user change Locale**
  - Pros
    - Expected/recommended Android approach
    - All localization works gracefully
  - Cons
    - Can only use languages that device manufacturer supports for the entire OS
    - Requires user to take several steps they might not know
- **Changing Locale in your code**
  - Pros
    - Can use any language you want
    - Can let user set language in app with simple button
  - Cons
    - Not expected/recommended approach
    - Settings do not live across app restarts (including rotations!), so requires you to remember and reuse it.

Changing the Language Programmatically: Code

- **Steps**

```
Locale locale = new Locale("es");  // Language code
Locale.setDefault(locale);
Configuration config = new Configuration();
config.locale = locale;
context.getResources().updateConfiguration(config,
                                                null);
```

  - context above is reference to the main Activity

- **More details**
  - http://adrianvintu.com/blogengine/post/Force-Locale-on-Android.aspx

Region

Idea

- **Values can be specific to region**
  - Not only to general language
  - Examples
    - Words
      - US English: garbage truck
      - Australian English: rubbish lorry
    - Images
      - Flag of the country
    - Colors
      - Colors matching flag
    - Audio files
      - US English: Yankee Doodle
      - Australian English: Waltzing Matilda
- **Good news**
  - Android lets you define res/values-en-au, etc.
- **Bad news**
  - Most phones support few or no regional settings

Steps

- **Put default values in res/values**
  - As before. List all the names
- **Put general lang. values in res/values-*xx***
  - Where *xx* is ISO 639-1 language code as before
  - List *only* the names that change from the default language
- **Put regional values in res/values-*xx*-r*YY***
  - Where *xx* is the language code and *YY* is the country code (case insensitive). Note the "r" for "region".
    - Codes are specified by ISO 3166-1
      - http://en.wikipedia.org/wiki/ISO_3166-1_alpha-3
  - List *only* the names that change from the base language
- **Process**
  - Android will load from most general to most specific

Example: The Android Resort

- **Idea**
  - Add support for Mexican Spanish, which uses "alberca" instead of "piscina" for "swimming pool".
- **Approach**
  - res/values/strings.xml defines
    - resort_name, welcome, our, pool, reserve, confirmed
  - res/values-es/strings.xml defines
    - welcome, our, pool, reserve, confirmed
  - res/values-es-rMX/strings.xml defines
    - pool

Strings File: English/Other (res/values/strings.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="resort_name">AndroidResort.com</string>
    <string name="welcome">Welcome to </string>
    <string name="our">Our </string>
    <string name="pool">swimming pool</string>
    <string name="reserve">Reserve Now!</string>
    <string name="confirmed">Registration Confirmed</string>
</resources>
```

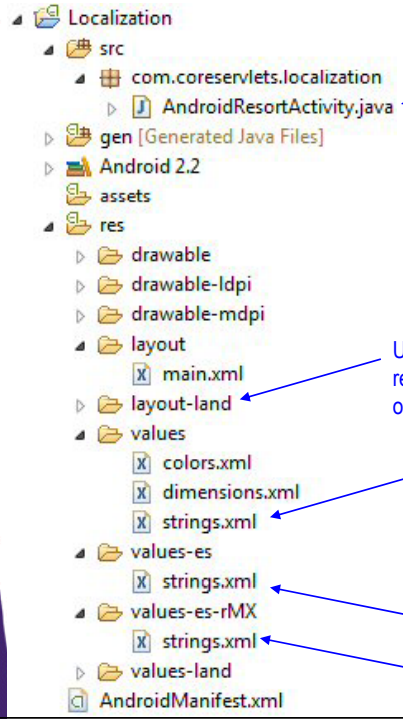

Strings File: General Spanish (res/values-es/strings.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="welcome">Bienvenido a </string>
    <string name="our">Nuestra </string>
    <string name="pool">piscina</string>
    <string name="reserve">¡Reserva Ahora!</string>
    <string name="confirmed">Registro Confirmado</string>
</resources>
```

Strings File: Mexican Spanish (res/values-es-rMX/strings.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="pool">alberca</string>
</resources>
```


Project Layout



Uses the named strings via getString(R.string.string_name). Uses same syntax regardless of whether the string came from values/strings.xml or was overridden in values-es/strings.xml or values-es-rMX/strings.xml.

Uses the named strings via @string/string_name. Uses same syntax regardless of whether the string came from values/strings.xml or was overridden in values-es/strings.xml or values-es-rMX/strings.xml.

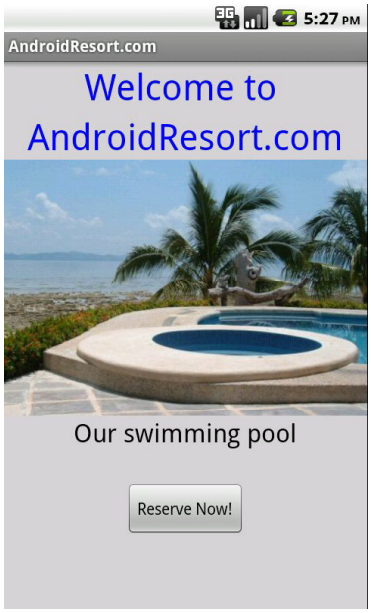
Defines *all* named strings. Applies to all languages except Spanish, and even one of the names applies to Spanish, since the Spanish versions do not override resort_name.

Overrides the named strings that are different in Spanish.

Overrides the named strings that are different in Mexican Spanish from in general Spanish.

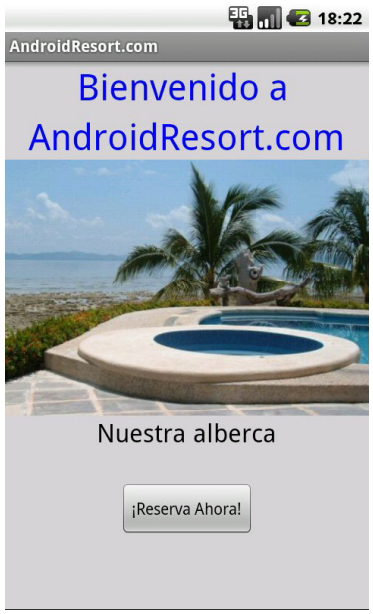
Results

Any language other than Spanish



General Spanish

Mexican Spanish



Screen Orientation

Overview

- **Idea**
  - Change the display based on whether the device is being held in portrait or landscape mode.
- **Resources that typically change**
  - Layout files (in res/layout)
  - Images (in res/drawable – image file or XML file)
  - Dimensions (in res/values, e.g., …/dimens.xml)
  - Video (in res/raw)
- **Resources that do not usually change**
  - Strings (in res/values, e.g., in res/values/strings.xml)
  - Colors (in res/values, e.g., in res/values/colors.xml)
  - Audio (in res/raw)

Steps

- **Make two folders: default and landscape**
  - res/layout and res/layout-land
    - You can do layout-port, but more common to use default
- **Define different layouts (of same name) in each**
  - For portrait mode
    - res/layout/main.xml (and maybe other names)
  - For landscape mode
    - res/layout-land/main.xml (and maybe other names)
- **Use similar approach for dimensions, images, etc.**
  - Use res/values/ and res/drawable for portrait mode and for things that are the same in both orientations.
  - Override in res/values-land and res/drawable-land

Steps (Continued)

- **In XML, refer to base dimensions or images**
  - textSize="@dimen/heading_size"
  - src="@ drawable/some_image"
    - No reference to folder or orientation. Android handles.
- **In Java, refer to base layout name**
  - setContentView(R.layout.main)
    - Android will provide the proper version automatically. For layout files, only one version applies. For values files, names are combined as before, with later values overriding earlier values of the same name.

Precedence of Folder Names

- **Idea**
  - Many folders with a single qualifier could apply (values, values-es, values-land, etc.).
  - You can also combine the names (e.g., values-es-land)
    - Language must be first, so use values-es-land, not values-land-es
  - So, which entries win?
- **Layout files and drawable files**
  - Only one file applies. Language entries win over screen orientation, so if there is layout-es and layout-land, and you are in both Spanish and landscape, layout-es wins
    - But, remember advice to avoid language-based layouts
- **Values files**
  - Many files may apply. They are loaded from least specific to most specific, and later values override earlier ones of the same name. Language is more specific than orientation.

How User Changes Screen Orientation

- **On phone (or other Android device)**
  - Physically rotate the device
- **On emulator**
  - Hit Control-F12 *or*
  - Hit 9 on the number keypad (Num Lock must be off)
- **In visual layout editor in Eclipse**
  - Choose Portrait or Landscape from the second combobox above the display

Example: The Android Resort

- **Idea**
  - Use two layouts: one for portrait and one for landscape
  - Change font sizes for heading depending on orientation
  - Keep support for English and Spanish
- **Approach**
  - res/layout/main.xml defines layout for portrait mode
  - res/layout-land/main.xml defines layout for landscape
  - res/values/dimens.xml defines font size for portrait
  - res/values-land/dimens.xml defines size for landscape
  - res/values, res/values-es, and res/values-es-rMX define strings and colors as shown previously

Layout File: Portrait Mode (res/layout/main.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://..."
              android:orientation="vertical" ...>
    <TextView android:text="@string/welcome"
              android:textSize="@dimen/heading_size" ... />
    <TextView android:text="@string/resort_name"
              android:textSize="@dimen/heading_size" ... />
    <ImageView android:src="@drawable/android_resort_pool"
               android:scaleType="fitXY" ... />
    <LinearLayout android:gravity="center_horizontal" ... >
        <TextView android:text="@string/our"
                  android:textSize="@dimen/body_size" ... />
        <TextView android:text="@string/pool"
                  android:textSize="@dimen/body_size" .../>
    </LinearLayout>
    <Button android:text="@string/reserve" .../>
</LinearLayout>
```


Layout File: Landscape Mode (res/layout-land/main.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://..." ... >
    <LinearLayout android:orientation="vertical" ... >
        <TextView android:text="@string/welcome"
                  android:textSize="@dimen/heading_size" ... />
        <TextView android:text="@string/resort_name"
                  android:textSize="@dimen/heading_size" ... />
        <Button android:text="@string/reserve" ... />
    </LinearLayout>
    <LinearLayout android:orientation="vertical" ... >
        <ImageView android:src="@drawable/android_resort_pool"
                   android:scaleType="fitXY" ... />
        <LinearLayout android:gravity="center_horizontal" ... >
            <TextView android:textSize="@dimen/body_size"
                      android:text="@string/our" .../>
            <TextView android:textSize="@dimen/body_size"
                      android:text="@string/pool" ... />
        </LinearLayout>
    </LinearLayout>
</LinearLayout>
```

Recall that horizontal is the default orientation

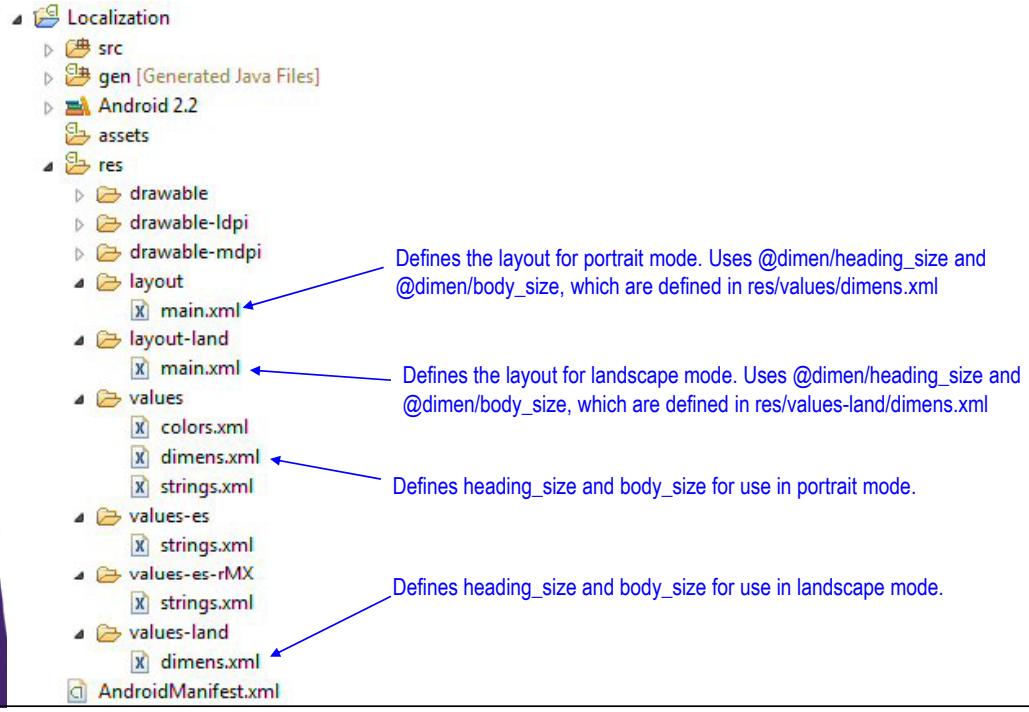
Dimensions File: Portrait Mode (res/values/dimens.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="heading_size">32dp</dimen>
    <dimen name="body_size">22dp</dimen>
</resources>
```

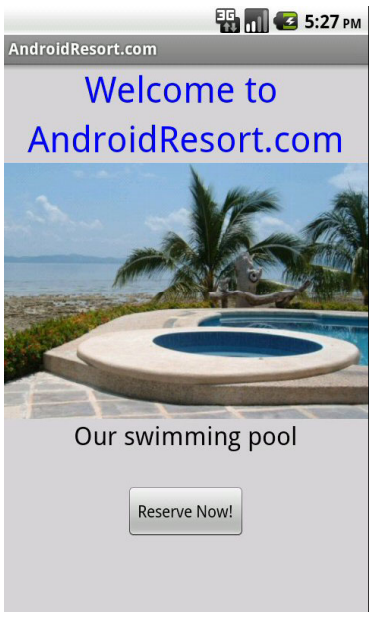

Dimensions File: Landscape (res/values-land/dimens.xml)

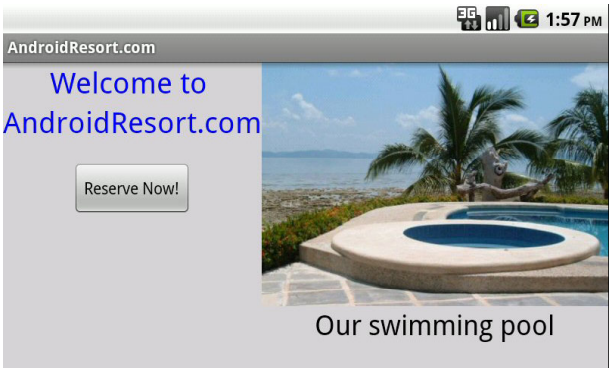
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="heading_size">26dp</dimen>
    <dimen name="body_size">24dp</dimen>
</resources>
```

Project Layout



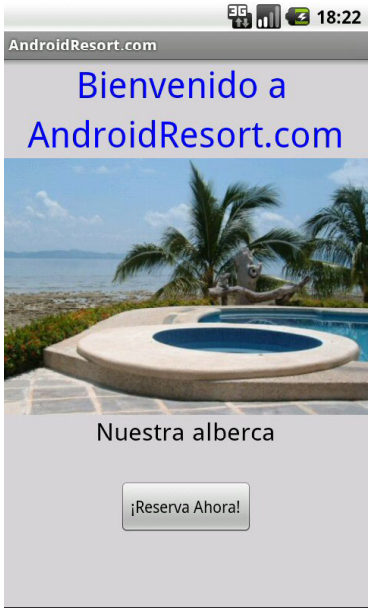
Results: English (Really Non-Spanish)





Results: General Spanish

Results: Mexican Spanish



Preventing Screen Rotations

- **Issue**
  - Screen rotations usually require a new layout
  - They also cause the app to be shutdown and restarted
    - Handling this will be discussed in the next lecture
- **Problem**
  - What if you do not have landscape layout?
  - Or have not yet handled shutdown and restart?
- **Solution**
  - Put an entry in AndroidManifest.xml saying that app runs only in portrait mode (or only in landscape mode)

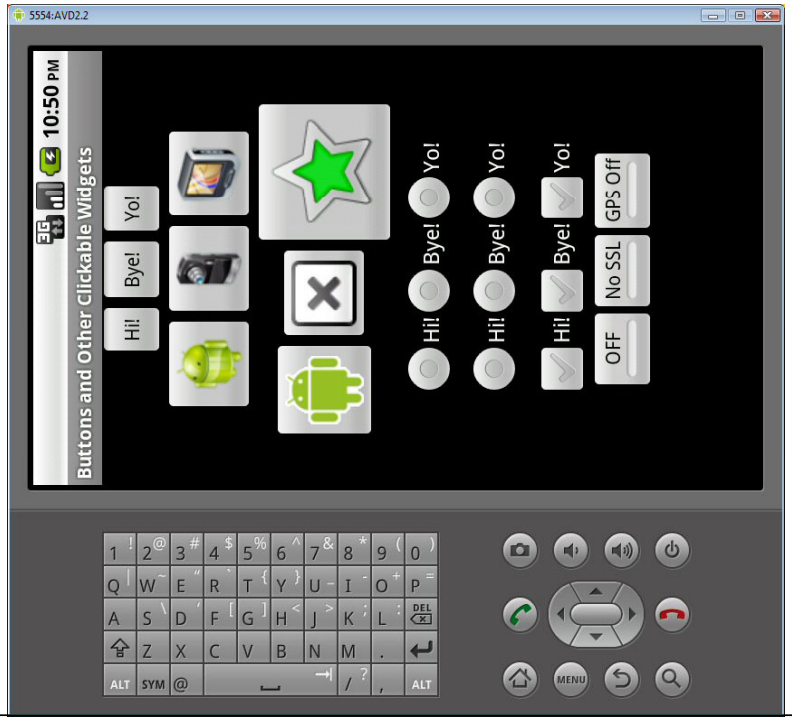
```
<activity android:name=".YourActivity"
          android:label="@string/app_name"
          android:screenOrientation="portrait">
```


Example: Screen from Widgets Lecture (Manifest File)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.coreservlets.widgets"
      android:versionCode="1"
      android:versionName="1.0">
    <uses-sdk android:minSdkVersion="8" />

    <application android:icon="@drawable/icon"
                 android:label="@string/app_name">
        ...
        <activity android:name=".ButtonActivity"
                  android:label="@string/button_app_name"
                  android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.VIEW" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
        ...
    </application>
</manifest>
```

Example: Results (in Landscape Mode)



Display Resolution

Overview

- **Idea**
  - Change images based on the resolution of the display
- **Resources that typically change**
  - Images (in res/drawable – image file or XML file)
  - Video (in res/raw)
- **Resources that do not usually change**
  - Dimensions
    - If you use *xx*dp, then it will scale to screen resolution
      - There are also small, normal, large, and xlarge qualifiers for screen size. Even with *xx*dp, you might want to adapt to those
    - But if you use *yy*px or *zz*in or *aa*mm, then you need to change the dimensions to match the resolution
  - Layout files, strings, colors, audio, etc.

Common Approaches

- **Alternative 1: Default plus two others**
  - drawable
    - Put images for high-res and extra-high-res (dots-per-inch) here
  - drawable-mdpi
    - Put images for medium-res (dots-per-inch) here
  - drawable-ldpi
    - Put images for low-res (dots-per-inch) here
- **Alternative 2: Three res-specific folders**
  - drawable-hdpi
    - Put images for high-res and extra-high-res (dots-per-inch) here
  - drawable-mdpi
    - Put images for medium-res (dots-per-inch) here
  - drawable-ldpi
    - Put images for low-res (dots-per-inch) here

Meanings of Resolution Qualifiers

- **xhdpi (extra-high dpi)**
  - 320 dots per inch
  - But, Android uses best match, so if you put images in drawable-hdpi and it is extra-high, images are still used.
    - Another common option is to use a single set of images for both extra-high and high, and put them in drawable
- **hdpi (high dpi)**
  - 240 dots per inch
- **mdpi (medium dpi)**
  - 160 dots per inch
- **ldpi (low dpi)**
  - 120 dots per inch

Best Practices

- **For small icons (e.g., for ImageButton)**
  - Use multiple versions
  - Realize that the size will be slightly different on different displays, so leave enough room to compensate
- **For larger images (e.g., for ImageView)**
  - Use multiple versions
  - Scale the image if it is larger than the available space

```
<ImageView android:src="@drawable/some_name"
           android:layout_height="wrap_content"
           android:layout_width="wrap_content"
           android:adjustViewBounds="true"
           android:scaleType="fitXY"/>
```

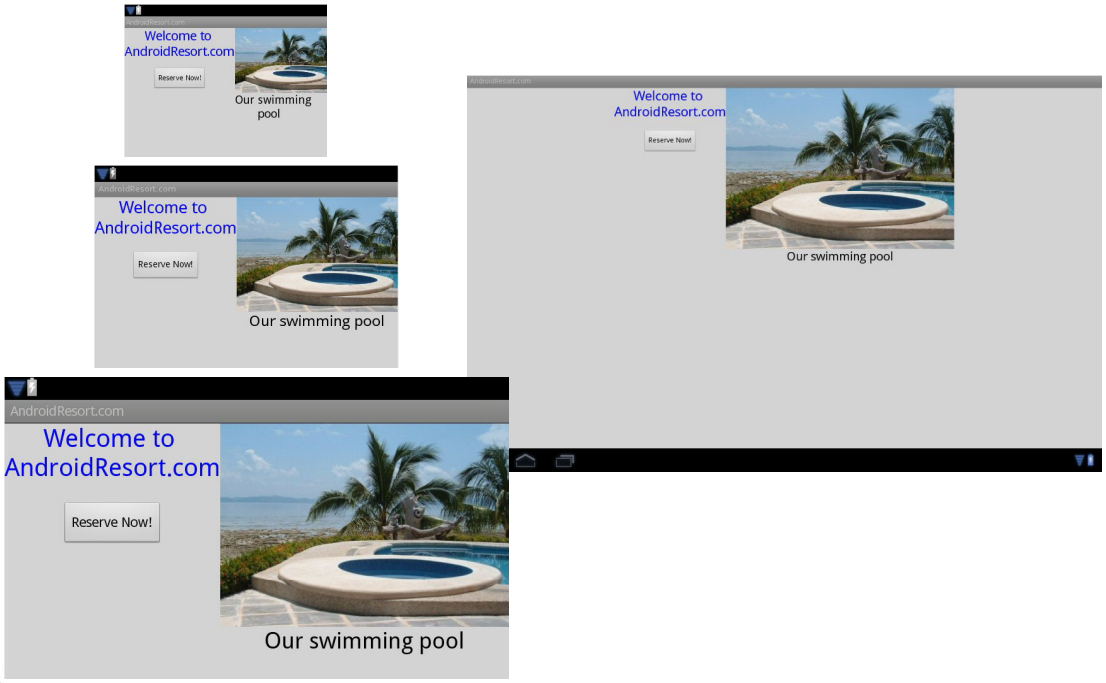
Precedence of Folder Names

- **Idea**
  - You may have many folders that apply
    - drawable, drawable-mdpi, drawable-es, drawable-es-land-hdpi…
  - So, which entries win?
- **Precedence rules**
  - Language is highest precedence
    - drawable-es wins over drawable-land or drawable-mdpi
  - Orientation is second highest
    - drawable-land wins over drawable-hdpi
  - Density is last (of these three)
    - But, try to chain the qualifiers so there is never a conflict.
      - drawable-land-hdpi, drawable-land-mdpi, etc.

Example: The Android Resort

- **Idea**
  - Have three versions of the swimming pool image: for low, medium, and high (or extra-high) resolution devices
- **Approach**
  - drawable/android_resort_pool.jpg
    - For high-res or extra-high-res devices
  - drawable-mdpi/android_resort_pool.jpg
    - For medium-res devices
  - drawable-ldpi/android_resort_pool.jpg
    - For low-res devices
  - Scale the image so that if it is still too big, it will not be cropped

Results (Choosing Different Densities in Visual Editor)



Wrap-Up

Configuration Types

- **Most common qualifiers (in order of precedence)**
  - Language and region
    - es, es-rMX, etc.
  - Screen size
    - small, normal, large, xlarge
  - Orientation
    - port, land
  - Resolution (density)
    - ldpi, mdpi, hdpi, xhdpi
  - Touchscreen type
    - notouch, stylus, finger
  - Full list
    - http://developer.android.com/guide/topics/resources/providing-resources.html
- **Using multiple qualifiers**
  - Order of qualifiers must match precedence order above
    - values-es-port-ldpi, not values-es-ldpi-port or values-port-es-ldpi

More Reading

- **Developer's Guide: Localization**
  - http://developer.android.com/guide/topics/resources/localization.html
- **Developer's Guide: Application Resources**
  - http://developer.android.com/guide/topics/resources/
- **Tutorial: Hello L10N**
  - http://developer.android.com/resources/tutorials/localization/index.html
- **Chapter: Localization**
  - From *Android in Action* by Ableson et al

Summary

- **Language**
  - Provide multiple versions of strings file
    - Maybe images, colors, and sounds as well
    - Default folder should provide values for all names
- **Screen orientation**
  - Provide multiple versions of layout files
    - Maybe dimension files and images as well
- **Screen density**
  - Provide multiple versions of images (drawable files)
    - Maybe videos and dimension files as well
- **Best practices**
  - Test in all combinations, esp. unexpected languages
  - Use Eclipse graphical layout editor for initial testing

Questions?