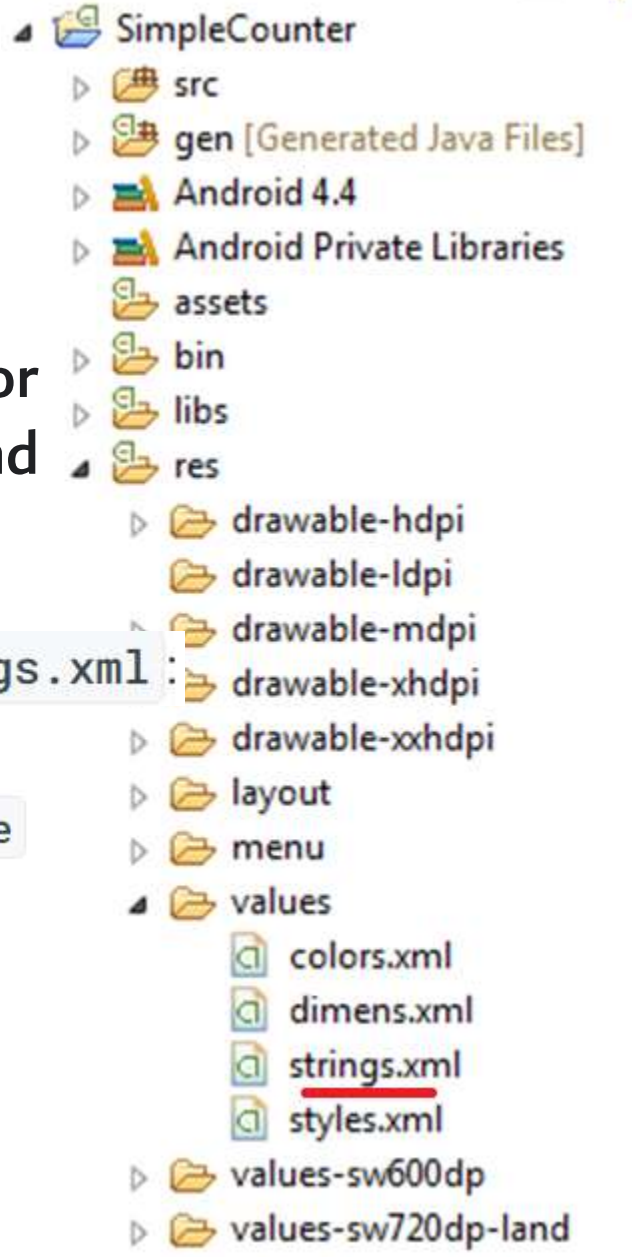
05_Android Resources

Week 05

Strings

- **A string resource provides text strings for your application with optional text styling and formatting. It is defined in XML.**



file location: XML file saved at `res/values/strings.xml`:

`res/values/filename.xml`

The filename is arbitrary. The `<string>` element's `name` is used as the resource ID.

compiled resource datatype:

Resource pointer to a `String`.

resource reference:

In Java: `R.string.string_name`

In XML: `@string/string_name`

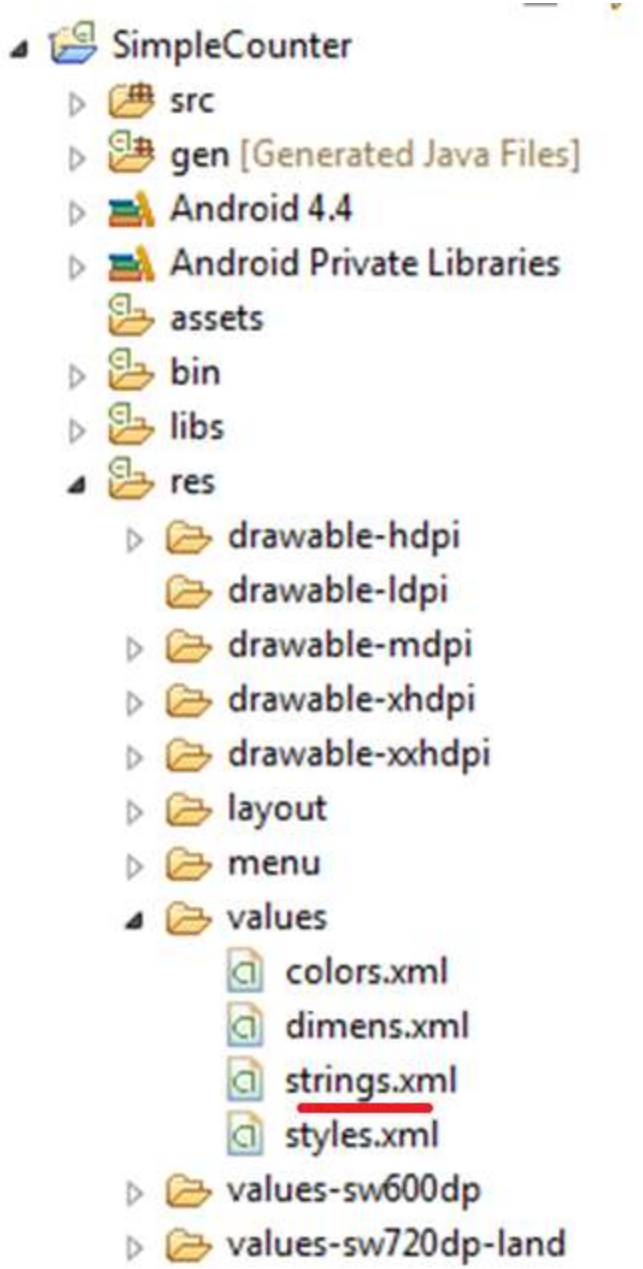
Strings

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

```xml
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello" />
```

Kotlin Java

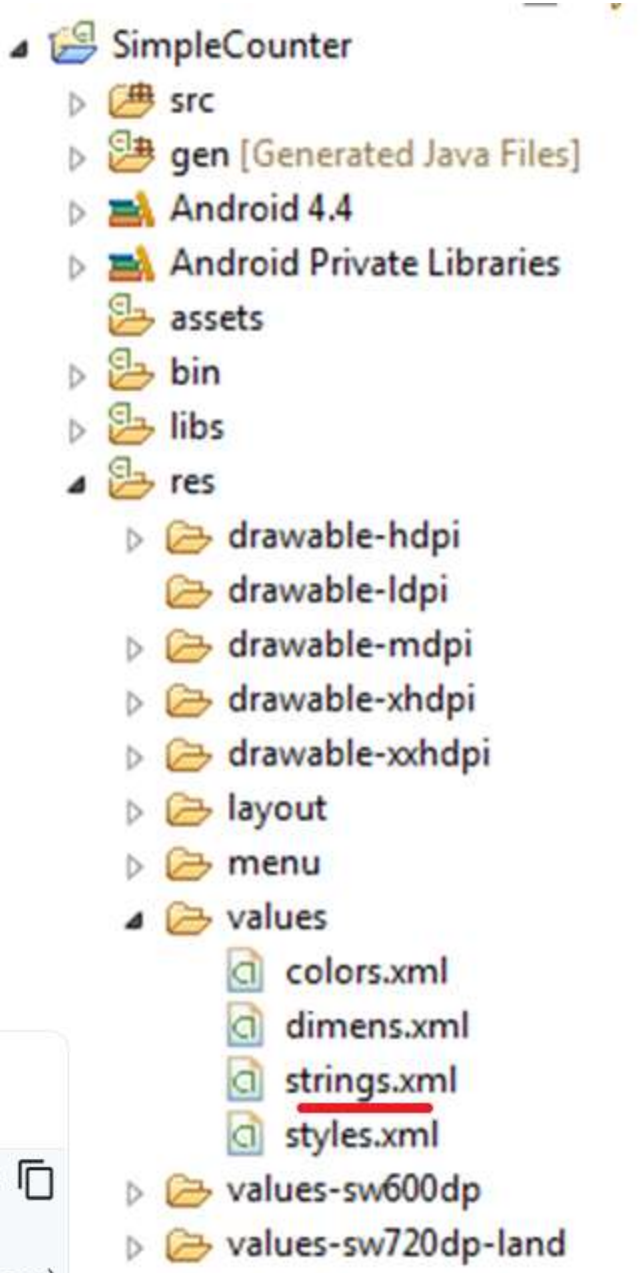
```java
String string = getString(R.string.hello);
```



You can use either `getString(int)` or `getText(int)` to retrieve a string. `getText(int)` retains any rich text styling applied to the string.

Array of Strings

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="planets_array">
        <item>Mercury</item>
        <item>Venus</item>
        <item>Earth</item>
        <item>Mars</item>
    </string-array>
</resources>
```



Kotlin Java

```java
Resources res = getResources();
String[] planets = res.getStringArray(R.array.planets_array)
```

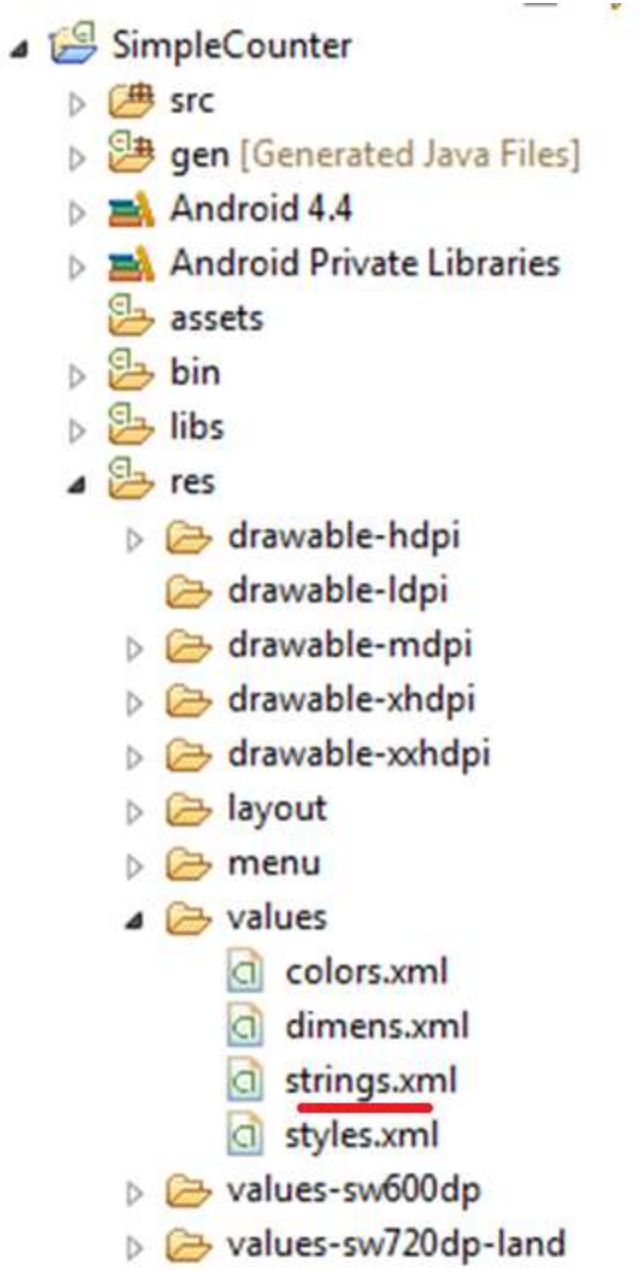
Array of Strings

In your strings.xml define:

```
<string-array name="array_name">
<item>Array Item One</item>
<item>Array Item Two</item>
<item>Array Item Three</item>
</string-array>
```

In your layout:

```
<Spinner
        android:id="@+id/spinner"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:drawSelectorOnTop="true"
        android:entries="@array/array_name"
    />
```



Color

- A color value defined in XML.

file location:

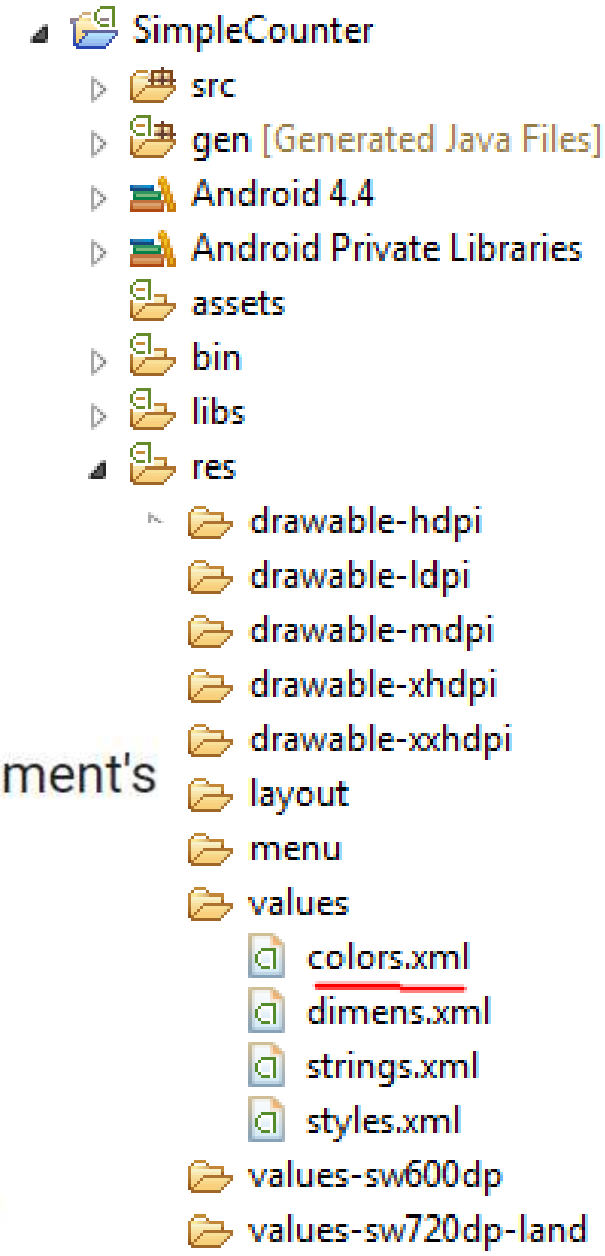
`res/values/colors.xml`

The filename is arbitrary. The `<color>` element's `name` will be used as the resource ID.

resource reference:

In Java: `R.color.color_name`

In XML: `@[package:]color/color_name`



Color

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="opaque_red">#f00</color>
    <color name="translucent_red">#80ff0000</color>
</resources>
```

```
<TextView
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:textColor="@color/translucent_red"
android:text="Hello"/>
```

Java

```
Resources res = getResources();
int color = res.getColor(R.color.opaque_red);
```

Dimension

- A dimension value defined in XML.
- A dimension is specified with a number followed by a unit of measure. For example: 10px, 2in, 5sp. The following units of measure are supported by Android:
  - dp, sp, pt, px, mm and in.

https://developer.android.com/guide/topics/resources/more-resources#Dimension

Dimension

example:

XML file saved at `res/values/dimens.xml` :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="textview_height">25dp</dimen>
    <dimen name="textview_width">150dp</dimen>
    <dimen name="ball_radius">30dp</dimen>
    <dimen name="font_size">16sp</dimen>
</resources>
```

Dimension

Java

```
Resources res = getResources();
float fontSize = res.getDimension(R.dimen.font_size);
```

This layout XML applies dimensions to attributes:

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

dp

- **dp (Density-independent Pixels)**
  - An abstract unit that is based on the physical density of the screen.
  - 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. The ratio of dp-to-pixel will change with the screen density, but not necessarily in direct proportion.

dp vs px

- Always use dp to pecify dimension of Views.
- Using dp units (instead of px units) is a simple solution to making the view dimensions in your layout resize properly for different screen densities. In other words, it provides consistency for the real-world sizes of your UI elements across different devices.

sp

- **sp (**Scale-independent Pixels **)**
  - This is like the dp unit, but it is also scaled by the user's font size preference. It is recommend you use this unit when specifying font sizes, so they will be adjusted for both the screen density and the user's preference.

Use **sp** for text size…because but it is scaled by the user's font size preference.

Use **dp** for everything else.

EditText (android:inputType)

For example, if you'd like an input method for entering a phone number, use the `"phone"` value:

```
<EditText
    android:id="@+id/phone"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/phone_hint"
    android:inputType="phone" />
```

Figure 1. The `phone` input type.

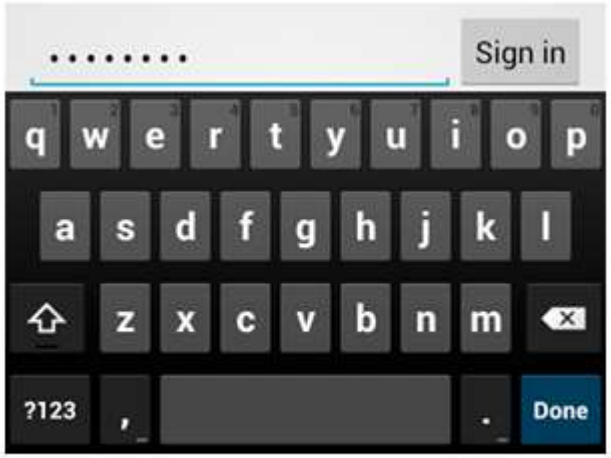


Figure 2. The `textPassword` input type.

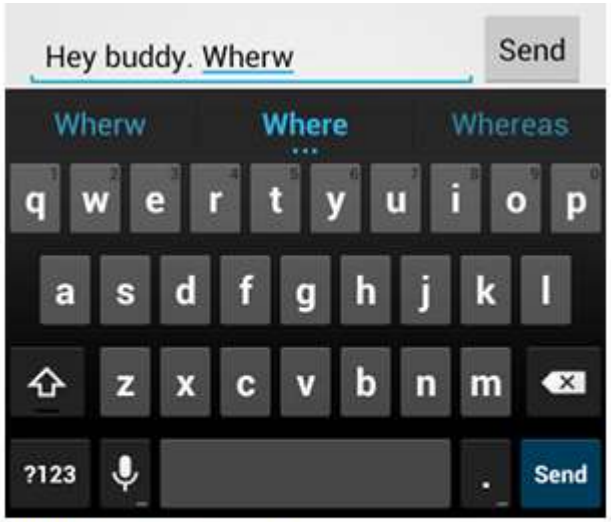


Figure 3. Adding `textAutoCorrect` provides auto-correction for misspellings.

Logcat

- Displays system messages, such as when a garbage collection occurs, and messages that you added to your app with the Log class.
- It displays messages in real time and keeps a history so you can view older messages.
- To display just the information of interest, you can create filters, modify how much information is displayed in messages, set priority levels, display messages produced by app code only, and search the log. By default, logcat shows the log output related to the most recently run app only.
- When an app throws an exception, logcat shows a message followed by the associated stack trace containing links to the line of code.

Log Class

- API for sending log output.

The `Log` class allows you to create log entries in your code that display in the logcat tool. Common logging methods include:

- `Log.v(String, String)` (verbose)
- `Log.d(String, String)` (debug)
- `Log.i(String, String)` (information)
- `Log.w(String, String)` (warning)
- `Log.e(String, String)` (error)

Log Class

Tip: A good convention is to declare a `TAG` constant in your class:

```
private static final String TAG = "MyActivity";
```

and use that in subsequent calls to the log methods.

Tip: Don't forget that when you make a call like

```
Log.v(TAG, "index=" + i);
```

```
Toast toast = Toast.makeText(getApplicationContext(), "Your Message", Toast.LENGTH_LONG)
toast.show();
```