INDICE

1.1. Introducción

Como habíamos comentado anteriormente la buena práctica es tener todas las cadenas de caracteres con texto para etiquetas, botones y en general, cualquier elemento de la interfaz de usuario, fuera del código de nuestra app. Deberían estar almacenadas en un archivo XML llamado **/res/values/**strings.xml.
Este fichero deberá contener todas las cadenas de caracteres en el idioma que consideres que será el de uso mayoritario. Es lo que se llama "recurso por defecto".

Si queremos internacionalizar nuestra aplicación, es decir, seleccionar el idioma del usuario dependiendo de la zona geográfica en la que se encuentre o del idioma que tenga personalizado en su dispositivo móvil, solo necesitamos que crear una nueva car - peta llamada **values- ??** donde ?? es el idioma que deseamos. Por ejemplo para el inglés tendríamos **/res/values-en**, **/res/va- lues-es** para español, ...
Como acabamos de decir, el **idioma predeterminado** será el almacenado *en /res/values*.

- Solo se deben cambiar los valores de las constantes, los identificadores deben ser los mismos.

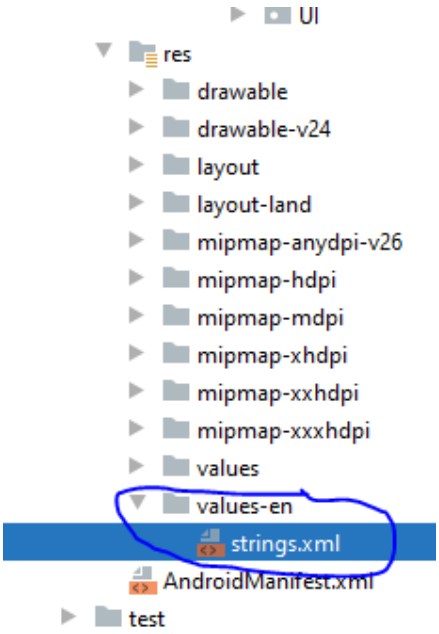
Referencias:

- **Localización**: https://developer.android.com/guide/topics/resources/localization.html

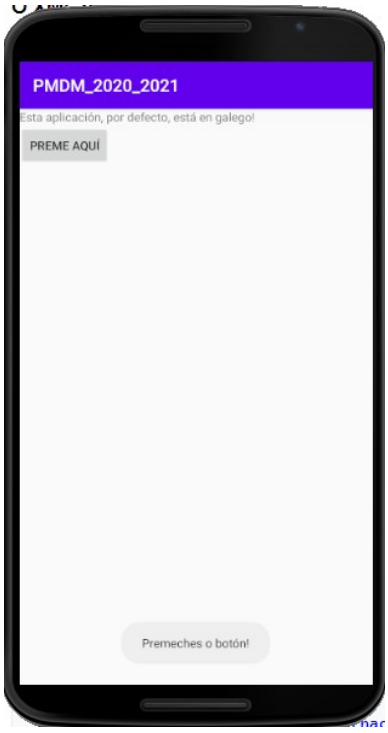
1.2. Caso práctico

Crea un nuevo proyecto: **U2_10_International**
En **/res/values** definimos las cadenas de texto en gallego (idioma por defecto) y definimos esas mismas cadenas en inglés en **/ res/values-en**.

Internacionalización



Carpetas de recursos: values e values-en. Cada uno con su fichero de cadena.



La aplicación está en gallego, tanto en el layout como en código Toast (otro tipo de control, que veremos con más detalle en la siguiente unidad del curso. Se trata de un pequeño mensaje pop-up).

Para inglés sería:

El layout y el código Java es el mismo para ambos idiomas.

A continuación los distintos códigos para esta app:

strings.xml de /res/values/

```
<resources>
    <string name="app_name">U2_10_Internacional</string>
    <string name="action_settings">Settings</string>
    <string name="idioma">Esta aplicación, por defecto, está en galego!</string>
    <string name="boton">Preme aquí</string>
    <string name="mensaxe_toast">Premeches o botón!</string>
</resources>
```

strings.xml de /res/values-en

```
<resources>
    <string name="app_name">U2_10_Internacional</string>
    <string name="action_settings">Settings</string>
    <string name="idioma">This app is by default in galician language!</string>
    <string name="boton">Click here</string>
    <string name="mensaxe_toast">You have clicked on the button!</string>
</resources>
```

El código XML del layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".U2_41_Internacional.U2_41_Internacional">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
            android:text="@string/idioma" />

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/boton"
            android:onClick="onBotonClick" />

    </LinearLayout>
```

Explicación de las líneas 10 y 15: ahora trabajamos con constantes, no podemos poner el texto directamente.

El código Java

```
package inesvalino.iessanclemente.pmdm_2021_2022.U2_10_Internacional;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

import inesvalino.iessanclemente.pmdm_2021_2022.R;

public class U2_10_Internacional extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_u2_10_internacional);
    }

    public void onBotonClick(View v) {
        Toast.makeText(this, R.string.mensaxe_toast, Toast.LENGTH_SHORT).show();
    }
}
```

Explicación de la línea 24: observad como el *Toast* también coge la cadena de texto de los recursos tipo string.

1.3. Consideraciones a tener en cuenta

- Cuando queremos utilizar un texto con una función que no acepta un recurso de clase R de tipo R.string.resource_name, debemos hacer uso de la función **getResources().getString("R.string.resource_name")** la cual devolverá una cadena (String) en el idioma adecuado. A través de esta función (getResources ()) podemos recuperar, llamando a diferentes métodos, todo tipo de recursos almacenados en /res/.

  Nota 1: La función getString ("R.string.resource_name") también podría usarse directamente.

  Nota 2: Recordad que hasta el momento las Activities creadas tienen una etiqueta 'puesta a mano' por nosotros en el archivo **AndroidManifiest.xml**. Deben estar referenciados en un archivo externo de tipo 'values' => 'string'. Recordad cambiarlo en vuestro proyecto (solo debe llevar una etiqueta a nivel de proyecto, no a nivel de activity).

- Los sufijos que estamos usando (-land; -es) se pueden combinar entre sí y con otros muchos diferentes.

  Este orden de prefijos aparece en el siguiente enlace.

Por ejemplo:

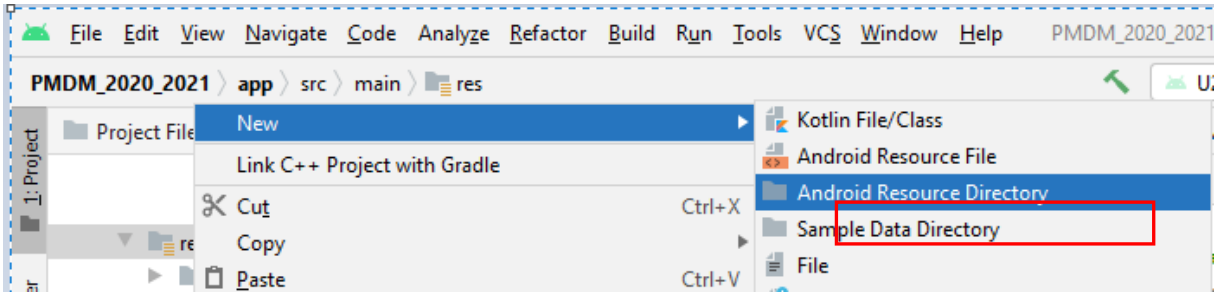
- res / values-en-rUS / strings.xml
- res / values-en-rUK / strings.xml

donde el segundo sufijo añadido se refiere al inglés "americano" (en-rUS) o al "inglés" del Reino Unido (en-rUK).

Nota 3: La letra r se refiere a la región.

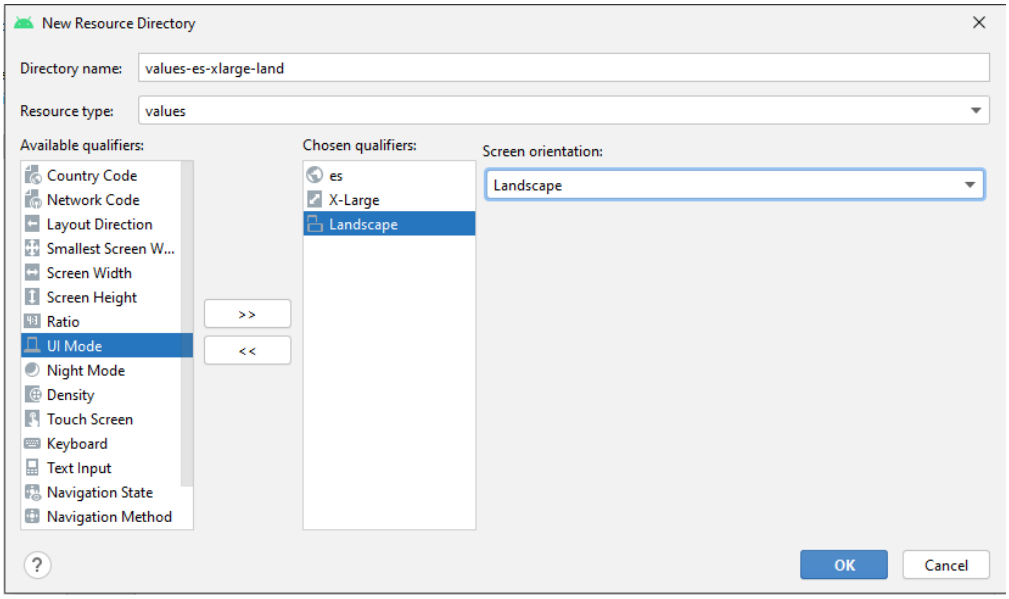
Para facilitar nuestro trabajo, Android Studio cuenta con una pantalla donde podemos indicar qué 'características' queremos que tenga nuestro archivo de recursos y agregar automáticamente los sufijos necesarios para nosotros.

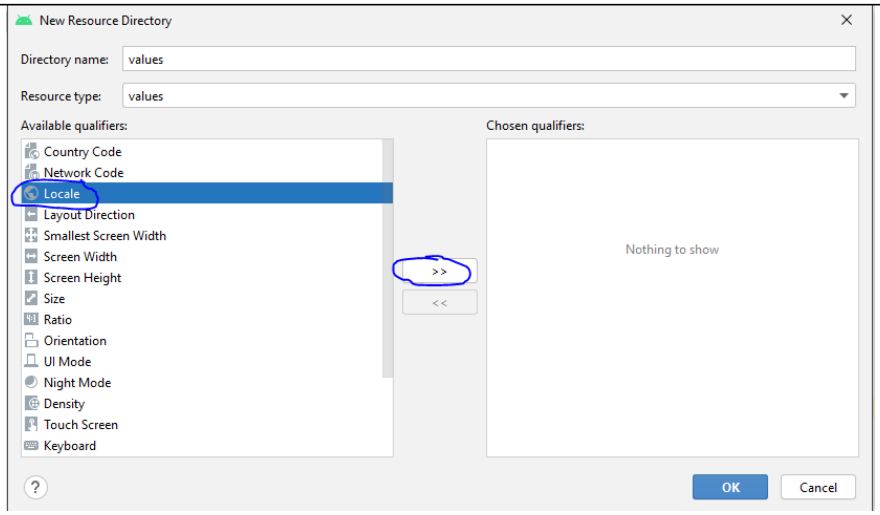
Creando un nuevo archivo de recursos



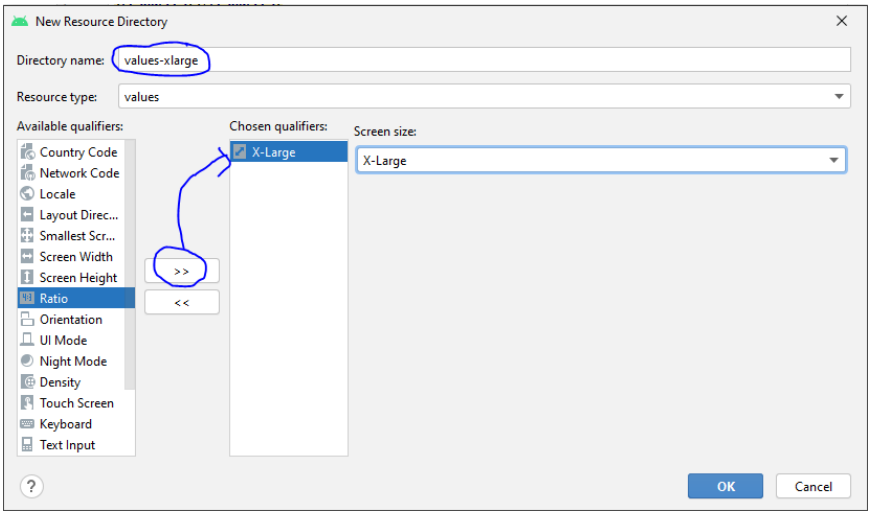
Haced clic con el botón derecho del ratón en la carpeta **/res/** o en **/res /values/**. Si lo hacemos sobre **/res/** aparecerá un combo extra en el que tendremos que elegir el tipo *'values'*.

En este ejemplo crearemos un recurso (archivo xml) para cuando la pantalla del dispositivo sea x-large, el idioma esté en español y el dispositivo tenga una orientación horizontal.



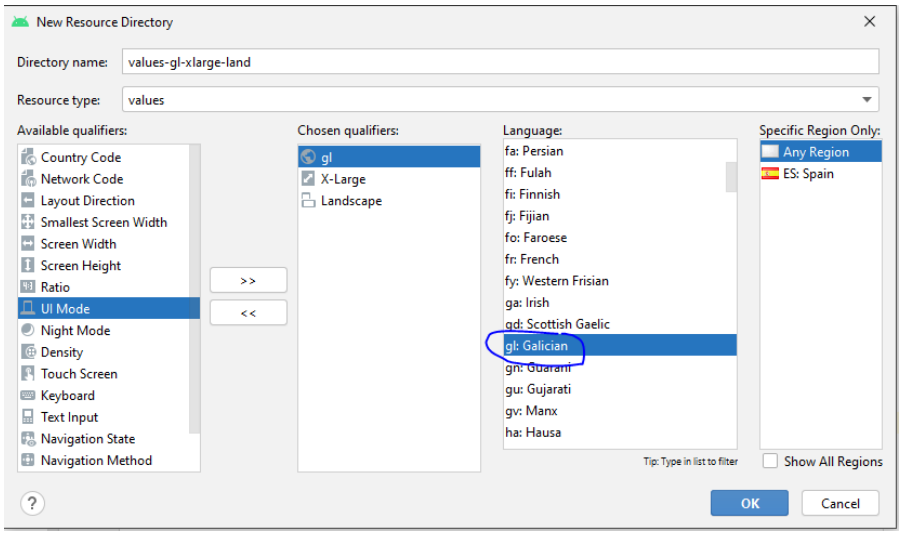


Debemos pasar la propiedad 'SIZE' desde la lista de la izquierda a la derecha (haciendo clic en el botón con la dirección derecha).



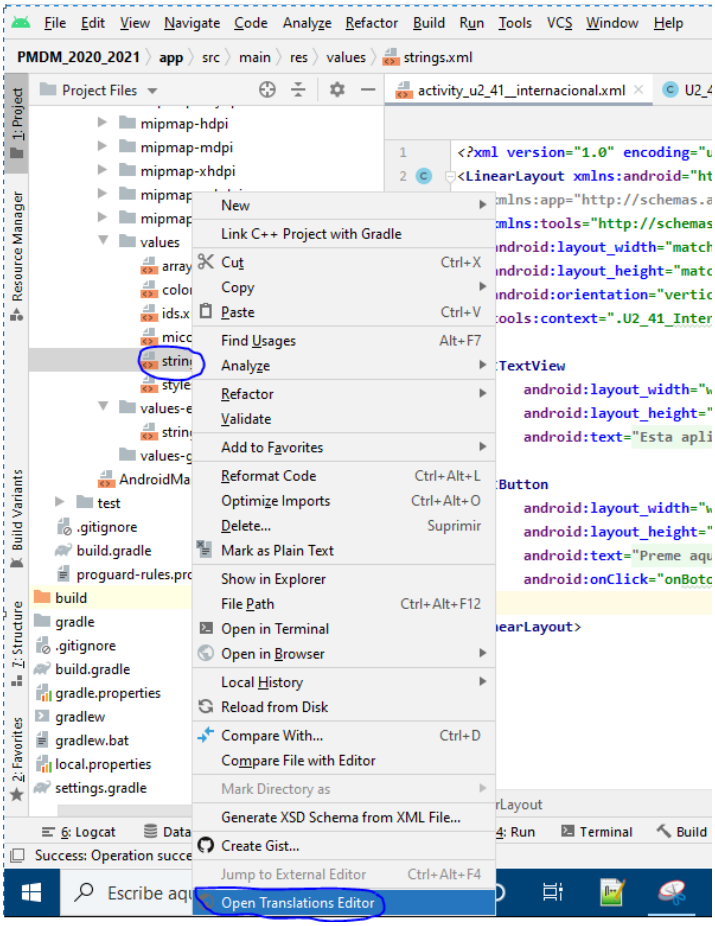
En concreto, en este ejemplo, estamos creando un recurso de tipo de layout para pantallas de 7 a 10 pulgadas (tamaño x-large), por lo que tendríamos que crear un layout en una carpeta "layout-xlarge". Al hacerlo, se puede ver cómo aparece el nombre con el sufijo correcto en la parte inferior. Además, se creará una carpeta física con el sufijo.

Nota 4: A partir de la versión Lollipop (API 21, android 5.0) ya se incluye el idioma gallego.

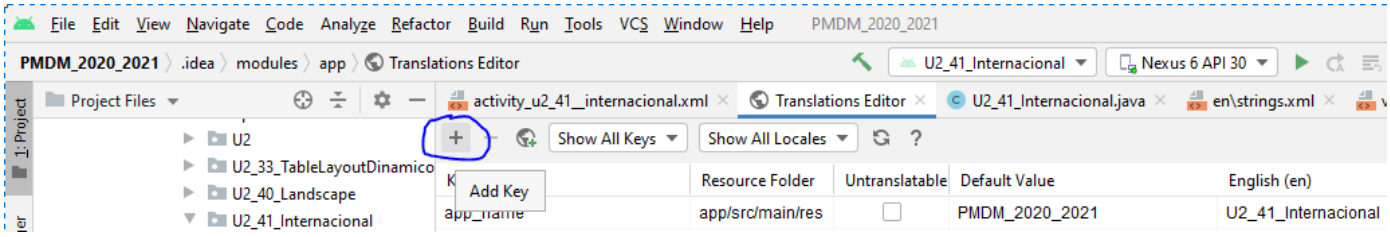


También tenemos la opción de crear estos archivos de idioma directamente en un **editor**. A través de esta herramienta podemos editar cualquier archivo en **/res/values** y asociar gráficamente diferentes idiomas e ingresar su traducción:

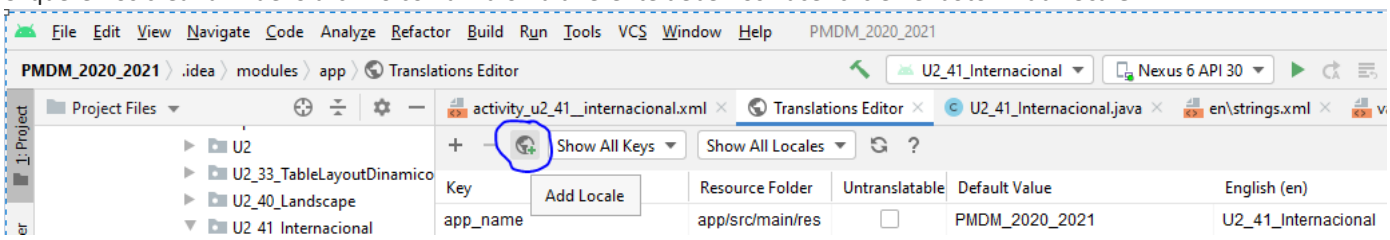
Creando nuevos archivos de idiomas con Translation Editor



Al hacer clic en el botón **+** creamos una nueva entrada en el archivo XML.



Si queremos crear un nuevo archivo con un idioma diferente debemos hacer clic en el botón **Add Locale**.



Una vez agregado el idioma, aparecerá una nueva columna para cada una de las entradas en el archivo. El archivo se creará en la carpeta del idioma agregado y podemos indicar si alguna de las entradas no tiene traducción (opción Untranslatable) lo

que indicará que esta entrada tomará el valor del archivo predeterminado en cualquier idioma que estemos.