Chapter – 12
Localization

Localization

- Localization is the process of making an app available in multiple languages.
- To sell an application worldwide successfully, you must target it to a broad and diverse audience.
- English enjoys a broad acceptance and practice, and therefore you can expect that an English language application will sell well globally, but why give up on sales opportunities to the broader non-English-speaking Android market?
- In its simplest approach, this means translation into multiple languages. But there's more! Beyond language translation, an application needs to properly handle dates and times, number and currency formats, and for some applications unit of measure.
- The reasons for localizing an application are manifold. Your application may be bound for some cultural reasons to a specific region.

String resource

- Keeping your labels and other bits of text outside the main source code of your application is generally considered to be a very good idea. [strings.xml]
- In particular, it helps with Internationalization (I18N) and localization (L10N).
- Even if you are not going to translate your strings to other languages, it is easier to make corrections if all the strings are in one spot instead of scattered throughout your source code.

Android Way

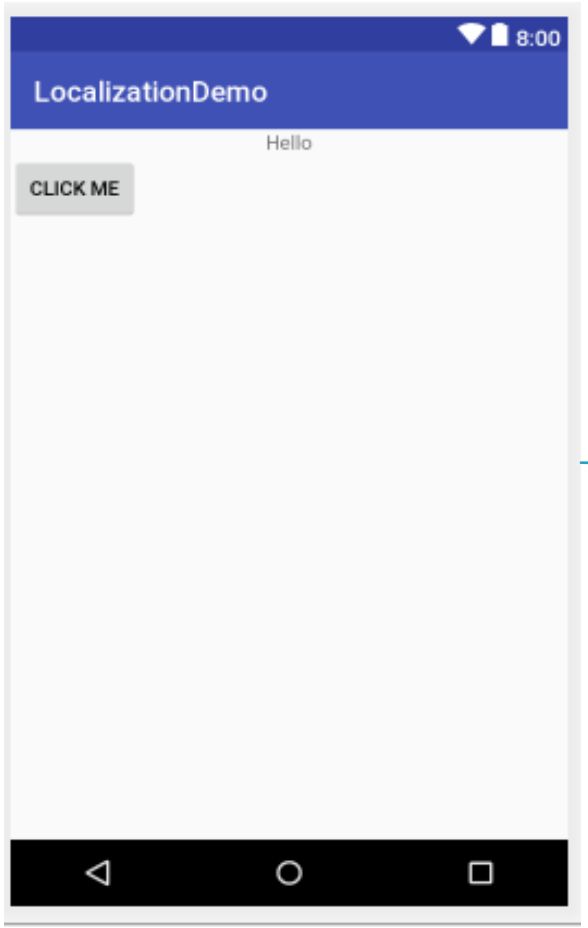
- The way Android currently handles this is by having multiple resource directories, with the criteria for each embedded in their names.
- Suppose, for example, you want to support strings in both English and Spanish.
- Normally, for a single-language setup, you would put your strings in a file named res/values/strings.xml.
- To support both English and Spanish, you would create two folders, res/values-en/ and res/values-es/, where the value after the hyphen is the ISO 639–1two-letter code for the language you want.
- Your English-language strings would go in res/values-en/strings.xml and the Spanish ones in res/ values-es/strings.xml. Android will choose the proper file based on the user's device settings.

Android Way

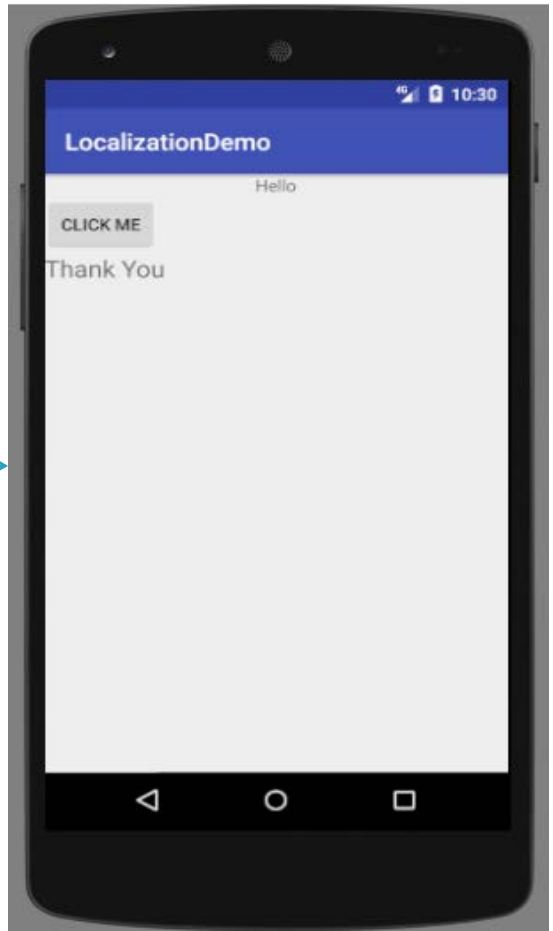
- An even better approach is for you to consider some language to be your default, and put those strings in res/values/strings.xml.
- Then, create other resource directories for your translations (e.g., res/values-es/strings.xml for Spanish).
- Android will try to match a specific language set of resources; failing that, it will fall back to the default of res/values/strings.xml.
- This way, if your app winds up on a device with a language that you do not expect, you at least serve up strings in your chosen default language.
- Otherwise, if there is no such default, you will wind up with a ResourceNotFoundException, and your application will crash.

Hands on Example(English)

This for the default Language(English), you have store everything in String Resources. To make localization here we use French.

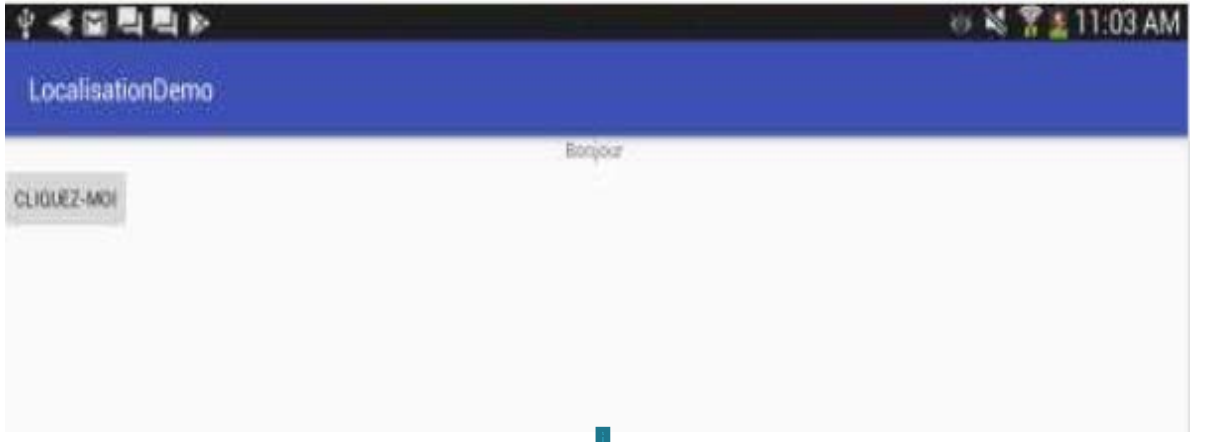


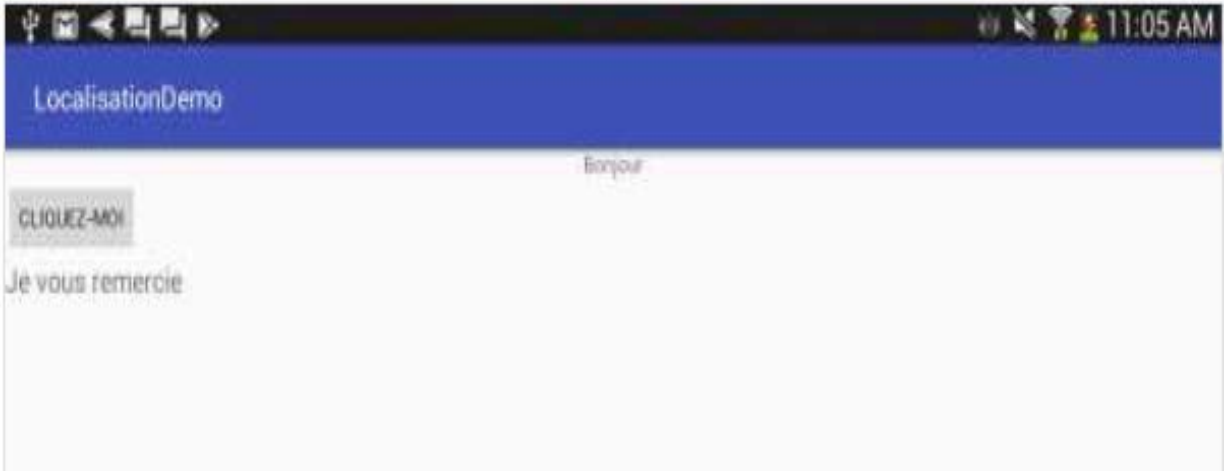
After CLICK ME



Hands on Example(France)

Once you changed the Language settings to France, you will get the following output





activity_main.xml

LocalizationDemo › app › src › main › res › layout › activity_main.xml

Android

- app
  - manifests
  - java
    - com.example.rmohanraj.
      - MainActivity
    - com.example.rmohanraj.
    - com.example.rmohanraj.
  - res
    - drawable
    - layout
      - activity_main.xml
    - mipmap
    - values
      - colors.xml (2)
      - strings.xml (2)
        - strings.xml
        - strings.xml (fr)
      - styles.xml (2)
- Gradle Scripts

activity_main.xml | MainActivity.java | values\strings.xml | gradle.properties

LinearLayout

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="fill_parent"
4      android:layout_height="fill_parent"
5      android:orientation="vertical" >
6      <TextView
7          android:layout_width="fill_parent"
8          android:layout_height="wrap_content"
9          android:gravity="center"
10         android:text="Hello" />
11     <Button
12         android:id="@+id/button1"
13         android:layout_width="wrap_content"
14         android:layout_height="wrap_content"
15         android:onClick="displayResponse"
16         android:text="Click Me" />
17     <TextView
18         android:id="@+id/textView1"
19         android:layout_width="match_parent"
20         android:layout_height="wrap_content"
21         android:text=""
22         android:textSize="20sp" />
23 </LinearLayout>
```

Design | Text

6: Logcat | Android Profiler | 4: Run | TODO | Terminal | 0: Messages

String Resources

strings.xml

```
<resources>
    <string name="app_name">LocalizationDemo</string>
    <string name="title">Localization</string>
    <string name="hello">Hello </string>
    <string name="button"> Click Me</string>
    <string name="response">Thank You</string>
</resources>
```

strings.xml(fr)

```
<resources>
    <string name="app_name">LocalisationDemo</string>
    <string name="title">Localisation</string>
    <string name="hello">Bonjour </string>
    <string name="button"> Cliquez-moi</string>
    <string name="response">Je vous remercie</string>
</resources>
```

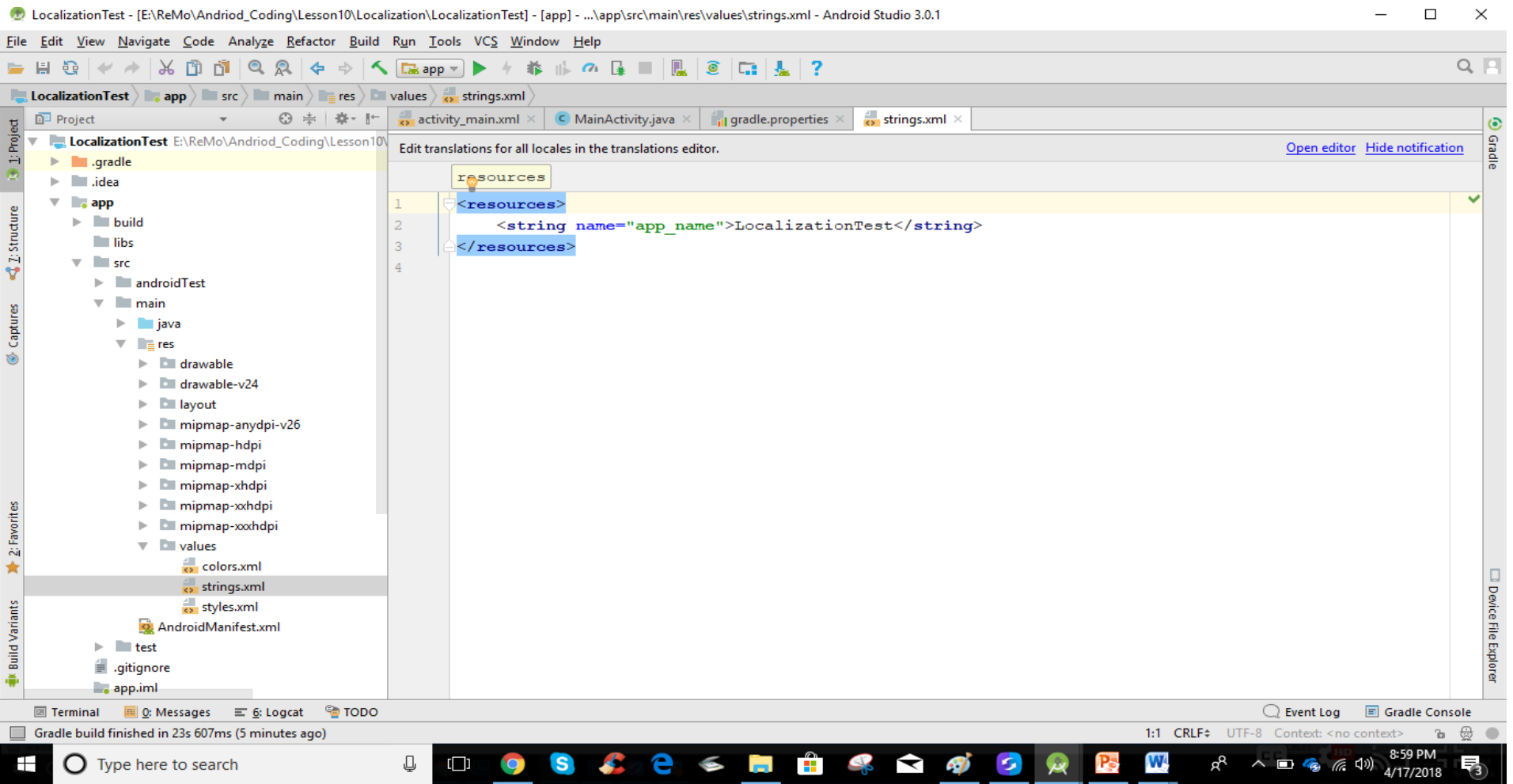
MainActivity.java

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    public void displayResponse(View v) {
         textview1.setText(R.string.response);
    }
}
```

Demo Code: DaysApp, LocalizationDynammic

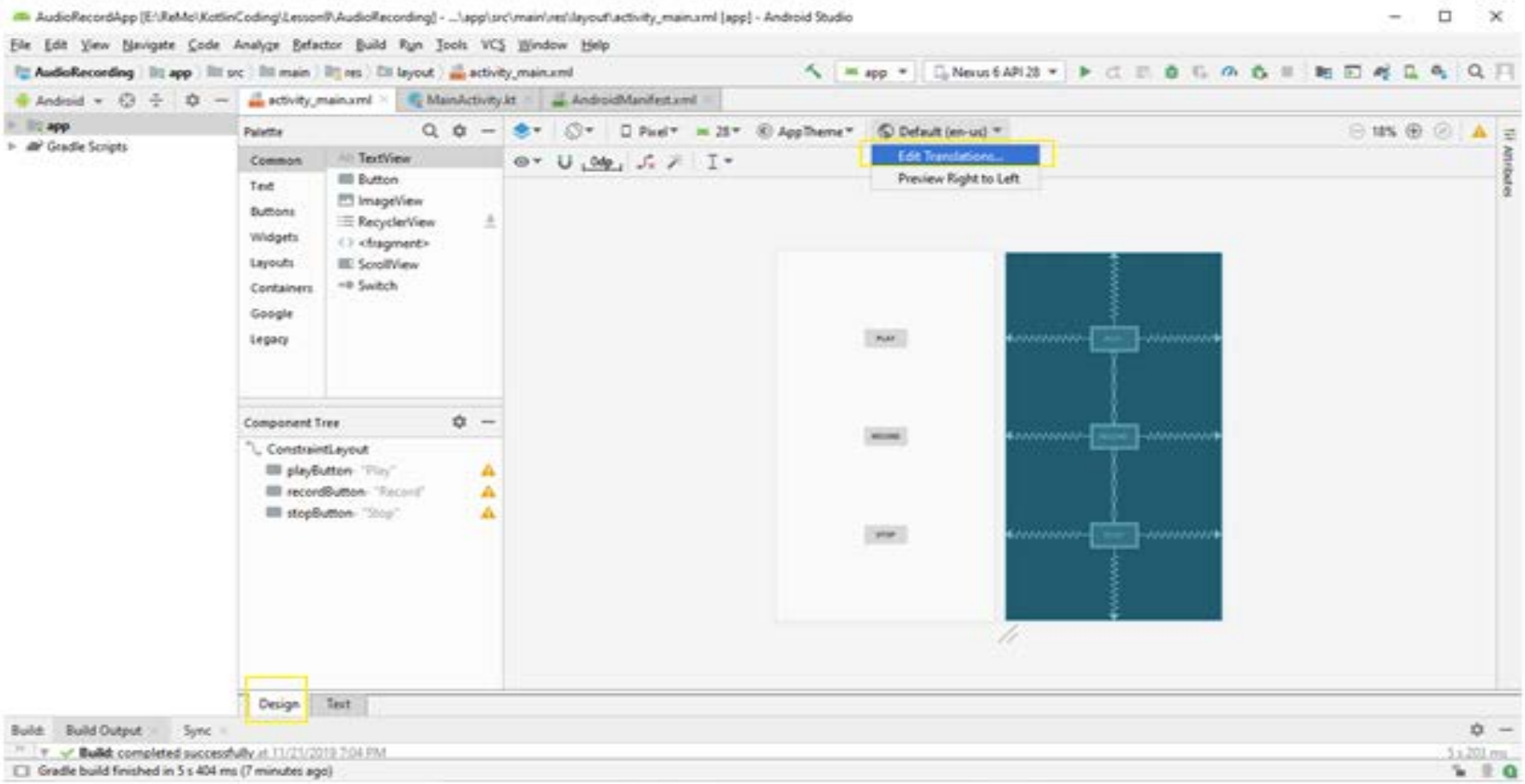
Localization Step by Step Screens

Method 1 : Enter Data into String resource directly.



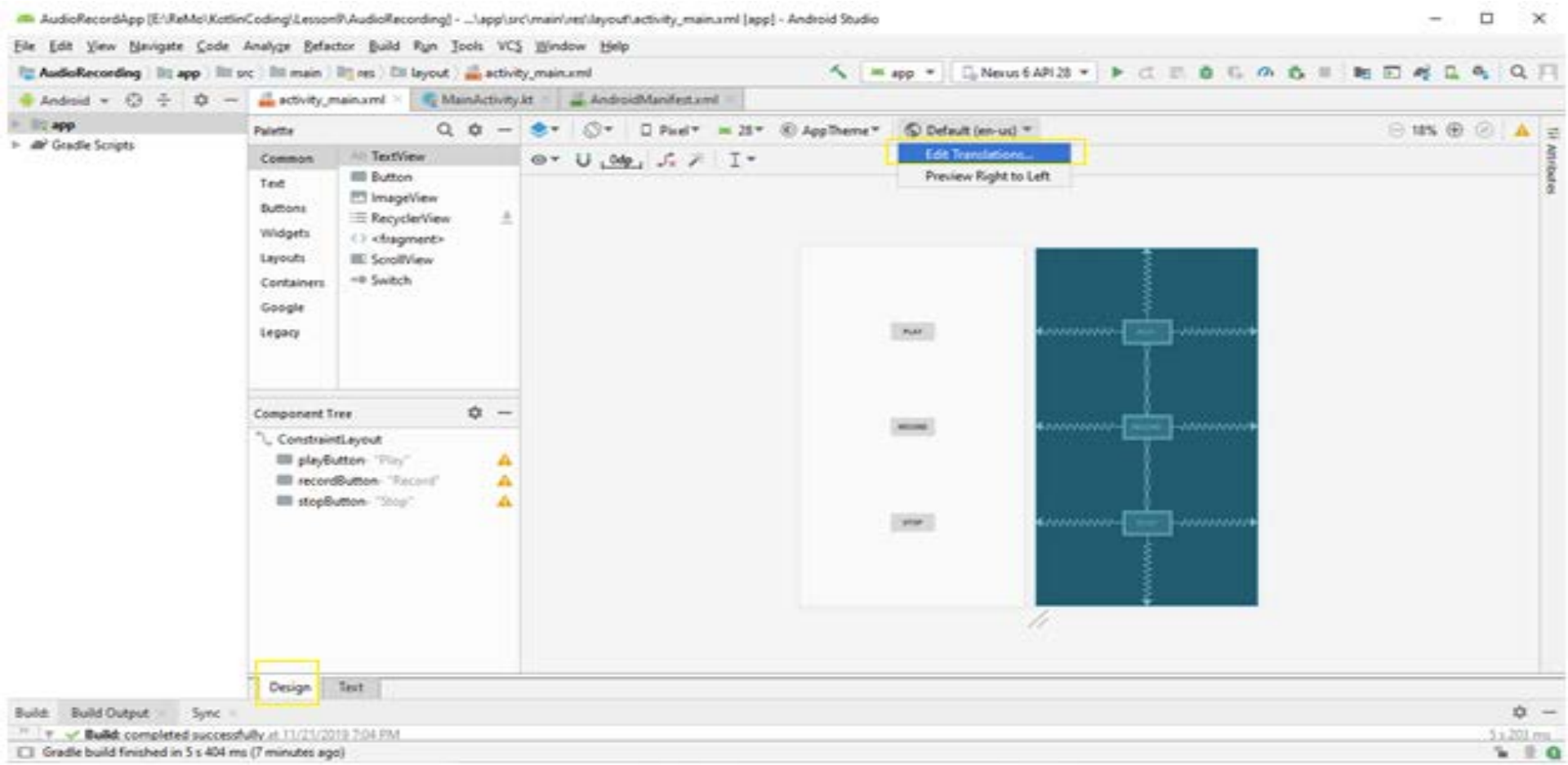
Localization Step by Step Screens

Method 2 : Step 1 : Go to your Layout Design View and Select Default(en-us) → Edit Translation



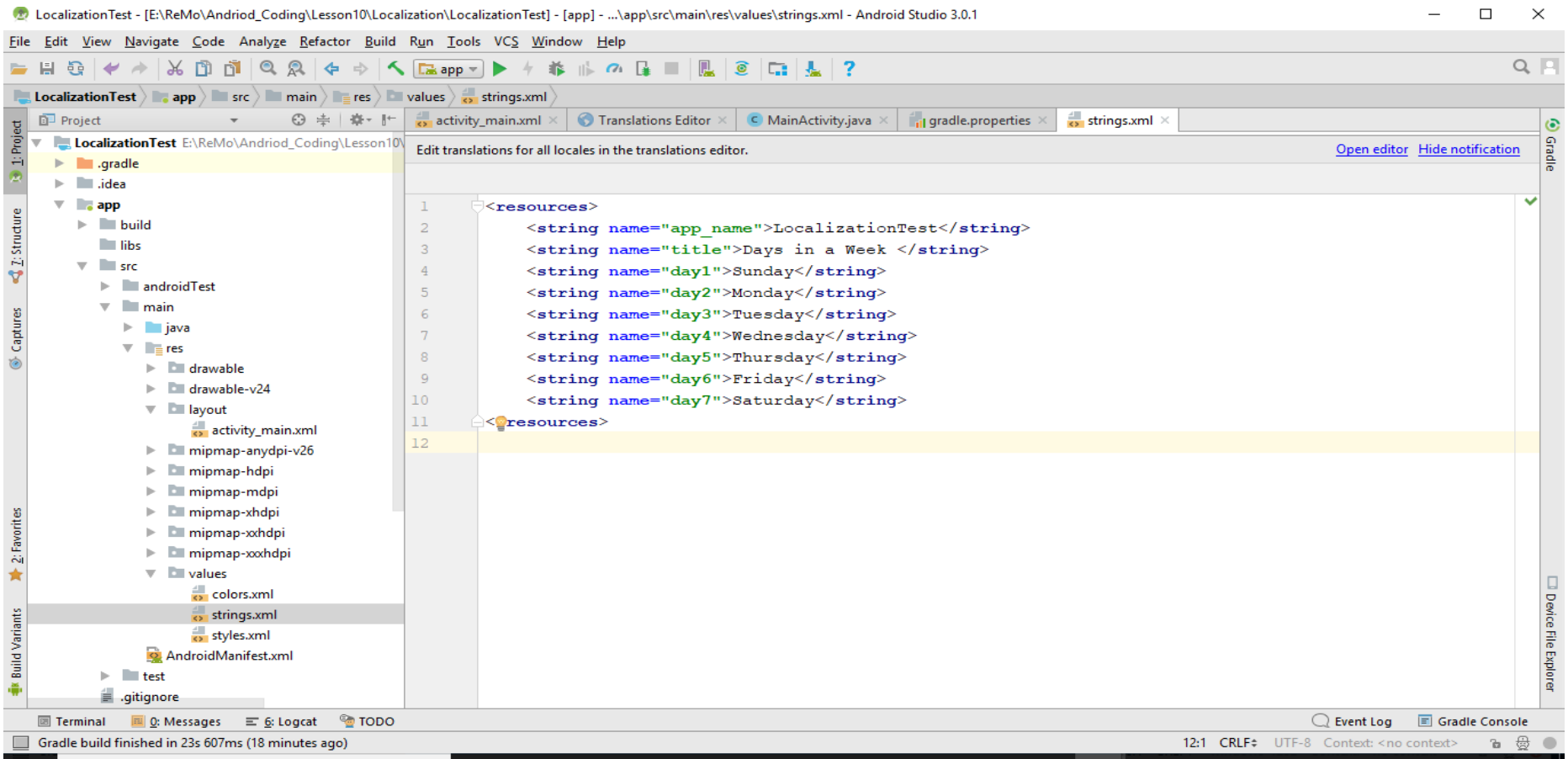
Localization Step by Step Screens

Step 2 :Click + to add more String inputs as a Key/Value Pair into your resource.



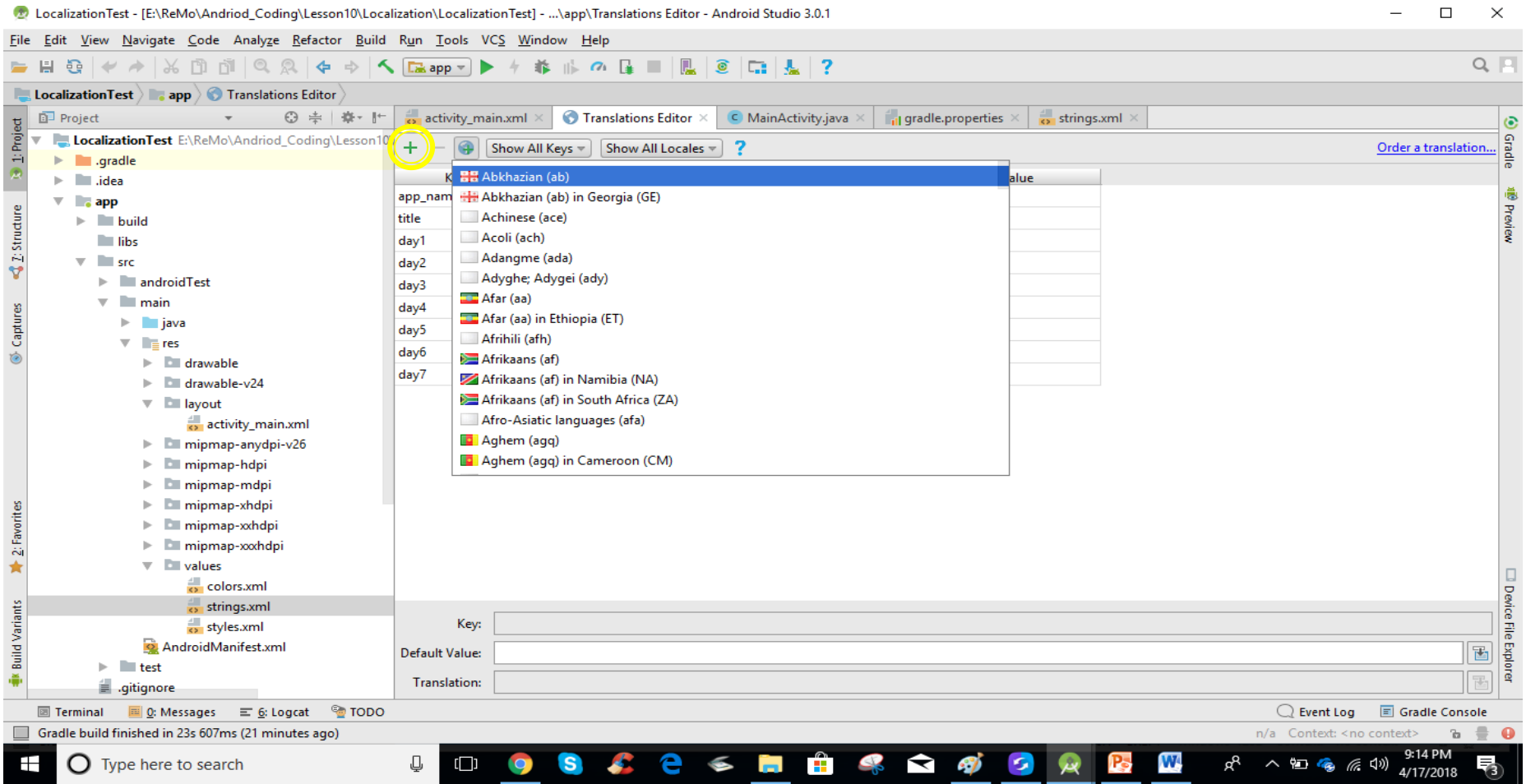
Localization Step by Step Screens

Step 3 :If you click on your strings.xml, you will notice with all the values.



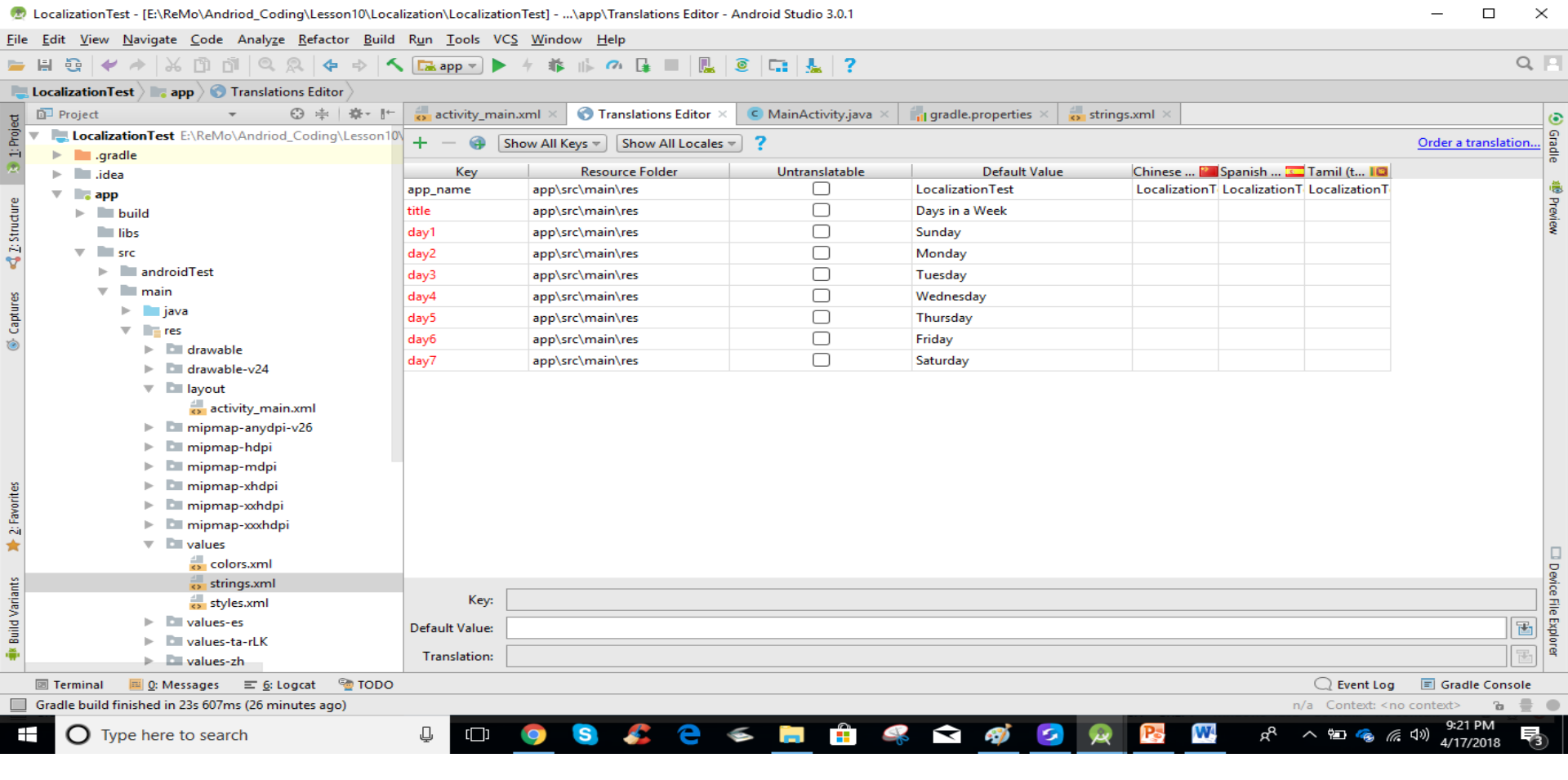
Localization Step by Step Screens

Step 4 : To add Locale support, go back to your Translation Editor and click the (+)highlighted icon and choose the desired language.



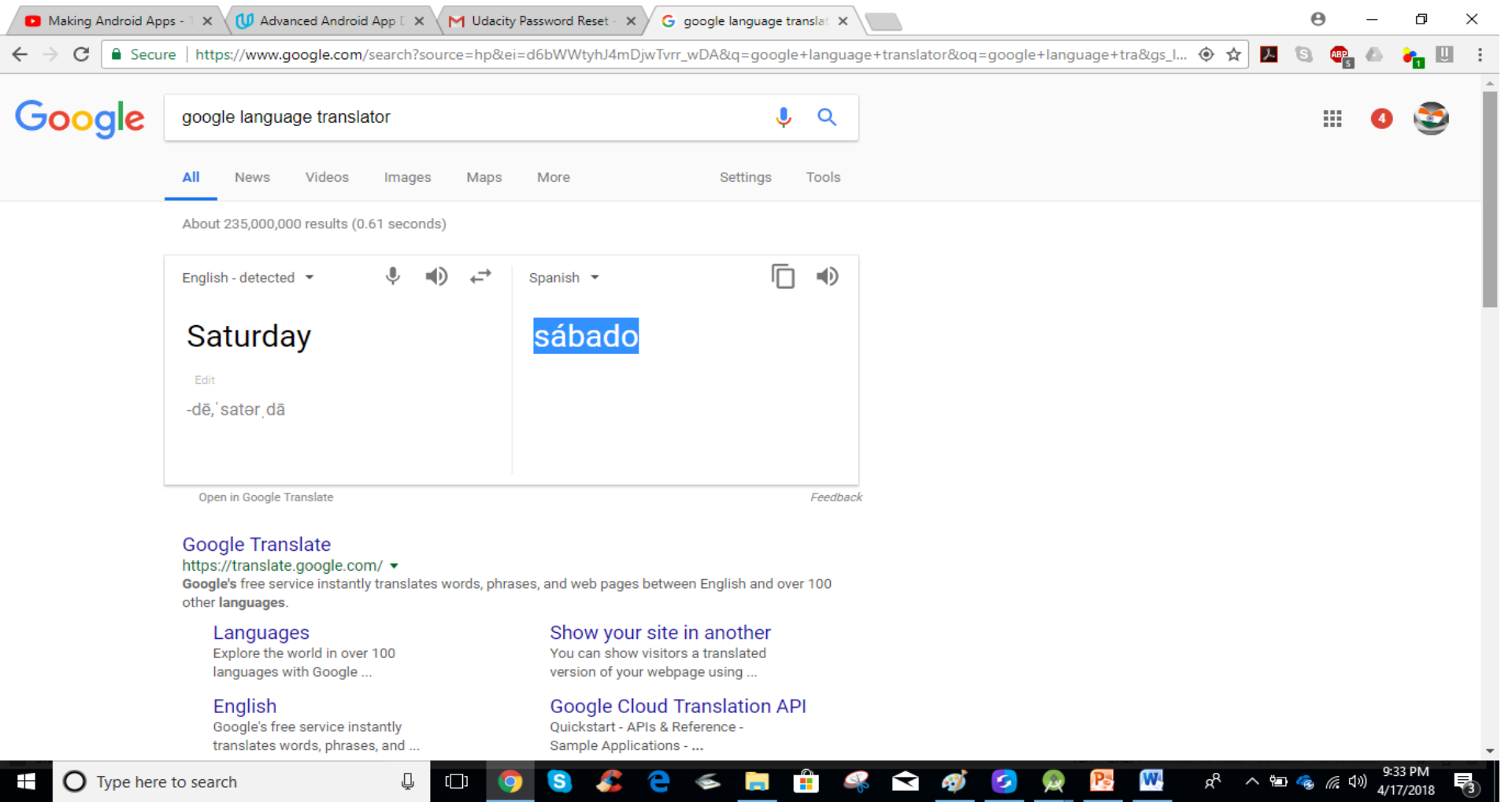
Localization Step by Step Screens

Step 5 : After Choosing the Language, you will get screen looks like below. Red color indicates that need to provide terms for each Locale.



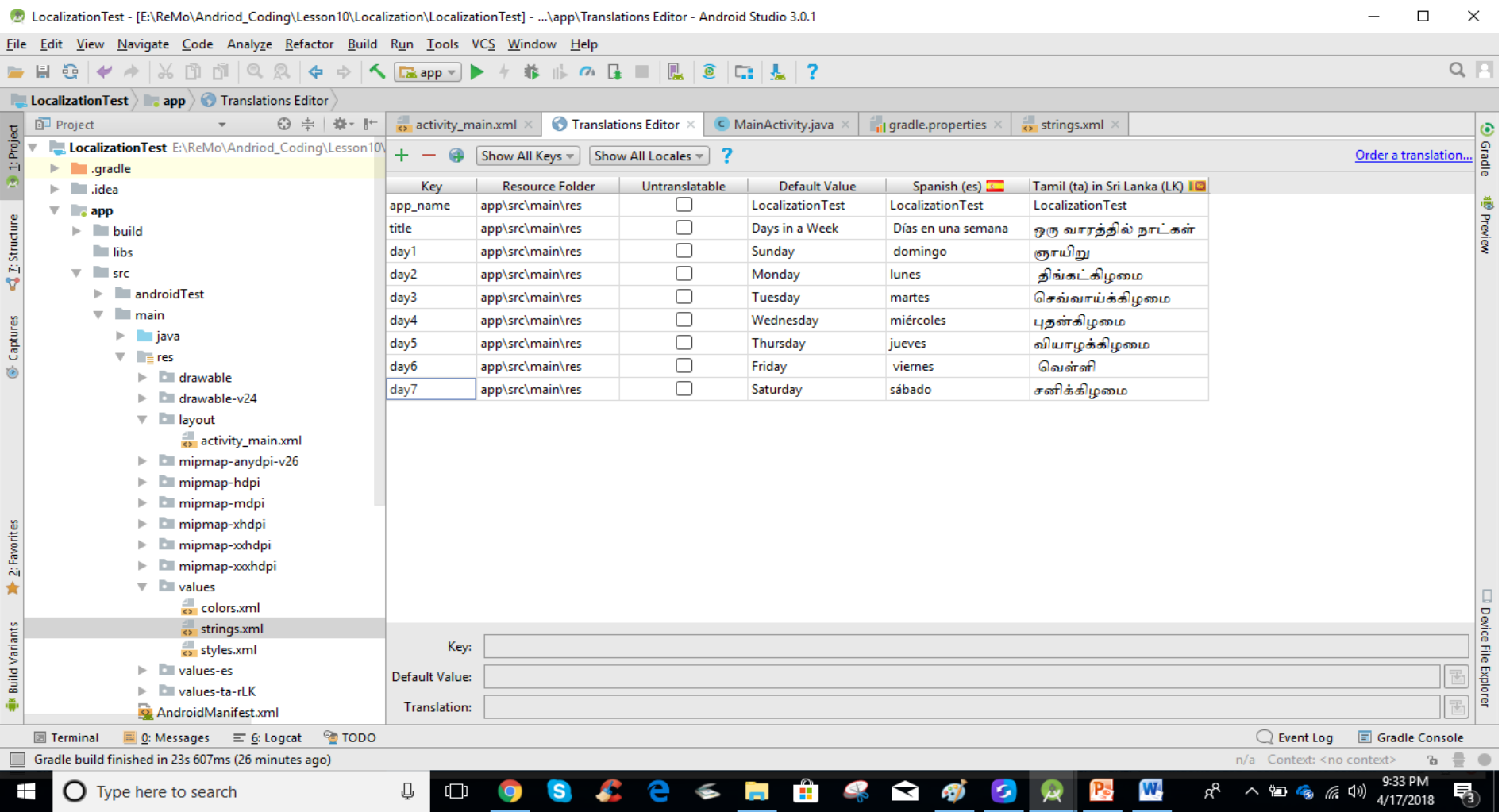
Localization Step by Step Screens

Step 6 :Use Google Language Translator to fill out the values in each locale.



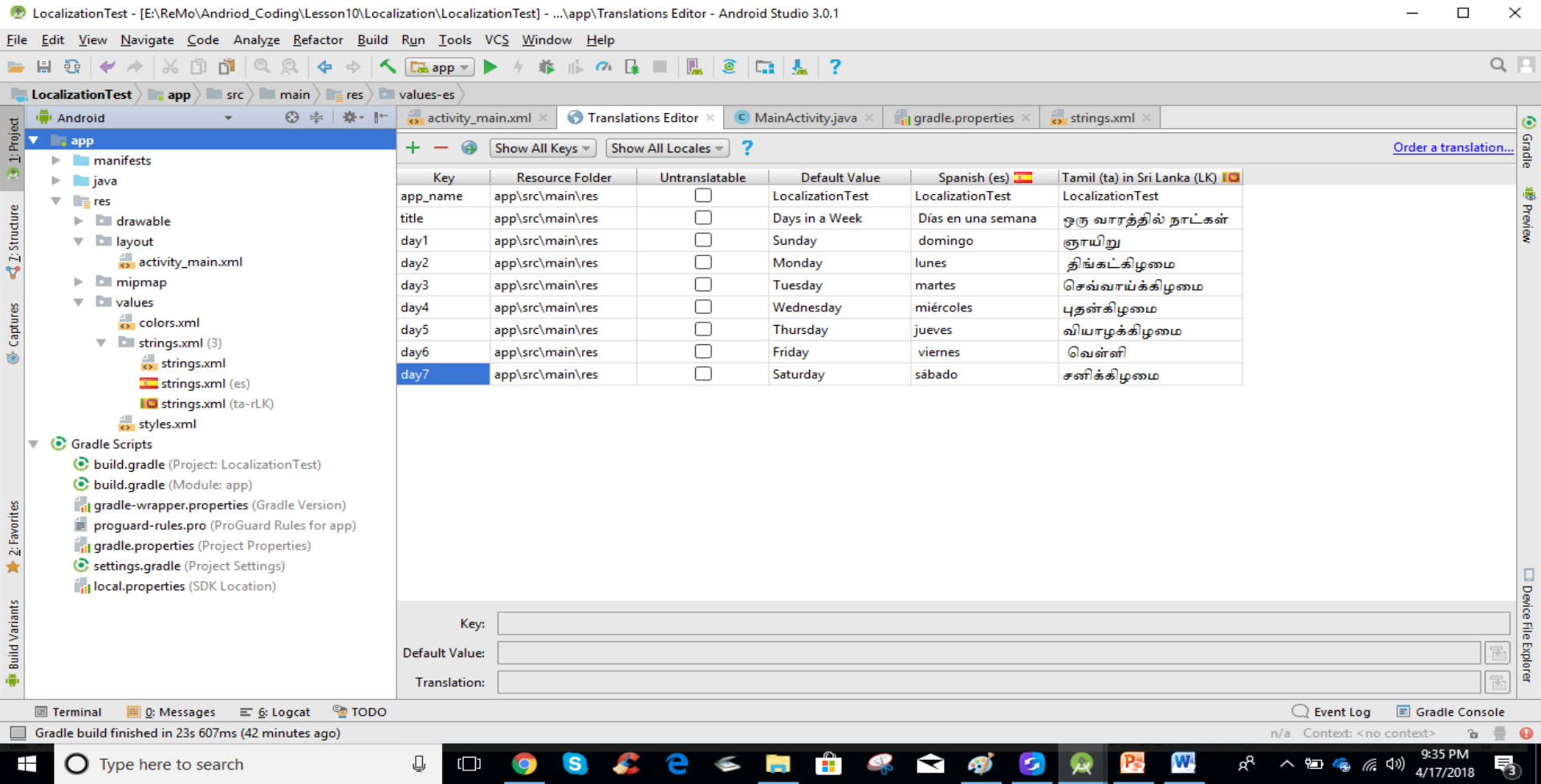
Localization Step by Step Screens

Step 7 :Once everything is done, the screen looks below.



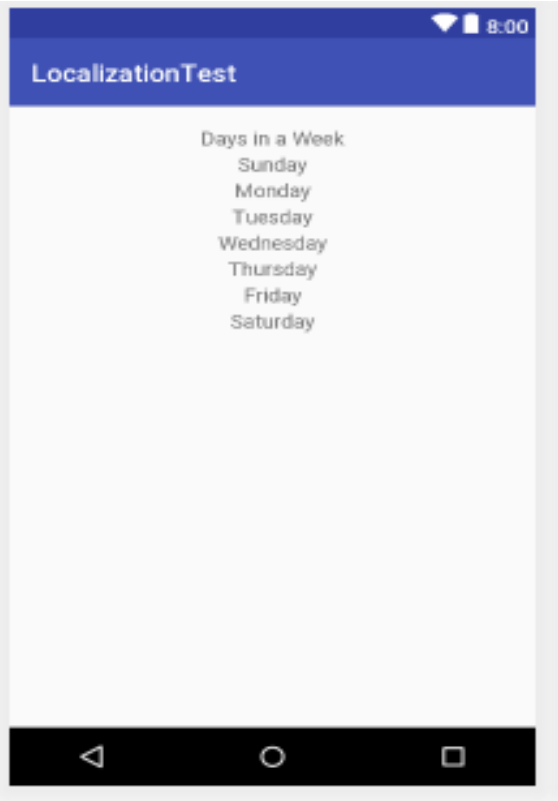
Localization Step by Step Screens

Step 8 : Open your res -> values->strings.xml, the screen as looks as below



Localization Step by Step Screens

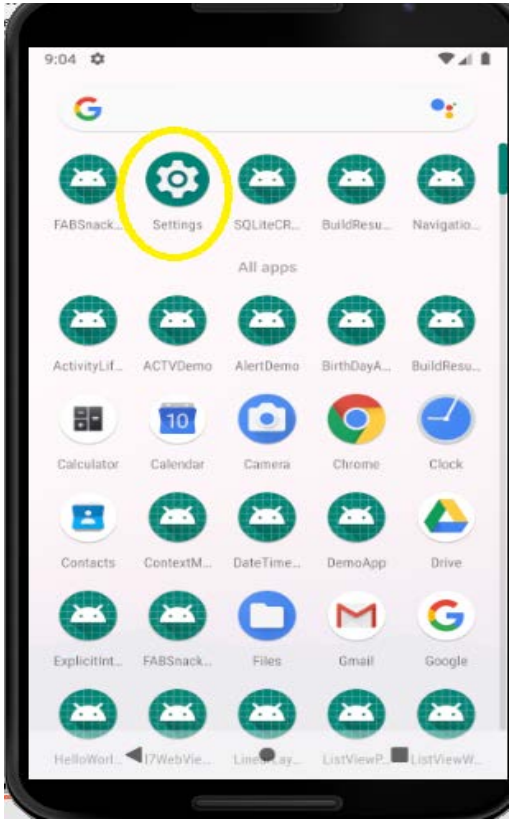
Step 9 :Go to your activity_xml and design layout as shown below. Add eight TextView components and make all component text attribute reference with its string resource.

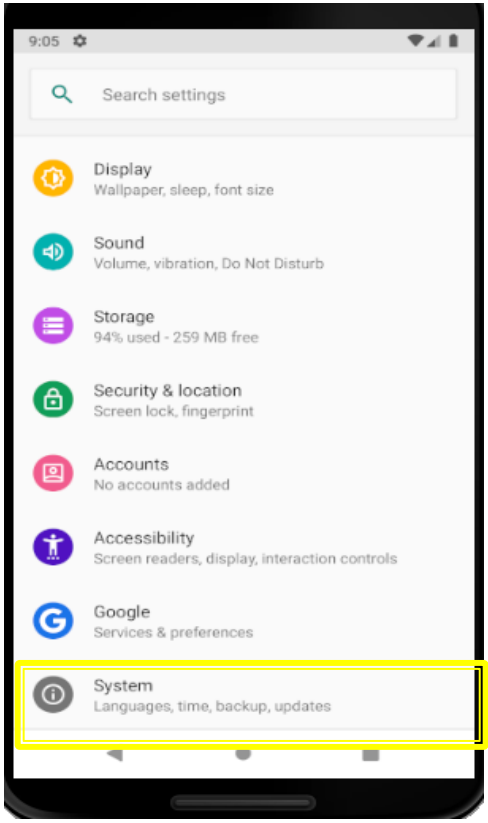


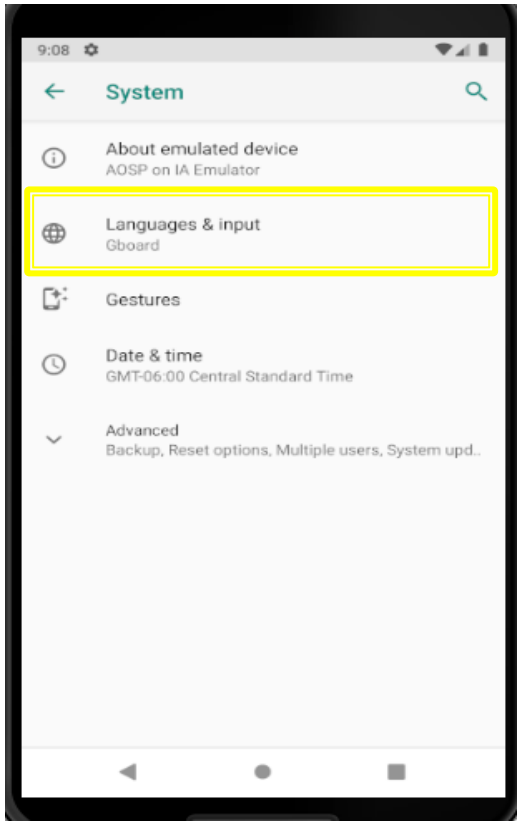

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:gravity="center"
    android:text="@string/title"
    android:textSize="16sp"/>
  <TextView
    android:id="@+id/textView1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:gravity="center"
    android:text="@string/day1"
    android:textSize="16sp"/>
```

How to change the Language Settings

From your emulator or real device,
Choose Settings → System → Language & Input







How to change the Language Settings

Continuation of Previous slide Screenshots,
Choose Languages, English is the first priority now. Drag your preferences and click Add a language to insert more languages. Make sure that, you should have the String resources for the selected language in your app.