NoteLingual

A tutorial by Claire Stanhope

Table of Contents

[Themes 2](#_Toc511636732)

[Making your theme 2](#_Toc511636733)

[Implementing the new theme 3](#_Toc511636734)

[Applying the changes 4](#_Toc511636735)

[Too many *NoteListActivity* instances 5](#_Toc511636736)

[Too many *UserPrefsActivity* instances 6](#_Toc511636737)

[Locales and Languages 6](#_Toc511636738)

[Making the new resource file and folder 7](#_Toc511636739)

[Filling in your new files 10](#_Toc511636740)

[Implementing your new locale 11](#_Toc511636741)

[Overriding the phone’s locale 12](#_Toc511636742)

[Mipmap locale 13](#_Toc511636743)

[Right-to-left languages 14](#_Toc511636744)

[Supporting right-to-left layouts 15](#_Toc511636745)

[Does this look right? 15](#_Toc511636746)

[Further Reading 17](#_Toc511636747)

# Themes

Themes are essentially the colour schemes of your app. Most apps tend to offer a light and dark theme, with contrasting text colours and different backgrounds. This section will teach you how to create your own themes in Android Studio.

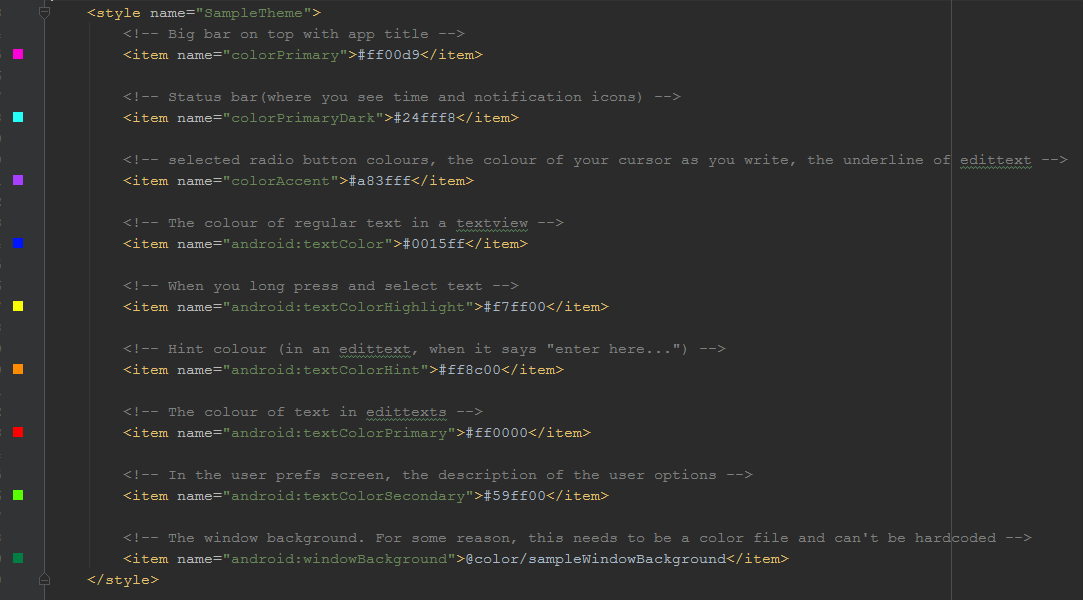
Android Studio apparently can’t decide if they want to use the word “theme” or “style” for app colour schemes, so they are used interchangeably. I will try to stick to “theme” but some elements are called “styles”. It’s confusing, but just think theme = style.

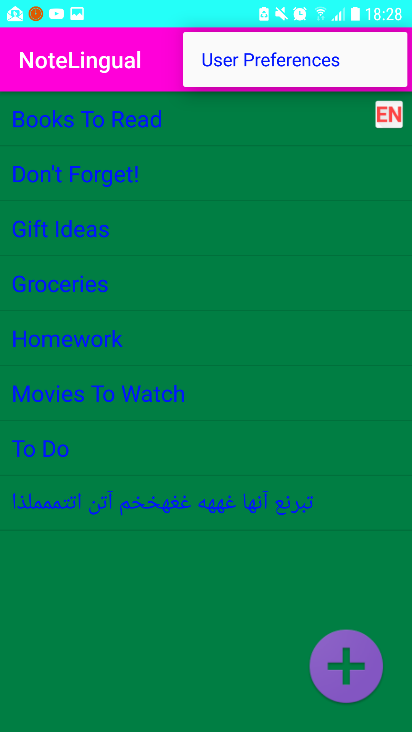
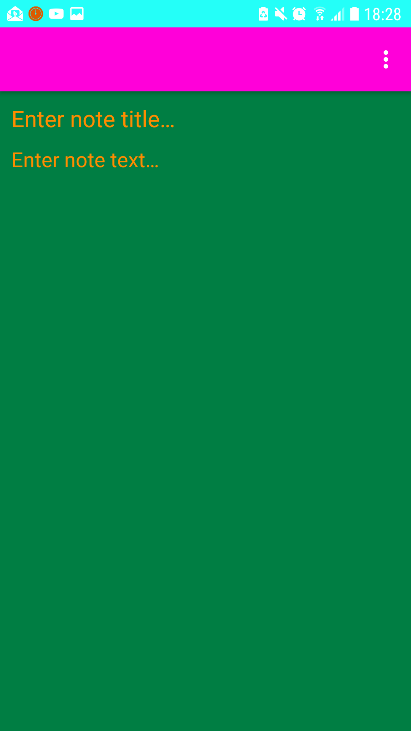
## Making your theme

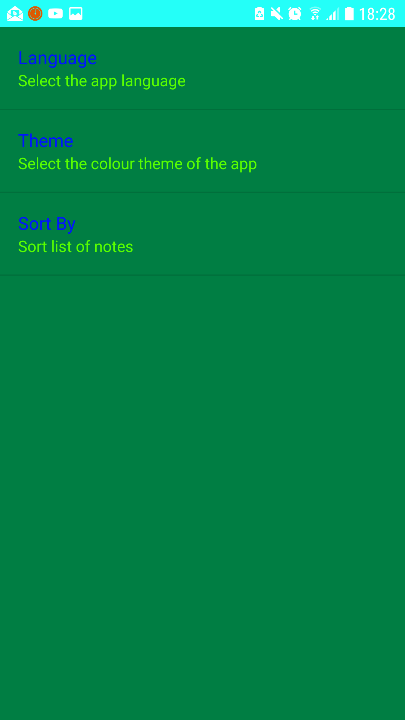
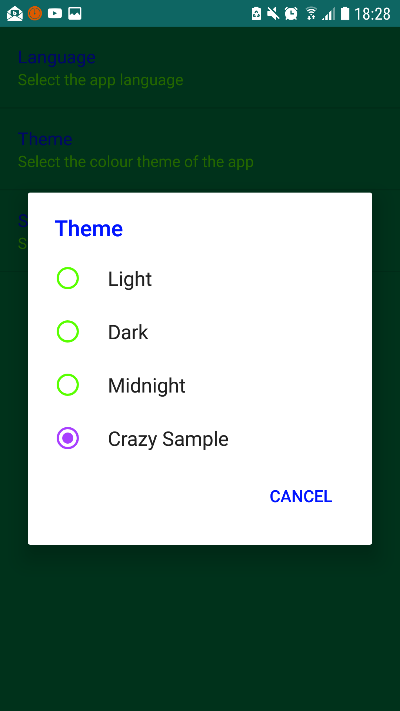
Before you code anything, make at least one new theme so that you can test your code actually works. Go to your *styles.xml* file, located in *res > values*.

When you create a new project, the only theme you should see is called “AppTheme” which has been left in this application for reference. I have added four of my own themes: Light, Dark, Midnight, and Sample.

Theme colours can either be hard-coded straight into the *styles* file, or colours can be added to the *colors* xml file. The second option is better practice and will be done for all of our themes except the Sample theme shown below. These values are hard-coded because this theme is terrible and shouldn’t actually be in a real app. I’ve added it to the final app so you can see what you can do and manipulate with a theme. Below is a screenshot of some colour explanations and some screenshots of the theme on my phone so you can see what they look like.

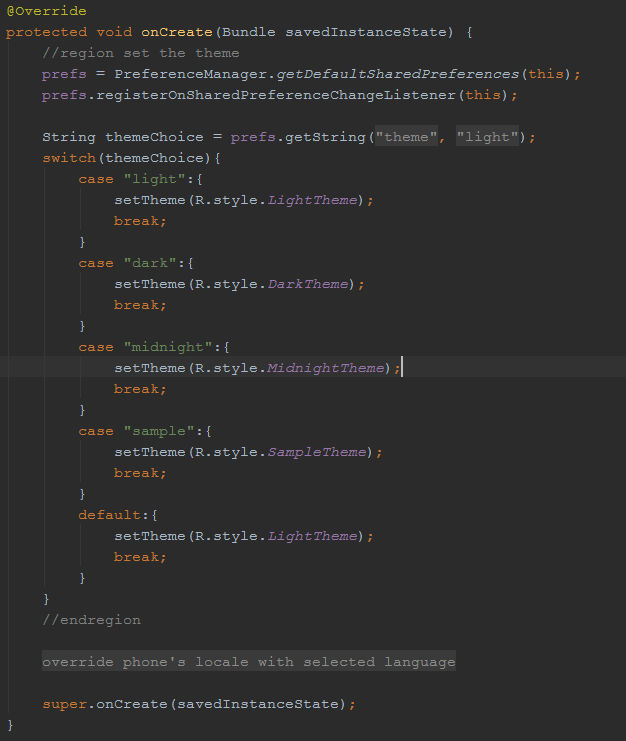


## Implementing the new theme

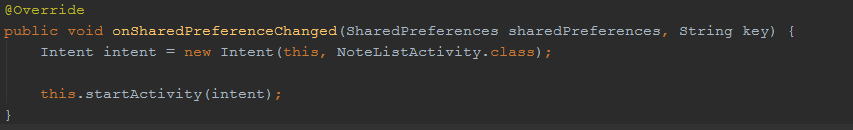
In order to implement themes, we’re going to create a regular base activity which will determine the user’s chosen theme and apply it to the activity. Just like the *BaseActivity* we’ve made in class so many times.



Note, in **onCreate**, our code appears ***before*** the **super.onCreate** call. This is extremely important and your themes will not work if they appear after this super call.

## Applying the changes

In order for these changes to apply, the activities must have a chance to go through the **onCreate** method of *BaseActivity*. We must add the code below to the *BaseActivity* java file.If the following code is not added, the newly selected theme will only apply to the activities launched from the “startup” activity and not the startup activity itself. The user would have to close the activity fully and reopen the app (not just hit the home button and then click on the app again, actually hit that task manager button and close it) in order to get the theme to apply.



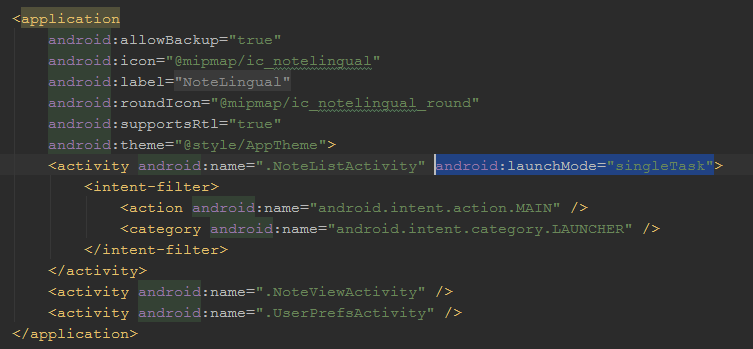
Note: in this app, our startup activity is *NoteListActivity*.

By adding the code above, the main activity is forced to open again and therefore the theme is changed.

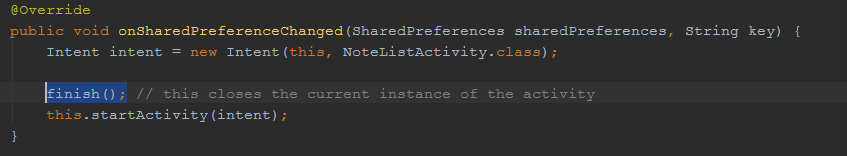
This solution causes a bit of a bug. Notice if you go into the preferences and change the theme multiple times then hit the back button, you’ll see all the themes you have selected. To get around this, we need to throw in more code (of course).

### Too many *NoteListActivity* instances

To stop the multiple pages of activities, we want to set it so that *NoteListActivity* can only have one instance running at a time. Essentially, we’re making it “static”. This is done by adding the code highlighted below into the manifest file.



We also need to add the highlighted code below in *BaseActivity* to close the current instance to allow a new instance to be created.



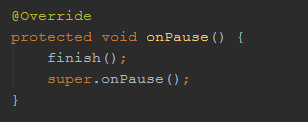
If this isn’t added, the *startActivity* call below it won’t create a new instance and completely defeat the purpose of this code. The theme will stay the same as before the user changed it.

### Too many *UserPrefsActivity* instances

Now the *NoteListActivity* should only appear once when you change the theme multiple times and hit the back button. But you probably noticed: you had to hit back on a *UserPrefsActivity* a bunch of times. There are two ways to fix this…

**Option One**: we could do something similar to the *UserPrefsActivity* and add the *singleTask* launch mode in the manifest. But when the user hit the back button the preferences activity would still show up once, which in my case is not what I want.

**Option Two**: we add the following code to the *UserPrefsActivity* to force the activity to finish when the user navigates away from it.



This is what I want in my app, so it’s what I’ve done.

# Locales and Languages

To start off, “locales” is the word used in Android development to mean the area on Earth the user has chosen as their “home” location. This is essentially an extension of languages because a language can change depending on where the person is from. For example, English has multiple locales, but the most commonly offered locales are en-US (United States English) and en-UK (United Kingdom English). When talking about the portable metal object used to shine light in a focused area, an en-US locale would use “flashlight” and en-UK would use “torch”. You can find a list of Android locales [here](https://web.archive.org/web/20120814113314/http:/colincooper.net/blog/2011/02/17/android-supported-language-and-locales/).

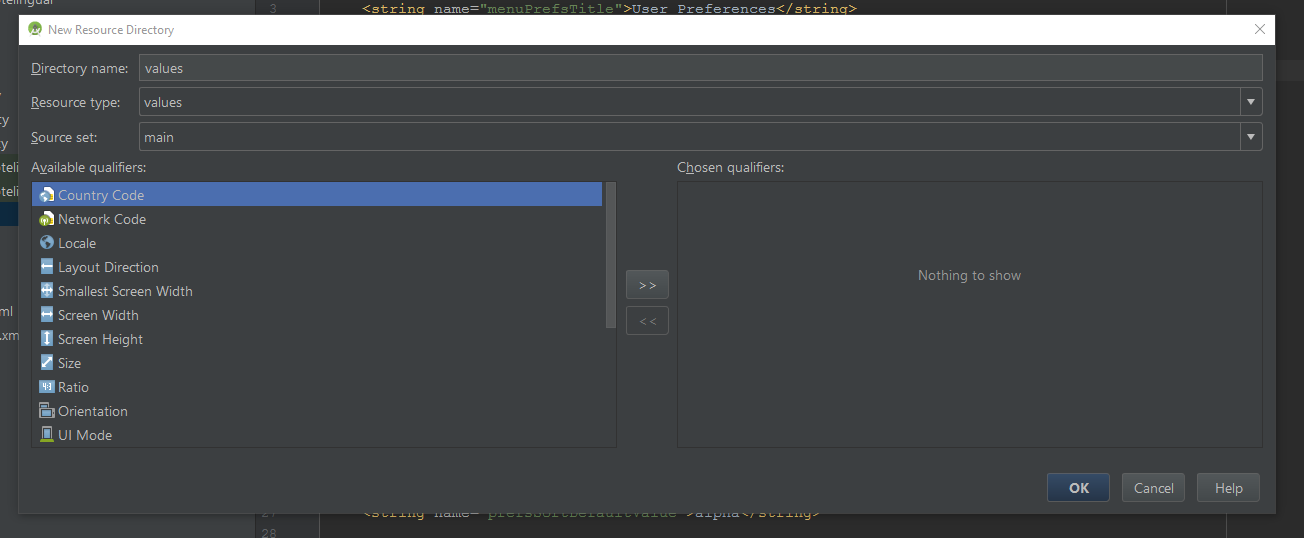
In this app, we are going to offer English and French to start, then add Finnish and Persian.

## Making the new resource file and folder

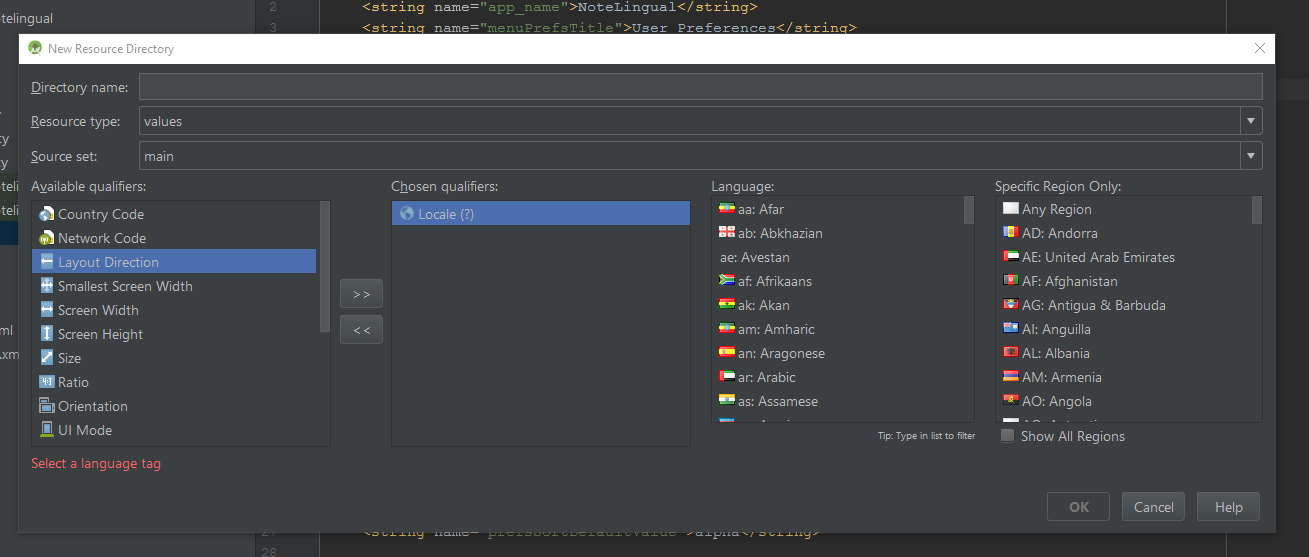
Android handles different locales by making multiple copies of files including the same IDs but with different contents. For example, the textarea with the hint string *enterNoteTitleHint* appears as “Enter note title…” in an English locale but “Le titre de la note…” in a French locale.

You’re going to need to make a new resource file for everything that the user sees. For this app, it’s just the *strings.xml* file and the *arrays.xml* file. These steps below take you through making a *strings.xml* file for the French language. The same steps apply for any other new language or .xml file.

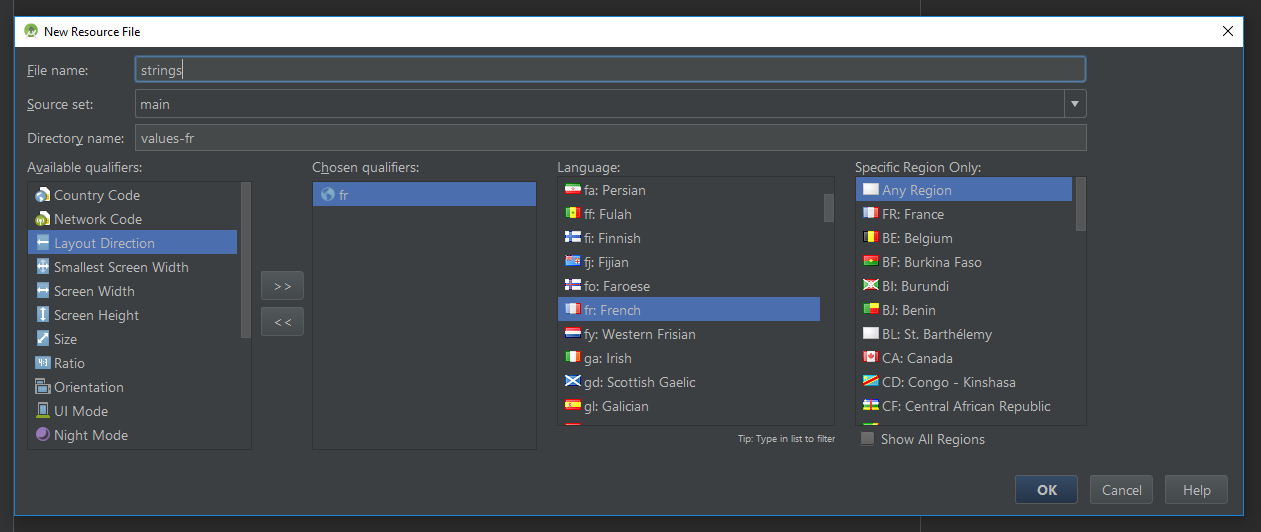
1. Right click on the **res** folder and select *New > Android resource directory*. The image below shows what you should see right after clicking it.



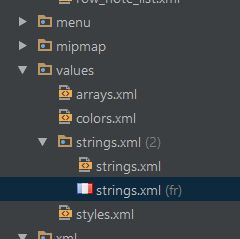
1. For now, we’re going to skip over the **Directory name**, **Resource type**, and **Source set** because they are set to the defaults we need for *strings.xml* and *arrays.xml* files. In the future, we will change the **Resource type**. But for now, jump right down to the **Available qualifiers** section. Select **Locale** from the list on the left and then click the **>> button** in the centre of the page. Your window should change to something like this:



1. Now select the language and region from the long lists on the right with all the flags. For this example, we will select **French** and **Any Region**. Notice that once a language and a region are selected, the **Directory name** changes. Leave this to the default; you’ll have serious problems if you change the directory name. Just don’t do it!
2. Now you hit **OK** and notice… that no new folder has shown up in your res directory. Don’t fret. This is normal.
3. We want to make a new French *strings.xml* file that we can use, so we’re going to right click on the **values** directory and select *New > Values resource file*. In this case, we ***must*** enter a **File name**, and since it’s a strings file we’ll call it “strings”. From here, we’re going to do the same thing by selecting the **Locale** and moving it over with the **>>** **button** and selecting **French** and **Any Region**. Notice how the Directory name field changes when we select different languages.

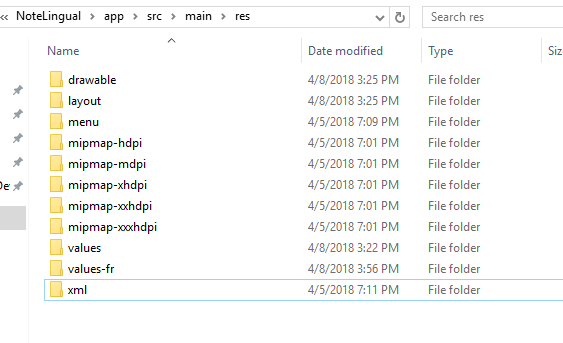


1. Now click **OK** and notice the *strings.xml* file is now a folder with two different strings files.



TADA! Steps 1-4 aren’t actually needed and you can skip right to steps 5 and 6 for any other languages you add. But it’s still good to know!

You’ll notice that if you navigate to the res folder in Windows Explorer, you can see each of these *values* folders containing your files! The path to get there is *YourAppName > app > src > main > res.*

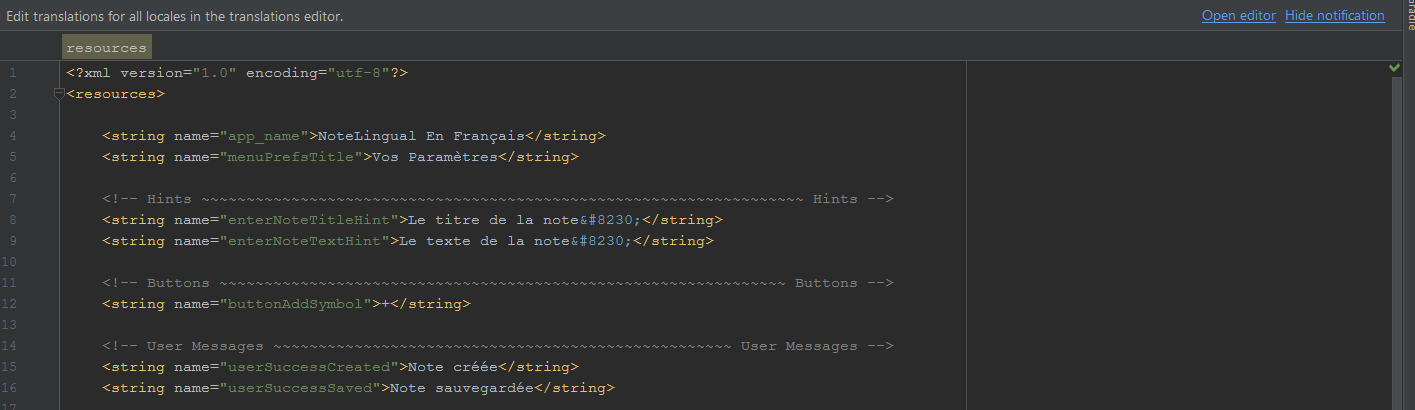


Not super important to know now, but could prove handy in the future…

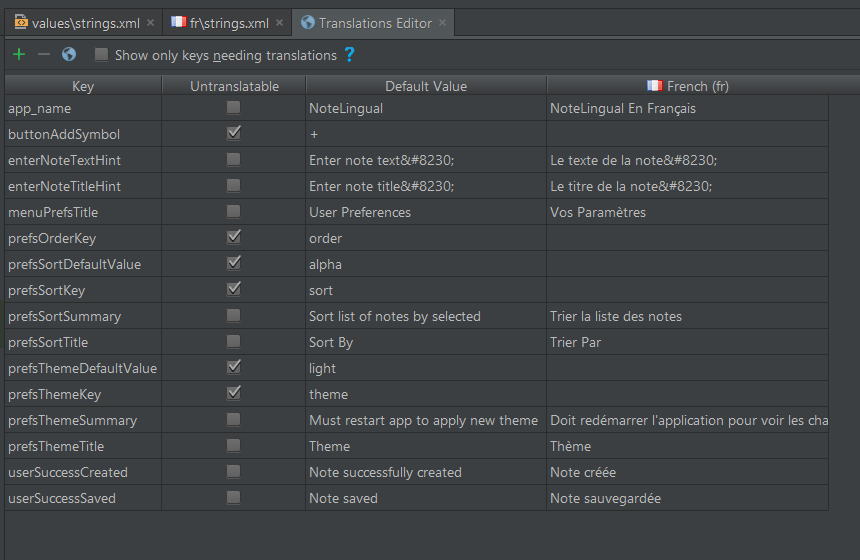
## Filling in your new files

Open your default *strings* file and copy all the strings into your new French *strings* file. Go through and change the contents of the strings to your new language. ***Do not change the IDs***. You might notice, some of these strings are used internally and not displayed to the user. This string should be deleted from the file.

Once you have gone through the file and changed the user-displayed strings to match the new language, you might notice there is a message at the top of the screen asking if you would like to **Open editor**.



If you click on this, you will be taken to a file that allows you to decide which strings should be translated and which should not. Here’s a screenshot of what my file looked like:

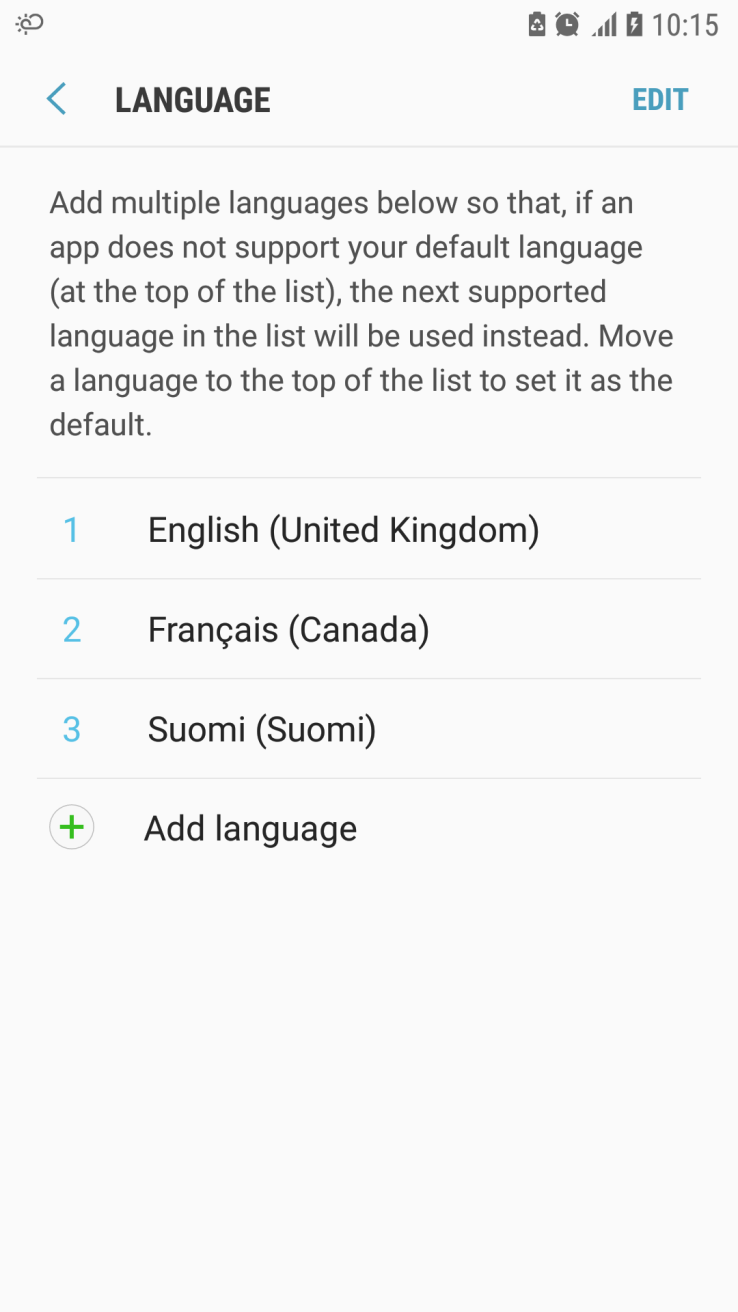


You’ll notice at the top left of the window there is a **+** **button**, a **-** **button**, and a **globe** **button**. If you click on that globe you can add a new *strings* file. I know I’ve taken you through many ways to do this, but it’s important to understand multiple strategies. Never know when they might come in handy!

## Implementing your new locale

Surprise: you’ve already done it. Go into your phone’s settings and change your phone’s language to the new language you’ve just created. Open the app. It’s in your newly set phone language! Woop!

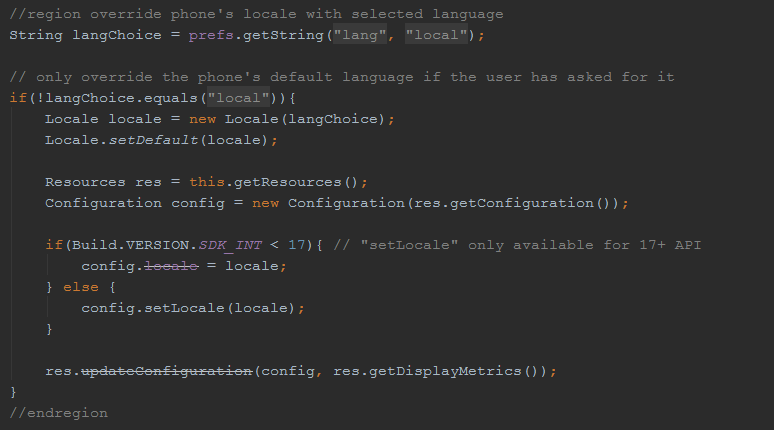
The Android OS does the phone locale detection for us by checking our phone’s locale and the available locales of the app. If the app we’ve opened offers a locale from one of our “selected locales”, the Android OS will select that for us automatically. Below is a screenshot of what the “selected locales” screen looks like on a Samsung Galaxy S7.

****

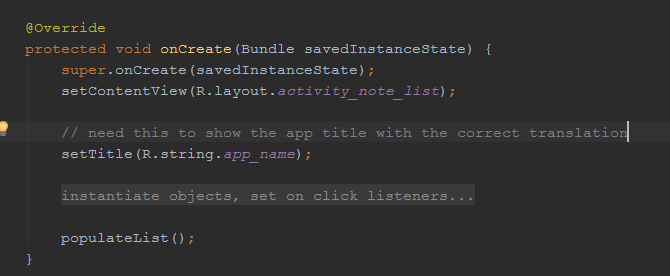
But what if I speak Finnish and my phone doesn’t give me the option to set my language to Finnish? Maybe my phone only supports English, French, and German, but I would much prefer the Finnish option. So, let’s give the user the choice to override their phone’s language if they want.

## Overriding the phone’s locale

To override the user’s phone locale to what is selected in preferences, we’ll add the code below into *BaseActivity*.



You’ll likely notice that even though this code works on almost everything, the title of the app on the top does not change from your phone’s default locale. This is a known issue and can be overwritten by placing the code below into the activity whose title you want to change:



**NOTE**: Like themes, the over-written languages are applied in the *onCreate* method in the *BaseActivity* file. If you are not implementing themes and just want to do languages, you must still follow the steps in the *Implementing the new theme* and *Applying the changes* sections of this document in order for languages to work smoothly.

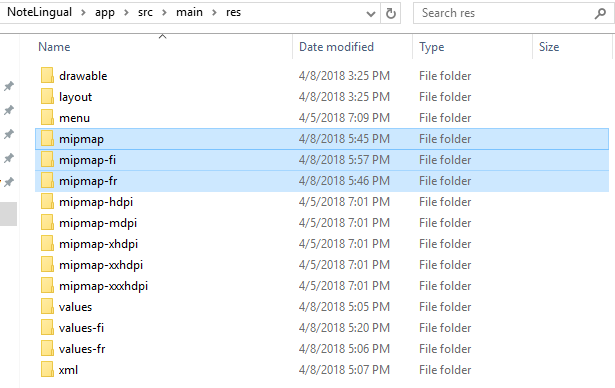
## Mipmap locale

These images on the top right are pretty ugly, but they’re there to demonstrate how we can do the same locale formatting with images (mipmaps).

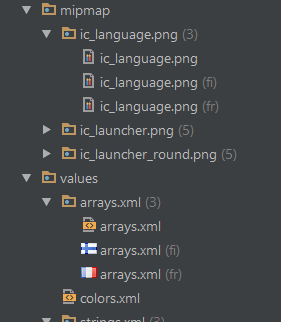
** **

To make a new mipmap folder for our language icons, go back up to the *Making the new resource files* section and follow steps 1-4, but under **Resource type** select **mipmap**. Continue through these steps once you have done steps 1-4 above.

1. You now have a new mipmap folder that you totally can’t see. That’s fine though, we’re going to look at it in Window’s Explorer. Go to *YourAppName > app > src > main > res* to see the new mipmap folder. We’ll also make a folder called “mipmap” here just by clicking *New > Folder*. Make a mipmap for every language you’re going to have a different image for. For me, I’ll start with three: English, French, and Finnish.



1. Now, take the mipmap files you want to add to your application and drag them into the appropriate folders in Window’s Explorer.
2. Once each mipmap has been placed in the correct folder, you’ll notice that your mipmap folder in Android Studio has multiple options for one image:



And that’s it! Now the mipmap should change when a new language is selected in the app.

## Right-to-left languages

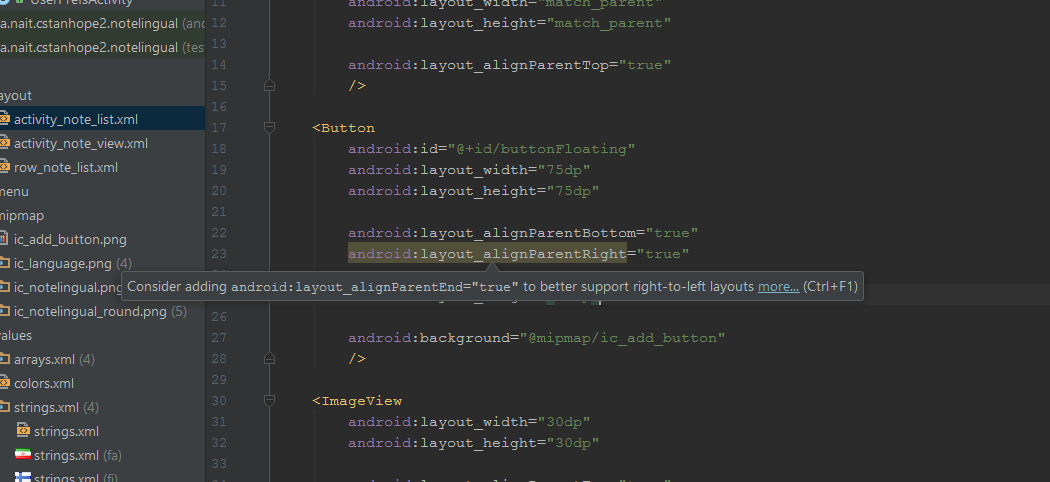
Now that we have the easy languages out of the way, let’s try a right-to-left language. For this example, we’re going to use Persian. To implement this new language, simply follow the steps we took to make the Finnish language available on our app. We’ll use the Persian Any Region locale.

Please note: I speak no Persian. These translations were done by Google Translate. The sample notes in the screenshots are just me button-mashing a Persian keyboard. Maybe I’ll speak Persian one day, but for now you’ll have to live with my button-mashing.

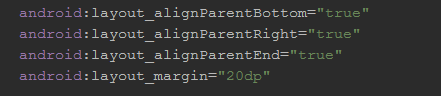
Once you add the new *strings* and *arrays* files, you’re done. Let’s all thank Google for making it that easy. Now you can select Persian and the phone will mirror your display automatically. However, for this to work automatically, you must support it in your layout files…

### Supporting right-to-left layouts

Remember all those android hints you saw, saying “Consider adding […] to support right-to-left layouts”?



Well, you need those for this to work. So don’t be lazy: put them in!



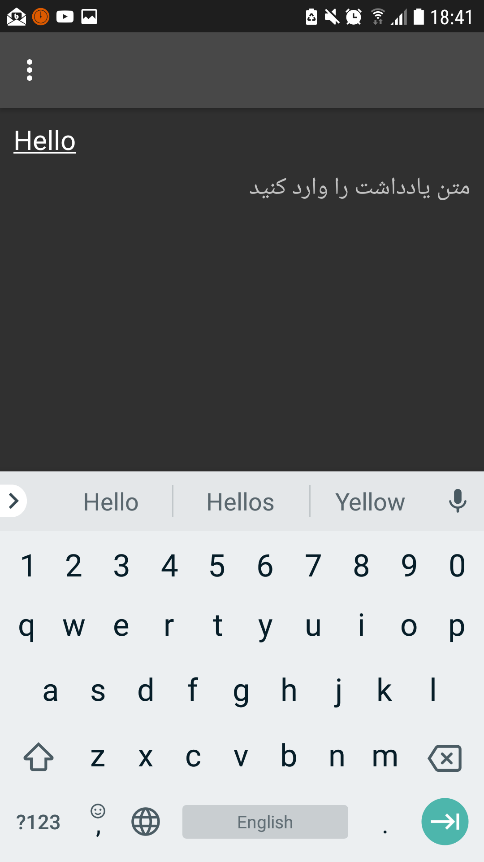
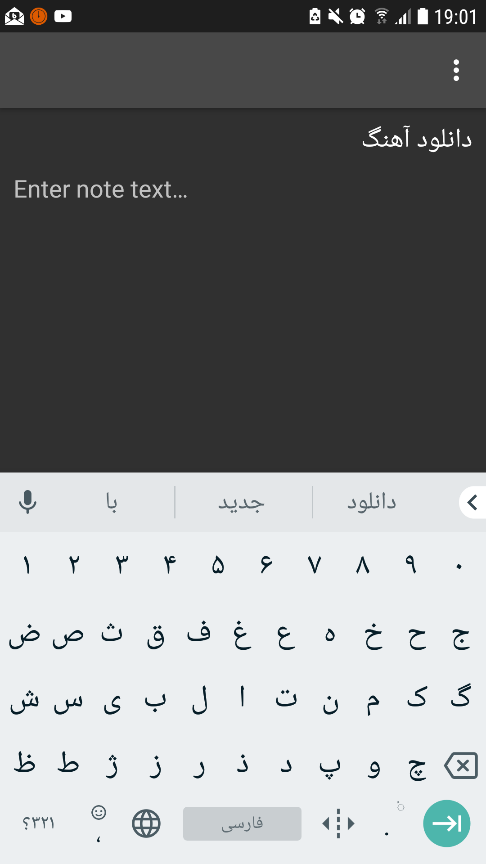
It’s very simple to do and, if you get into the habit of it, you won’t even have to worry about it in the future when you try to implement other languages!

### Does this look right?

One thing you will notice is that your notes in left-to-right languages will simply be right-aligned in the right-to-left language layout. This is normal, don’t fret. If you speak Persian, you’ll notice the same thing is done for right-to-left languages in a left-to-right layout.

**** 

You might also notice that when the Persian language is selected in the app and you enter any text with an English or left-to-right language, the direction of the text in the edit-text file changes. This is what we want, as the user controls their own keyboard. The same happens with a Persian keyboard in the English version of the app.

# Further Reading

The Android Developers page on styles and themes:

<https://developer.android.com/guide/topics/ui/look-and-feel/themes.html>

The Android Developers page on locale and language support:

<https://developer.android.com/training/basics/supporting-devices/languages.html>

The Android Developers page on how locale is processed:

<https://developer.android.com/guide/topics/resources/multilingual-support.html>