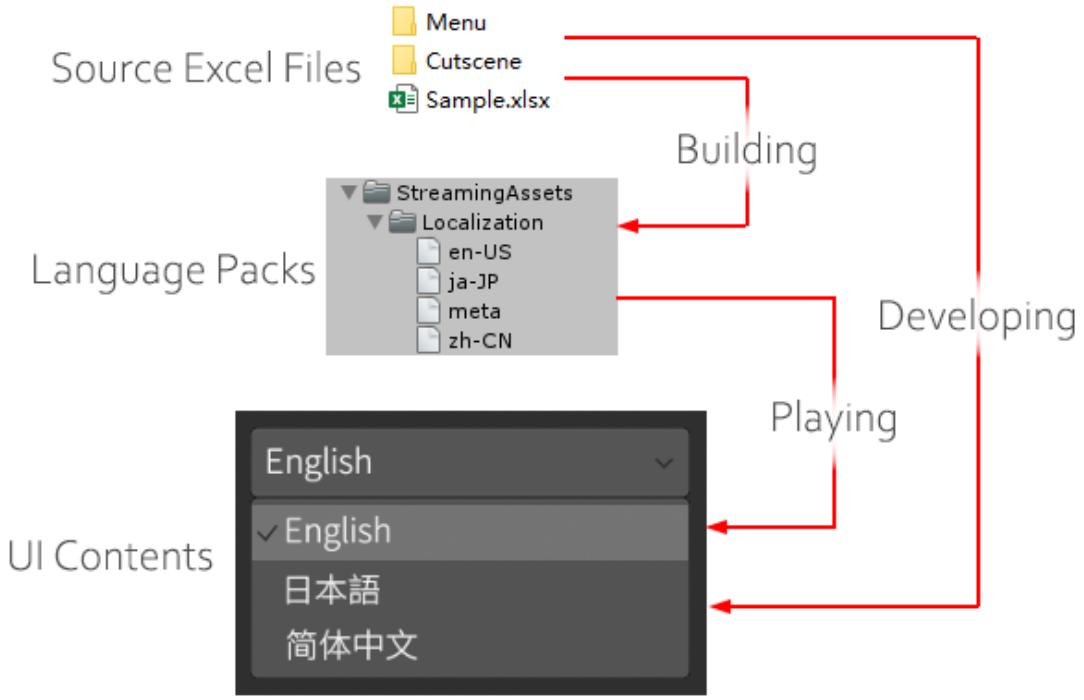
Localization Manager

This is a localization system for Unity that uses excel in development. When Unity starts building, the localization system converts source excel files into language packs; when game is playing, the localization system loads a specified language pack and updates all localized contents in game. You can load excel files directly in development, this is more convenient for localization developers and testers.



Features

- You can use multiple excel files, each containing multiple sheets, and each sheet containing only parts of contents of parts of languages. As the localization system works, it reads all excel files in the Localization directory (include sub-directories) and merges their contents, which means the file organization structure can be customized.

- You can define keywords and then use "{ }" to reference them later.

EngineName	Unity
GameName	A Localization System for {EngineName}
	You can reference a text which has already referenced other texts.
UserManual	To use "{GameName}", please charge it first.

- You can make texts as atrributes. Marking a TextName with an "@" tells the localization system that it is a language attribute. The difference between normal texts and attributes is that attributes can always be accessed without loading the specific language pack. The "LanguageName" attribute is indispensable, which is the localized name of the language.

The localization system sorts languages by LanguageName by default, so you can easily create and display a user-friendly list of language choices.

@LanguageName	English	简体中文
@TextureBundle	Textures/Localization/en-US	Textures/Localization/zh-CN

- Can use the following escape characters: "\n"-line break, "\t"-tab, "\"-backslash, "{{"-a single "{", "}}"-a single "}".

```
This is a line.\nThis is another line.
Here is a tab\tin this sentence.
Here is 8 backslashs: \\\\\\\\
It's not a reference: {{EngineName}}
```

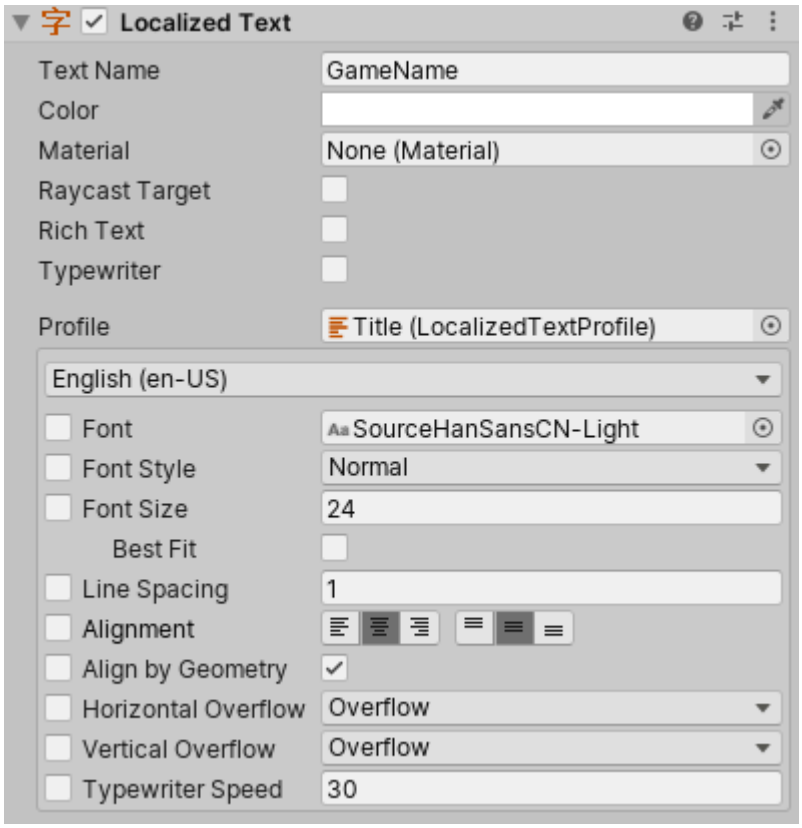
- Use "^" as TextName to inherit the prefix of the previous TextName and automatically increments the number of the suffix. This is automatic numbering function. This requires that the first of a series of automatic numbering must explicitly specify the prefix and initial suffix number. The advantage of using this function is that you don't have to manually modify TextNames to ensure that the numbers are continuous when inserting a line, deleting a line, or adjusting line orders. Your scripts can also access these lines through a loop.

Dialog0	This is the first in a series of automatic numbering.
^	The final TextName of this line will be "Dialog1".
^	The final TextName of this line will be "Dialog2".

- You can preview localized contents in edit-mode. (Window > Localization).

- Supports Unity UI & TextMeshPro internally, you can add per language settings for every

UI contents. (The toggle before every option means “override this for current language”.)



- Add Typewriter effect for Unity UI (Component > UI > Typewriter Controller, need toggle on “Typewriter” in LocalizedText). This is achieved via shader so there is no performance impact.

Quick Start

0. You have to set your project’s Scripting Runtime Version to .Net 4.x Equivalent to use this package. (this is the default setting.)
1. Create a "Localization" folder in your project root directory, it is the same directory as the "Assets". (You can open the Localization window and click “Open Localization Folder”.)
2. Create and edit excel files, save them in the Localization folder. (When the package is installed, it should create “Sample.xlsx” in the “Localization” folder, if you want to run the sample scene, please do not delete or modify this file.)
3. Open the Localizaton window, click "Build Packs" (If you checked “Load Excels Instead of Packs” then you don't have to do it), then click “Load Meta” (If you checked “Auto-load Meta in Edit-mode” then you don’t have to do it manually).
4. Create a LocalizedText or LocalizedTMPText(“TMP” means “TextMeshPro”, you need install it first) GameObject, input a valid TextName. Now you can test it in edit-mode.
5. Call `LocalizationManager.LoadMetaAsync` when your game starts running.
6. Call `LocalizationManager.LoadLanguageAsync` to load a language.
7. If you want to change language at runtime, just call

`LocalizationManager.LoadLanguageAsync` again.

8. There are some other methods you might need in `LocalizationManager`, please have a look at the comments.

If you run into any issues using my assets, please submit feedback here.