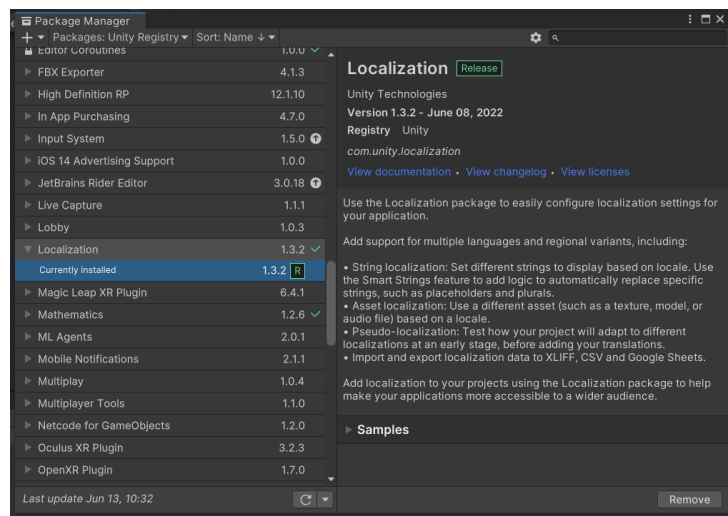


UNITY LOCALIZATION INSTRUCTION

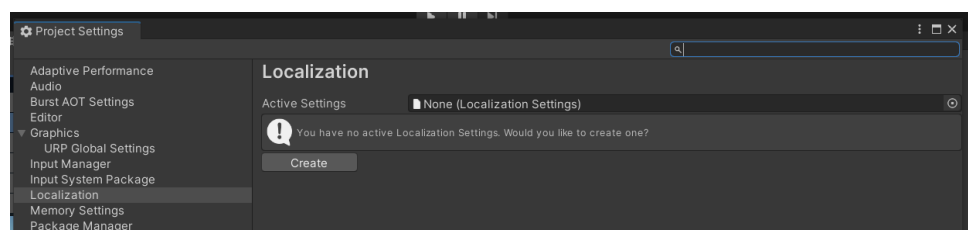
Using Unity localization is actually quite simple, but there are a number of things that must be done correctly to be able to use the features provided by Unity. The following are some instructions for using Unity localization and you can follow them.

- Installation and Setup

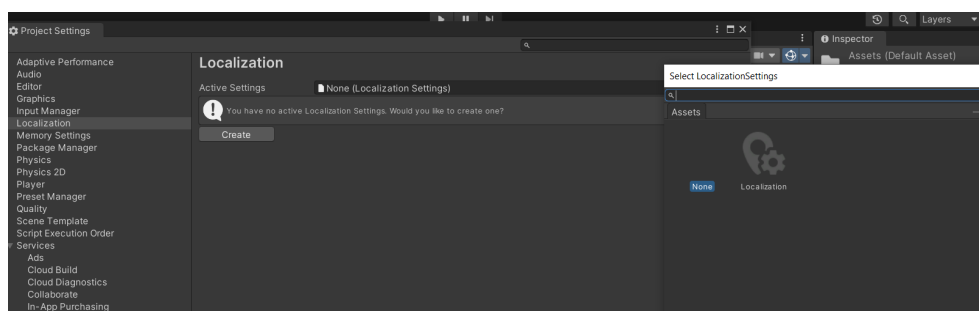
To use the Localization feature, we must install the Unity localization package from the Package Manager.



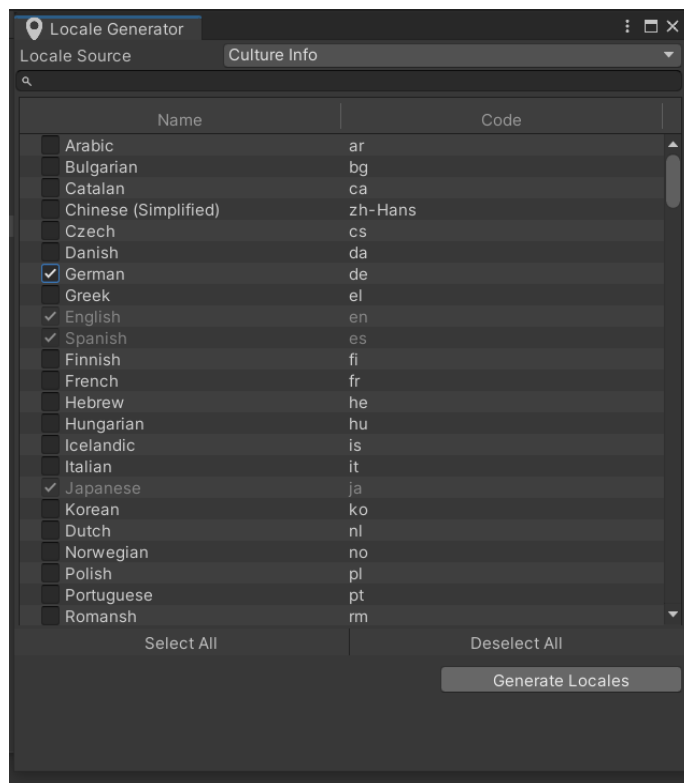
Next is to create localization setting assets that will be used in the project.



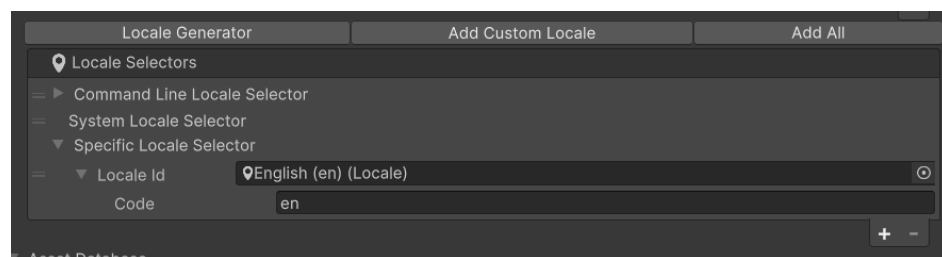
If we use a template, localization asset settings have been provided and all that remains is to enter them in the Active Settings section.



After that we can start entering the locale or language that we will include in the project, we can always change this parameter later. Go to *Edit* → *Project Settings* → *Localization window* and click on Locale Generator to create a Locale asset. Check the locale you want to use and click Generate locales to save and create Locale Assets.



We can specify the Default Locale used in the project, go to the specific locale selector section in Locale Selectors, click on locale ID and select the locale we want to be the default.

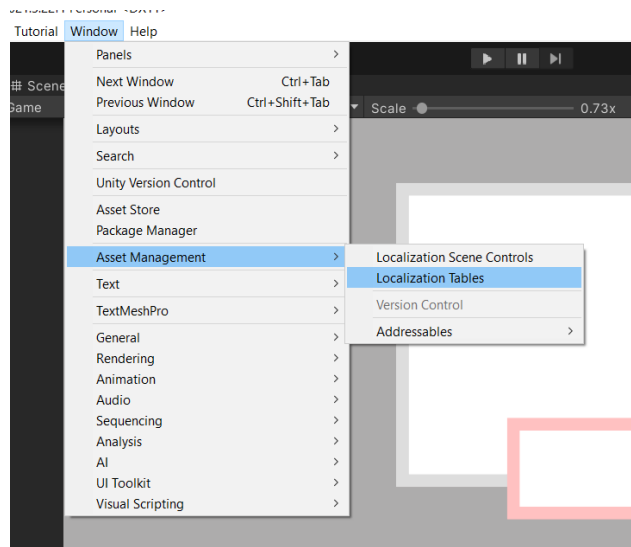


- Managing Translation

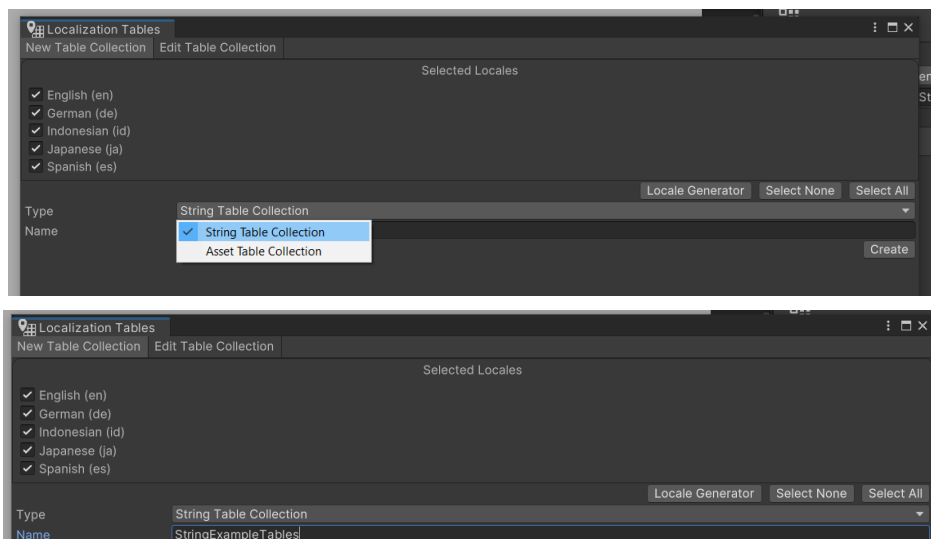
The official localization package will have us creating string table collections and populating them with translatable strings that we can then use in our components.

- Create String Table Collections

To use localization, we must create a localization table which we will use to store the data we need. Go to *Window* → *Asset Management* → *Localization Tables*.

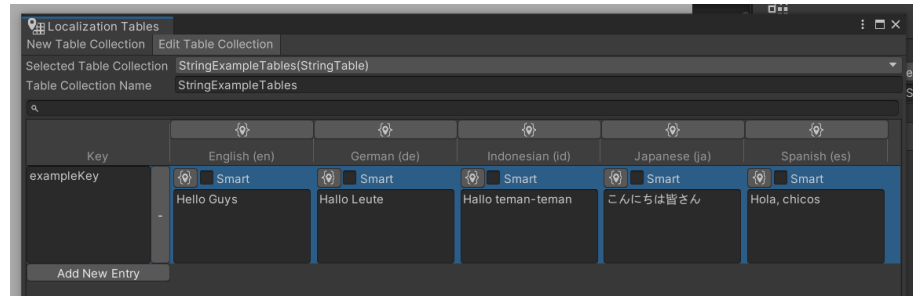


We can choose two data types that we can use, namely strings and assets. Choose according to your needs.



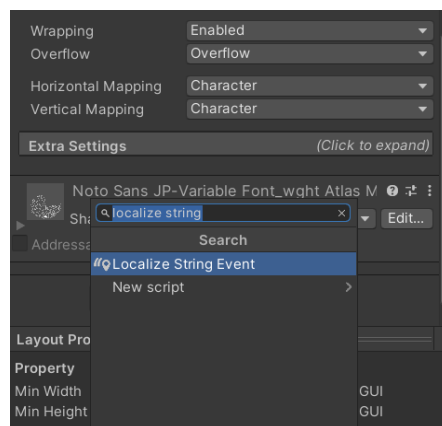
- Added Translated Strings / Asset

Enter data in the form of strings / assets that we will use, the important thing that must be entered is the key of each row of data that we will use.

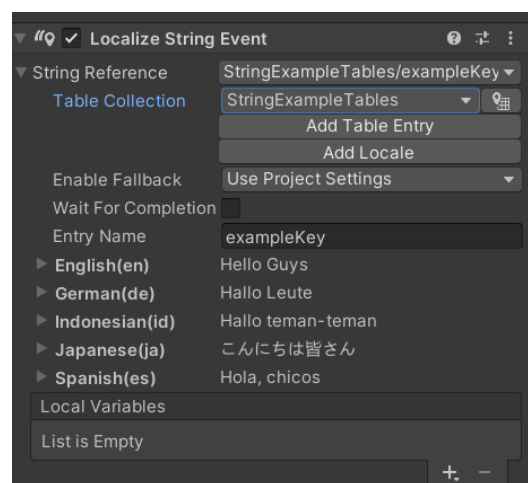


- Localize the Text and Image.

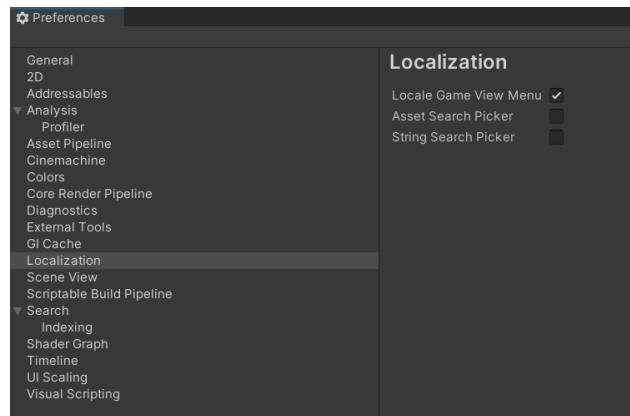
In the text that we want to connect with Localization, add the localize String event script, this is a script that comes from the Unity Localization package.



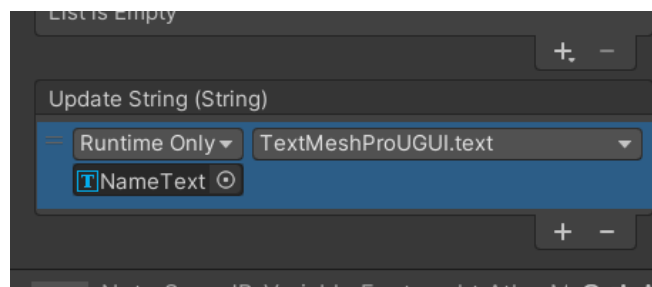
In the String Reference, select the table and key that we created earlier.



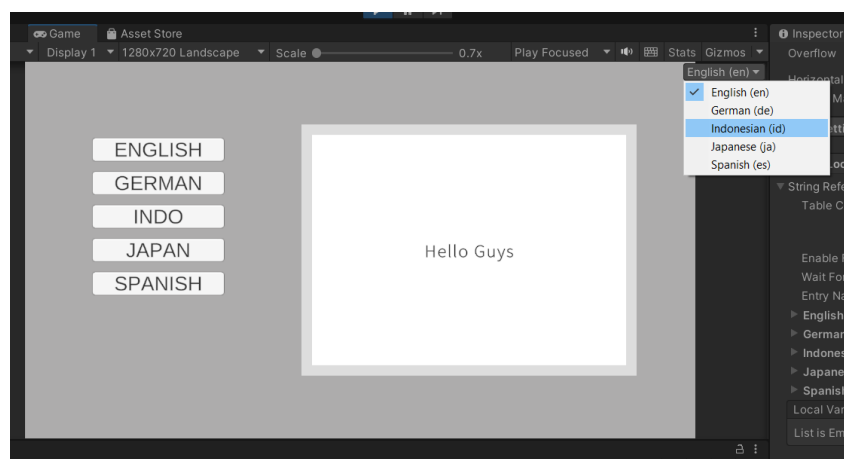
If the table and key we created are not found, go to *Edit* → *Preferences* → *Localization*, then uncheck the asset and string search picker. Now when we are in the inspector, we should be able to find the table and key that we have created.



In the update string event, enter the corresponding text object and select TextMeshProUGUI then select text.



We can try using game view, and localization should work. Without creating buttons, we can use the menus in the scene to change the locale currently in use. But the menu can only appear and be used in Unity only.



We can also modify assets, in a way that is more or less the same as string localization and only differs in the tables and keys used, also in the script used, where for sprites / assets, the Localize Sprite Event script is used.

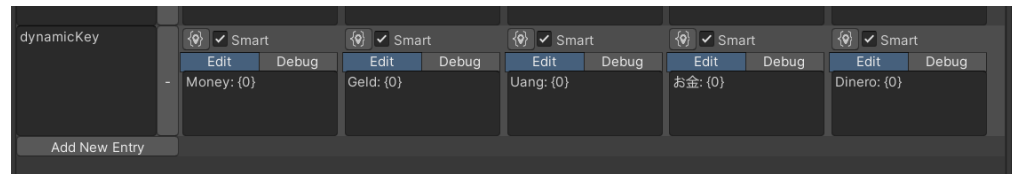


There are several other features that we can use in this localization

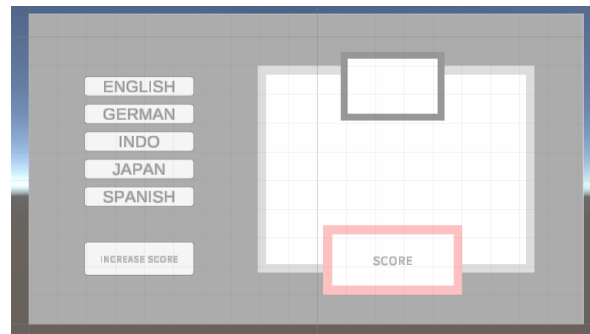
- Dynamic Text with Smart String

In the previous section, we could change static text, but this method doesn't necessarily work for dynamic text, such as score text which can change during Play Time. We can localize a Dynamic text in the game by using the smart string provided by Unity.

Create a new key in the existing localization table, enter the string data you want to use. If we want to use more than one data, we can also enter and change other variable data, by adding the name as well as the index. The index will be used to determine which argument will be changed first. Checklist in the smart box section for using the smart string feature, with this we can change the string value through code.



For example, we can create a string in the form of a score and create a simple script that we will use to change the score via code.

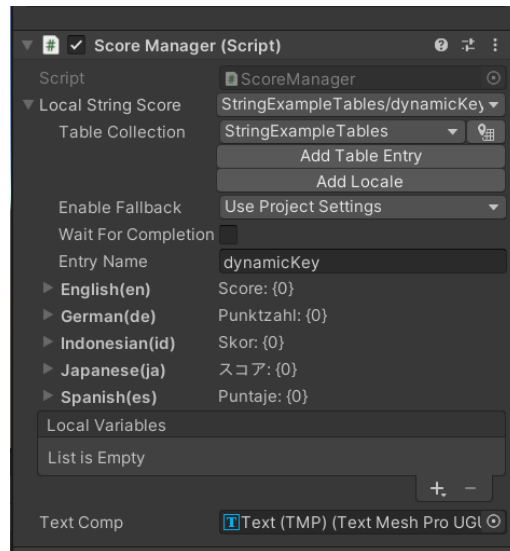


```

Assembly-CSharp
Unity Script | 0 references
7 public class ScoreManager : MonoBehaviour
8 {
9     [SerializeField] private LocalizedString localStringScore;
10    [SerializeField] private TextMeshProUGUI textComp;
11
12    private int score;
13
14    Unity Message | 0 references
15    private void OnEnable()
16    {
17        localStringScore.Arguments = new object[] { score };
18        localStringScore.StringChanged += UpdateText;
19    }
20
21    Unity Message | 0 references
22    private void OnDisable()
23    {
24        localStringScore.StringChanged -= UpdateText;
25    }
26
27    2 references
28    private void UpdateText(string value)
29    {
30        textComp.text = value;
31    }
32
33    0 references
34    public void increasingScore()
35    {
36        score++;
37        localStringScore.Arguments[0] = score;
38        localStringScore.RefreshString();
39    }
40 }

```

Assign the script to text score and select the table as well as the key that we created earlier.

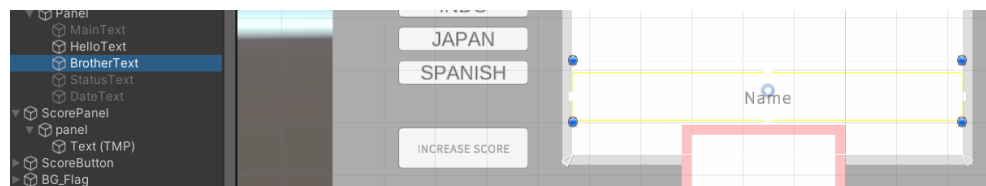


Don't forget to also create a button to change the score data.








- Plural Text

Plural is a feature provided by smart strings that can be used to identify data in the form of an int, where the data will be used to output the type of string according to the int that comes out / gets. Here we can create a simple example, we can create a string that will be useful to provide information about the number of siblings.



The first step is still the same, namely by creating a new key, and entering the string data used.

In this section, we use interpolation to get the data we will use. Interpolation itself is a concept that we can use to retrieve data from scripts that have been assigned to the object.

brotherTextKey	 ✓ Smart	 ✓ Smart	 ✓ Smart	 ✓ Smart	 ✓ Smart
	EditDebug	EditDebug	EditDebug	EditDebug	EditDebug
-	You have {brotherVar.brotherCount: plural:a brother {} brothers}	Du hast {brotherVar.brotherCount: plural:nein Bruder {} Brüder}	Anda memiliki {brotherVar.brotherCount} saudara	あなたには {brotherVar.brotherCount} 兄弟 があります	Tienes {brotherVar.brotherCount: plural:un hermano {} hermanos}

We can first create a value script, which contains the data variables that we will use.

```
Assembly-CSharp
1 using System.Collections;
2 using System.Collections.Generic;
3 using System;
4 using UnityEngine;
5
6 using Random = UnityEngine.Random;
7
8 public class Values : MonoBehaviour
9 {
10     public int brotherCount;
11
12     // Start is called before the first frame update
13     void Start()
14     {
15         //score = scoreManager.sc
16         brotherCount = Random.Range(1, 4);
17     }
18
19 }
```

Assign the script to the created text object, then on the Local Variable in the script event section, click the plus button and select the object reference. Enter the script value in the Object reference section.

✓ Localize String Event

String Reference

StringExampleTables/brotherTextl

Table CollectionStringExampleTables

Add Table Entry

Add Locale

Enable FallbackUse Project Settings

Wait For Completion

Entry NamebrotherTextKey

English(en)You have {brotherVar.brotherCount:pl

German(de)Du hast {brotherVar.brotherCount:plur

Indonesian(id)Anda memiliki {brotherVar.brotherCou

Japanese(ja)あなたには {brotherVar.brotherCount}

Spanish(es)Tienes {brotherVar.brotherCount:plur

Local Variables

Variable NamebrotherVar

Object ReferenceNameText (Values)

+ -

Update String (String)

Runtime OnlyTextMeshProUGUI.text

NameText

+ -

Remember that each data variable must have the same name that we will enter in the table. The key to this feature is to add a "plural" command to the table after the data variable. Also add the plural format by using "|" and "{}" to separate the two types of plural sentences to be used. Every language has its own plural format and we have to adjust to it.

brotherTextKey	Smart	Debug	Smart	Debug	Smart	Debug	Smart	Debug	Smart	Debug
-	You have {brotherVar.brotherCount: plural:a brother {} brothers}		Du hast {brotherVar.brotherCount: plural:einen Bruder {} Brüder}		Anda memiliki {brotherVar.brotherCount} saudara		あなたには {brotherVar.brotherCount} 兄弟 があります		Tienes {brotherVar.brotherCount: plural:un hermano {} hermanos}	

You have {brotherVar.brotherCount:plural:a brother|{} brothers}

- Number Formatting

To use this feature, the process is still the same as Plural and only differs in the command used. Where in this feature the command used is "C" after the data variable.

statusKey	Smart	Debug	Smart	Debug	Smart	Debug	Smart	Debug	Smart	Debug
-	Your money is {moneyVar.moneyCount:C }		Ihr Geld ist {moneyVar.moneyCount:C }		Uang Anda {moneyVar.moneyCount:C }		あなたのお金は {moneyVar.moneyCount:C } です。		su dinero es {moneyVar.moneyCount:C }	

Your money is {moneyVar.moneyCount:C}

- Date Formatting

The date formatting is still the same, where the only difference is the command format used, the format used is a custom date format specifier that is entered after the data variable. The format for each locale is also different and we have to adapt it.

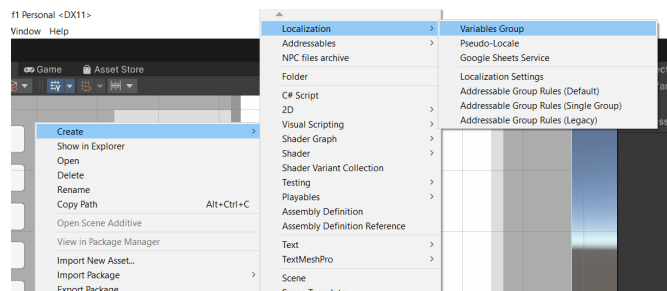
dateKey	Smart	Debug	Smart	Debug	Smart	Debug	Smart	Debug	Smart	Debug
-	Today is {dateVar.dateTime:MM M d}		Heute ist {dateVar.dateTime:MM M d}		Hari ini adalah {dateVar.dateTime:d MMM}		今日は {dateVar.dateTime:MM M d} です		Hoy es {dateVar.dateTime:d MMM}	

Today is {dateVar.dateTime:MMM d}

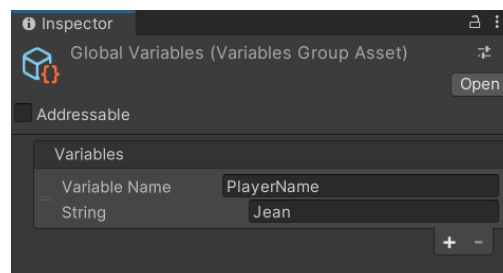
- Global Variables

In the examples above we were able to provide dynamic values that are used in our translation strings at runtime. However, these values are a one-and-done deal: runtime changes to these values will not cause our translation strings to re-render. If we want reactive dynamic interpolation in our translation strings, we can use global variables.

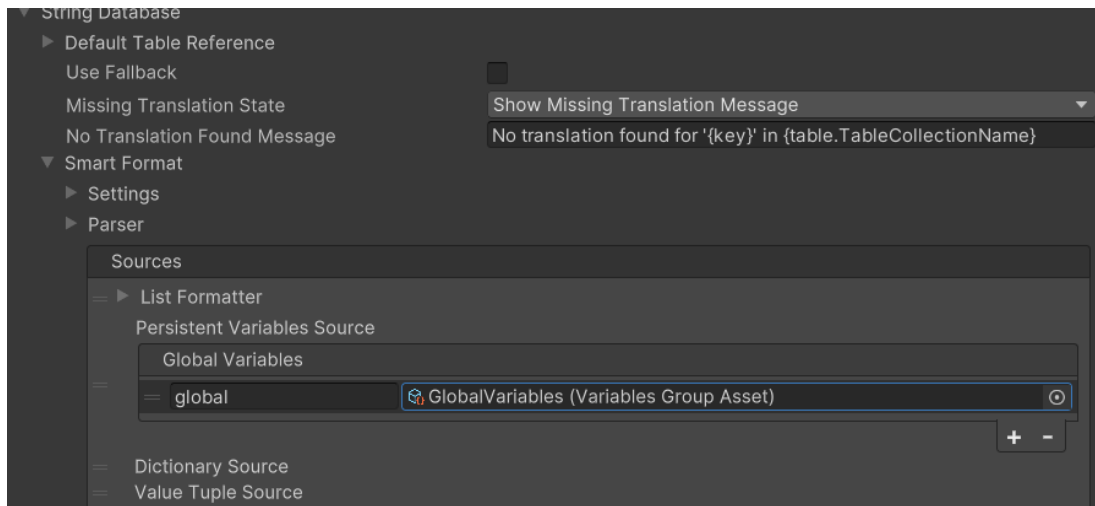
The process for creating a global variable is to create a global variable group. In our project view, we can right-click and select *Create* → *Localization* → *Variables Group*. This will create a new global variables group asset in our project



In Global Variable, Select Add (+) in the Variables list and create a new String variable.



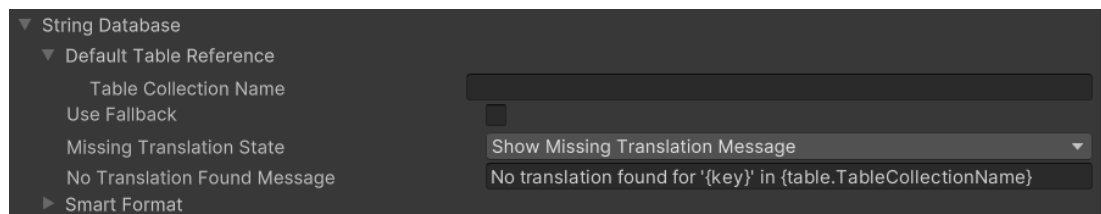
In order to expose our global variables to our translations, we have to wire them up to our settings. Let's open our *Localization Settings* and find the (+) under *String Database* → *Sources*. Clicking the (+) icon will reveal a menu, from which we can select *Global Variables Source*.



Select (+) in the Global Variables Source to create a new item. Keep the default name “global”, this is the value used in the first part of the Smart String `{global.playerName}` and drag the Global Variables Source asset into the slot. The Global Variables source is now configured.

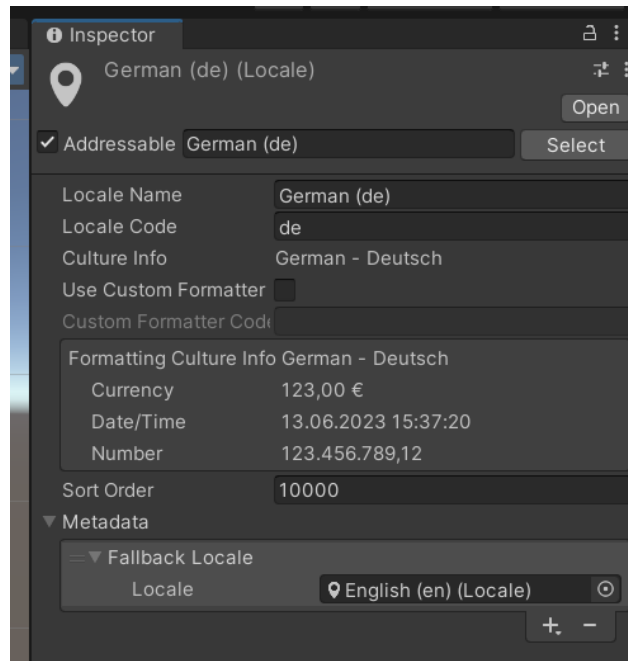
- Fallback

Sometimes we will have missing translations in our projects. There are a few ways to deal with this, and we can set our chosen strategy in *Localization Settings*.



The *Missing Translation State* field can be set to show a warning message instead of the translated string (default). This warning will appear in production as well.

We can also check the *Use Fallback* checkbox to use locale fallbacks for the whole project. This will cause a translation missing in say, French, to “fall back” to its English counterpart. If we go this route we need to make sure to set our fallback locales on each local we want to fall back. Selecting one of the locale assets in our project reveals its details in the *Inspector*. From there we can find the *Metadata* collection and add a *Fallback locale* to it.



- Previewing and Building

Because the localization package uses addressables, we have to build our addressables groups before we can get our updated translations in our standalone builds. To do this, we head over to *Window* → *Asset Management* → *Addressables* → *Groups*. From there, we can click *Build* → *New Build* → *Default Build Script* to build our addressables groups.

