To access to the most recent version of this documentation, visit https://mnkasikci.github.io/QuickLocalizationDocs/

Overview

Quick Localization is a Unity asset that eases the localization process for you with a *minimalistic* and *practical* approach. You can purchase the asset from Unity Asset Store.

See Rest in Joy's YouTube channel for a video tutorial on using this asset as well as many other tutorials on game development and Unity.

Features

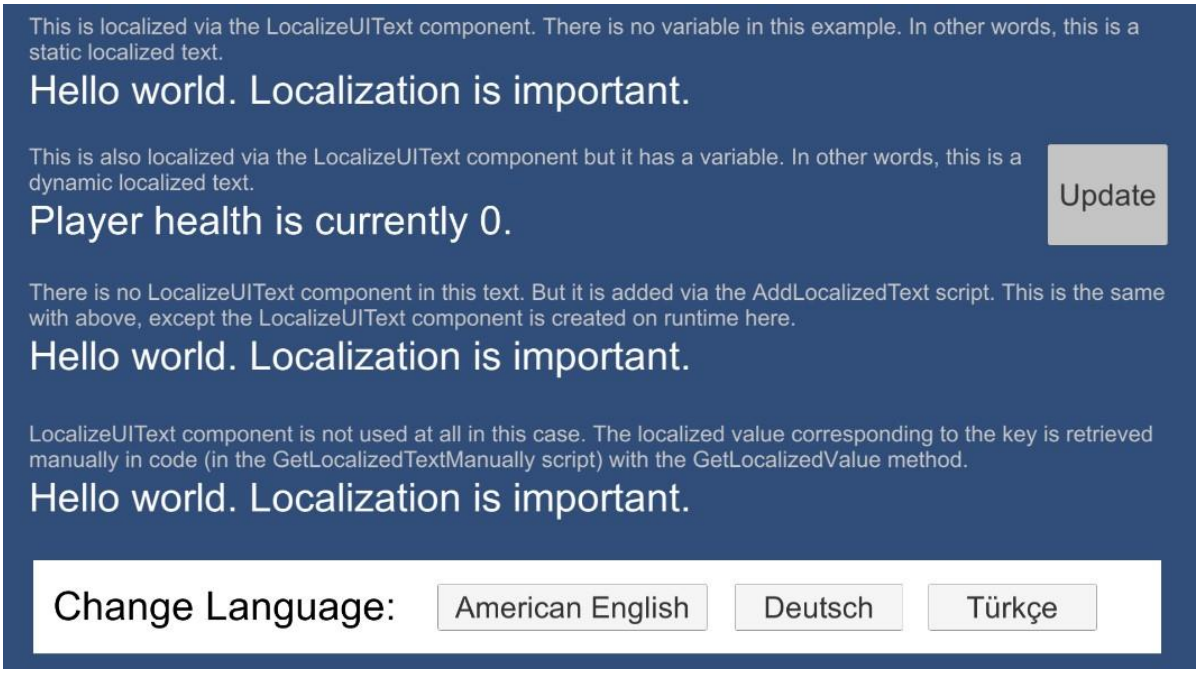
- Simple and Intuitive: The asset implements the well-known key and value pairs system to store and retrieve localized texts.
- Custom Localization Editor Window: Language les can be created and edited through a custom editor window that comes with the asset. Localization Editor allows you to add, edit, or remove localization items (key / value pairs) via a graphical user interface.
- System Language Detection: The asset detects the system language on the rst run and tries to retrieve the localization le for that language. If that fails, the game will load with the main language speci ed by the developer.
- Fallback Languages: If the asset cannot nd the translation for a speci c key in the player's chosen language, it will return the translation in the system language. If that also fails, it will return the translation in the main language speci ed by the developer.
- Remembering Chosen Langauge: Any language chosen by the player in the game is automatically saved in the `PlayerPrefs` le and remembered when the game runs again.
- Localization without Coding (via Inspector): The asset comes with a `LocalizeUIText` component that can be added to any gameobject that has a Text or Text Mesh Pro component. The key and any variables in the localized text can be entered in the inspector without any coding at all.
- Supports Static and Dynamic Texts: In addition to static texts like "Hello world", the asset can localize dynamic texts like "Hello {0}, you have {1} gold".
- Debugging Missing and Used Keys: You never need to worry about any missing keys as the asset logs these to a le in a "Missing Keys" folder. You can also create an empty language le with all the keys used in the game.

Core Components

- Localization Manager: Handles all localization-related tasks in the game.
- Localization Editor: Assists with creating and editing the language les.
- Localize UI Text: A component that is added to any GameObject and localizes the Text or Text Mesh Pro component of that GameObject based on the given key and variables (if any).

Demo

A comprehensive demo is included in the asset under the *Demo* folder. Take a look at the demo in detail to better understand how the asset works.



Contact

The Quick Localization asset is developed and maintained by Rest in Joy and Epiphany at Night.

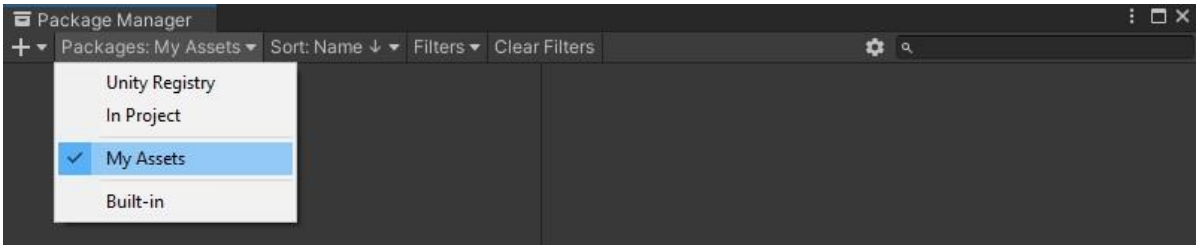
Contact us for all your pre-sale or after-sale inquiries at restinjoy20@gmail.com.

Changelog

- **1.0 Initial release**

Installation

After you purchase the asset from the Unity Asset Store, it will appear in the package manager. You can access the package manager from *Window > Package Manager*. Once the window is open, make sure you are viewing *Packages: My Assets* as the Quick Localization will be listed there.
Select Quick Localization and click the *Import* button located at the bottom right corner.



Once the import is completed, you can access the Localization Editor from the Tools menu, add Localize UI Text component in the inspector, and use the Localization Manager in the scene.

How Localization Works

This will be a primer on how a localization system works in general so you can get the most out of Quick Localization.

Language Files

For each language you will need a le that contains the text in that language. For instance, you might have a le for English and another le for German. The important thing is that you keep translations separate and organize them by language. You do not mix English and German translations in a single le.

Localization Dictionaries

Dictionaries hold key and value pairs. For instance, the key can be "Warm Hello" and the value can be "Hi friend, how are you doing?" This allows you to give the key to the dictionary and request the corresponding value.

Dictionaries are widely used in localization. Each language le acts essentially like a dictionary and the keys are the same throughout all language les. This data structure allows you to use the same key with different language les to retrieve the localized version in the intended language.

See the below example:

Language File	Key	Value
English	Warm Hello	Hi friend, how are you doing?
German	Warm Hello	Hallo Freund, wie geht es dir?

Loading Language Files

There is something special with loading language les. You need to make sure that the language les are loaded and ready to give out values before you load any scene in your game. Otherwise, if the localization les are not loaded properly, the objects in the scene will fail to obtain the localized texts and the result will be localization chaos.

This is why any localization asset should alert you when it loaded the language les and that you can proceed with loading your scenes.

Localization Manager

Localization

Localization Manager is the script that will handle all your localization-related tasks. This page explains how to configure Localization Manager as well as its variables and methods that might be relevant for you.

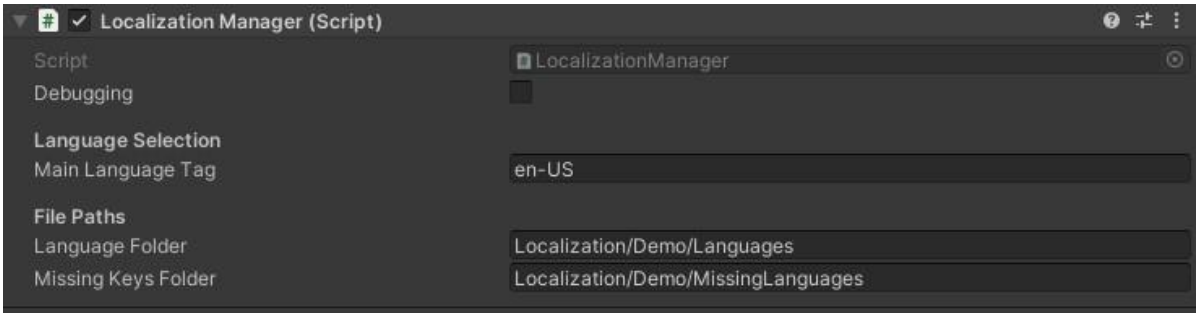
Note that Localization Manager implements the singleton pattern. This means

Localization Manager will always exist throughout your game and will not be destroyed when you load a new scene. You can access Localization Manager from anywhere through its Instance variable without resorting to any FindObjectOfType or similar trick.

Configuration

Create an empty gameobject in the scene and add Localization Manager as a component.
Alternatively, you can drag and drop the Localization Manager prefab in the Prefabs folder.

Afterwards, you will see that Localization Manager has some fields you need to fill in. Below is a ready to use example.



Debugging

This is for debugging purposes only. If checked, `Localization Manager` will log certain messages to the console to help you understand what exactly is happening in the scene.

Language Selection

`Main Language Tag` is where you set the main language of your game. What's important here is that you should use the language tag here, not the name of the language. This means if you want the main language to be American English, you will enter `en-US` not `English` or `American English` .

Language tags allow `Localization Manager` to sort out even the small differences between use of a language in different regions (such as English in Britain and USA). The most common language tags can be found below but for a full list of the language tags, see this website.

English - United Kingdom	**en-GB**
English - United States	**en-US**
French - France	**fr-FR**
German - Germany	**de-DE**
Spanish - Spain	**es-ES**

Note that your language file should exactly match this tag so for American English, the language file should be named en-US.

File Paths

File paths are where in your directory you will store your language les and missing keys logs. All paths are relative to your assets folder. So you only need to write `Languages` if your language les are located at `/Assets/Languages` . Never start a path with `/` because then the path will not be relative.

There are two le paths:

Language Folder is where all your language les should be located.

Missing Keys Folder is where Localization Manager will keep the logs of the keys that were not translated in the given language le.

Readiness

Whether Localization Manager is ready and the language les are properly loaded can be checked by accessing IsReady property and checking if it is true . This can simply be done like this in any of your scripts:

```
if(LocalizationManager.Instance.IsReady == true)
{
// Your code here
}
```

Methods

There are several methods you might want to make use of.

GetAvailableLanguages()

This method does not take any parameter and will return a Dictionary<string, string> containing the native names of the available languages and their language tags. For instance, if you have en-US and de-DE in your Language Folder , this method will return the following key and value pairs:

Key	Value
English	en-US
Deutsch	de-DE

ChooseLanguage(string languageTag)

If you would like to change the language and set a new language at runtime, you can use the ChooseLanguage method. It has only one string parameter, which is the languageTag . This is the same with the language tag we mentioned in the con guration of Localization Manager . So if you would like to change the language to German, you can use the following code:

```
LocalizationManager.Instance.ChooseLanguage("de-DE");
```

Once the ChooseLangauge method is called, Localization Manager will automatically save the chosen language in the PlayerPrefs le and remember it the next time.

AddLocalizeUIText(GameObject gameObject, string key, params object[] variables)

If you would like to add a LocalizeUIText component to a gameobject on runtime, you will need to use this method. You simply need to specify the relevant gameobject, key for the localization, and the variables (if any).

Note that this is a static method so you do not need to access the Instance rst. You can just use the following code:

```
LocalizationManager.AddLocalizeUIText(menuStartButtonGameObject, "Start Button");
```

GetLocalizedValue(string key, params object[] variables)

You can use this method if you would like to retrieve the localized value of a key manually. You can provide the variables in the method as well.

Scene Load

Before you load any scene in the game, you need to make sure that the language les are loaded and Localization Manager is ready to give out localized values . Otherwise, if the localization les are not loaded properly, the objects in the scene will not be able to retrieve the localized values.

What you need to do is to check if Localization Manager is ready before you load the scene in another script. This is typically done by a Game Manager script that continuously checks if the Localization Manager is ready and loads the scene only when it is. For this purpose, Localization Manager has an IsReady property that will return true once it is ready.

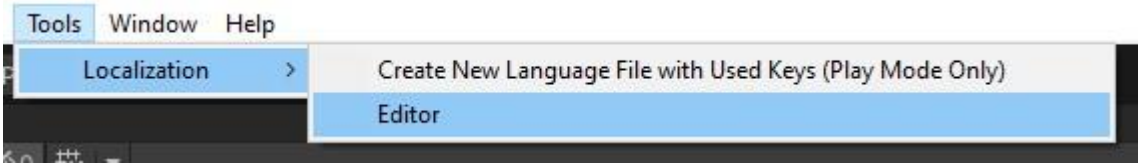
You can use an if statement like below to check if the Localization Manager is ready:

```
if(LocalizationManager.Instance.IsReady == true)
{
// Load the scene here
}
```

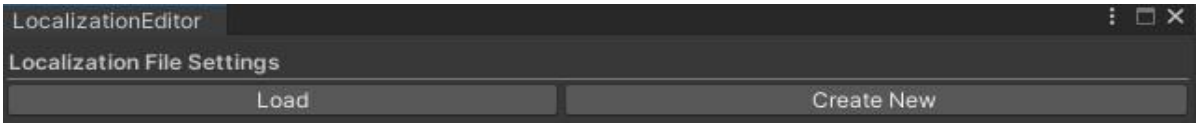
See the demo for a sample implementation of this system.

Creating and Editing Language Files

Use Localization Editor custom window to create and edit language files. You can access Localization Editor from the Tools menu:



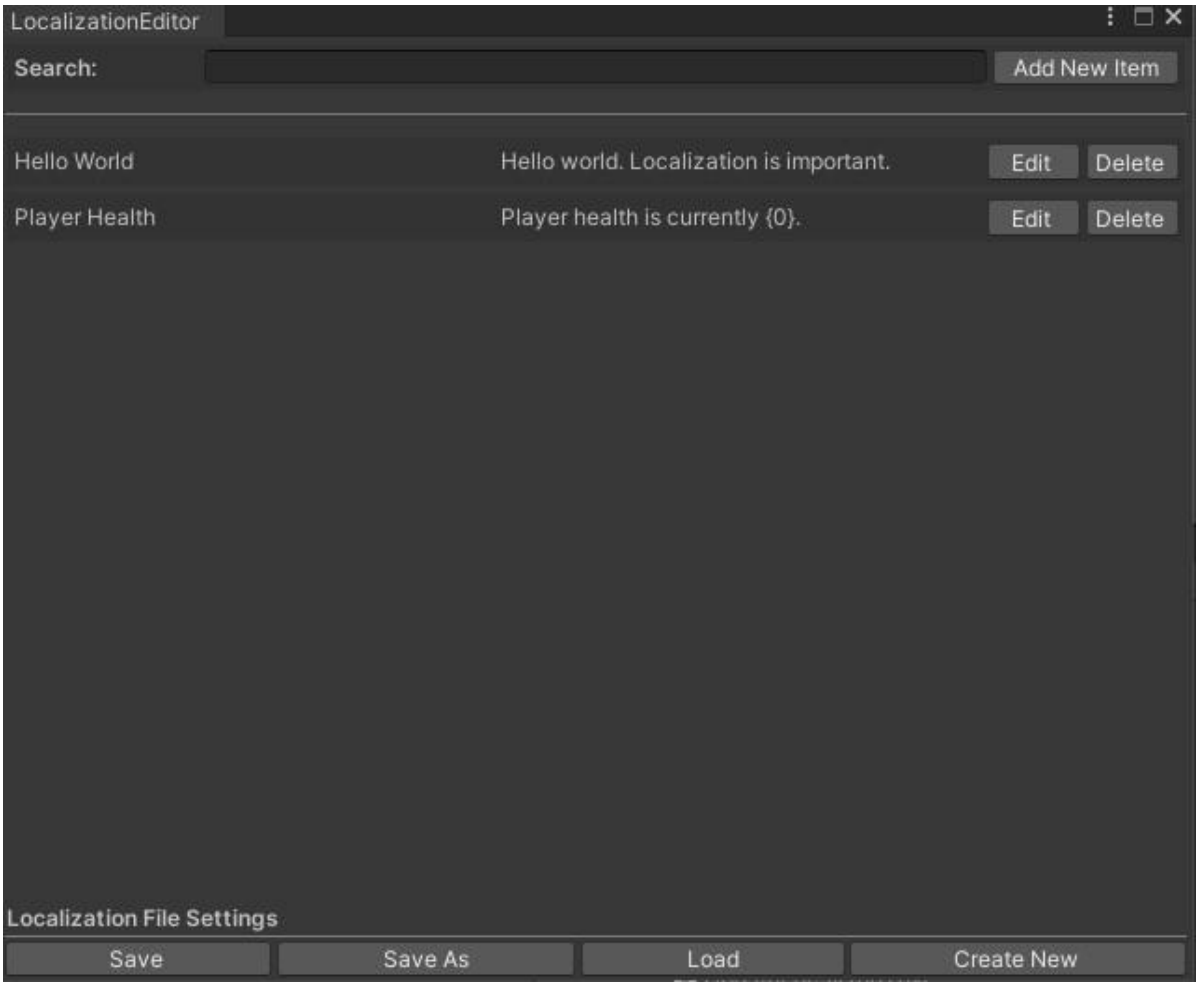
Once the editor is open, you can load language les or create a new language le. If you click on Create New , a new language le will be created and then you will be able to add key and value pairs to it. Remember that the language le should exactly match the language tags so for American English, the language le should be named en-US .



The rest is self-explanatory:

- You can add a new key and value pair by clicking on Add New Item .
- You can edit an existing key and value pair by clicking on Edit next to that item.
- You can search keys and values in the search bar at the top of the window.

- You can use le methods like save, save as, load, or create new at the bottom of the window.



All language les are serialized (saved) in JSON format.

Debugging Features

Missing Keys

You never need to worry about any missing keys in a language le as Localization Manager logs these to a separate le in the Missing Keys folder you speci ed in Localization Manager .

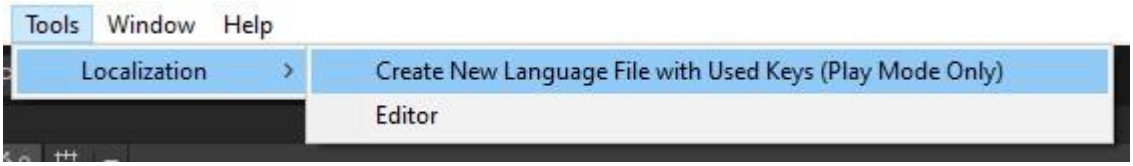
For instance, let's assume that your game was translated from American English to German. And the translator missed the Warm Hello key and did not translate it. When your game runs in German, some script will try to get the localized value for Warm Hello and will fail. In that case, two things will happen:

1. The fact that Warm Hello was not found in the German language le will be logged in a le in the Missing Keys folder.

2. Localization Manager will return the localized value for the fallback language, which is rst the system language of the player and if that also fails, the main language you set in Localization Manager

Used Keys

You can create an empty language le with all the keys used in the game by going to Tools > Localization:



For instance, you might be working on a prototype where you do not want to invest any time in sorting out the translation just yet. In that case, you can just use any key you would like and when you run the game, you can create a language le with all the keys used.

Note that this feature only works in Play Mode.

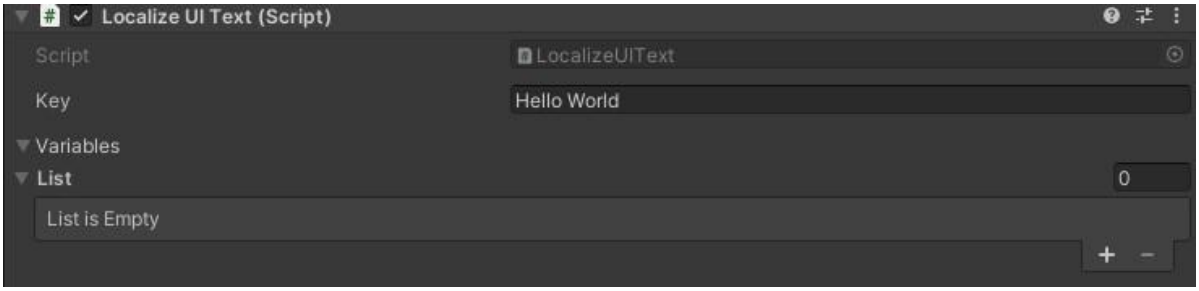
Using Localize UI Text Component

LocalizeUIText is a ready to use component that can localize any Text or Text Mesh Pro text with the given key. All you need to do is to add LocalizeUIText to the gameobject with the Text or Text Mesh Pro component and enter the key and variables (if any).

Static Localized Text

'Static' here means there is no variable in the localized text, meaning there is no external data that needs to be taken into account when retrieving this localized text. For instance, *"Hi friend, how are you?"* is a static localized text.

For static localized text, you only need to enter the key and nothing else. You can leave the variables list empty.



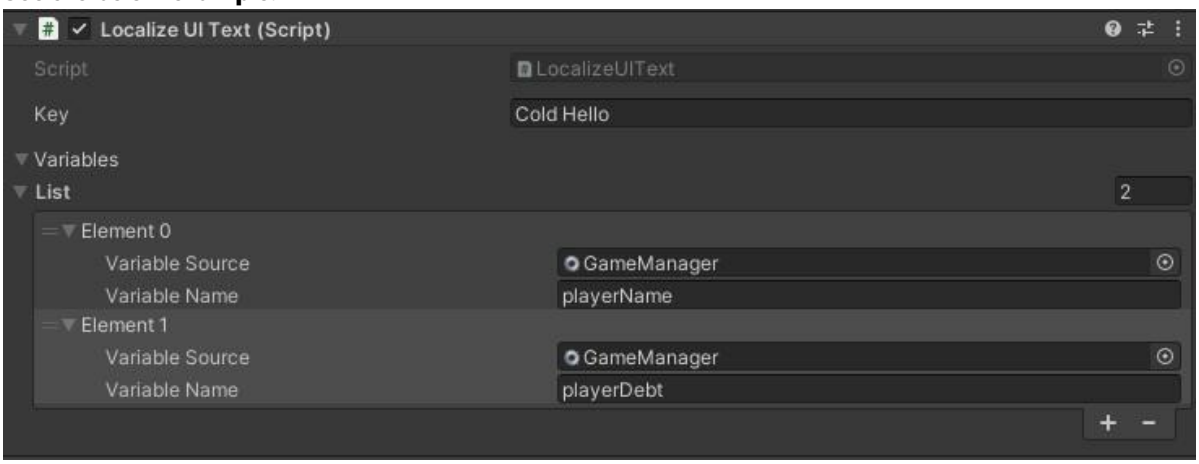
Dynamic Localized Text

'Dynamic' means there will be variables in the localized text so we will need external data to create the nal localized text. For instance, *"Hi* Joseph*, how are you doing?"* can be a dynamic localized text if Joseph will need to change based on the name the player given itself. If the player named itself George , the localized text will need to be *"Hi* George*, how are you doing?"*. This is done by using a special markdown in the localized value and giving the relevant data in the LocalizeUIText component.

In our example, the localized text will be *"Hi {0}, how are you doing?"* {0} is a placeholder here and will be replaced with the data we provide by Localization Manager . The localized value can have more than one variables like *"Hi {0}, you owe me {1} gold."* Here, we will need to provide the name of the player as well as the amount it owes as variables in the exact same order because the rst variable will replace {0} and the second will replace {1} .
For each variable, you need to provide a source and its exact name in that source:

- Source: The source can be a script that implements the singleton pattern (with an Instance variable), a scriptable object, or a gameobject component.
- Name: Name of the property or the eld should be written exactly as it is in the source.

See the below example:



Updating Dynamic Localized Values

Some variables you use in localized values might need to be updated such as player health. If you have a localized value like *"Player health is {0}"*, it will not be enough that this is retrieved once but it will need to be updated in intervals of your choice. For this purpose, you can simply call the UpdateText() method of the relevant LocalizeUIText component.

If you are updating a LocalizeUIText component that you *created in the inspector*, you do not need to give any parameters to the UpdateText() method as it will remember the variables you provided in the inspector:

```
[Serializefield] LocalizeUIText playerHealthUI;
...
playerHealthUI.UpdateText();
```

However, if you are updating a `LocalizeUIText` component that was *created on runtime* with `LocalizationManager.AddLocalizeUIText(...)`, then you will need to provide the relevant variables in the UpdateText() method:

```
[Serializefield] LocalizeUIText playerHealthUI; public
int playerHealth;
...
playerHealthUI.UpdateText(playerHealth);
```

Retrieving Localized Text Manually

You can use `LocalizeUIText` to automate the localization by leveraging components but sometimes you may need to manually retrieve the localized value of a key. In such cases, you can use the `GetLocalizedValue(string key, params object[] variables)` of `Localization Manager`.

For static localized text, you can leave the variables empty and for dynamic localized text, you can provide the variables by separating them with a comma. For instance:

```
Localization Manager.GetLocalizedValue("Warm Hello"); // This is static and will get "Hi friend, how are you doing?"

Localization Manager.GetLocalizedValue("Cold Hello", playerName, playerDebt); // This is dynamic and will get "Hi {playerName}, you owe me {playerDebt} gold"
```