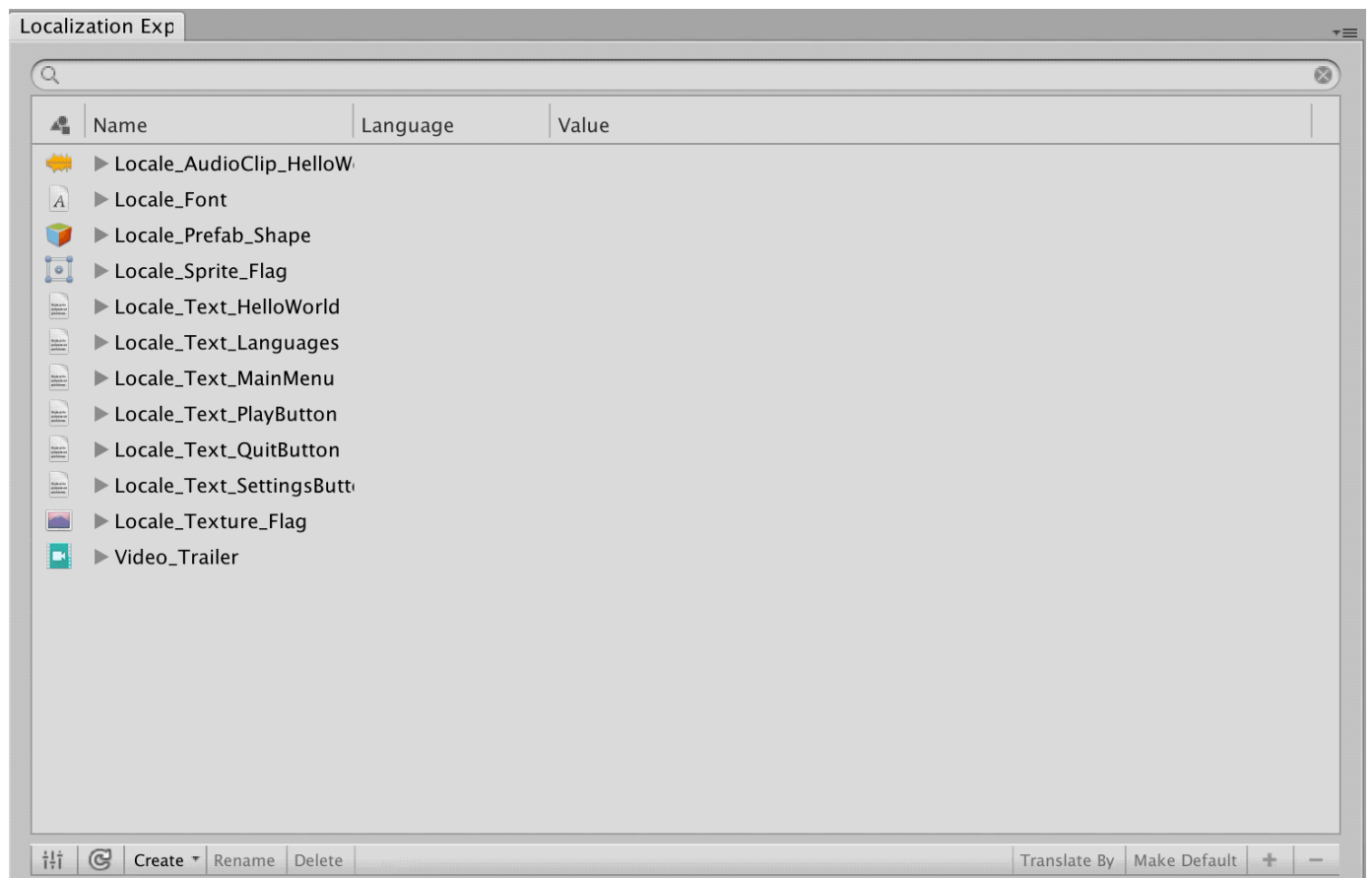


What is Asset Localization?

Easy-to-use and generic asset localization system for Unity.

With [Asset Localization](#), you can easily localize your built-in Unity game assets. Also you can create custom localizable asset functionality for your custom assets.

Available on [Asset Store](#).



Preview

Features:

- Built-in [Text](#), [TextMesh](#), [TextMeshPro](#), [TextAsset](#), [VideoClip](#), [AudioClip](#), [Sprite](#), [Texture](#), [Font](#) and [Prefab](#) localization
- Edit everything through [Localization Explorer](#) window
- [Quick text translation](#) inside the Editor with [Google Translate](#) engine
- [Extensible localized asset](#) support
- Used locales are automatically added to Info.plist on iOS build
- Lightweight and easy to use
- No coding required

 Required Unity 2017.1 or newer version.

Getting Started

1. Importing the Package in Your Project

Open or create your project in Unity. Then import the Asset Localization package from the Asset Store window in Unity editor. You can find detailed explanation [here](#).

2. Creating Localization Settings

Localization settings is created automatically when package is imported. Also you can manually create via

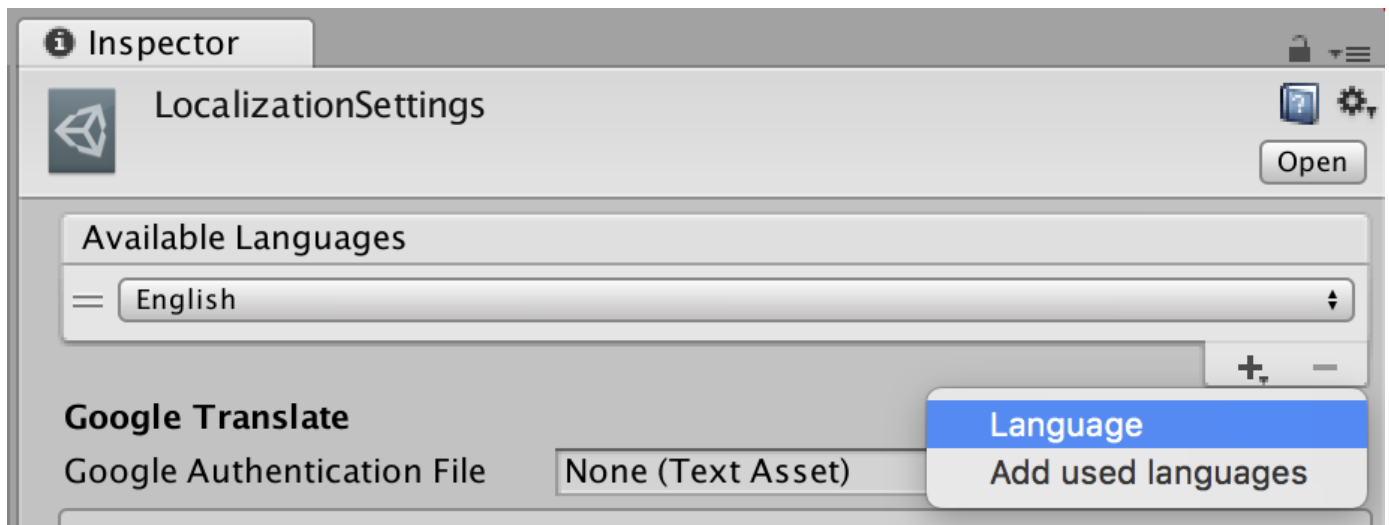
```
Project Window -> GameToolkit -> Localization -> Localization Settings
```



Settings file must be kept under any **Resources** folder. You can read detailed explanation about [Resources](#).

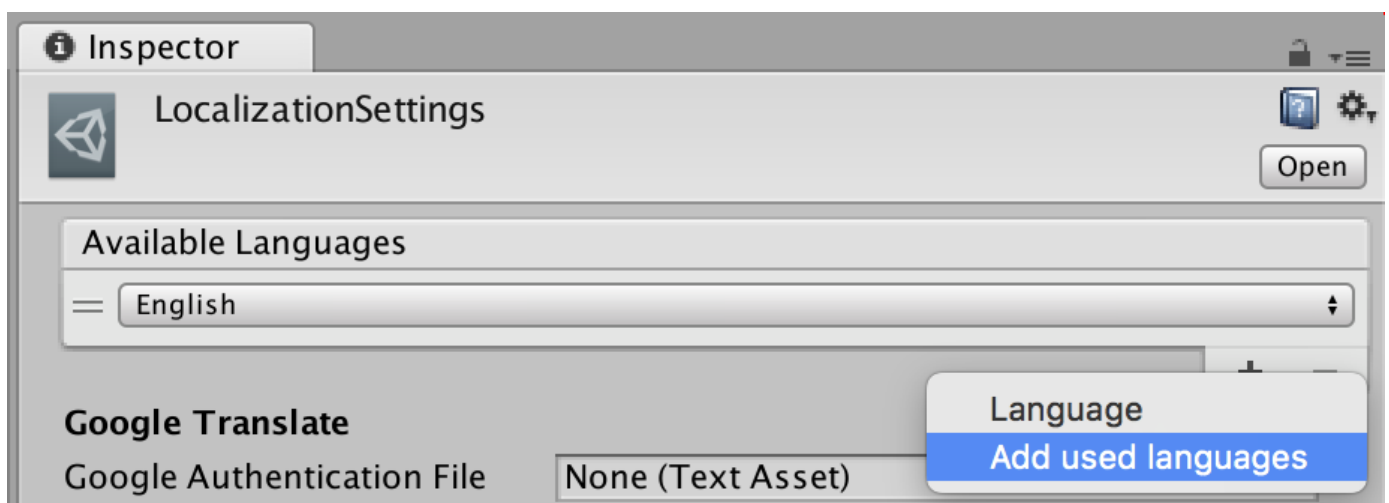
3. Localization Settings

You can add languages that must be available in you game as many as you want via:



Adding a language

or add used languages in your assets via:



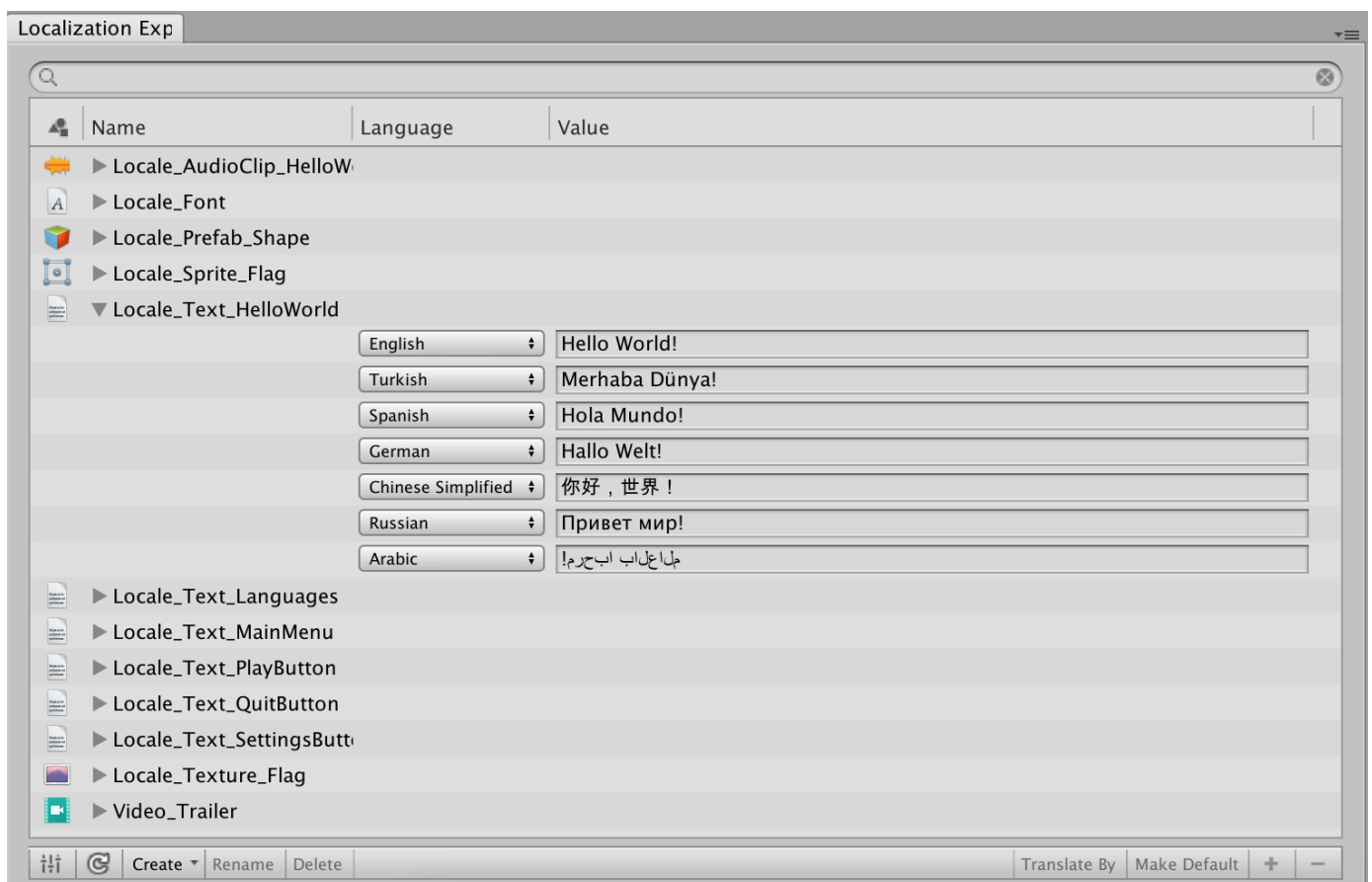
Adding used languages

If you want to use **Quick Translate** option, you should set **Google Authentication File** claimed from **Google Cloud**. For more information look at:

[→ Quick Translate](#)

/quick-translate

Localization Explorer



Localization Explorer

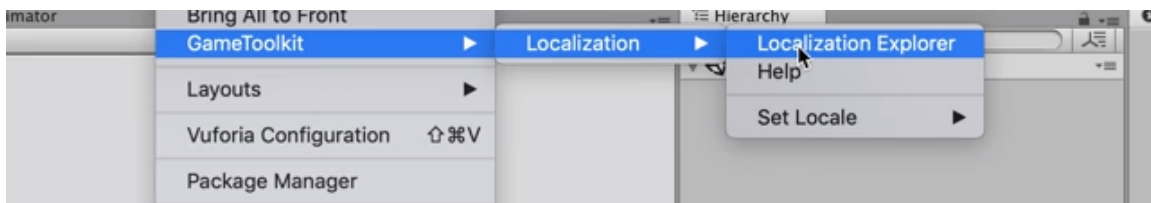
1. Opening Localization Explorer

Localization explorer lets you manage your localized assets in a single window. You can access the window via

```
Window -> GameToolkit -> Localization -> Localization Explorer
```



H. Ibrahim Penekli
@ibrahimpenekli



4.81 s

SD

10 views



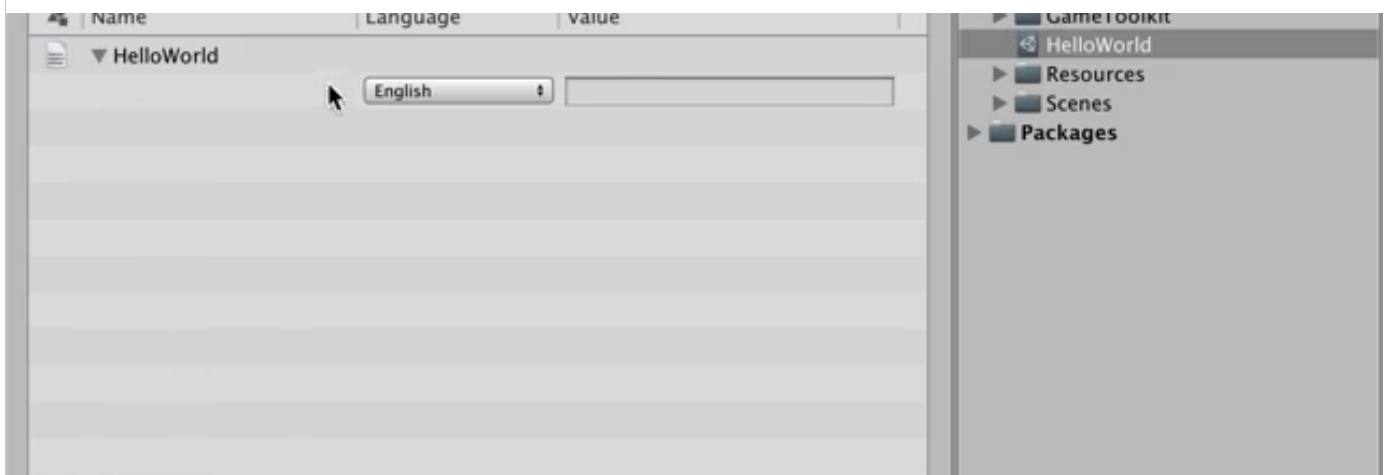
2. Add/Modify/Delete localized asset

Add a localized asset by clicking `Localization Explorer -> Create`

- Select an asset type from drop-down menu opened.
- Give a name for the created asset. The asset created is shown in the localization explorer.
- Selected localized asset can be renamed by clicking **Rename** button and deleted by clicking **Delete** button.
- Built-in **Duplicate (Ctrl+D)**, **Delete (Shift+Del)** and **Frame Selected (F)** commands works with multi-selection.
- Expand the created asset and add **locale (language & value pair)** as many as you want by clicking **+** button. Also you can remove the selected locale by clicking **-** button through the explorer.



H. Ibrahim Penekli
@ibrahimpenekli



6.00 s

SD

9 views

gfykot

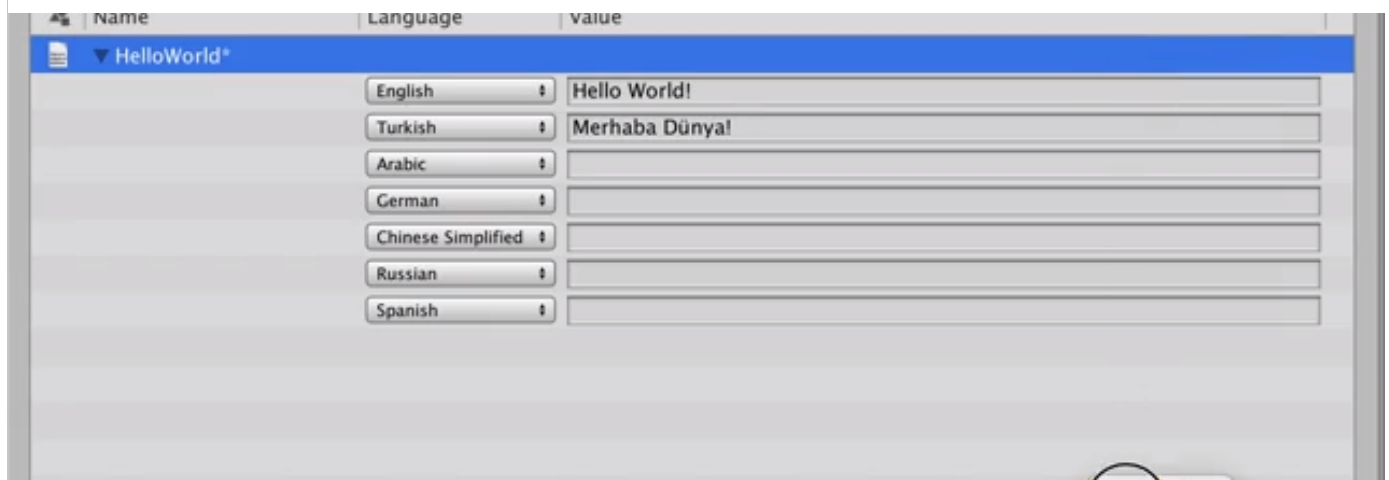
3. Quick translate missing locale(s)

You can translate missing locale(s) automatically by clicking **Translate By** button:

- Choose a language that has already translated



H. Ibrahim Penekli
@ibrahimpenekli



3.13 s

SD

8 views

gfykot



Quick Translate option is available only for **Text** assets and it must be configured. See [Quick Translate](#) configuration to learn how to configure it correctly.

Localizing Components

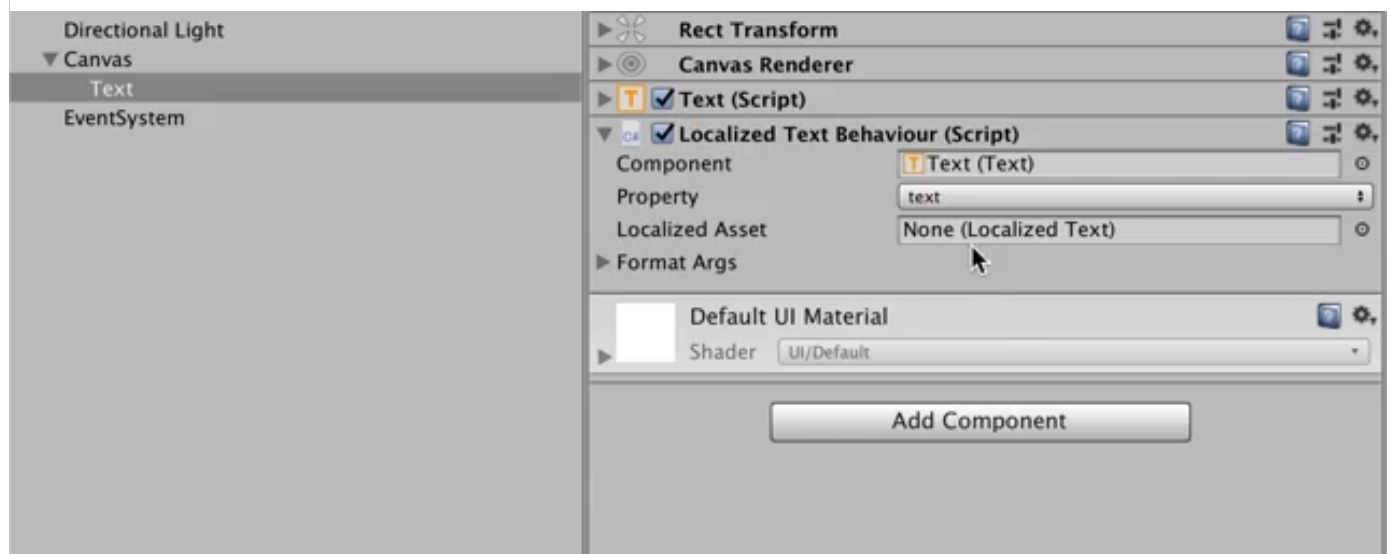
Localized assets must be attached to the actual game objects in the scene in order to set localized value to the appropriate component's attribute. The system offers various ready-to-use components.

Localizing a Text Component

- While the Text game object selected, click **Add Component** button in the inspector.
- Search for **Localized Text** and add it.
- **LocalizedTextBehaviour** needs a component and it's property to set localized value to the property.
- Drag the `Text` component from the game object to the **LocalizedTextBehaviour's Component** field.
- The localization system searches for the text (`string`) based properties. **Property** drop-down lists available properties. Simply select **text** property.
- Finally, select localized text asset from the **Localized Asset** field.



H. Ibrahim Penekli
@ibrahimpenekli





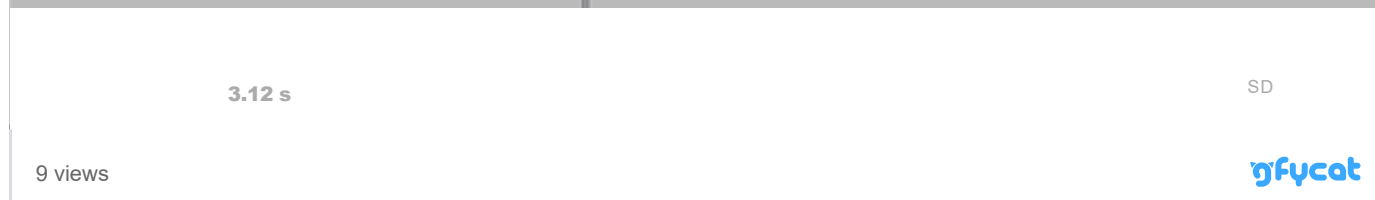
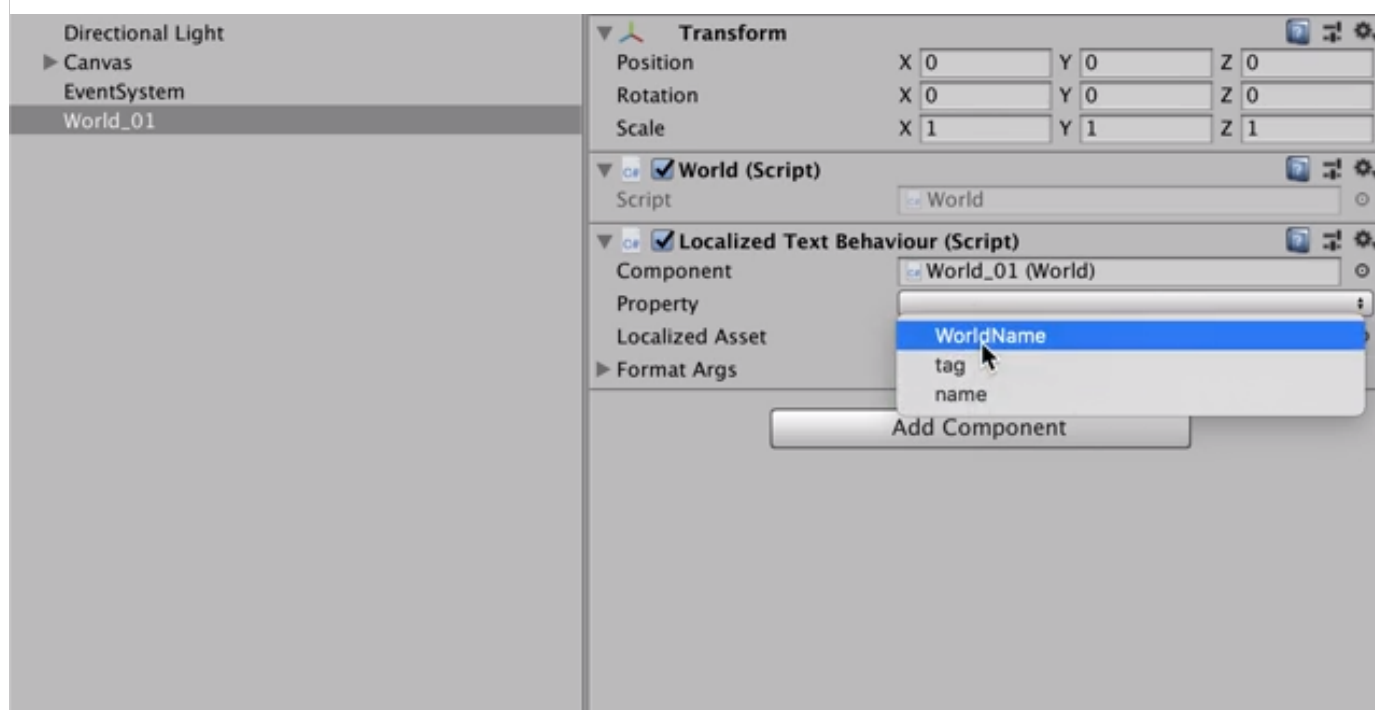
Localizing Text Component

When the game starts, the localization sets the **text** property of the selected component with selected localized asset's value. Updates the value whenever game language has changed.

This generic component & property selection provides us to use at any component that has text based property. For example; we can use same behavior with same steps for the [Text Mesh Pro](#) component. You should follow the similar steps for all localized asset types. For instance; attach **LocalizedSpriteBehaviour** for the [Image](#) component and select [sprite](#) property and done!



H. Ibrahim Penekli
@ibrahimpenekli



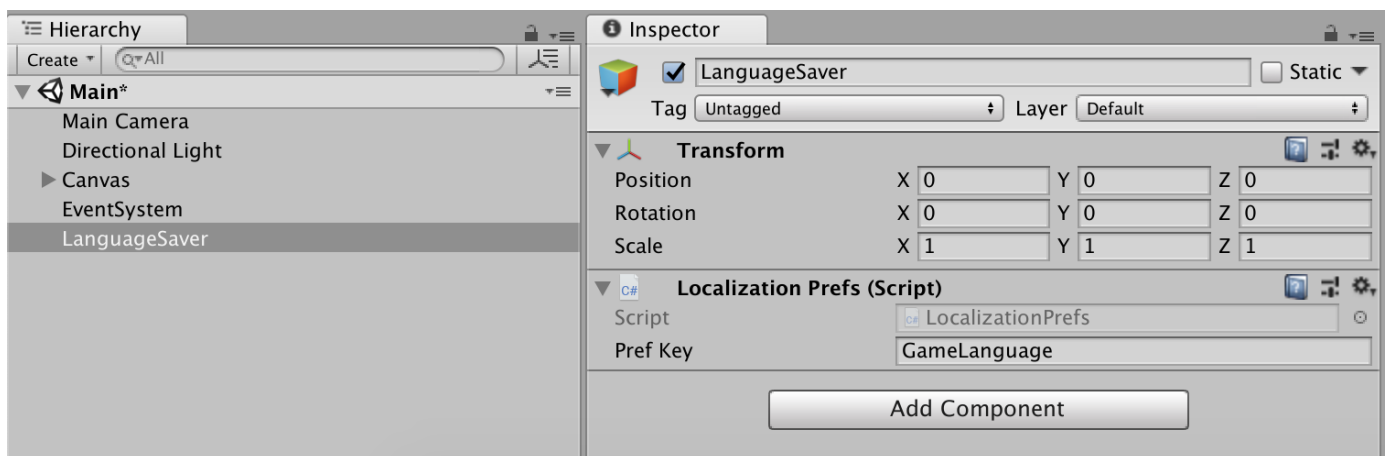
Localizing Custom Component & Property

If you want to go further and create your custom localization behavior, see [here](#).

Language Persistence

Localization Preferences

If you would like to persist language across the game runs, use **LocalizationPrefs** component. Attach the component to any game object you want. **LocalizationPrefs** uses the [PlayerPrefs](#) to keep application language on the device storage. So you can set [PlayerPrefs](#) key from **Pref Key** field from inspector.



LocalizationPrefs

When the user selects a language from e.g. drop-down, **LocalizationPrefs** detects the language change and saves the selected language to the [PlayerPrefs](#) with key you chosen. Whenever application starts, **LocalizationPrefs** restores the selected language and sets the application language. You should place **LocalizationPrefs** game object into the initial scene of your game.

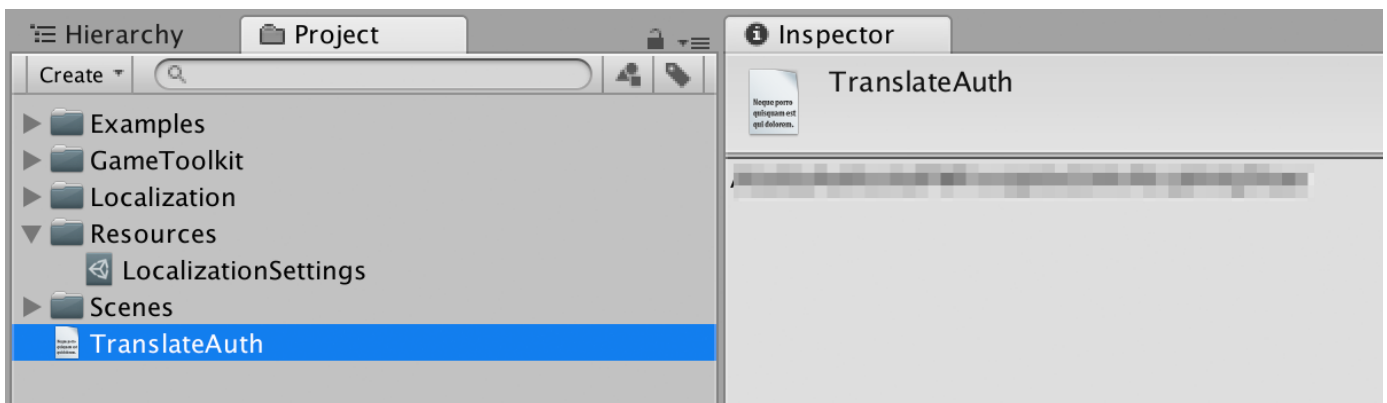
Quick Translate

Google Cloud Translation API

Google Cloud Translation API provides a simple programmatic interface for translating an arbitrary string into any supported language. Quick translate uses the API for translating missing locales. Google Translate API requires an authentication for pricing. More detailed information [here](#).

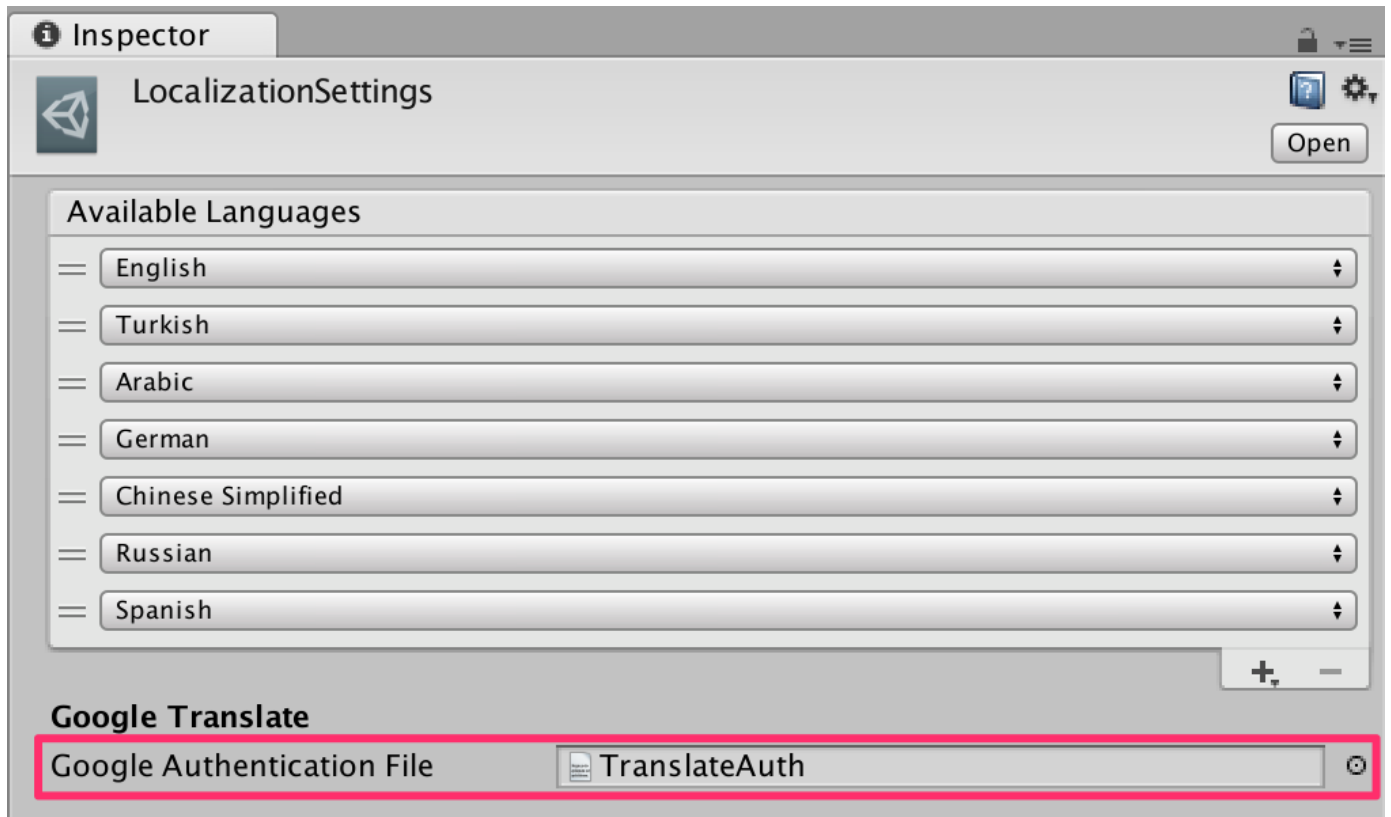
Configure localization settings for Quick Translate

- First obtain your API key from [Google Cloud Console](#). See [how-to guide](#).
- Create a text file in project assets and paste your API key into here. (Make sure that the copied key is correct one)



Google Cloud Auth File (Contains API key)

- Open **LocalizationSettings**
- Attach the created key file to **Google Authentication File** field.



Settings - Google Authentication File

Finally, you can translate missing locales automatically. See [here](#).

- i If you want to use quick translate feature only in the Editor, exclude the auth file when deploying your game for production. For more information how to exclude an asset see [here](#).


Scripting Reference

Localization Settings

```
1 // To get localization settings:
2 LocalizationSettings localizationSettings = LocalizationSettings.Instance;
3
4 // You can get available languages as:
5 List<SystemLanguage> availableLanguages = localizationSettings.AvailableLanguages;
6
7 // To access Google auth file:
8 TextAsset authFile = localizationSettings.GoogleAuthenticationFile;
```

Localization Manager

```
1 // You can get current language:
2 SystemLanguage currentLanguage = Localization.Instance.CurrentLanguage;
3
4 // or you can set current language:
5 Localization.Instance.CurrentLanguage = SystemLanguage.English;
6
7 // or set by system language:
8 Localization.Instance.SetSystemLanguage();
9
10 // or set by default language defined in LocalizationSettings (first item is the default)
11 Localization.Instance.SetDefaultLanguage();
12
13 // Register application locale changed event:
14 Localization.Instance.LocaleChanged += (object sender, LocaleChangedEventArgs e) =>
15 {
16     Debug.Log("Application locale has changed from " + e.PreviousLanguage + " to " + e.CurrentLanguage);
17 };
```

- 
- Use
- `Localization.Instance`
- if only if application is playing. See
- [Application.isPlaying](#)
- .

Extending Custom Localized Asset

Creating custom localized asset is involved simple steps:

- Extend class from `LocalizedAsset<T>` , enter your asset type for generic parameter:

```
1 [CreateAssetMenu(fileName = "MyLocalizedCustomAsset", menuName = "GameToolkit/L
2 public class MyLocalizedCustomAsset : LocalizedAsset<MyCustomAsset>
```

- Your custom localized asset automatically registered under
Localization Explorer -> Create menu if you set `menuName` property of
`CreateAssetMenu` attribute as `"GameToolkit/Localization/<your_asset_name>"`
- Create serializable asset item by extending `LocaleItem<T>` , enter your asset type for generic parameter again (it is necessary for the Unity to serialize object):

```
1 [Serializable]
2 private class MyCustomLocaleItem : LocaleItem<MyCustomAsset> { };
```

- Define locale items array with concrete type you declared:

```
1 [SerializeField]
2 private MyCustomLocaleItem[] m_LocaleItems = new MyCustomLocaleItem[1];
```

- Finally, implement getter method for getting locale items with the base class type:

```
public override LocaleItemBase[] LocaleItems { get { return m_LocaleItems; } }
```

Complete code:

```
1 using System;
2 using UnityEngine;
3 using GameToolkit.Localization;
4
5 [CreateAssetMenu(fileName = "MyLocalizedCustomAsset", menuName = "GameToolkit/L
6 public class MyLocalizedCustomAsset : LocalizedAsset<MyCustomAsset>
7 {
8     [Serializable]
9     private class MyCustomLocaleItem : LocaleItem<MyCustomAsset> { };
10
11     [SerializeField]
12     private MyCustomLocaleItem[] m_LocaleItems = new MyCustomLocaleItem[1];
13
14     public override LocaleItemBase[] LocaleItems { get { return m_LocaleItems;
15 }
```

Congratulation! You have a custom localized asset that can use your game.

Extending Custom Localized Asset Behaviour

If you want to extend localized asset behavior, you have two options:

- If you want to implement completely custom behavior, you should extend from `LocalizedAssetBehaviour` .
- If you want to create generic component & property based behavior for your custom localized asset, then you should extend from `LocalizedGenericAssetBehaviour` .

1. Extending from LocalizedAssetBehaviour

You must extend your class from `LocalizedAssetBehaviour` and override the `UpdateComponentValue()` appropriately. This method is invoked every-time when game starts or application language has changed. You should update the component property with your custom localized asset's value.

```
1 public class MyLocalizedAssetBehaviour : LocalizedAssetBehaviour
```



```

2  {
3      public MyLocalizedCustomAsset LocalizedAsset;
4
5      protected override void UpdateComponentValue()
6      {
7          // Update the specified property with current value.
8          ... = LocalizedAsset.Value;
9          // or using safe value getter
10         ... = GetValueOrDefault(LocalizedAsset);
11     }
12 }

```

2. Extending from LocalizedGenericAssetBehaviour<TAsset, TType>

The only step you need to take is the extend from `LocalizedGenericAssetBehaviour` and specify your custom asset type for the first generic parameter, and specify your asset value type as the second generic parameter. That's it!

```

1  public class MyLocalizedAssetBehaviour : LocalizedGenericAssetBehaviour<MyLocal
2  {
3  }

```