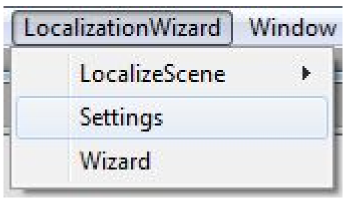
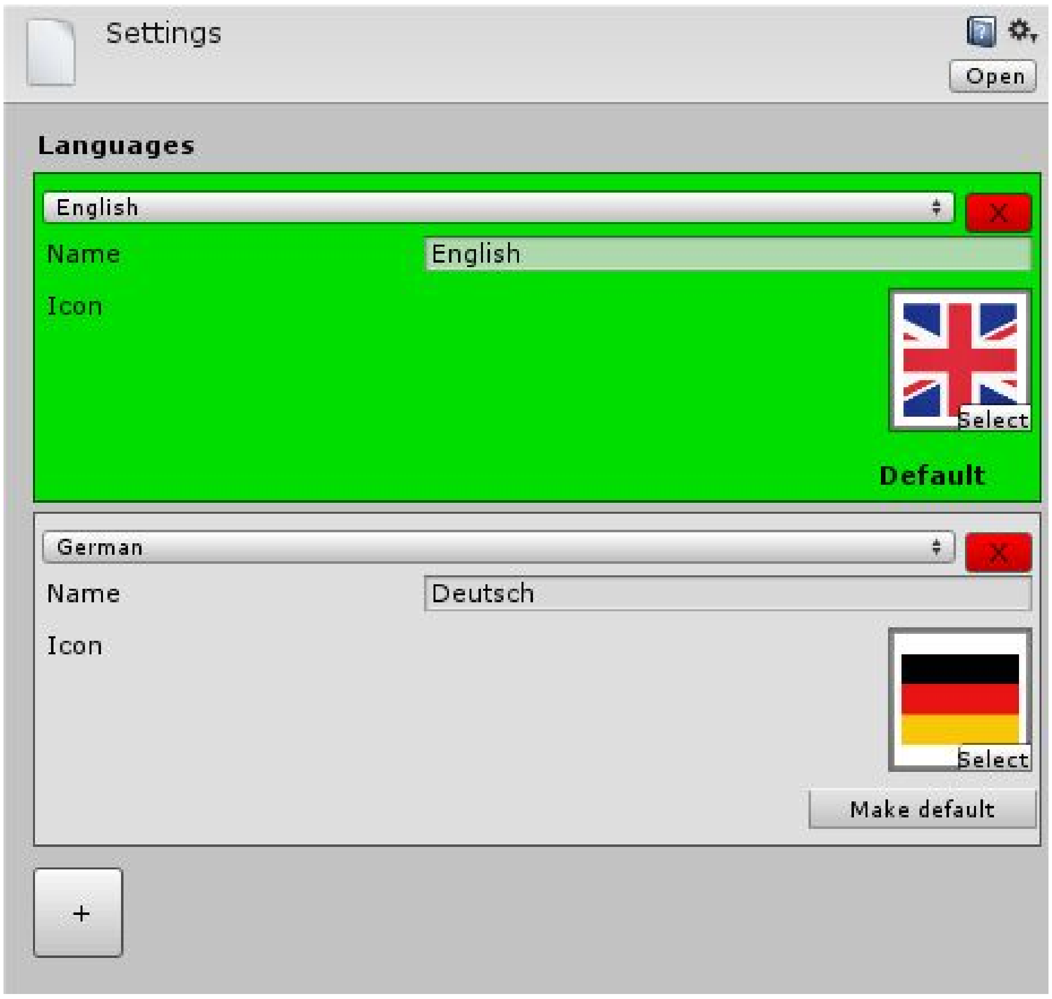
LocalizationWizard

LocalizationWizard is the tool for fast and easy localization of your Unity3d project. With LocalizationWizard you can quickly add localization of user interface text, sounds and textures. Also, if necessary, you can add other types of localization. With generic classes it would be quick and easy.

1 How to configure

The first thing you need is to specify the languages available for localization and the default language. To do this, open the settings window by pressing the menu button **LocalizationWizard/Settings** or click “**Open settings**” in the wizard.



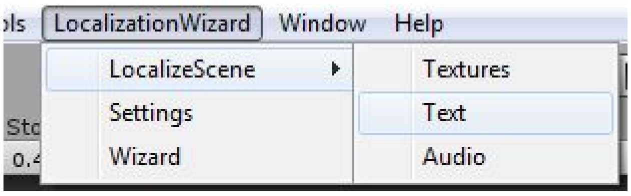


You can also specify the icon and name for the language for more compact display of the current language of the project.

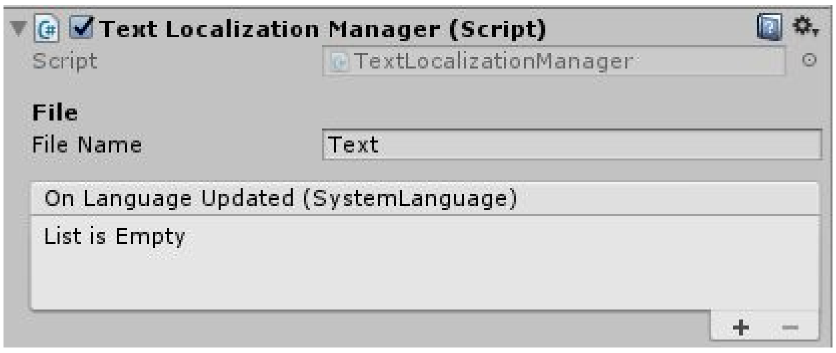
When the program is started for the first time, the language will be automatically set to the one that is selected in the system. If the system language is not supported by your application, then the language marked as default will be installed.

2 How to use.

First you need to add to the scene your LocalizationManager. You can place manager manually or select menu item **LocalizationWizard/LocalizeScene/Text**.



If you want to use your custom localization file you need to specify a name for the XML file in the **File Name** field. Your custom localization xml files will be saved in path Assets/LocalizationWizard/Resources/**{language}**/**{filename}**.xml.


Text Localization Manager (Script)

Script: TextLocalizationManager

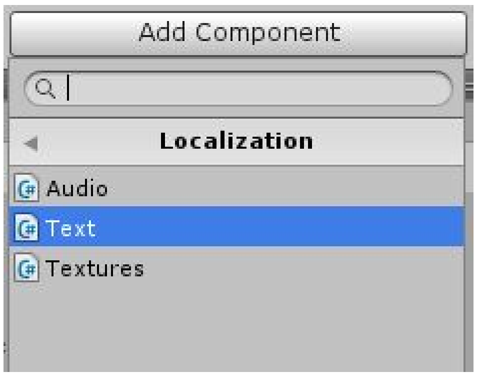
File

File Name: Text

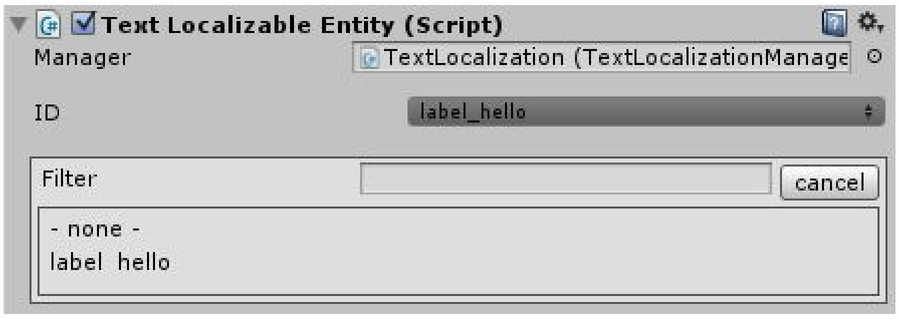
On Language Updated (SystemLanguage)

List is Empty

Next, place **LocalizableEntity** on the scene. You can add LocalizableEntity component using **Add Component** menu. Select *Localization item*.



Now you can edit placed entity. Select **ID** of value that you want to be localized in this LocalizableEntity.


Text Localizable Entity (Script)
Manager
TextLocalization (TextLocalizationManage
ID
label_hello
Filter
cancel
- none -
label_hello

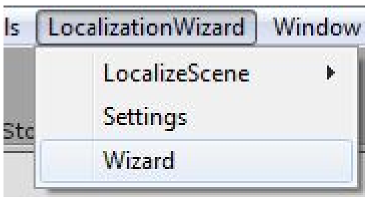
If you select " **- none -** " then this entity will not localize any value. In run-time it will do nothing.

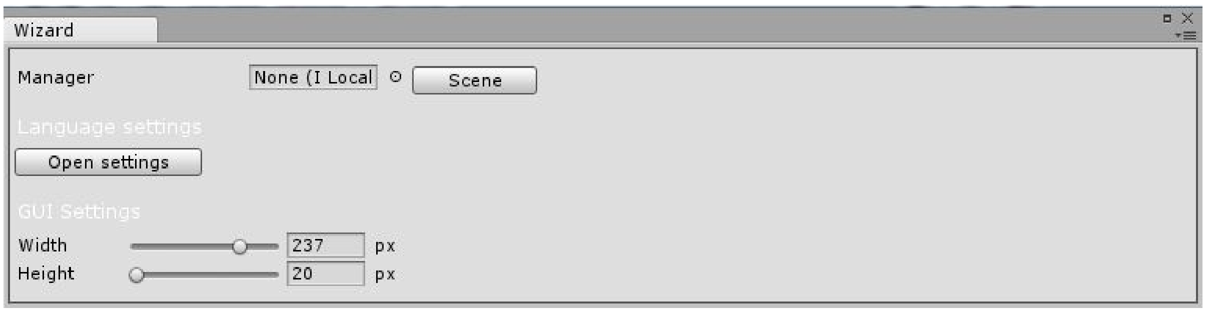
The value in the field **Manager** is the LocalizationManager, with whom **LocalizableEntity** will interact.
You don't need to select it manually, because this field fills automatically when LocalizableEntity placed to scene.

Selected value will be applied when object became enabled or you changing language in run-time.

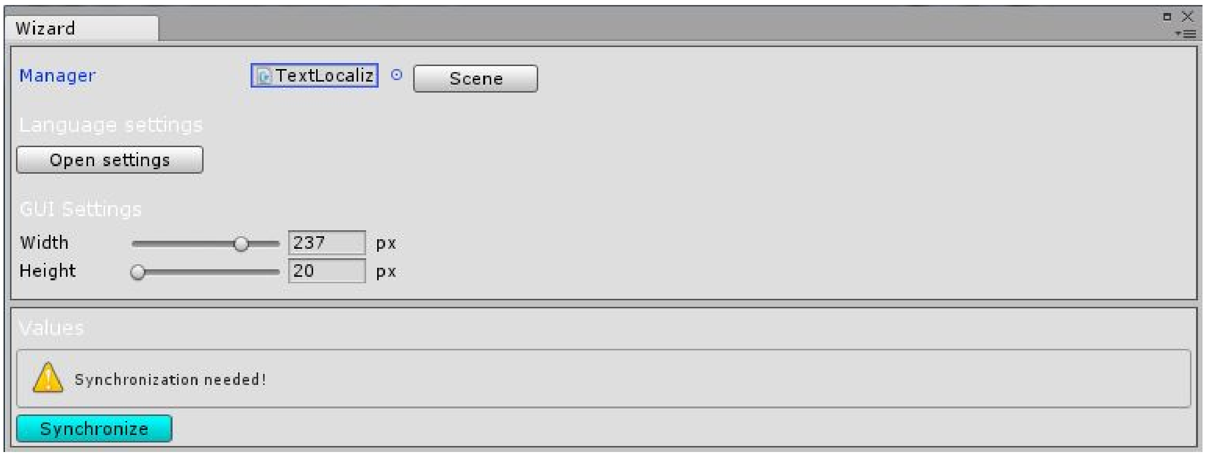
3 How to edit localization files.

Open the wizard by clicking on the button in the menu **LocalizationWizard/Wizard**.





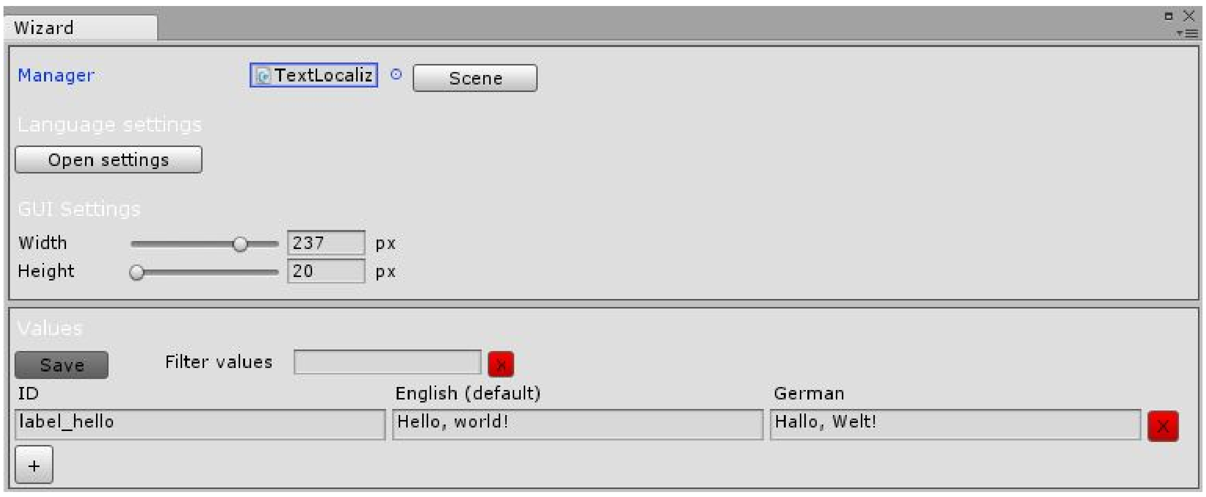
In the *Manager* field, select your **LocalizationManager**.



Synchronization needed!

If you don't have any localization files or change the available set of languages, you need to synchronize the XML files. **After synchronization tool will generate XML files for all languages available and unused - will be removed.**

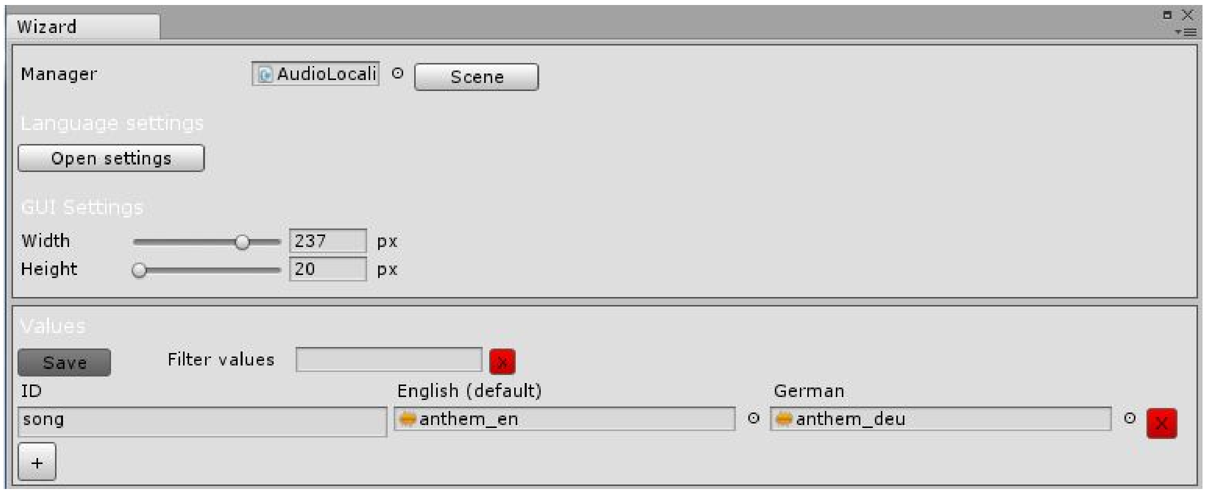
After synchronization, you can fill the table.



ID	English (default)	German
label_hello	Hello, world!	Hallo, Welt!

For managers with different localizable types window will looks different. *This window for TextLocalizationManagers*.

This - for AudioLocalizationManager.


Wizard

Manager AudioLocali Scene

Language settings

Open settings

GUI Settings

Width 237 px

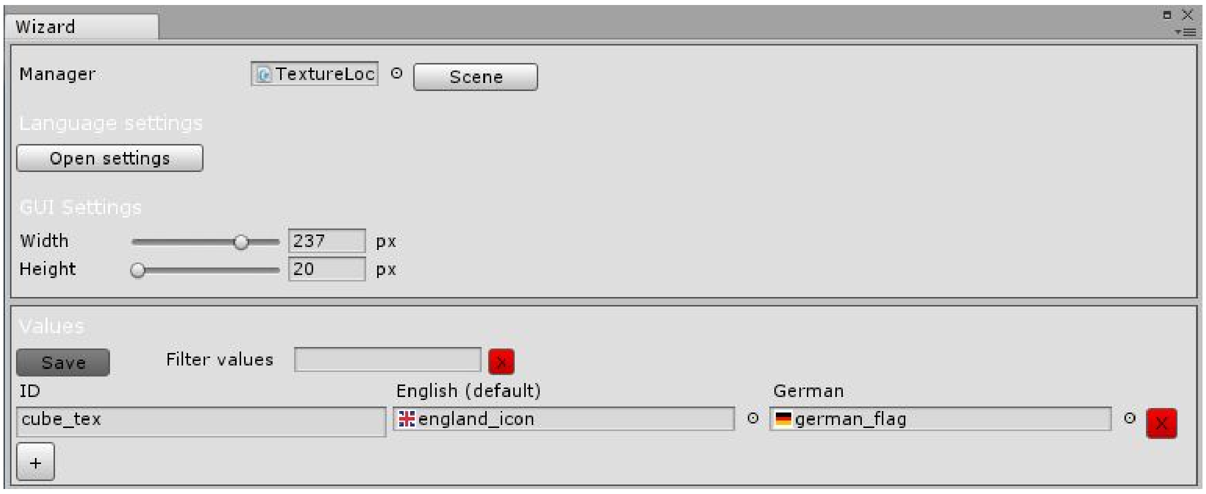
Height 20 px

Values

Save Filter values

ID	English (default)	German
song	anthem_en	anthem_deu

Window for TextureLocalizationManager.


Wizard

Manager TextureLoc Scene

Language settings

Open settings

GUI Settings

Width 237 px

Height 20 px

Values

Save Filter values

ID	English (default)	German
cube_tex	england_icon	german_flag

Important: if you use localization of AudioClip or Texture from the presented example, remember that all the textures and audio files must be in **Resources folder**.

4 How to add your own localization type **(for advanced users)**

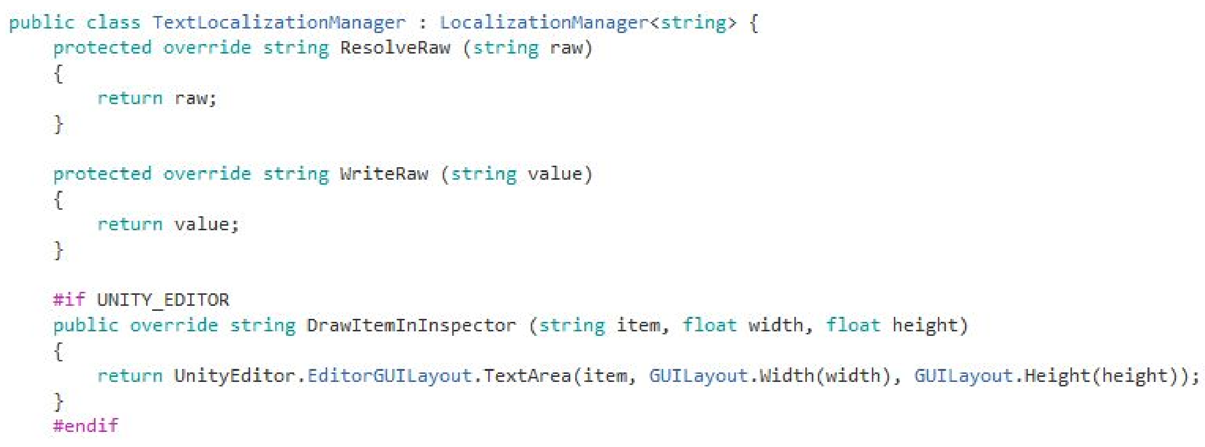
To add a new type of localization will need to create several classes:

- **LocalizationManager**
- **LocalizableEntity**

In **LocalizationManager** you must describe how your value will be recorded and read to/from XML file and describe how your value will show in the Wizard window.

Consider the example of localizing the text.

You must inherit your class from LocalizationManager and provide generic type which you want to localize.
(In the example the type is string)



```
public class TextLocalizationManager : LocalizationManager<string> {
    protected override string ResolveRaw (string raw)
    {
        return raw;
    }

    protected override string WriteRaw (string value)
    {
        return value;
    }

    #if UNITY_EDITOR
    public override string DrawItemInInspector (string item, float width, float height)
    {
        return UnityEditor.EditorGUILayout.TextArea(item, GUILayout.Width(width), GUILayout.Height(height));
    }
    #endif
```

WriteRaw method takes a value and writes it to the XML as a string.

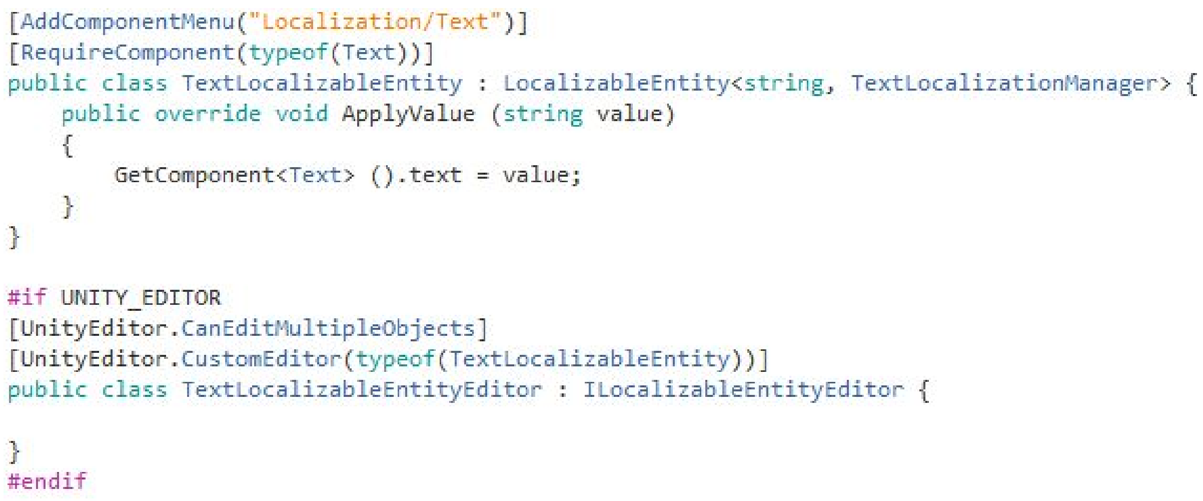
ResolveRaw - returns your value, reading it from the string of XML.

For string in any action not necessary, but if you need to localize (for example) numeric values in these methods you need to convert your values into a string and parse them.

The *DrawItemInInspector* method takes you localizable value type, height and width of GUI elements. For the string type is used TextArea. For types AudioClip and Texture2D is used ObjectField.

This method must be compiled only in UnityEditor. Obsolete method using **#if UNITY_EDITOR** and **#endif** directive.

The next class, which you need to create is **LocalizableEntity**. This class must to be inherited from LocalizableEntity with generic types of localizable value and the type of LocalizationManager which localizes this value.
(In the example the type of localizable value is string and type of LocalizationManager is TextLocalizationManager)



```
[AddComponentMenu("Localization/Text")]
[RequireComponent(typeof(Text))]
public class TextLocalizableEntity : LocalizableEntity<string, TextLocalizationManager> {
    public override void ApplyValue (string value)
    {
        GetComponent<Text> ().text = value;
    }
}

#if UNITY_EDITOR
[UnityEditor.CanEditMultipleObjects]
[UnityEditor.CustomEditor(typeof(TextLocalizableEntity))]
public class TextLocalizableEntityEditor : ILocalizableEntityEditor {

}
#endif
```

ApplyValue method accepts a value of your type (e.g. string) and you just need to write it somewhere. In the example I write the text in *Text.text*.

To correctly display your entity in the inspector you need to add the Editor class that is inherited from **ILocalizableEntityEditor**.

This class must be compiled only in UnityEditor. Obsolete it using **#if UNITY_EDITOR** and **#endif** directive.

If your entity contains additional fields, you need to override ***DrawExtensionsGUI*** method to display them in inspector.

For example there is the code of TextureLocalizableEntity class, which contains three additional fields.

```
using UnityEngine;
using System.Collections;

namespace LocalizationWizard {
    [AddComponentMenu("Localization/Textures")]
    [RequireComponent(typeof(Renderer))]
    public class TextureLocalizableEntity : LocalizableEntity<TextureData, TextureLocalizationManager> {
        public int materialndex = 0;
        public string textureName = "_MainTex";
        public bool useSharedMaterial = false;

        public override void ApplyValue (TextureData value)
        {
            Renderer r = GetComponent<Renderer> ();
            if (materialndex >= 0 && materialndex < r.materials.Length) {
                if (useSharedMaterial)
                    r.sharedMaterials [materialndex].SetTexture (textureName, value.texture);
                else
                    r.materials [materialndex].SetTexture (textureName, value.texture);
            }
        }
    }

    #if UNITY_EDITOR
    [UnityEditor.CanEditMultipleObjects]
    [UnityEditor.CustomEditor(typeof(TextureLocalizableEntity))]
    public class TextureLocalizableEntityEditor : ILocalizableEntityEditor {

        UnityEditor.SerializedProperty textureProp, sharedProp, materialIndexProp;

        protected override void OnEnable ()
        {
            base.OnEnable ();
            textureProp = serializedObject.FindProperty ("textureName");
            sharedProp = serializedObject.FindProperty ("useSharedMaterial");
            materialIndexProp = serializedObject.FindProperty ("materialndex");
        }
        protected override void DrawExtensionsGUI ()
        {
            TextureLocalizableEntity entity = target as TextureLocalizableEntity;
            Renderer r = entity.GetComponent<Renderer> ();

            GUILayout.Space (10f);
            UnityEditor.EditorGUILayout.PropertyField (materialIndexProp);
            UnityEditor.EditorGUILayout.PropertyField (textureProp);
            GUILayout.Space (5f);
            UnityEditor.EditorGUILayout.PropertyField (sharedProp);
        }
    }
    #endif
}
```

Important: if you localize some object type *(e.q. AudioClip, Texture, GameObject)*, you (probably) have to create additional class to associate your object with another value that you can get in run-time. *For AudioClip and Texture I have created an object that connects them with their name in the Resources folder*. For this method requires that all objects be in the Resources folder. You can also use AssetBundle system for this.

Thanks for using this asset! Rate it in **AssetStore**! Good luck!