One Click Localization

Last updated: Aug 28, 2020

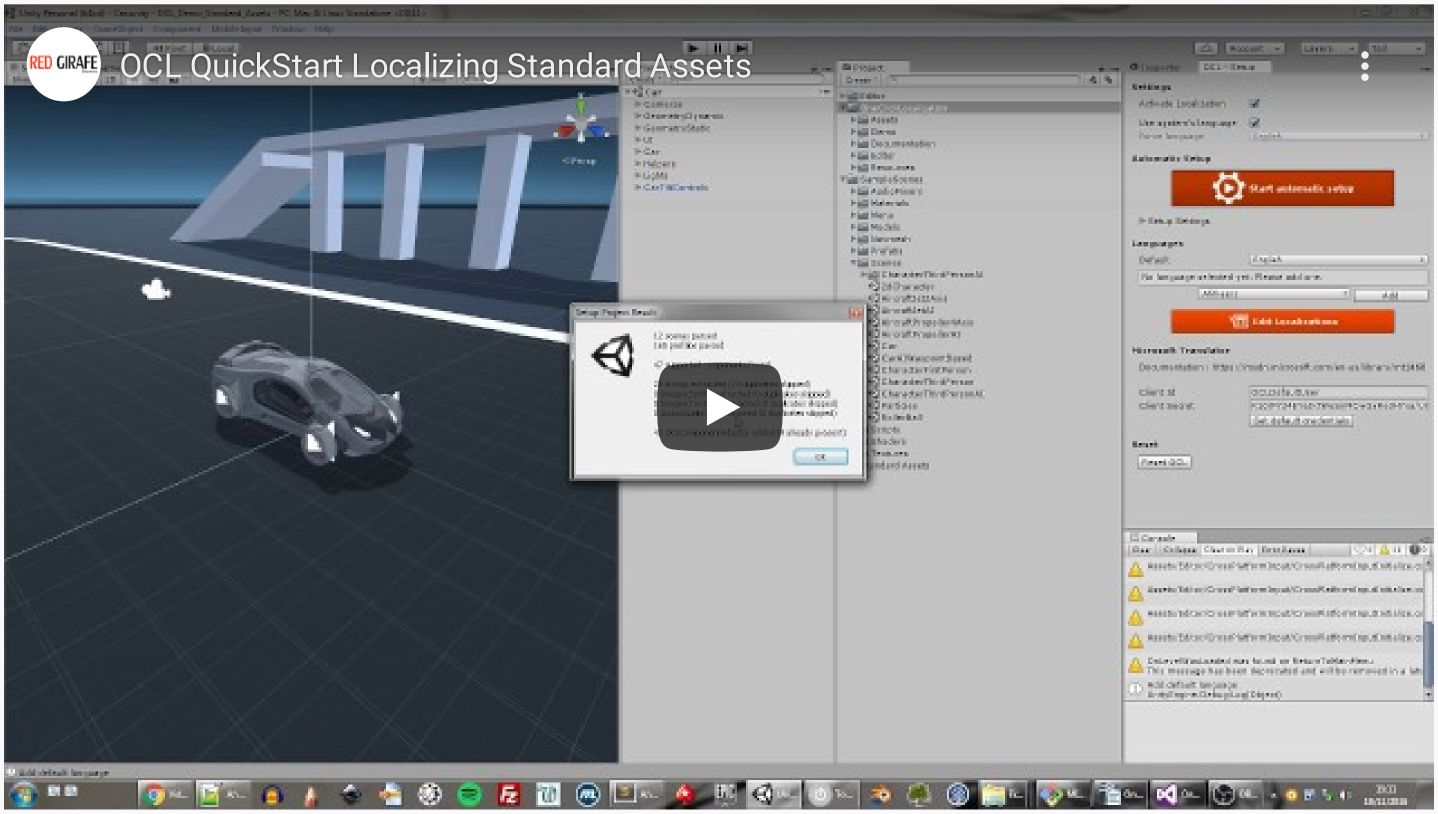
What is it?

One Click Localization (OCL) is a Localization Tool for Unity designed to be easy and fast to use.
The aim of OCL is that... **you don't think about localization**... OCL is based on values instead of keys like most of the localization systems, that's why you don't need to take care of it until you really want localize your project.

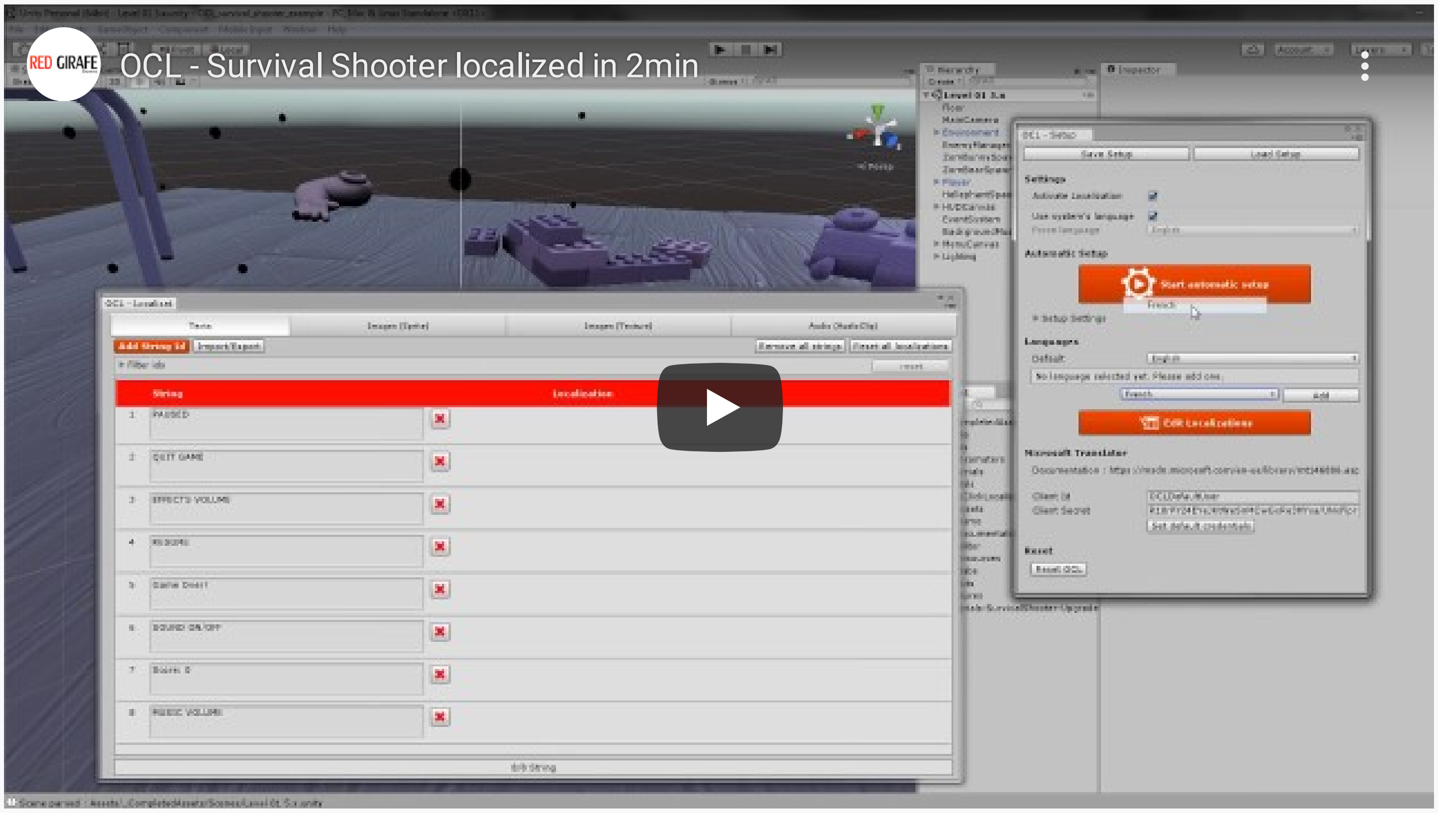
Its customizable automatic setup and its generic component adapter will let you localize a project in minutes, whithout a single line of code.

Demos

Quick Start Video - Unity Standard Assets



Example - Survival Shooter localized in 2min



Test App

▶ WebGL

Use the Test App in Editor

To use the Test App in the Editor, you must first load its setup located in
OneClickLocalization/Demo/Saved_Setup/DemoSetup.asset
You can then build and run it for any platform.

Quick Start

Default configuration

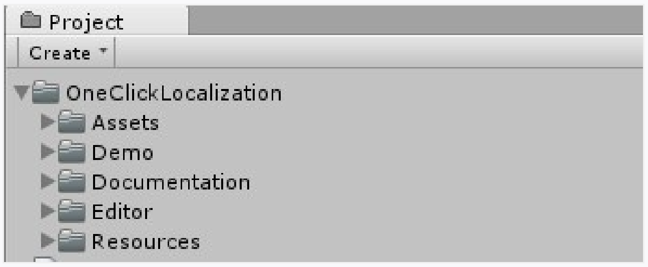
This quick start uses OCL default configuration, which only supports strings. Check the Automatic Setup section for more information on configuration.

1. Download OCL from the Unity Asset Store

OCL in Asset Store

2. Import OCL Package

   All OCL content is located in a single directory **"OneClickLocalization"**



3. Open Setup Window : **Window / OneClickLocalization / Setup**
4. Click on the **Start Automatic setup** button (this is the One Click ;))
5. **Voilà! Youre project is Localization Ready**!
6. Now you can add languages and start localize your texts from the Localization Window
7. Finally, you can run your project to test localization (if your default language equals your system's language, uncheck **Use system's language** and select the right language)

Unity Versions compatibility

Warning Popup on last Unity versions

With lastest Unity versions, during the package import, you'll get the Unity's Warning PopUp "Have you made a backup? API Changes". Click "Yes i made a backup", everything works fine.

To avoid many version content problems, there is only one version deployed to the asset store now, for the latest version of unity.

✓ 2019.4 - **Tested**
✓ 2019.3 - **Tested**
✓ 2018.2 - **Tested**
✓ 2017.1 - **Tested**
✓ 5.6 - **Tested**
✓ 5.5 - **Tested**
✓ 5.4 - **Tested**

Editor - Setup

The Setup Window is used to configure OCL and start automatic setups.

Save / Load

OCL saves all its configuration data and localizations (used at runtime) in a single asset file **OCLSetup.asset** located in Resources folder.
Be sure to add this file to your Source Control, to use Cloud Build and keep setup and localizations.

- **Save Setup**

  Saves the current configuration and localizations to the specified path.
  You can't save outside of the project Assets folder

- **Load Setup**

  Load the specified setup asset and replaces the current one.
  You will lose current configuration and localizations.

Settings

This options can be modified during runtime via the Script API

- **Activate localization**

  If unchecked, OCL is deactivated.

- **Use system's language**

  If checked, OCL automatically detects language by using **Application.systemLanguage**

- **Force language**

  Available if **Use system's language is unchecked**. Select the language used by OCL.

- **Default language for nulls**

  If checked, OCL API will return default language value when translation is null.

  Usefull when you don't want check null values by yourself and avoid blank texts in your UI.

Automatic Setup

- **Add OCL Component**

  If checked, setup process will add a OCLComponentAdapter to GameObjects found with a supported type.

- **Extract Data**

  If checked, setup process will extract data from the supported types to the localization list. You can then edit this localizations with the Localization Window

  - **Configure Includes / Exlude**
    Specify lists of components to include or exclude from setup process.
    By default, only components supported by OCLComponentAdapter are in the include list :
    UnityEngine.UI.Text, UnityEngine.UI.Image, UnityEngine.UI.RawImage, UnityEngine.TextMesh, UnityEngine.AudioSource, UILabel (NGui)

- **Parse Scene objects**

  If checked, setup process will parse objects from scenes

  - Selected scenes
    List of scenes to parse. Only scenes from the build settings **Scenes in build** are available.
  - Parse inactives

    If unchecked, inactive objects will be skipped

- **Parse Prefab assets**

  If checked, setup process will parse prefabs from the **Assets** folder.

  - Assets subpath
    Let you define a subpath of prefabs to parse.
  - Parse inactives
    If unchecked, inactive objects will be skipped

- **Parse Scriptable Objects assets (disabled by default)**

  If checked, setup process will parse ScriptableObjects from the **Assets** folder. It will extract strings from field (going recursively though lists, arrays and objects) and though root object properties (no recursivity here).

  - Assets subpath
    Let you define a subpath of ScriptableObject to parse. You should always specify this option when using this parsing, as it avoids all ScriptableObject from other packages (TextMeshPro, etc.).

Languages

- **Default**

  The language of your content before localization.

- **Languages table**

  The table displays all the selected languages, the ratio of ids localized and the option to add/remove a language. When you remove a language, you lose all the OCL data related to it, use carefully.

- **Edit Localizations**

  Opens the Localization Window

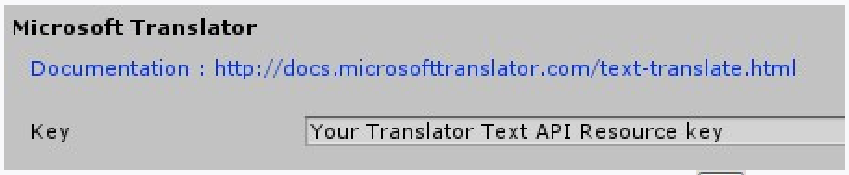
Microsoft Azure Translator

Azure Account needed

You must have an Azure portal account to use translator. There is a 12 months free evaluation when you create one with 2 millions character free/month

See this documentation to generate a key : Text Translator API Doc

Add your Resource key from Azure portal in the **Key** input (key is something like "adf86b4e3d1a488f84db076092e19f16", you can grab it from the "Manage keys" section in Azure portal)


Microsoft Translator

Documentation : http://docs.microsofttranslator.com/text-translate.html

Key Your Translator Text API Resource key

Translator can be used though the Localization Window :

You can also use the "Translate all" button, it will translate all the entries visible in the table (based on the filters)

WARNING : be sure to use "global" configuration when you create a new Azure Translator cognitive service on the portal, or you could get "401 errors"

Reset

No undo possible!
There is no Undo possible after a reset, you'll lose all OCL related data, be sure to have backups of your data (with export or source control of the OCLSetup asset) before you use it.

The reset will :

- Search for **ALL** OCLComponentAdapter in the project (from build scenes and prefabs) to remove them.
- Reset Setup Window data
- Reset languages configuration
- Reset all localizations

Editor - Edit Localizations

The Localization Window is an integrated editor for all your localizations : string, Sprite, Texture or AudioClip

A localization can have two states for a language :

- **null** : if GetLocalization is called for this id, it will answer null.
- **not null** : the value will be returned when GetLocalization is called

When using OCL from script API, be sure to check if returned value is null.

For strings : null and empty are differenciated, if you want a string to be null : use the **Reset** button ∅

Edit Localizations Window supports Undo / Redo and line returns.

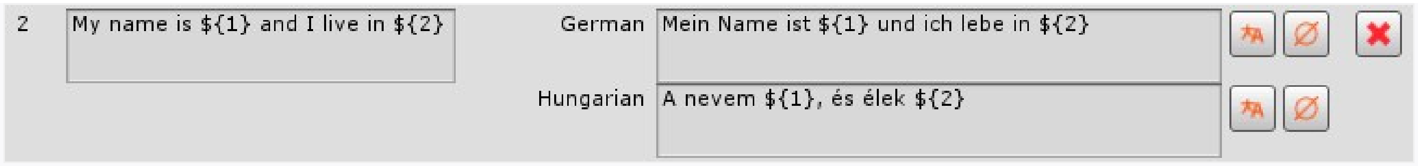
Filters

- **Untranslated only** : If checked, the table only show localization with a null value
- **Search** : Filters the table with the specified string
- **Languages** : Table displays only localizations for the selected languages
- **Apply** : Apply the current filter values to Ids
- **Reset** : Reverts filter values to default

String Parameters

Strings support parameters for dynamic content.
Use bracket numbered ids : **${#}**

Example :


2 My name is ${1} and I live in ${2}
German Mein Name ist ${1} und ich lebe in ${2}
Hungarian A nevem ${1}, és élek ${2}

OCL will then automatically replace parameters using regular expressions.

String Import/Export

You can import and export your strings to **xml** and **csv** formats

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<localization>
  <strings>
    <string stringId="String to localize 1">
      <value lang="French">String à localiser 1</value>
      <value lang="German">I can't speak German 1</value>
    </string>
    <string stringId="Another string to localize">
      <value lang="French">Une autre String à localiser</value>
      <value lang="German">I still can't speak German</value>
    </string>
  </strings>
</localization>
```

CSV

Uses coma separator, to avoid any problem during import, it is recommended to wrap all your strings in double quotes.

```
StringId,French,German
String to localize 1,String à localiser 1,I can't speak German 1
Another string to localize,Une autre String à localiser,I still can't speak German
"String to localize with, special ' characters","String avec caracteres speciaux,'",I can't
speak German... really
```

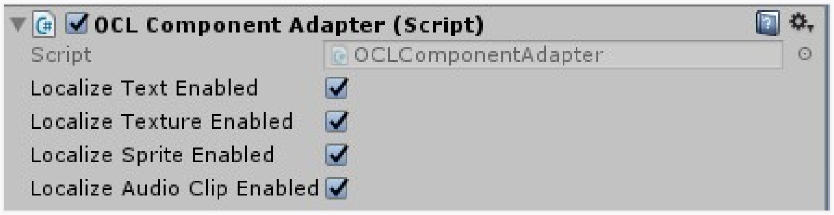
Runtime - Components

Components are based on the OCL Script API, they apply the localization data to the GameObjects.

Only one component currently exist **OCLComponentAdapter** designed for ease of use and fast integration. More will come soon dedicated to performances

OCLComponentAdapter

Automatically localize supported components on the same GameObject, **no configuration needed**.



OCL Component Adapter (Script)

Script	OCLComponentAdapter
Localize Text Enabled	✓
Localize Texture Enabled	✓
Localize Sprite Enabled	✓
Localize Audio Clip Enabled	✓

Localization is currently supported for the following components (with corresponding type)

- **UnityEngine.UI.Text** (string)
- **UILabel** (string) - NGui component, tested on free version 2.7.0
- **UnityEngine.TextMesh** (string)
- **UnityEngine.Image** (Sprite)
- **UnityEngine.RawImage** (Texture)
- **UnityEngine.AudioSource** (AudioClip)

To add it to a GameObject :

- Automatic : using Automatic Setup with the option **Add OCL Component**.
- Manual : you'll find it in the **OneClickLocalization** Category

You enable or disable types for the component. This is not dynamic during runtime for performance reasons.

Performances remarks

- OCLComponentAdapter test on each update if supported components values have changed and cache data to optimize the test as much as possible. String comparison is very fast and therefore has very low impact on performances. Texture, Sprite and AudioClip comparisons can impact performances when heavily used, they should be used wisely
- Impact on performances grows with the number of OCLComponentAdapter running, therefore adding OCLComponentAdapter automatically with Automatic Setup should be used carefully and its impact tested with the profiler.

Runtime - Script API

OCL exposes full access to its configuration and data though a complete C# API

All the API is accessed from the static class **OneClickLocalization.OCL**

OCL only uses **UnityEngine.SystemLanguage** to determine language, no locale (en_En, fr_FR, etc...)

Delegates

Delegate	Description
`delegate void ActiveChanged(bool isActive) ActiveChanged onActiveChanged`	Called when Active state changes

`delegate void LanguageChanged(SystemLanguage oldLang, SystemLanguage newLang) LanguageChanged onLanguageChanged`	Called when selected language changes
`delegate void LanguagesChanged() LanguagesChanged onLanguagesChanged`	Called when a language is added or removed from the the languages list
`delegate void LocalizationChanged(object id, SystemLanguage language, object newValue) LocalizationChanged onLocalizationChanged`	Called when a localization is modified

Methods

Method	Description
`bool IsActive()`	Get OCL active state
`void SetActive(bool value)`	Set OCL active state
`void SetLanguage(SystemLanguage language)`	Set the language used by OCL, has not effect if IsLanguageAuto is true
`SystemLanguage GetLanguage()`	Returns language used by OCL. If IsLanguageAuto is true : returns Applicati If IsLanguageAuto is false : returns language defined with SetLanguage. Default value is SystemLanguage.English
`bool IsLanguageAuto()`	If true, OCL uses Application.systemLanguage for localization. If false, OCL uses GetCustomLanguage for localization

`void setLanguageAuto(bool isAuto)`	Defines if OCL should use Application.systemLanguage or GetLanguage for
`void AddLanguage(SystemLanguage language)`	Add a new language. Has no effect if language is already in GetLanguages
`void RemoveLanguage(SystemLanguage language)`	Removes a language. Has no effect if language is not in GetLanguages.
`List GetLanguages(bool addDefaultLanguage = true)`	Returns supported languages. Use AddLanguage to add a new one and RemoveLanguage to remove one. If addDefaultLanguage is true, defaultLanguage will be in the list even if it ha
`object GetLocalization(object itemId)`	**Main method of the API** Generic version of GetLocalization Returns the translation of the given object if its type is supported, its id is pr

`string GetLocalization(string originalString)`
`Sprite GetLocalization(Sprite originalSprite)`
`Texture GetLocalization(Texture originalTexture)`
`AudioClip GetLocalization(AudioClip originalAudioClip)`
	Typed versions of GetLocalization

`string GetLocalization(string originalString, SystemLanguage language)`	Returns the translation of the given string for the given language. This method shouldn't be called directly as it won't handle active, defaultLar Call it only if you need to access localization data directly without taking car
	Typed versions of GetLocalization(Sprite, lang)

```
Sprite GetLocalization(Sprite
originalSprite,
SystemLanguage language)
Texture
GetLocalization(Texture
originalTexture,
SystemLanguage language)
AudioClip
GetLocalization(AudioClip
originalAudioClip,
SystemLanguage language)
```

```
void SetLocalization(object
id, SystemLanguage language,
object translation)
```

Generic version of SetLocalization
Set the translation for the given id and language.
Has no effect if language is not in GetLanguages. Use AddLanguage to add

```
void SetLocalization(string
id, SystemLanguage language,
string translation)
void SetLocalization(Sprite
id, SystemLanguage language,
Sprite translation)
void SetLocalization(Texture
id, SystemLanguage language,
Texture translation)
void
SetLocalization(AudioClip id,
SystemLanguage language,
AudioClip translation)
```

Typed versions of SetLocalization

Examples

Get a localization for current language

```
string localization = OCL.GetLocalization("my text to localize");
```

Get a localization for a specific language

```
string localization = OCL.GetLocalization("my text to localize", SystemLanguage.German);
```

List languages in a Dropdown

```
// Languages list init
List<string> languagesStrings = new List<string>();
// Add supported languages
foreach (SystemLanguage supportedLanguage in OCL.GetLanguages())
{
    languagesStrings.Add(supportedLanguage.ToString());
}
languageDropdown.AddOptions(languagesStrings);
```

Change language from a Dropdown selection

```
string selectedLanguage = languageDropdown.options[languageDropdown.value].text;
OCL.SetLanguage((SystemLanguage) Enum.Parse(typeof(SystemLanguage), selectedLanguage));
```

Add new language

```
SystemLanguage language = SystemLanguage.German;
if (OCL.GetLanguages(false).Contains(language))
{
    Debug.Log("Selected language already there.");
}
else
{
    OCL.AddLanguage(language);
}
```

Set new localization value

```
if (OCL.GetLanguages().Contains(language))
{
    OCL.SetLocalization("localized text", language, "My new value for this localized text");
    Debug.Log("Localization updated");
}
else
{
    Debug.Log("Selected language is not supported, use AddLanguage()");
}
```

For complete examples, see the Demo App code shipped with OCL

Contact

Found a bug? Any question? Contact us 7/7, day and night! (but don't expect immediate answer at night... or on sunday... and saturday... And maybe not too early in the morning... at least not before first coffee)