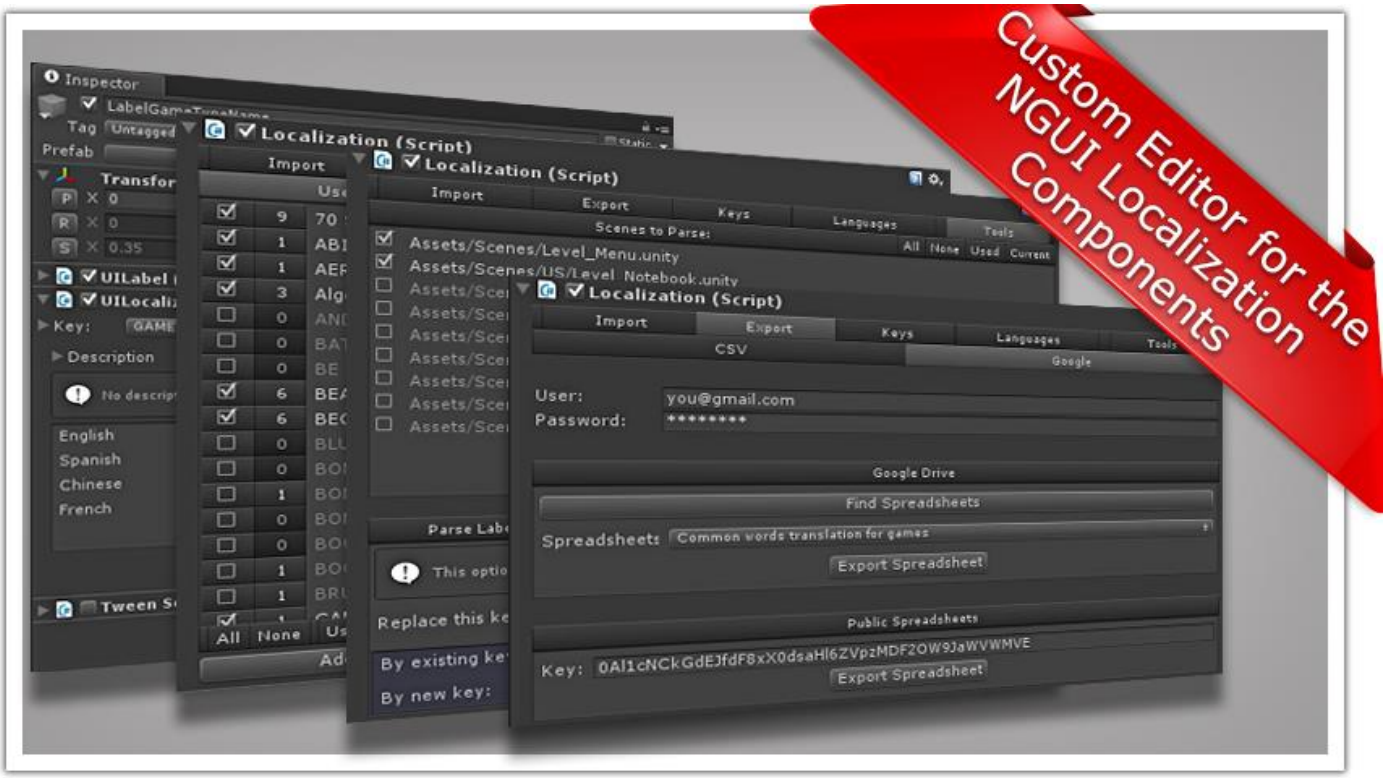
I2Localization

Documentation Draft
(There are sections I'm still writing or are just temporal, but should help you get started)



Contents

Features

Localize

Compatible with NGUI, DF-GUI, uGUI and the Unity Standard Components.
Almost everything can be localized: Text, Images, Sounds, Fonts, Prefabs, Animations, Atlases, among others

Parsing Scenes

By using the batching tools, all scenes can be automatically parsed to find how many times a term is used, what objects are missing the localization data and lot of other useful information to find and fix localization errors.

Edit and Preview

Right from inside the editor all Terms can be listed and modified. Changing the translations and adding new languages. There are lot of visual tips for highlighting localization errors and tooltips to explain each feature.

Sub-Object Translations

The localize component look deep into the targets and allows translating not just the texts, but everything in them. That way Labels can change Text and Font according to the language, while Sprites could modify their Atlas and SpriteName.

	A	B	C	D
1	Terms	Description	English	French
2	Desc	Title Text	English localization	La localisation française
3	English	Language: English	English	French
4	Flag	Flag Sprite	Flag-US	Flag-FR
5	French	Language: Spanish	Français	Français
6	Info	Tooltip	Localization example	Par exemple la localisation
7	Language	Language Name	English	Français
8	Music	Settings Button	Music	Musique
9	Sound	Settings Button	Sound	Son

Spreadsheets

While keeping the internal localization data in optimized custom files, spreadsheets can be linked as external sources for easy editing and sharing of translations. Google Spreadsheets, CSV, Excel and Open Office files can be synchronized with just one click.

Translation Modifiers

Concatenation for adding names and values to the translation could break the output as the word order changes based on the language. This plugins provides configurable callbacks for correct concatenation and replacements.

Multiple Sources

Language descriptions can be baked into the App Resources, stored as scene objects or even deployed as Bundles. That allows adding new translations to the game by downloading bundles and reduces memory used by only keeping in memory the translations needed for each scene.

Auto Translation

Any term or language can be automatically translated by clicking a single button or by linking the language source with the Google Spreadsheet's translation features.

Easy Exploring

Countless iteration have target making the editor as easy to use as possible. By using Terms Categories to group the data and multi-filters to combine the visualized groups, exploring and previewing the localization becomes a breeze.

```
string GetPlayerColor( int PlayerIndex )
{
    if (PlayerIndex==0)
        return ScriptLocalization.Get ("Color/Red"); // Find by String
    else
        return ScriptLocalization.Color.Blue; // Find with Compile-Time checking
}
```

Script Parsing

The plugin automatically finds any constant string used in the scripts and converts them into localized strings. Also, terms can be moved into script constants to get compile time checking for the used terms.

Overview

<space>

How it works

<space>

Whats a Term

<space>

What it's included in the plugin

(editor, game scripts, examples)
<space>

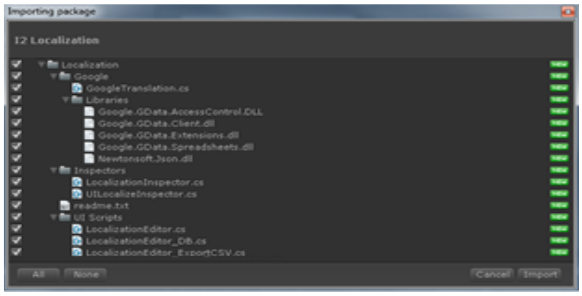
Installation

Buying the I2 Localization plugin

The plugin can be bought directly from the Asset Store

Steps	Action

1. In Unity, select **Window > Asset Store** to open the Unity Asset Store.
2. Search for "*I2 Localization*", and click the **Buy** button.
3. After purchase, import the downloaded Package into your project.



Installation

The full plugin is installed at *Assets / I2 / Localization*

With the following structure:

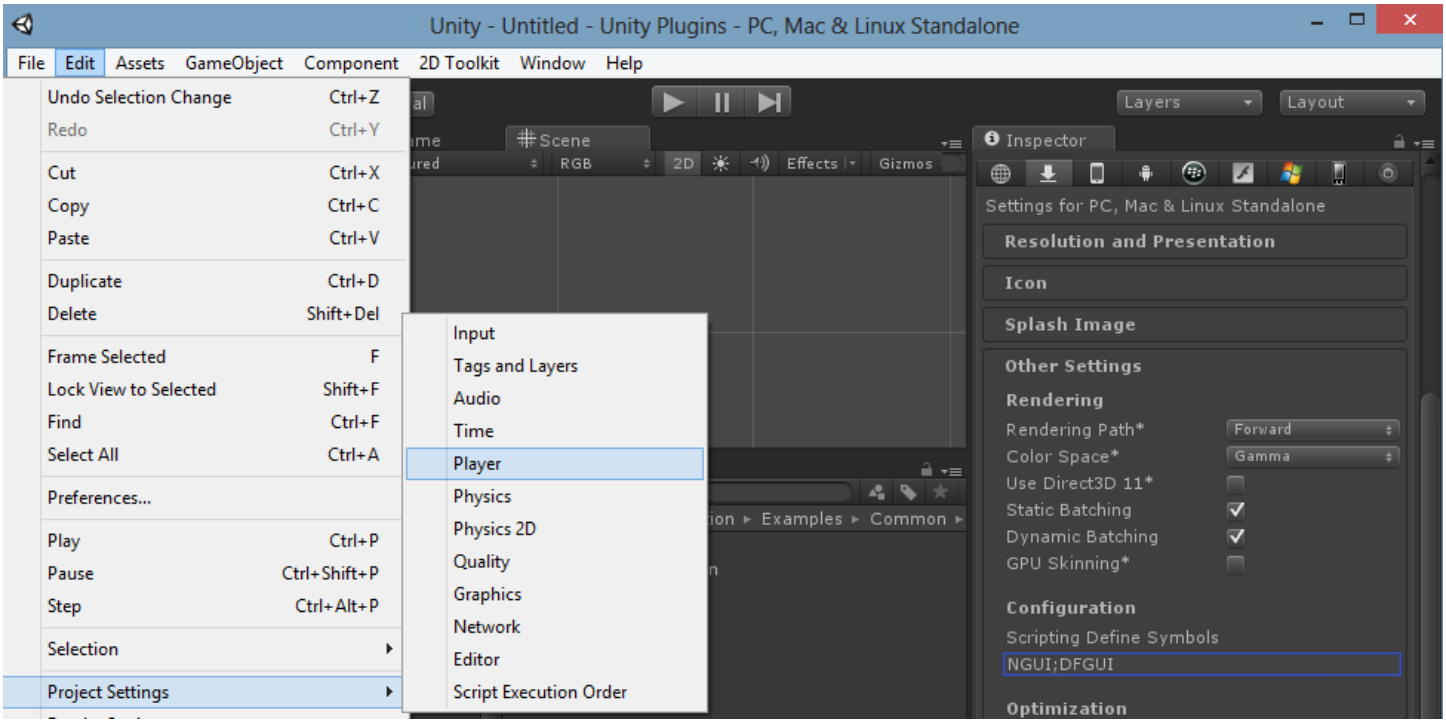
- **Editor**: The scripts to create the Custom Editors and the needed third party DLLs
  - ***DB***: Scripts used to Import/Export and manage the terms data
  - ***Inspector***: Code for the Custom Inspectors
  - ***Localization***: Custom Inspector use this scripts for sharing controls and GUI elements
- **Scripts**: These are the scripts that are executed when in Play mode
  - ***Targets***: Each localization target (NGUI, Unity Standard, DFGUI, uGUI, 2D-ToolKit)
- **Resources**: Needed assets for running the game (Main Language Source and its languages)
- **Examples**
  - ***Common*** : Shared data and Scripts
  - ***DFGUI*** : Example scene and assets for DFGUI
  - ***NGUI*** : Example scenes and assets for NGUI
  - ***2DToolKit*** : Example scene for 2D-ToolKit
  - ***Unity Standard*** : Examples for components included in the default unity (Audio, Animation, Prefab, TextMesh, etc)

Enabling Localization Targets

I2 Localization is capable of localizing **NGUI**, **DFGUI**, **uGUI**, **2D-ToolKit** and some **Standard Unity** components.

However, to avoid compilation issues, the features that require third party plugins are disabled by default. If you project has any of those plugins already installed, then you can safely enable the corresponding feature in the code.

By default, when the plugin gets opened in the editor for the first time it will call the **UpgradeManager.EnablePlugins()** which will detect the third party plugins installed in your project and enable the corresponding I2 Localization targets by adding Script Define Symbols on the player settings.



The allowed symbols are: **NGUI, UGUI, DFGUI, TK2D**

If the plugin fails to detect any of the installed plugins, you can force the detection by using the menu command (***Tools/I2 Localization/Enable Plugins***) or manually adding the correct symbols to the Player Settings.

Please note that if your project doesn't contain the enabled plugins, they will generate compile errors as the I2 Targets depend on classes included in those plugins.

Quick Start

This section walks you through the steps involved in localizing a basic scene. If you haven't already installed the I2 Localization plugin, see the Installation page.

This Quick Start guide is designed to start with a project that hasn't been localized. If you want to port an existing NGUI localization into the I2 Localization system, please refer to the Porting from NGUI topic.

Start by opening or creating a new scene with NGUI, DFGUI, uGUI, 2D ToolKit o Standard components like TextMesh, GUIText, etc. Alternatively for this example, you can open the "UnityStandard Localization" Scene that is included in the Plugin examples folder:

Assets \ *I2* \ *Localization* \ *Examples* \ *Targets* \ *UnityStandard* \ UnityStandard Localization

Creating a Language Source

Language Sources are the base of the localization. They store all the Terms used in the game and the translation of each of those Terms into the selected languages.

Sources could be stored in a variety of places

- They can be created on the **Resource** folder so that they get auto-loaded when the game starts. This Source has to be named I2Languages in order to be recognized by the game.
- Sources can also be included as part of a **GameObject** in the scene to make some Terms available only on those scenes. This could save memory when handling huge localization files on RPGs, MMO or other big games.
- They can be set as part of a **Bundle** and loaded dynamically from there. This allows adding more terms for localizing new contents on games that are expanded by downloading bundles.

On most projects it's recommended to use the Language Source stored in the Resources folder so that the same source became available to all your scenes without having to keep a localization prefab up-to-date in all scenes.

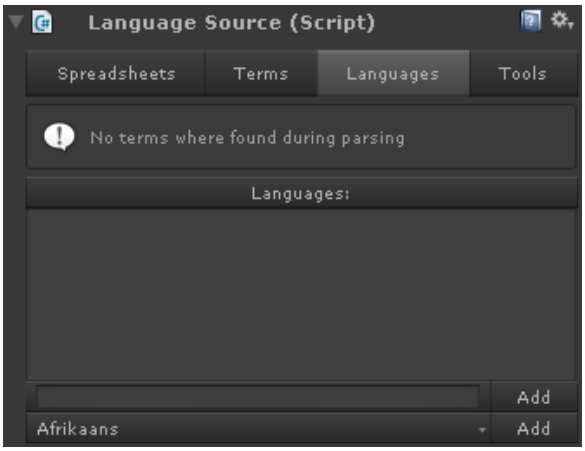
A ready to use Language Source is provided at:

Assets \ I2 \ Localization \ Resources \ I2Languages.prefab

To start localizing the project, click on the **I2Languages.prefab**

or

Create a new GameObject in the scene and add a Language Source component (***I2 \ Localization \ Source***)



Adding Languages

After selecting the Language Source, it time to define a few languages. By default the Language Tab is selected.

Languages are custom TextAssets that store each Term in the source and its translation to that language.

Steps	Action

1. In the Source editor select the Language Tab if it's not already opened
2. Click on the Dropbox at the bottom and the editor will show a list of the default languages
3. Select a language from the list and click the Add Button next to the dropbox.
4. A dialog box will appear asking for a location to create the TextAsset associated with the language. A good practice is to place them into a folder named "Languages" in your project.

   After the location is selected, the TextAsset will be created and associated with the Language Source
5. Alternatively you can type a custom name in the TextField of top of the Languages dropbox and then click the Add next to the TextField.

   That allows creating language variations and Made-Up-Languages.

Adding Terms

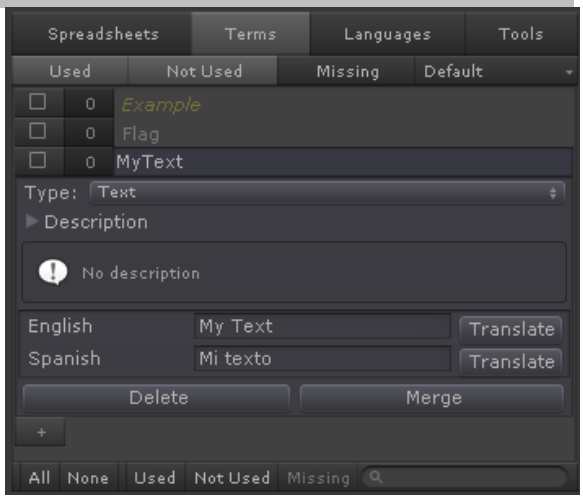
Terms are the keys that will be translated into each language. This are just IDs that when assigned to a label or other localizable component will replace the Component target value by the translation of the term in the current Language.

They don't need to be in any specific language or follow a predefined convention as far as they are unique to the source. Examples of terms are: Flag, ID_1, MissionText1, etc.

A good practice is to let Terms match the translation to the default Language as it will be easier to detect what that terms references. That way "Flag" is more descriptive than "ID_1", even when both could translate to "Flag" in English and "Bandera" in Spanish.

Steps	Action

1. Select the **Terms** Tab
2. Click the "**+**" Button, type a Term like "MyText" and click the Create Key
3. Type "My Text" on the text field next to the English Language if it was created. Or write the appropriate translation to any of the existing languages.
4. If on a platform other than Web, click the Translate button next to the other Languages. That will connect to google and find an automatic translation if available.



Repeat the above steps to add a few Keys and their translations. Once you are done, select any other GameObject and as soon as the Inspector closes the Language Source Custom Editor, all the data will be automatically saved.

Localizing a Label

Once the source is setup with a few languages and the Terms are defined, the I2 Localization plugin is ready to localize components using those Terms.

In this Quick Start Guide we will see how to localize a GUIText. The same procedure will work for other supported components

Steps	Action

1. Select a **GameObject** in the scene containing a **GUIText** Component.

2. Click **Add Component** in the Inspector and select the localize component

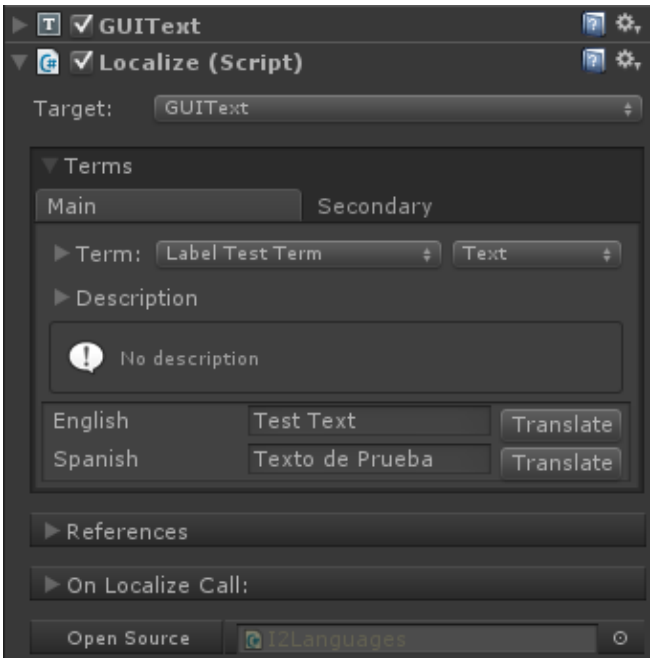
   I2 > Localization > Localize

3. Click on the **Main Term dropbox** and select one of the Terms defined in the Language Source.

   That will show a preview of how the label text will be displayed on each language.

4. Optionally by selecting **the Secondary Term** as well, the Font can be changed depending on the language.

   Secondary Term are target dependent and will localize Fonts when attached to a Label, Atlases when attached to an Sprite, etc



Changing the game Language

Now that the source, the languages, the terms and the localize components are setup, it's time to run the game and see how the label it's localized.

By default, when running the game for the first time, the I2 Localization plugin will try enabling the device language as reported by the unity (Application.systemLanguage). If that language is not included in the Language Source then the first language in the source is used instead.

While playing, a different language can be selected by running the following script. As soon a new language is set ad Current, all objects in the scene containing a Localize component will be updated automatically.

```
If (LocalizationManager.HasLanguage(LanguageName))
{
      LocalizationManager.CurrentLanguage = LanguageName;
}
```

The plugin contains several example scenes showing how to change the language by clicking on buttons or selecting them in a dropdown list.

Language Sources

All the localization data is stored in Language Sources. They store the supported languages, the list of terms, references to localized objects (Fonts, Atlasses, etc).

Sources could be created in a variety of places

- They can be created on the **Resource** folder so that they get auto-loaded when the game starts and be accessible by all Scenes. This Source has to be named I2Localization in order to be recognized by the game.
- Sources can also be included as part of a **GameObject** in the scene to make some Terms available only on those scenes. This could save memory when handling huge localization files on RPGs, MMO or other big games because only the terms used on that scene will be loaded into memory.
- They can also be set as part of a **Bundle** and loaded dynamically from there. This allows adding more terms for localizing new contents on games that are expanded by downloading bundles.

When installing the plugin, a default Source is created in a prefab named **I2Localization** at **Assets > I2 > Localization > Resources**.

For most cases, It's recommended to move that Resource Folder into the project Assets and create all the localization inside the I2Localization source. Given that this source is loaded by default as soon as the game starts, all scenes will be allowed to use the localization data set in that source.

That avoid having to duplicate the same localization data in each scene. Even when making prefabs, they could become out of synch between scenes if the user forgets to apply the changes after doing modifications.

For larger games that localize lot of terms, it could be an option to add another Source to selected scenes and keep in them the terms that are only going to be used in those scenes. When running the game, the Localization Manager will use the terms from both the Resources and Scene sources. So there is no need to duplicate the terms.

When selecting a Language Source it will show a Custom Editor with some tabs that allow switching between the source's features.

Languages Tab

When starting the localization, the first thing to do is to define the languages we want to translate the game into. By clicking the **Languages Tab** a list of all languages and its international code will be displayed.

Languages are stored internally as TextAssets containing a dictionary describing how each Term is translated into that Language. Normally the management of this file is done automatically by the editor and in future versions they are going to be converted into a compressed file with a custom format for quick access.

Default Languages

At the bottom of the list, there are two rows that allow creating new Languages. The one at the bottom is a **DropDown** containing most default languages and variations. By clicking the **Add button** next to the **DropDown** the Language file will be created. But first a dialog box will appear asking for a location to create the file.

A good practice is to place all the language files into a folder named **Resources > I2 Languages**. Although any folder inside the project will work.

When creating a language using the dropdown, the language code will be populated with the international code for that language. But it can be changed afterwards for tweaking regions.

Custom Languages

It could happen that the name of the language does not appear in the Dropdown list or that a Fictional language is needed. A language with a custom name could be created by writing the name in the EditField on top of the dropdown and clicking the Add Button next to it.

Given that the language is not a default one, then the editor will not be able to select a Language Code and will let that field empty. Language codes are used for automatic translation and not needed while in play mode. Therefore it could stay empty if the user will type or import from a spreadsheet all the translation.

The custom language TextField also works as a filter to the dropdown. That way it makes simple to type the first letters of the language and the dropdown will remove all languages that does not contain those letters. For instance, typing “span” will make a shorter list with only the Spanish languages. Regions can also be filtered by adding spaces: “eng c” will select the “English (Canada)” language.

Initial Language

The first time the game runs, the Localization Manager finds the system language used by the device where the game is running and if any of the sources have that language then it is enabled by default. If the language is not found in any source then the first language in the list will be used when starting.

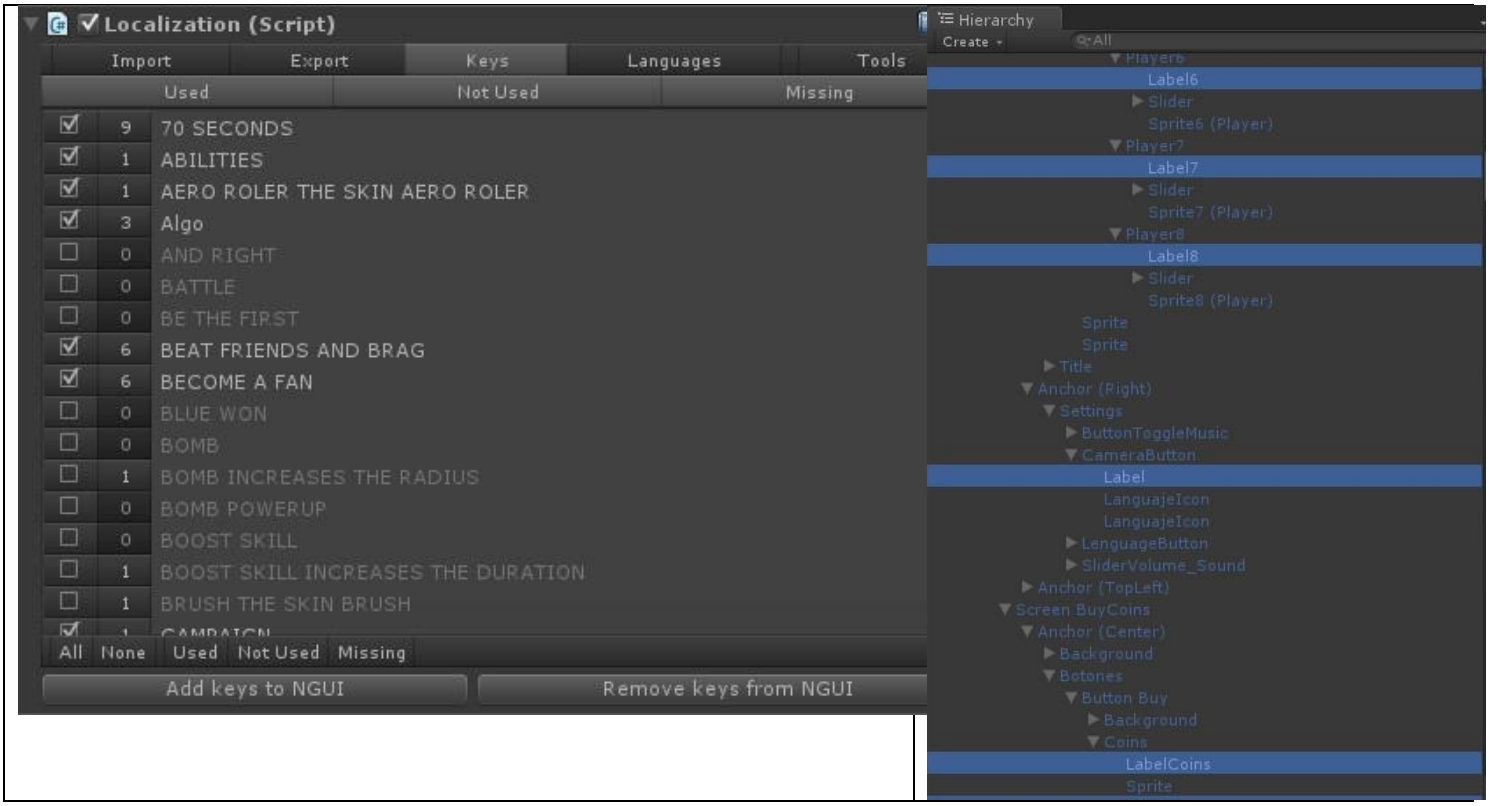
Next to each of the language codes there are buttons to move the languages up or down. That way the intended default language can be moved to the top of the list.

Terms Tab

The Terms tab shows all the terms stored in the languages and how many times they are been used.

When the Sources Custom Editor open in the Inspector View, it does a quick parsing of the current scene and counts the Terms used by all of the Localize components. That number is displayed on a button next to each of the terms.

By clicking on that button, all the Objects in the scene using that Term will be selected.



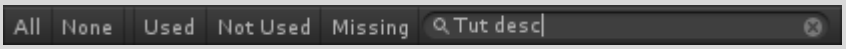
Filters

At the top of the Terms List there are three buttons to filter which keys should be listed.

- **Used**: Adds or Remove to the list the Terms that are actually used in the scenes. They will always show a number higher than 0 because they are used at least once.

- **Not Used**: These are the keys that are in the localization files but are not used in any of the labels or sprites in the scene. It's save to remove this keys as they are not used.
  To remove them, just select the checkbox at the left of each key and click the **Remove keys from Source** button
  As a shortcut, the button **Not Used** at the bottom of the list, will select all of the Keys that are not been used. This avoid having to select them one by one.

- **Missing**: These are the keys that are used in the scene but are not set in the localization files. So the I2 Localization will not found them when localizing a label and so the text of that label will not be translated.
  After selecting the checkbox of each of those missing keys, they can be added to the localization files by clicking the **Add Keys to Source** button.
  If the localization is exported to the spreadsheets after adding the missing keys to the languages. Those keys will appear empty in the spreadsheets to make them easier to spot.



Also in the toolbar bellow the list there is a textField that allows further refining the list by only displaying the Terms that contain the text typed there. It also allows multi-filters by separating words with commas, semicolon or spaces.

That way, by typing “Tut desc” will only show in the list terms containing “tut” or “desc”, so all Tutorial and Description terms will be displayed.

Creating Keys

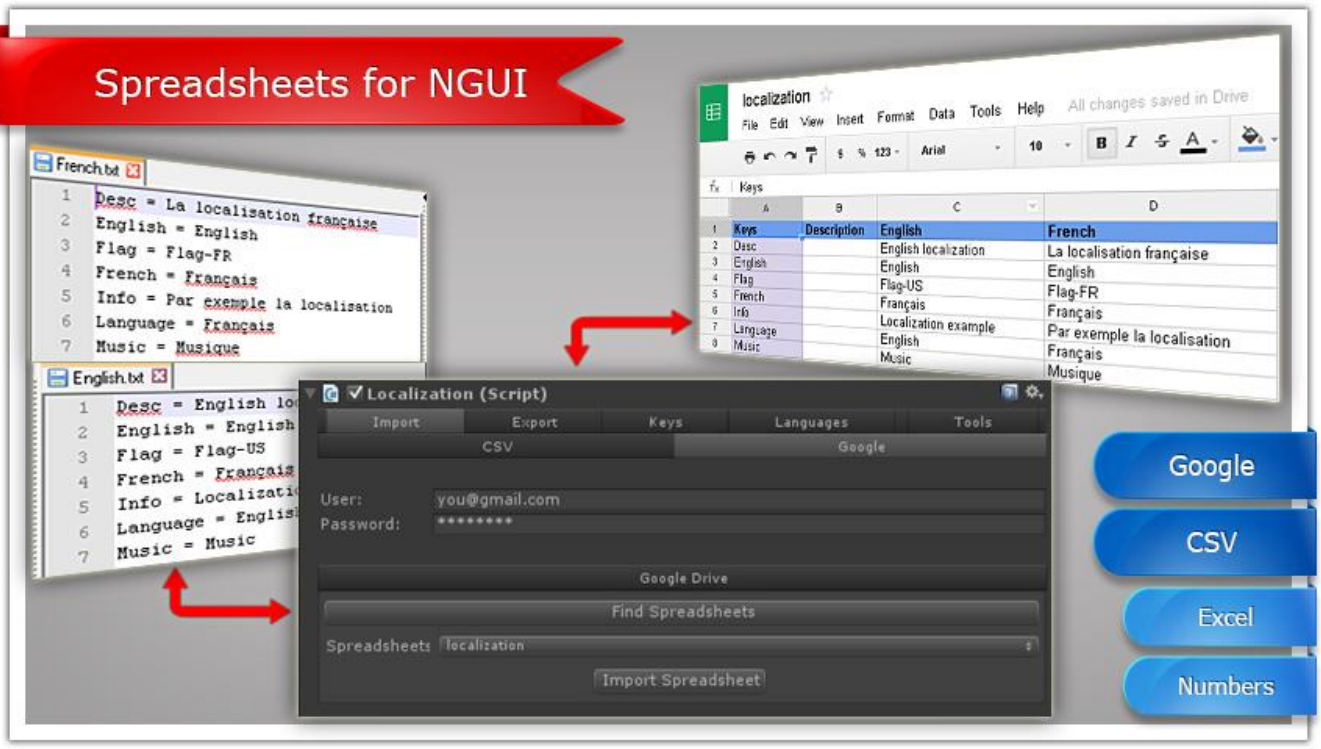
At the top of the Terms List there are three buttons to filter which keys should be listed.

Terms Data
    Type
    Description
    Languages
    Auto Translate
    Categories
Tools
SpreadSheets
    <data can be saved into external sheets for easy sharing/modification>
    Local Files
        CSV
            Example of format
            Can be opened with Excel or other softwares
    Google
        Credentials

Exporting Spreadsheets

While I2 Localization allows editing Terms and Languages inside the Unity Editor, most of the time is more convenient to have a copy of the data as an external spreadsheets. That makes easier to synchronize all keys in the different languages and at the same time, and the spreadsheets could be shared between several teammates to translate texts into all languages.

At this moment Google Spreadsheets are fully supported and the editor can automatically export and import from a public spreadsheet or look up for one a specified doc in the Google Drive.



Linking to a CSV file is also supported. Most spreadsheets programs can load CSV files so it's possible to get the localization data into and out of Microsoft Excel on Windows and Numbers on Mac.

Google Spreadsheets

To export to a spreadsheet in the Google Drive you need to create a Spreadsheet in your Google Drive and then use you Google credential to synchronize your data.

The following steps show how to do it. If you are already have an Spreadsheet that you want to use, then skip the first 3 steps.

Steps	Action

1. Log into Google and open the Google drive:

   https://drive.google.com

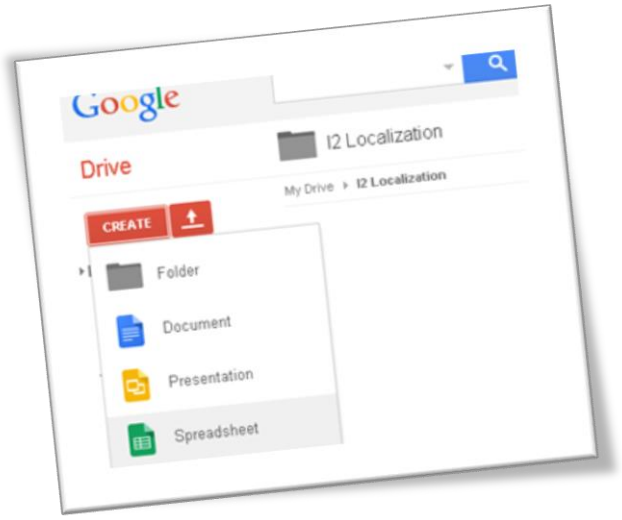
2. Click on the **Create** button and select **Spreadsheet**

3. A new Spreadsheet will be created will the name "*Untitled spreadsheet*".

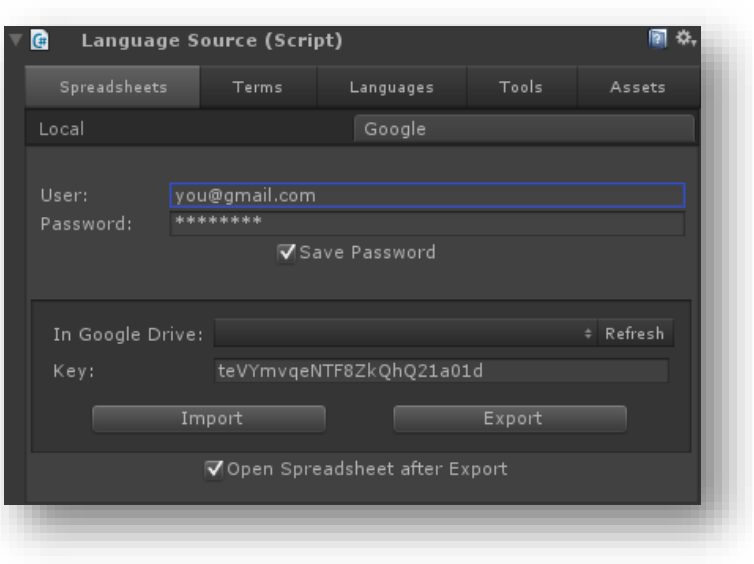
   Click in that name and it will become editable so that you could rename it.

4. Select the **Spreadsheets Tab** and then **Google**

5. Type your Google user and password



6. Then click the Refresh button and the DropDown list will be populated with all the Spreadsheets in the drive.



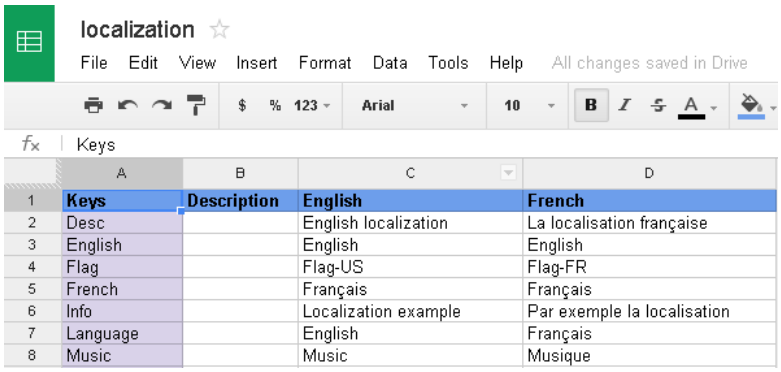
7. Select the target Spreadsheet and the key for that document will be display in the **Key** TextField.

   From that point on that key is what is needed to synchronize so It will be remembered by the language source.

8. Click the Export button and all the terms and languages will be exported to the Spreadsheet and the editor will open the document in the browser after the synchronization finishes.

   Categories will be exported as different sheets in the selected Spreadsheet.

After exporting is completed, if the Open Spreadsheet after Export is enable, then the browser will open the Spreadsheet.



	A	B	C	D
1	**Keys**	**Description**	**English**	**French**
2	Desc		English localization	La localisation française
3	English		English	English
4	Flag		Flag-US	Flag-FR
5	French		Français	Français
6	Info		Localization example	Par exemple la localisation
7	Language		English	Français
8	Music		Music	Musique

Shared Spreadsheets

Often, teams create spreadsheets in one Google account and share the spreadsheets for the rest of the team members as a public or shared spreadsheet.

To share a Spreadsheet just click the Share button at the top right of the corner of the Google viewer and select the correct permissions.

When a spreadsheet is shared a public link is generated. It usually has the following format:

https://docs.google.com/spreadsheet/ccc?key=0Ao0r8tFfhPVbdGVWWW12cWVOVEY4WmtRaFEyMWFwR5c&usp=sharing

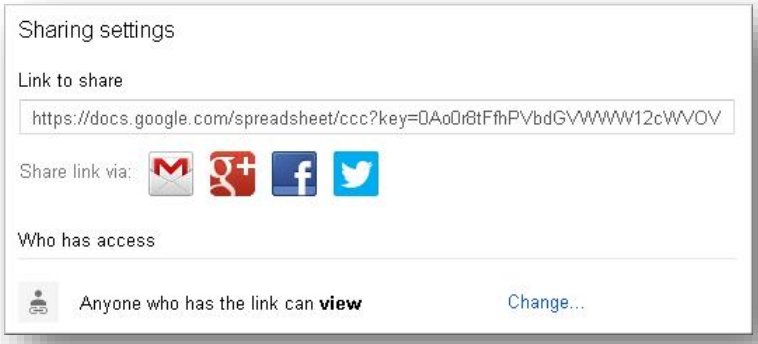
By using that link, I2 Localization can access to the Spreadsheet without accessing the Google Drive. The editor only needs the "*key*" parameter to access that document.

In that example, the key will be the text after **?key=** and before the **&**.

Example:
0Ao0r8tFfhPVbdGVWWW12cWVOVEY4WmtRaFEyMWFwR5c

To do the Synchronization with the Shared Spreadsheet, copy that key into the Key textfield on the Language Source custom editor.

Then click Import or Export.



Exporting CSV

Aside of exporting the Google Spreadsheets, it's possible to export to any other spreadsheet program that supports the CSV format. That makes possible to use Microsoft Excel in windows or Numbers in Mac to handle the localization.

First, select the **Export** tab in the Localization Editor and then **CSV**.

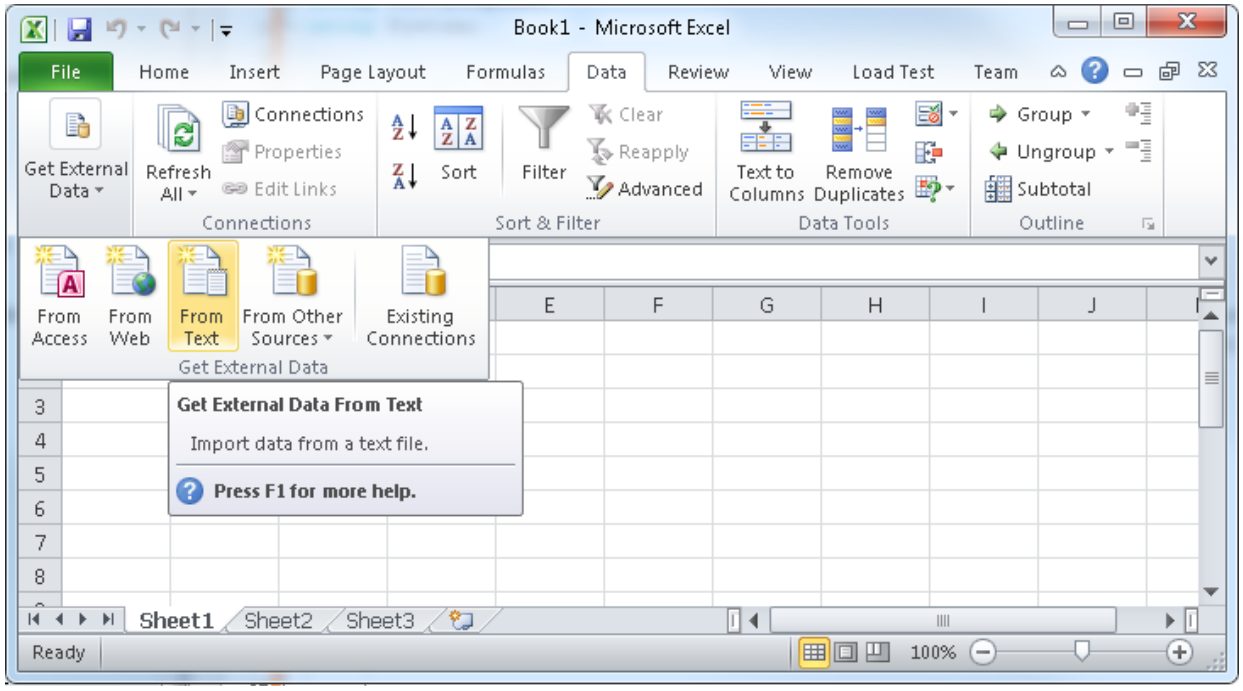
Then type the path of the CSV file you want to generate or click the **...** button which will open a Save CSV dialog box which will allow you to select the path and filename.

Once the target file destination is selected, click the **Export CSV** button. If the selected file is not valid or has not been selected then the Save CSV Dialog box will show up.

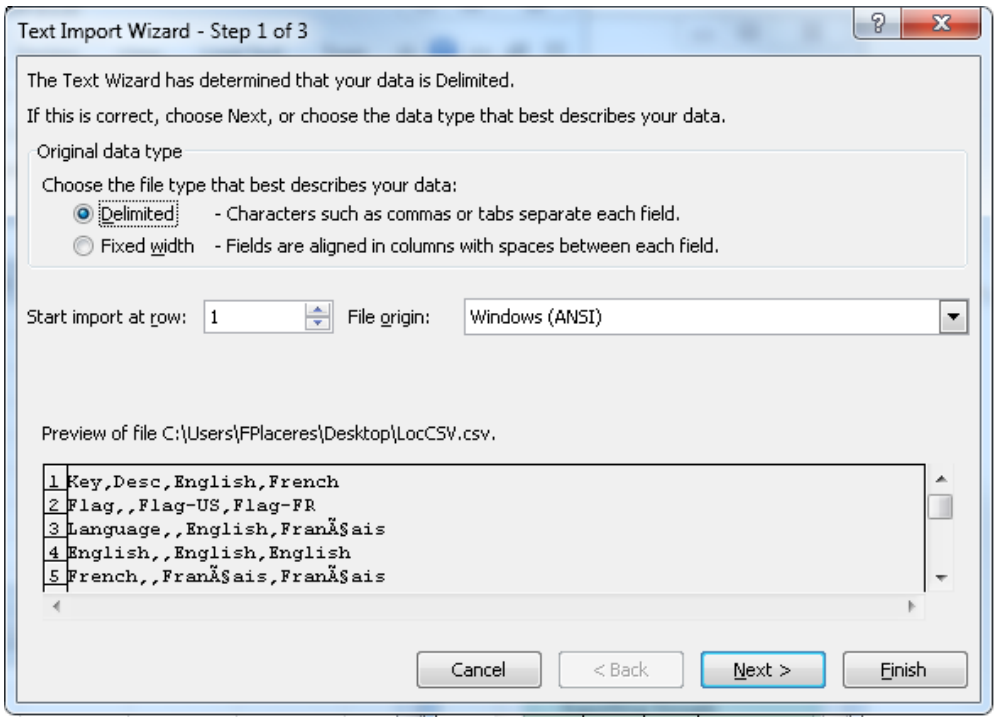
Microsoft Excel

Once the CSV file is generated, it could be used to update a Microsoft Excel spreadsheet.

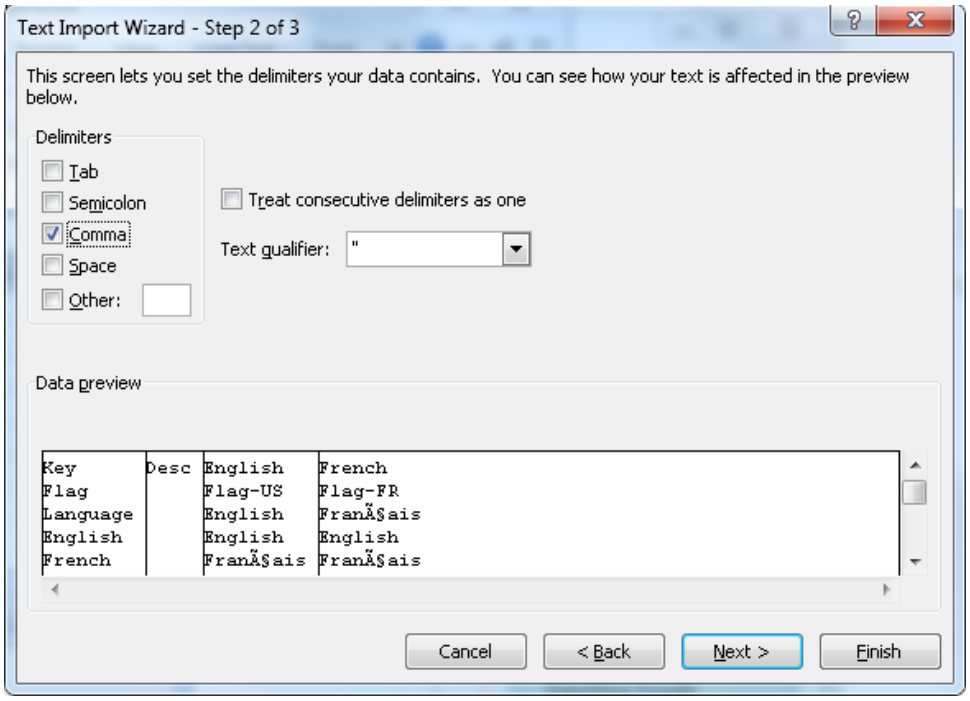
Either create a new excel document or open an existing spreadsheet. Then select the **Data** menu and in there click **Get External Data** and then **From Text**



It will ask you to select the generated CSV file and will show the following window. The CSV file format has all its columns delimited by Commas, so we have to select **Delimited** and click **Next**.



The next screen allows you selecting which character should be used as delimiter. Select **Comma** and the preview will show the text separated in all its columns.

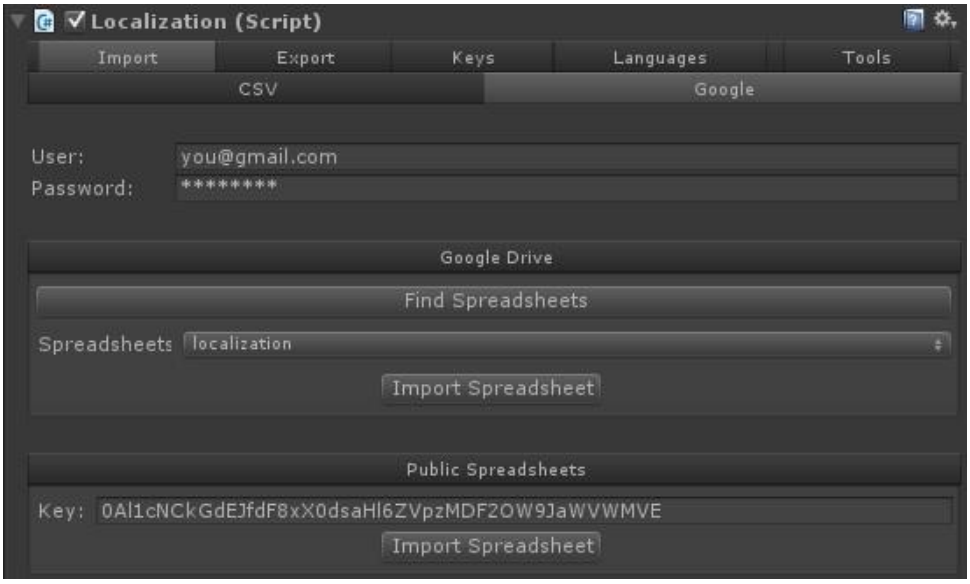


Then click **finish** and the Excel will be updated with the content of the CSV file.

Importing Spreadsheets

To import spreadsheets into NGUI, follows the same steps like when exporting them.

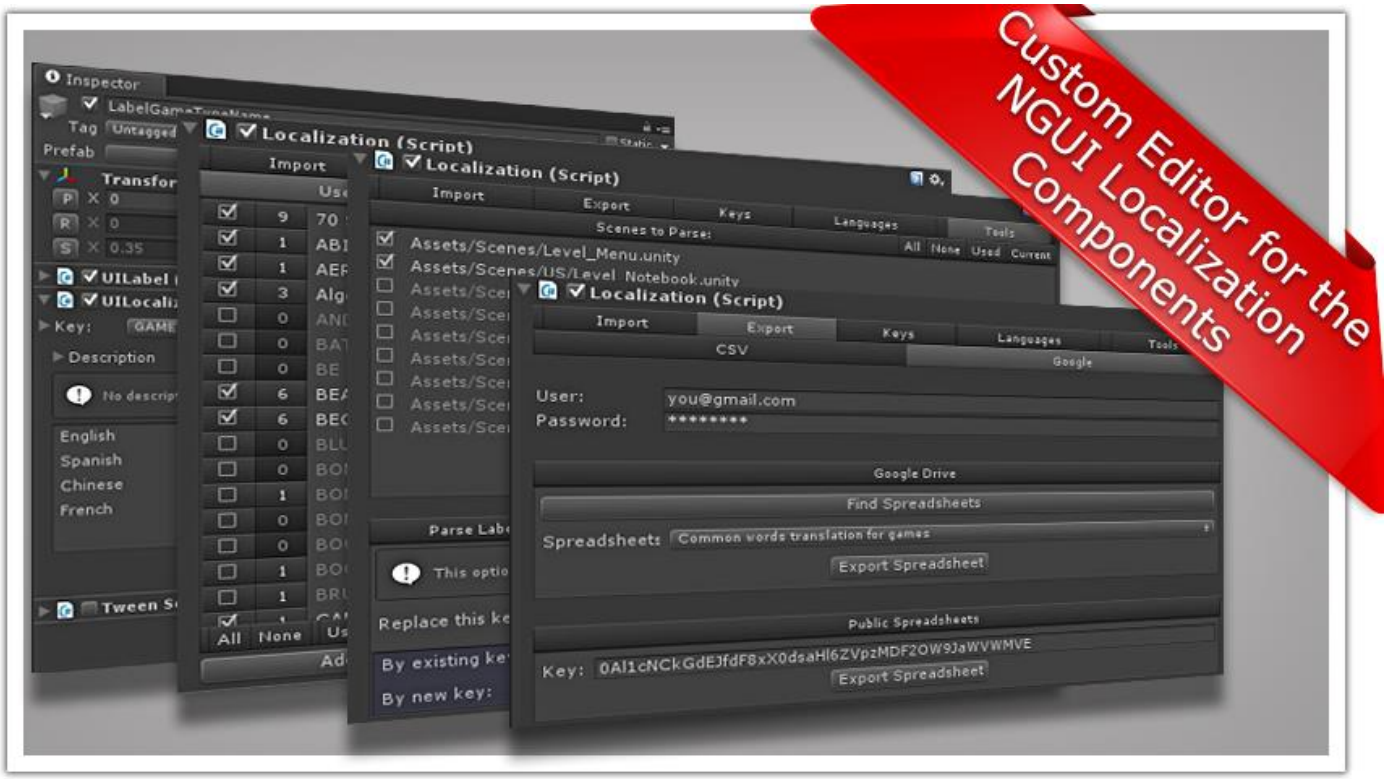
By selecting Import from CSV o from Google Spreadsheets, the editor will read the source and generate the NGUI localization files, linking them to the Localization component.



Check the export procedure for a full description of the buttons and settings involved in accessing the google drive and CSV spreadsheets.

I2Localization for NGUI

Documentation v1.0.0


Custom Editor for the NGUI Localization Components
Inspector
Localization (Script)
Import
Export
Keys
Languages
Tools
Scenes to Parse:
Assets/Scenes/Level_Menu.unity
Assets/Scenes/US/Level Notebook.unity
CSV
Google
User:
you@gmail.com
Password:
Google Drive
Find Spreadsheets
Spreadsheets:
Common words translation for games
Export Spreadsheet
Public Spreadsheets
Key:
0Al1cNCkGdEJfdF8xX0dsaHl6ZVpzMDF2OW9JaWVWMVE
Export Spreadsheet
English
Spanish
Chinese
French
Description
Replace this ke
By existing ke
By new key:
All
None

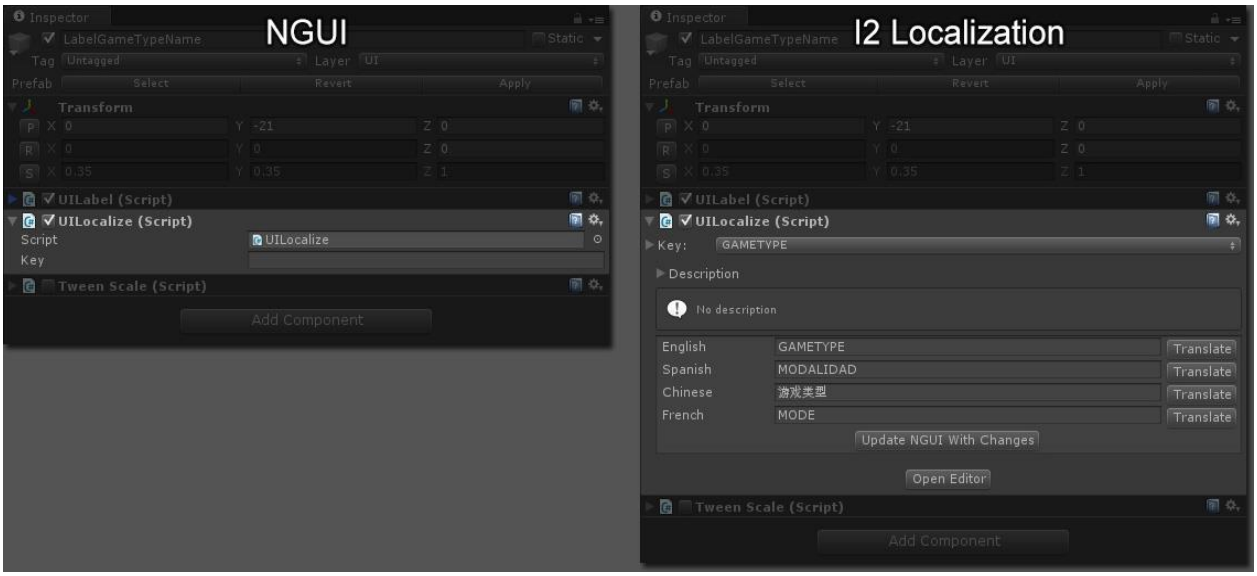
I2Localization for NGUI

NGUI localization works great at the base label and allows changing label's text and sprites based on the selected language. However, managing the keys that control which text is outputted for each language it's not tightly controlled as it involve updating a set of text files that should all match (i.e. contain the same keys).

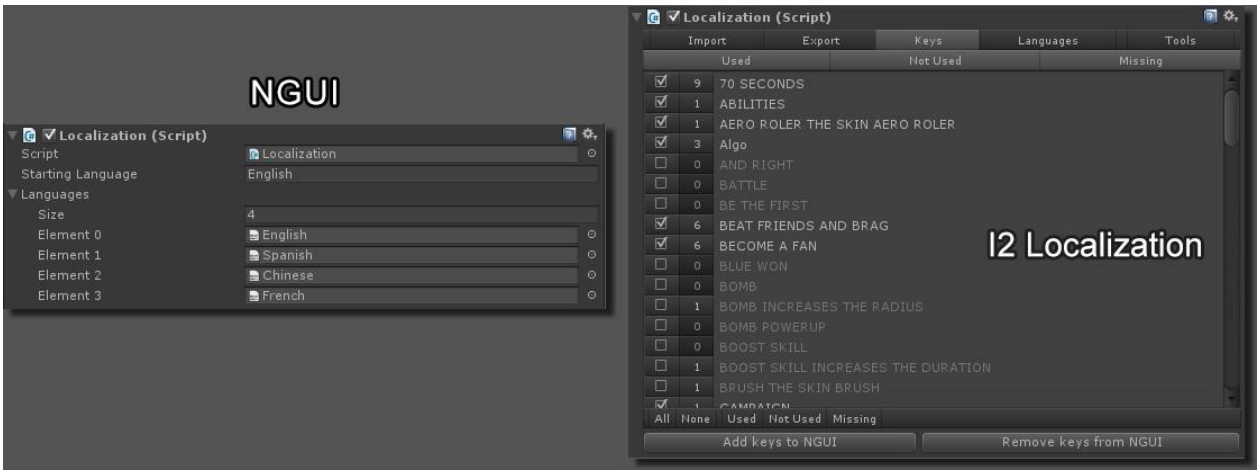
Often, after a while the project ends up having more keys than needed or having missing keys that are used in the labels but not defined in any or some languages.

I2 Localization for NGUI introduces two custom editors for NGUI components. And provide tools within them to parse, edit and fix localization errors.

The first editor provides an interface for the NGUI Localize component. And allows creating or selecting a translation key. It also allows previewing what text the label will show in the different languages.



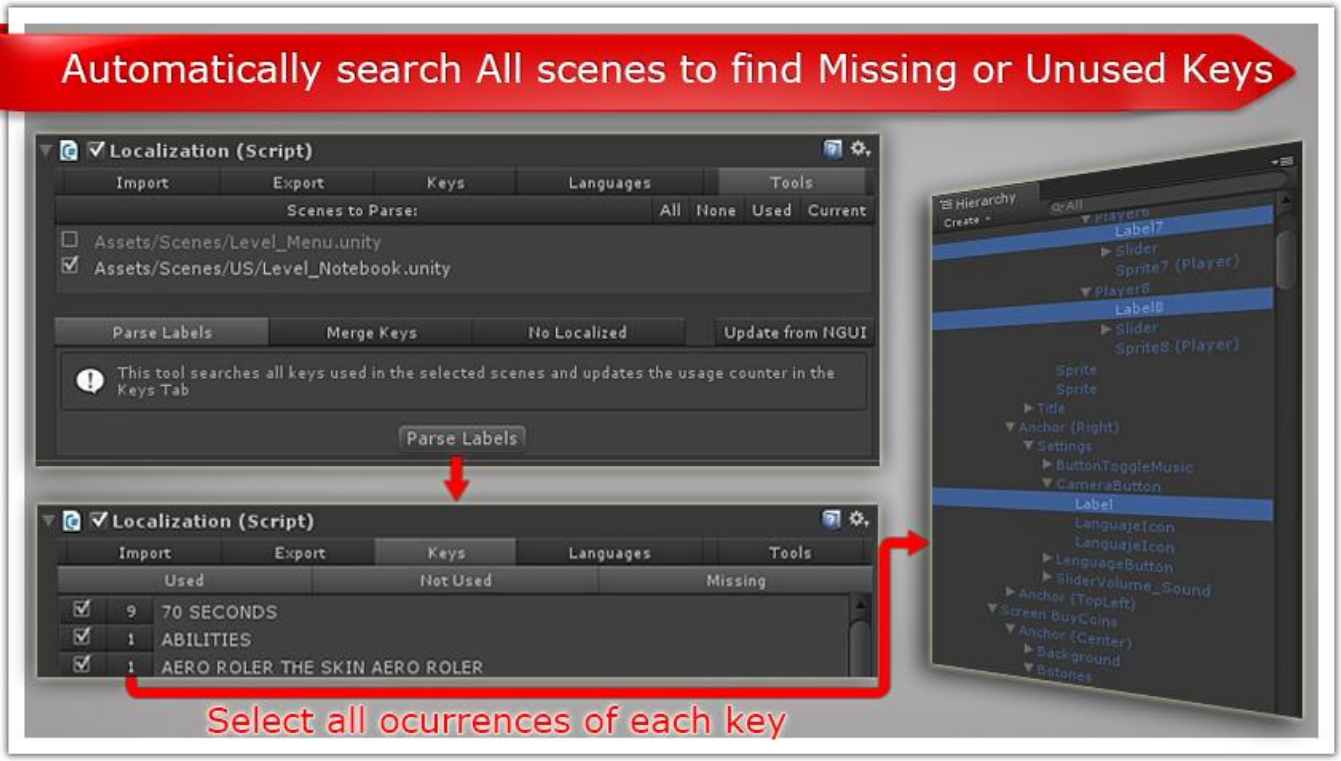
The second editor is for the Localization component. It handles the management of the NGUI localization files and presents a list of all keys, languages and tools to parse the scenes.



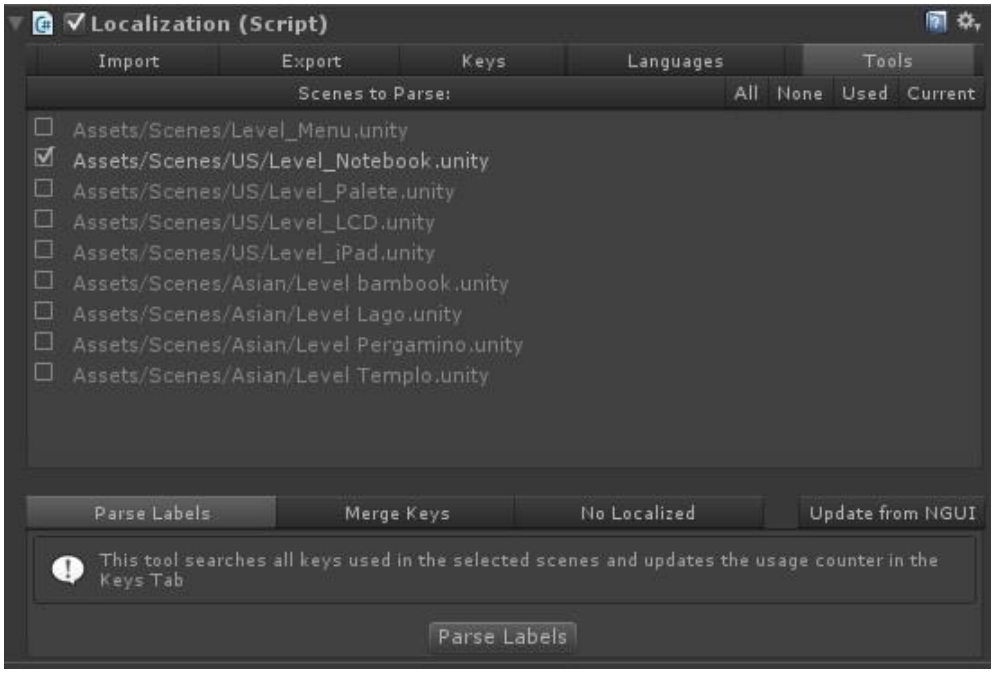
Parsing the scenes for Missing or Unused Keys

One of the major issues developers found when setting localization in NGUI is keeping track of which of the keys in the NGUI localization files are no longer used in any scene. As well as finding which keys are used in the scenes but hasn't been set in the localization Files.

To solve this problem, I2 introduces a tool that Parses the scenes and finds how many times those keys are been used or if they are missing from any of the NGUI files.



To execute the tool, Select the Localization object and go to the **Tools** tab.



The top section will show all scenes added to the project Build Settings.

And there are shortcuts to select them:

- **All** the scenes
- **None** of them, to clear the selection and make easier to hand pick a subset
- The **Used** scenes, which are the scenes enabled on the Build Settings.
- The **Current** scene. It is the one currently open in the editor.

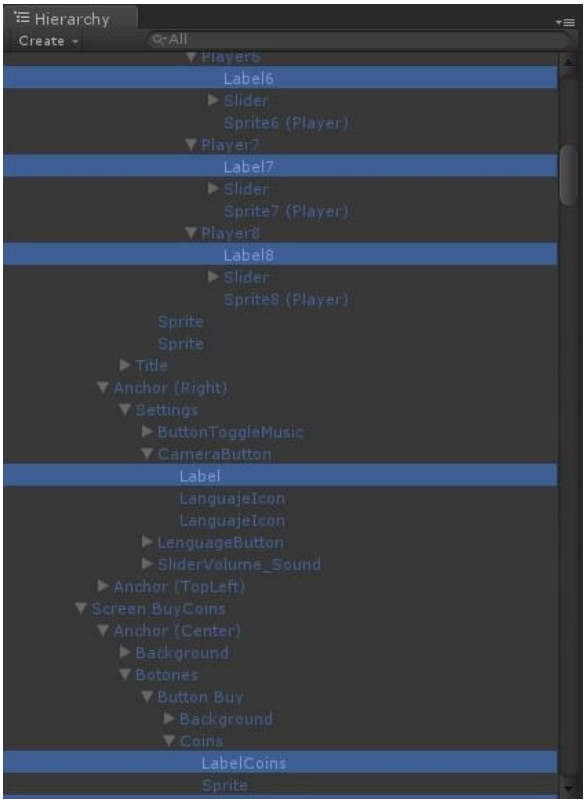
After selecting which scenes are going to be searched, select the **Parse Labels** tab in the bottom panel.

When clicking on the **Parse Labels** button, the editor will open each of the selected scenes, and search thru all of the UILocalize components to track Keys usage.

After the search is completed, the editor will return to the initial screen and open the **Keys** tab to show all the keys found during the parsing and how many times they are used if any.

In the Keys list, each of the keys will show a button with the number of times that key was used in the scenes.

By clicking that button, the editor will select all labels in the current scene that are using that Key.



At the top of the Keys List there are three buttons to filter which keys should be listed.

- **Used**: Adds or Remove to the list the keys on the NGUI localization files that are actually used in the scenes. They will always show a number higher than 0 because they are used at least once.

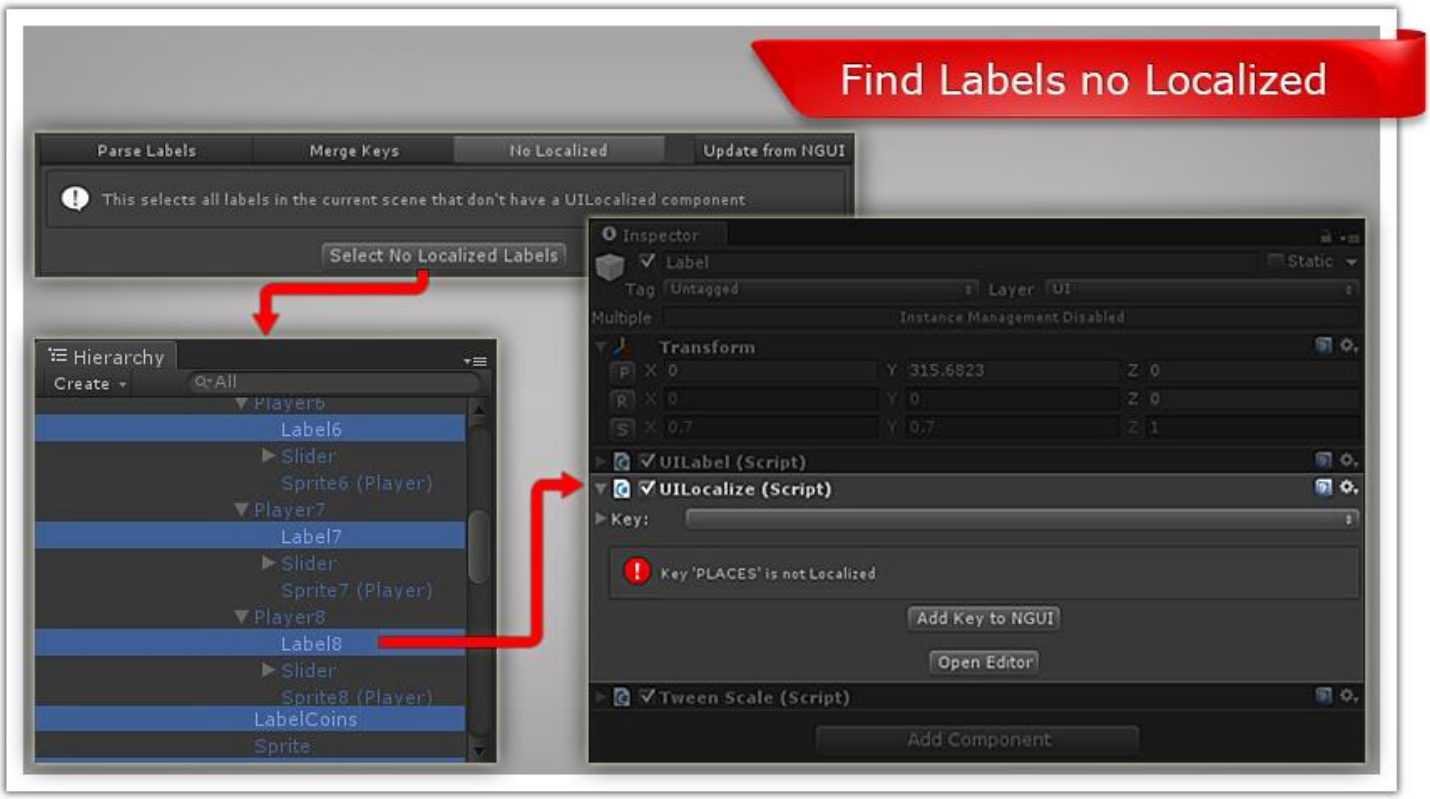
- **Not Used**: These are the keys that are in the NGUI localization files but are not used in any of the labels or sprites in the scene. It's save to remove this keys as they are not used.
  To remove them, just select the checkbox at the left of each key and click the **Remove keys from NGUI** button
  As a shortcut, the button **Not Used** at the bottom of the list, will select all of the Keys that are not been used. This avoid having to select them one by one.

- **Missing**: These are the keys that are used in the scene but are not set in the NGUI localization files. So NGUI will not found them when localizing a label and so the text of that label will not be translated.
  After selecting the checkbox of each of those missing keys, they can be added to the NGUI localization files by clicking the **Add Keys to NGUI** button.
  If NGUI is exported to the spreadsheets after adding the missing keys to NGUI. Those keys will appear empty in the spreadsheets to make them easier to spot.

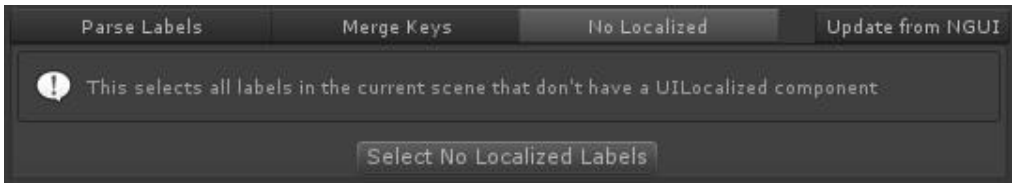
Finding Labels without localization

After the art team changed or added sections to the menus, the new labels have to be localized.

Most of the time, finding which labels aren't yet localized is a time consuming task. And often it gets down to play the game, switching between languages and checking which text don't change. That's not a reliable approach that could let to missing some un-localized texts.



One of the tools implemented in the Localization Editor, searches thru the scene and selects all the labels that does not have a UILocalize component assigned.



To run that tool, select the **Localization Editor** and in the **Tools** tab, open the **No Localized** section.

By clicking the **Select No Localized Labels** button, the editor starts searching thru all the scene and then selects all the labels not localized and expands the objects in the Inspector view.

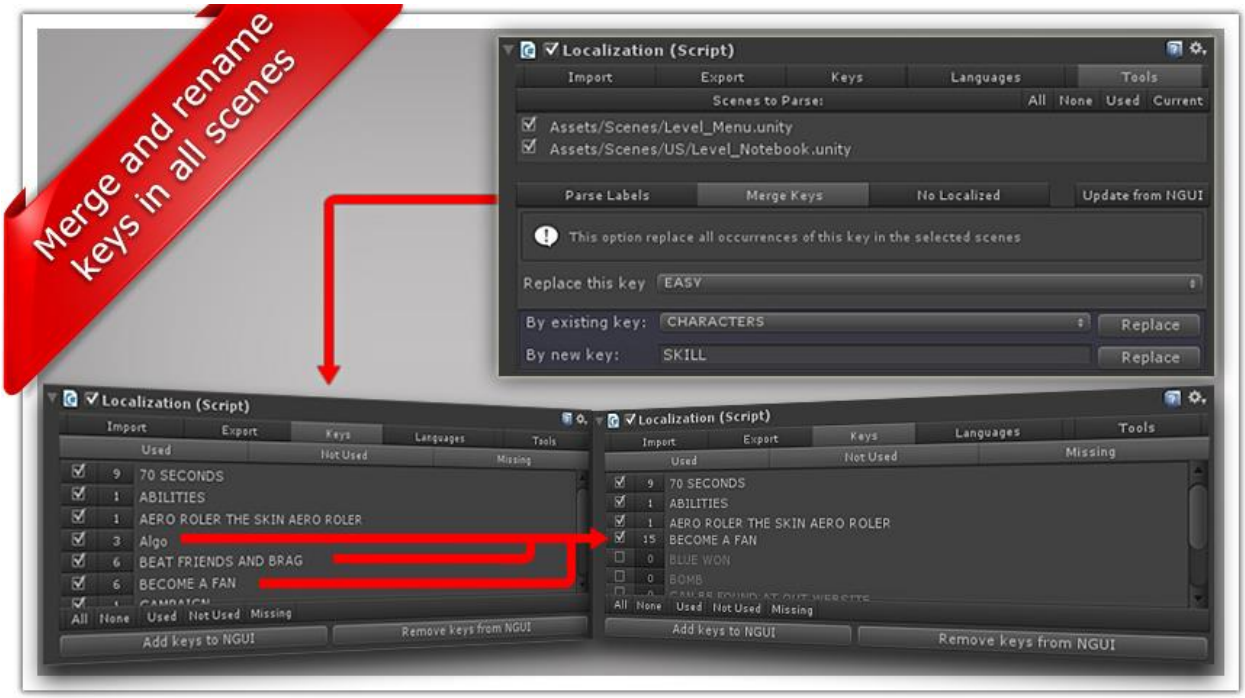

Hierarchy
Create
All
Background
Desc
Reward
label Reward
Sprite
TimeLeft
label TimeLeft
InviteFriends
Button Invite
Sprite2
Title
score
Line
Sprite
TopSplash
Description

Combining Keys

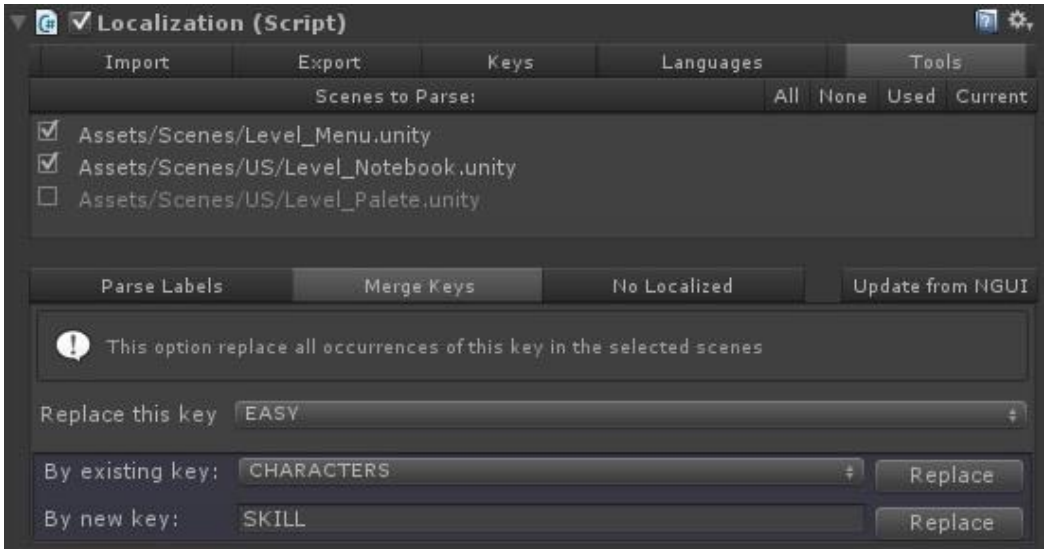
Sometimes keys need to be renamed to have a better naming convention or to make them have more sense.

The problem is that often a Key is used at several locations on all the scenes. Therefore, renaming that key means tracking all those locations and changing the key in there as well. Otherwise all those labels will not find a translation.

A similar issue happens when we need to replace a key by another one in all scenes. Like when you have an END_SCREEN and STOP_DIALOG key which both translate to the same text "Cancel". So both keys could be combined into a new key CANCEL.



On the **Tools** tab in the **localization Editor** there is an option to search all the scenes and do the merging or renaming.



Select the Scenes where the key is going to be replace. Most of the time, all scenes should be selected unless it's known that the key is used only in a few scenes.

After selecting the scenes, open the **Merge Key**s tab and select the Key that has to be replaced.
And either select an existing key from the dropdown or write the name of a new key.

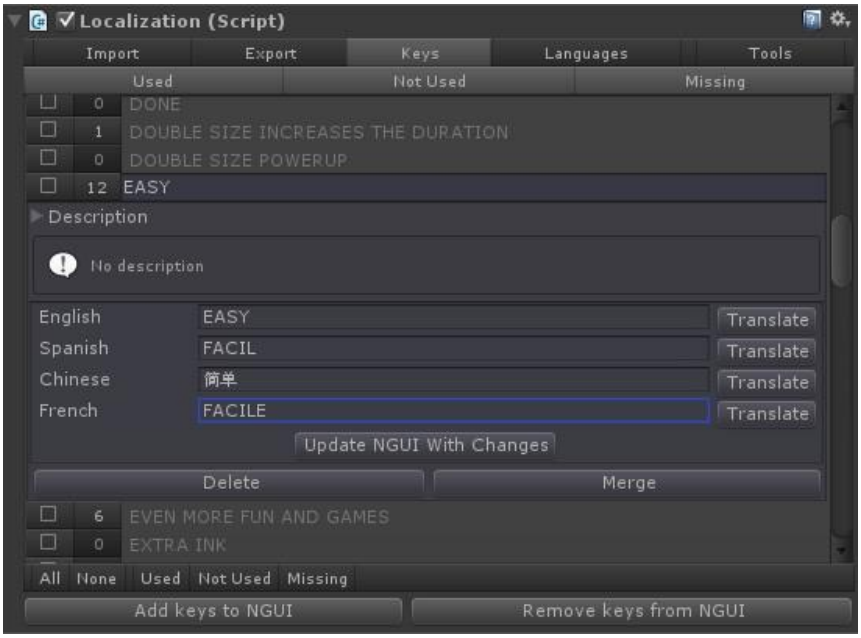
When the **Replace** button is clicked, the editor will open each of the screen, and replace the selected key by the new or the selected one.

Creating and Editing Keys

One of the features of the I2 Localization editor is to easily allow browsing, editing and creating Keys.
When selecting the NGUI Localization component, a list of all Keys is displayed on the **Keys** tabs.

That list is initially populated with all existing keys in the NGUI localization files. But if the **Parse Labels** tools is executed, then also **Used** Keys that are not set in the localization files are shown.

By clicking on one of the Keys in the list, the key is expanded and will show a description and its translation to each of the supported languages.

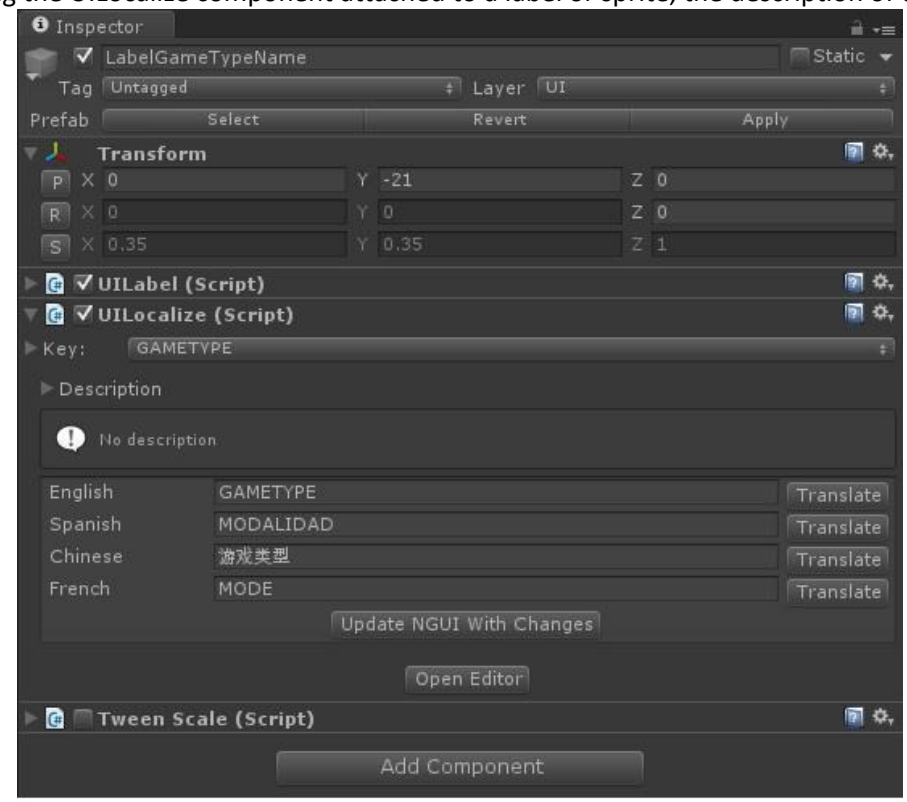


The translation can be edited by typing on the corresponding text field. Or by clicking the Translate button which will consult the google translation web to translate from any of other already set languages into the one that is been edited.

To avoid saving too often the NGUI localization files which could cause lags in the editor responsiveness, the translation is saved in NGUI only after clicking the button **Update NGUI with Changes**.

So, it's possible to do edit the values of several keys and when everything gets ready, click the **Update NGUI With Changes** and that will generate the TextAsset files used by NGUI.

Similarly, when clicking the UILocalize component attached to a label or sprite, the description of that label will show up.



By selecting the Key in the popup, the label or sprite will use a different entry from the Keys list. That will actually modify the Key value inside the UILocalize NGUI component.

If you want to create a new key instead of selecting an existing one. Click the arrow to the left of the Key and that will unfold a text field that allows typing the name of the new Key.