Babel Translation Toolkit

Ward Dehairs
`http://warddev.com`

1 Basic use guide

The babel translation toolkit is designed to automatically extract text from your scene for easy translation. If your project does not have any complicated interactions with text, you only need to go trough the following steps to set up your translations.

- import babel into your project
- Go to Window > Babel Translation Wizard
- Create a new key, with the key language being the language in which you implemented your game
- Make sure that all the scenes you wish to translate are added to your build settings, press ”Collect text from all scenes” and continue.
- Add the languages you want to translate to
- Add translations for your text
- Save the key
- Create a GUI element for the end user to select different languages (You can turn on the default GUI in stead for testing purposes).

The GUI element you create should connect back to a custom script, which may for example contain the following code

```
public void Translate(string language) {
    BabelTranslator.TranslateScene(language);
}
```

An alternative example:

```
public void Translate(int language) {
    string languageString = BabelTranslator.GetAllLanguages()[language];
    BabelTranslator.QuickRefresh();
    BabelTranslator.TranslateScene(languageString);
}
```

Congratulations! You now have working translations for your game!

1.1 Details of the text search

The text search works by going over all scenes added to the Build Settings of your project. Each scene is loaded, after which most of the components in the scene are recursively searched for string variables. Each unique string that is encountered is added to a list for you to translate. Note that this search will only yield strings that are in some level in your scene. It will not be able to automatically find strings buried in your assets nor in your scripts. To add these strings to the translation key, you should either find a way to temporarily add them to the scene, or simply add them to key manually.

2 Advanced use guide

2.1 Translation system details

The Babel Translation Wizard, is called such because it is simply a tool to help you do 2 things. Firstly, to make a translation key, secondly, to set up your scenes with Babel Translator (with the right settings) that do the string searching and translation at run time. Each Translator must have, at the very least, a valid key that it will use to translate. If desired, different scenes can use different keys. This is usually not recommended, since it will lead to a lot of duplicate translations and added work.
In principle, the translator and its translation key is all you need to use the Babel Translation Toolkit.

2.2 String search options

String search options can be used to filter out text that does not need to be translated. Think of strings like placeholders, full names, technical data such as numbers or hex values etc.
Here is an overview of all the search options that can be used to filter as you desire:

option	**explanation**
Search prefix	Ignore any string that does not start with a given prefix. The idea here is that all string will have a form such as ”tr_ hello!”. This can be used for the most extremely complex projects, where you want to work with a whitelist system for translating. When using white listing prefixes, the key language is a sort of technical language that should not be shown to the end user. The wizard has built-in tools for creating a prefix-free language quickly.
Ignore prefix matches	Invert the use of the search prefix so that it is used as a black list in stead of a white list. This can be used if there is a small amount of technical strings that you want to avoid including, which you may denote for example as ”dnt_ MyCrucialParameterNameThatShould-NotBeTranslated”.
Ignore all caps	Ignore strings that are all caps. This is often useful since all caps strings tend to be abbreviations or technical stuff, such as hex codes.
Ignore no letters	Ignore strings that contain no letters. Likewise to all caps, these are usually technical strings or numbers that require no translation.
Case sensitive	Should the search differentiate strings that differ by case? It is recommended to leave this on at all times.
Max recursion depth	How many ”classes” deep should the string search go? In general, this can be as little as 2 or 3, unless you are looking for specific string variables in custom scripts. It is recommended to try and reduce this number for performance, while making sure the text search continues to find everything you need.
Auto create translators	Automatically create a Translator in every scene that is searched. The parameters set in the search utility will be automatically copied to the translators. It is recommended to always keep this option on.
Custom rules	Define specific search rules on a scene by scene basis. This option is only provided on the translator objects themselves. A custom rule will white or black list text on a basis of the parent class and variable name. Note that you can easily find the exact names of these by searching new text and hovering the cursor over the string in question in the Translator Wizard. It is recommended to use custom rules only in the case where you want to exclude one or two sources of strings. In other cases, you should use the prefix system in stead.

2.3 Advanced translation options

The translator offers a few additional options, related to its functionality:

option	explanation
Starting language	Language to translate to when the Translator starts. This value is the key language by default, which means that no translation will automatically occur.
Allow cross translations	Allow the translator to translate text from languages that are not the key language. In essence, with this option turned on, any string in the translation key can be translated into any available language. This option is provided mainly to deal with translated text being copied into text components, in which case the corresponding key text is not available. Note that for cross-translations to work accurately, different keys should never map to the same translated text. If this is the case, the translator will yield warnings, as long as ”log warnings” is turned on.
Reset canvas on translate	This option will make the translator turn off and on all canvas objects after translating a scene. This is recommended, since text components may not automatically show their newly translated text contents otherwise. The option is provided in case toggling the canvas this way messes with your custom scripts in some way.
Show default GUI	Show translation buttons in the top left corner of the screen. This is included for easy testing. It is not recommended to use this option in your end product.
Log warnings	Log development warnings. These mostly included problems that may be encountered while searching strings or setting up cross translations. It is recommended to keep this option on while working on your translations and turning it off afterwards.

2.4 Translation methods

The core of the translation functionality lies in TranslateScene(). There are however more methods to consider in a slightly more complex project. First of all, TranslateScene() might miss text references that were newly Instantiated. In such a case, you should refresh the scene text before translating. This can be accomplished with either HardRefresh() or QuickRefresh(). The former will apply the same text search that was used to gather translatable text in the first place and should be very thorough. It is quite heavy though and can take a few seconds to finish. If this is a problem, you can try out QuickRefresh() which restricts search to Unity Text components.
Another potential problem is that you may be assigning text via a custom script. In this case, you're most likely working with string constants that are embedded in your code. You can easily translate these string using the Tr() function, such as in the following example.

```
using UnityEngine;
using UnityEngine.UI;

using static BabelTranslator;

public class GameScript : MonoBehaviour {

    public Text text;

    public void UpdateText(bool state) {
        if(state)
            text.text = "Correct!";
        else
            text.text = "False!";
    }

    public void UpdateTextTranslated(bool state) {
        if(state)
            text.text = Tr("Correct!");
        else
            text.text = Tr("False!");
    }
}
```

In this example, the line "using static BabelTranslator;" makes sure that static methods from the translator can be used without writing "BabelTranslator" each time. Because of this, we can encapsulate a string constant with "Tr(...)" to automatically translate the contents. Note that in this case, you might still have to add the translations in question to the key manually.

2.5 Translation key editing

A Babel Translation Key is saved as a simple csv file. The first line contains the key language and all translation languages. The rest are just the key strings and their translations. As for the standard CSV format, the elements are seperated by a comma. Quotation marks are used to encapsulate commas that appear in text, in which case double quotation marks can be used to denote a single quotation mark. The parser also responds to escaped characters such as a newline character (\n), which can added to the text.
You can always just open the translation key with a text editor, such as notepad and edit it manually. You can also import a csv format into google docs, or microsoft office. In particular, the spreadsheet/excel format is ideal for working with CSV files.

One useful tip is to import your translation key to an online google spreadsheet page. You can then use formulae of the following form to automatically translate your key:

```
= GOOGLETRANSLATE(A1; "en"; "nl")
```

Note that services such as google translate are not context sensitive and may lead to some awkward translations if not manually checked.

3 Implementation details

3.1 Auto translator

This was an attempt to automatically use an online service to provide translations at the click of a button. This feature is not yet working.

3.2 Translation key

At run time, the translation key is simply a dictionary that maps key strings onto translation sets, which contains a new string for each translation languages. The result is a 1:1 mapping of key strings to translations.
If cross translations are allowed, an additional reverse mapping is included, which maps translated strings back to their key source. In this way, you can translate between any 2 languages by using the key language as an intermediate.
The main complexity lies in finding text to be translated in the first place. The search utility uses csharp reflection to recursively search classes distributed throughout the scenes in your project. This allows the search to find string variables that are buried away in custom classes. Extra care is recommended, since such a complete recursive search is very finicky, and can be very heavy.

4 Troubleshooting

If you find that the translator is missing some text elements you can go over the following checklist to see what the problem might be.

- `Translation key`
  Double check that the given text is actually in the translation key you're using.
- `Cross translations`
  The translator will not be able to translate from anything but the key language when cross translations are turned off.
- `Search fail`
  The translator may not be finding the text in question, double check the search options, in particular the custom rules. Note that if a custom rule does not have "exact match" turned on it will match any class or variable that simply contains the given names.
- `Reference fail`
  The translator might not have a reference to the given text because it was instantiated at run time. Make sure to try QuickRefresh() or HardRefresh() before doing a scene translation.
- `Script order`
  The text might be a string constant that is being assigned via script. Note that you can use the Tr() function to easily translate string constants in your code.