Game Maker Studio Localization/Translation Tutorial

So you've made (or are making) a **Game Maker Studio** game and you want to make sure people all over the world will be able to enjoy your game... but localization is difficult and expensive, right? Actually, it's easier than you think, and the following methods make the process as painless as possible for both GMS programmers and the translators!

Here is a .gmz example of these scripts in action:
https://dl.dropboxusercontent.com/u/2294969/LocalizationTutorial.gmz

The Localized Strings File

The first thing we need to do is create an included file called "**LocalizedStrings.ini**" where we'll store the different languages. Create the headings (enclosed with square brackets"[]") of the languages you want to use. You don't need to include any other text yet. Example:

```
[English]

[French]

[Spanish]
```

Once you've done that you'll need to save it as a **UTF-8 Unicode** file or it won't be able to handle special characters. Use a text editor like **NotePad++** and set **Encoding>Encode in UTF-8**. After you've saved it, import it into GMS as an **Included File**.

Setting/Detecting Language

Game Maker allows you to automatically detect the language of the player's operating system, so it will choose the correct language automatically. It will return a ISO 639 language code. You'll want to run a piece of code like this when the game loads, change it up based on the languages you want to use.

```
//detect the language

switch os_get_language(){
    case "en":
        global.language = "English";
        break;
    case "fr":
        global.language = "French";
        break;
    case "es":
        global.language = "Spanish";
        break;
    default:
        //If we don't support the language code, use english
        global.language = "English";
```

```
        break;
}
```

If you like, you could also have an option to let the user set their language manually in a settings menu, but for this tutorial we'll stick with automatically detected language.

Loading Strings

Creating the Script

You'll need to create a script that will look up your key string and replace it with the localized version. I prefer to **use a string as a key** rather than a number to remove a layer of abstraction (it also makes it much easier for the translators if they don't have to look up numbers that correspond to strings).

```
///localizedString(String Key)
//DO NOT run this script in the draw event
var _stringkey = argument0;
var _localstring;

ini_open(working_directory+"LocalizedStrings.ini");
if ini_key_exists(global.language, _stringkey){
    //The key exists, so we'll grab the value.
    _localstring = ini_read_string(global.language, _stringkey, "Key not found.");

}else{
    //The key doesn't exist, so we'll add a placeholder to the ini so we can translate it later
    //In the Game Maker Studio IDE, Go to Help>Open Project Data to find the placeholder strings
    //The value of this key will be placeholder text that includes the string and language code
    _localstring = "["+global.language+"_"+_stringkey+"]";
    ini_write_string(global.language, _stringkey, _localstring);

}

ini_close();
return _localstring;
```

The additional feature of this script is that if it can't find a key, it will make one for you in the save file (just remember to get the file from the game data directory, which can easily be found going to **Help>Open project data**)

Using the Script

To load a string from the ini file, use the following script:

```
playtext = localizedString("Play");
```

Remember that every time you use this script, it opens and close the file on disk. So it's best to call it just **once** at the beginning of the room to load the strings. **NEVER use this in a draw event!**

Special Font Setup

Different Languages will require different fonts, Luckily Game Maker allows you to use your **LocalizedStrings.ini** to only include the fonts characters you will use (great for Chinese and Japanese font types).

1. Create a new font.
2. Choose a font family that contains the characters you will need to generate.
3. Click the **+** sign to bring up font range window.
4. Click "From File" and choose All Files (*.*) and load **LocalizedStrings.ini**.
5. Name the font with the format "**fntDefault_[lang]_[size]".**

Make sure you do this before you publish your game, or you may not have all the characters that need to be included!

If you can't find a font that fits all your languages, you may need to create separate fonts for special languages and switching them when you set the language.

```
switch os_get_language()
    {
    case "zh":
        global.language = "Chinese";
        draw_set_font(Arial_zh_12);
        Break;
    case "jp":
        global.language = "Japanese";
        draw_set_font(Arial_jp_12);
        break;
    }
```

Final Notes

Eventually your **LocalizedStrings.ini** will look something like this:

```
[English]
Bird="Bird"
Dog="Dog"
Cat="Cat"
Chocolate="Chocolate"
[French]
Cat="Chat"
Bird="Oiseau"
Chocolate="Chocolat"
Dog="Chien"
[Spanish]
Cat="Gato"
Bird="Pájaro"
Chocolate="Chocolate"
Dog="Perro"
```

Tips Adding Localization to a Finished Game

If your game has text "hard-coded" in, it may be a challenge to add localization after the fact. One way would be to search your scripts (**Scripts>Search in Scripts**) for "draw_text" since the functions that contain that text are the only way GMS has to display strings. Once you've found those, go into the objects' create events and make variables to hold the localized text.

Another challenge will be spacing your localized text. Some words are much longer or shorter in other languages. Consider the **string_width()** and **string_height()** functions which can return the dimensions of a string, allowing you to adjust the font size automatically (make sure to use **draw_set_font()** before running those functions to make sure the measurements are accurate).