Localization Support Package

README

Thanks for buying Localization Support package! This simple system allows you to create multiple translations for your game without having to deal with every text separately. Moreover it's easy to use, scripted in C# and comes with full source code. Here, I'll show you how to make best use of this package.

PLATFORMS & VERSIONS

Works with Unity 4.6+. Any platform is supported, however language detection will only work on platforms with Application.systemLanguage working. Tested on Standalone and Android, with Unity 4.6 and 5.2.1.

GETTING STARTED

1. In your main assets folder, create a Resources folder.
2. Open your favourite editor and create a xml file for your strings according to the following structure:

```
<strings>
    <language1>
        <string id="your string id">Your string</string>
        . . .
    </language1>
    <language2>
        . . .
    </language2>
. . .
</strings>
```

   Language names are not case sensitive, however ids are. Only languages from SystemLanguages enum are supported. If you are unsure about something, take a look at Strings.xml provided with this package.
3. Import this file into your Resources folder.

4. Add LocalizationSetup prefab to your first screen of the game (e.g. Intro, Main menu, preloader). Set any language options you want in the Inspector with this prefab selected.
5. Now add LocalizationSupport prefab to every scene where you want to replace your UI Texts' texts with strings for a given language. You don't have to do this if you only need access to strings from code.

   NOTE: LocalizationSetup and LocalizationSupport may exists both in a single scene provided you reference LocalizationSetup in LocalizationSupport's parameters in the inspector.
6. Set your UI Texts' texts you want to use Localization support to your strings ids.

   EXAMPLE: You have a string with id "new". You want this string to load on a button.

   To achieve this you must set your button's text to your string's id: "new".
7. You're done. If you set everything up correctly you should have strings replacing ids on UI elements when you hit Play button.

NEW: CHANGING IMAGES/SPRITES

With this new version of Localization Support Package you can now change UI Images depending on the current language. Let's suppose you have a button named PlayButton and you want to swap images for different languages. Here's how to do this:

1. Open the scene, where you want to change UI Images.
2. Drag LocalizationSpriteLibrary prefab to hierarchy.
3. In the inspector, expand Sprites under LocalizationSpriteLibrary, set array size and add all the sprites you want to be language dependent to this list from your assets directory.

   NOTE: For each scene where you want to use image swapping you should have one (and only one) LocalizationSpriteLibrary.

4. Now open your strings.xml and add to appropriate languages:

   *<image id="your ui image **gameobject name** (ex. PlayButton)">Your LocalizationSpriteLibrary **sprite name**</image>*

5. Make sure you have LocalizationSupport prefab in your scene. If you have it already, it's ok. If not, just drag it from the assets.
6. You're done. If you set everything up correctly, you should see your sprite changing on your scene in play mode.
7. If you need more help see how I did this in DemoRuntimeChange scene.

DEMO SCENES

The system comes with two demo scenes:

- **DemoSetupInEditor** shows you how localization support is set up using LocalizationSetup prefab in Editor. Just what is described in Getting Started.
- **DemoRuntimeChange** is a bit more complicated and involves changing language at runtime. Take a look at DemoScene.cs to see how it's done (comments included). *NOTE: Changing language requires reloading the scene.*

API REFERENCE

Class: LocalizationSupport

- **UseSystemLanguage** – whether to override current language and use system language. Immediately after this is set to true, changing current language HAS NO EFFECT.
- **StringsXMLPath** – path to the strings xml resource. This is a resource name. "Resources\" and ".xml" (extension) must not be included in this path.
- **CurrentLanguage** – sets current language.
- **StringsLoaded** – checks whether strings are loaded for the current language.
- **LoadStrings** – loads strings for the current language. Do not call it manually unless you need to get some strings in Awake(). It is called automatically in Awake() every time you load a level.
- **GetString** – Gets string value for current language.
- **GetImage** – Gets image as Sprite for current language.
- **StringToLanguage** – Converts string to SystemLanguage. Throws an exception if the string is invalid.

CORRECT WAY TO CHANGE LANGUAGE AT RUNTIME (from DemoScene.cs)

```
//First set whether to overwrite any language setting with system language.
LocalizationSupport.UseSystemLanguage = autodetectToggle.isOn;

//Set the language you want. Has no effect if UseSystemLanguage is on.
LocalizationSupport.CurrentLanguage =
       LocalizationSupport.StringToLanguage(languages[currentLanguage]);

//Reload the level to apply changes to UI. Note LocalizationSupport.GetString(string)
will work without reloading the scene.
Application.LoadLevel(0);
```