

Multi-Language System (Web / Local) – V1.0.0

Legal Notes:

This package is Open Source. You're allowed to modify it however you want and share it for FREE, NOT FOR PAID. Plus, for every redistribution of this package you will have to insert my name (silvematt).

You're allowed to modify it however you wish, include it in your games and then sell it, but without earning money directly from those Assets.

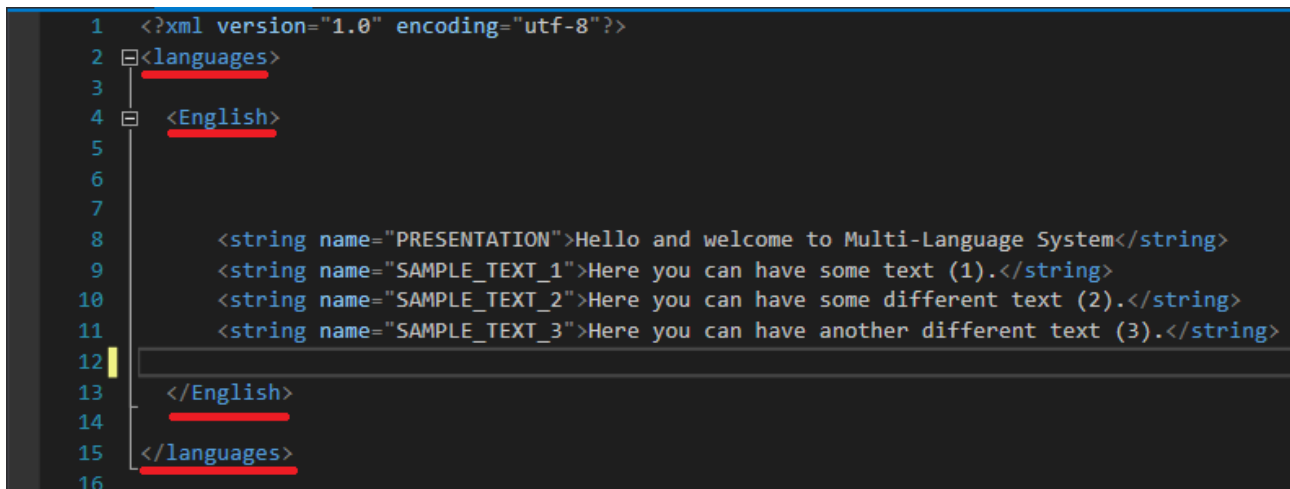
Hello and thank you for downloading this package!

I hope that it will be very usefully to you and a good helper for your project!

Chapter 1.0, Basic Concept.

The first thing that you have to understand is how an XML file is composed, as said in the description, you cannot know the XML and using MLS without any problem, but what is explained next must be known for understanding how MLS works.

Firstly, let's open a XML file that you'll find included in the package:



```
1  <?xml version="1.0" encoding="utf-8"?>
2  <languages>
3
4  <English>
5
6
7
8      <string name="PRESENTATION">Hello and welcome to Multi-Language System</string>
9      <string name="SAMPLE_TEXT_1">Here you can have some text (1).</string>
10     <string name="SAMPLE_TEXT_2">Here you can have some different text (2).</string>
11     <string name="SAMPLE_TEXT_3">Here you can have another different text (3).</string>
12
13  </English>
14
15  </languages>
16
```

You can see on the first line what is a “XML Declaration”, as this line identifies the document as a XML.

The second line is the declaration of an element, an element is nothing more than everything that starts from the element's start (“<languages>”) to the element's end (“</languages>”).

We gonna use the elements to group the texts of a language and then read everything inside them from Unity.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <languages>
3
4  <English>
5
6      String Index      String value
7
8      <string name="PRESENTATION">hello and welcome to Multi-Language System</string>
9      <string name="SAMPLE_TEXT_1">Here you can have some text (1).</string>
10     <string name="SAMPLE_TEXT_2">Here you can have some different text (2).</string>
11     <string name="SAMPLE_TEXT_3">Here you can have another different text (3).</string>
12
13 </English>
14
15 </languages>
16

```

What you see in the image is the declaration of a string in XML. It will contains the “Index” (the name) of the string and the value of it. You can add how many strings you want, it’s always a good practice to group all the strings that have the same scope and give them a self-explain name. For example:

```

<string name="PLAY_BUTTON">PLAY</string>
<string name="OPTIONS_BUTTON">OPTIONS</string>
<string name="EXIT_BUTTON">EXIT</string>
<string name="RETURN_BUTTON">RETURN</string>

<string name="SAMPLE_MESSAGE_1">Here you can have a message text (1).</string>
<string name="SAMPLE_MESSAGE_2">Here you can have a different message text (2).</string>
<string name="SAMPLE_MESSAGE_3">Here you can have another different message text (3).</string>

```

We have all the buttons and all the sample messages present in the login grouped.

This really makes your life easier to search and modify strings.

So basically watch a .XML file from the “XML” folder and see how it is composed, you’ll find three files, one that contains the textes in Italian, another in English and another in Espanol. It’s easy to understand that every language have his own XML file.

What does not change (and must not) from every XML file is the Index of the strings.

Chapter 2: Reading Language (Web/Locally)

Let's get deeper.

The "LanguageReader.cs" is the core of MLS, it allow us to read the XML file (using System.Xml) and to transfer all it contains in a hashtable.

We can get all this datas thanks to the "getLine()" function, that reads the hashtable with the index that we give as parameter and return us the string content

For more information on the hashtables read:

https://en.wikipedia.org/wiki/Hash_table

"LanguageReader.cs" class is not a class that derives from MonoBehaviour, indeed, it's a class that's instantiated from another script, the "LanguageManager.cs".

The main difference between those two scripts are that "LanguageReader" reads the file and save the strings in a hashtable, while "LanguageManager.cs" allow us to physically get the XML file to pass it to the Language Reader.

The script are explained with a lot of comments that explains with what and how they works.

Understanding how we "grab" the file from the web or locally is another important step, let's see how we do this.

For local-grab:

You may have noticed a folder named "Lang" that have the path "Assets/Lang" with inside the .XML files.

Well, let's open the LanguageManager.cs and look at the function "OpenLocalXML".

```

//Switch for the "Language" (as parameter), foreach language present in the game we have a different name file.
//Despite from the Web opening, here we instantiate the LanguageReader instantaniely, because the file must be
switch (Language)
{
    case "English":
        langReader = new LanguageReader(Path.Combine(Application.dataPath, "Lang/ENG.xml")) "English", true);
        break;
    case "Espanol":
        langReader = new LanguageReader(Path.Combine(Application.dataPath, "Lang/ESP.xml"), "Espanol", true);
        break;
    case "Italian":
        langReader = new LanguageReader(Path.Combine(Application.dataPath, "Lang/ITA.xml"), "Italian", true);
        break;
    default:
        langReader = new LanguageReader(Path.Combine(Application.dataPath, "Lang/ENG.xml"), "English", true);
        break;
}

```

What is highlighted is the location of the file “ENG.xml” that is the folder “Data” (Application.dataPath) -> to the folder “Lang” (“Lang”) -> and then file “ENG.xml”.

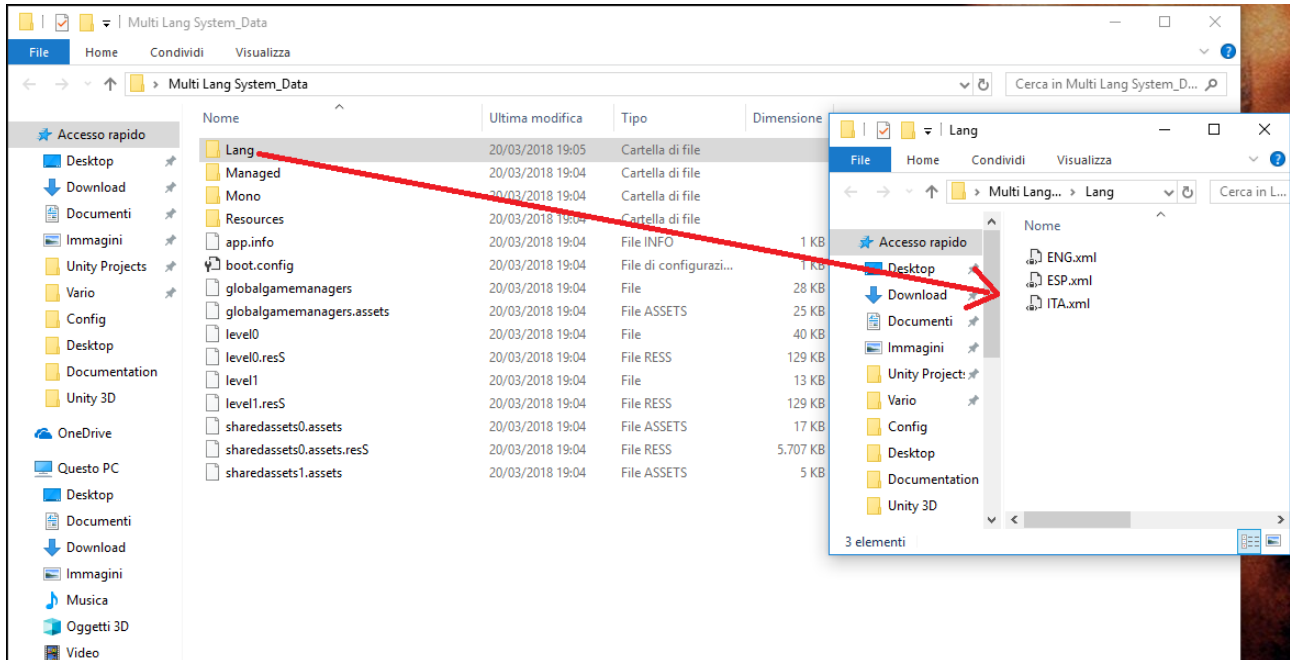
When you hit play from the Unity Editor, the folder “Data” will be the base “Asset” folder, so it will look for that:



Note:

When you import the .unitypackage file you will have “Multi-Language System” as a sub-folder of your Assets folder. What you have to do is to locate at least “Lang” in the main “Asset” folder.

When you have a build of your project (for example for windows), you will have a physical folder called “Data”, inside that you should create a “Lang” folder and insert the .XML files



Please note: You can also make an automatically creation of that folder with the files in it by using “StreamingAssets” for more informations read: <https://docs.unity3d.com/Manual/StreamingAssets.html>

For web-grab:

The web grab is a more slow process of the local one, in fact we'll have to wait the download of the file from the web, this waiting is really short, usually about 0.5-1.5 sec, but you should know that there is this wait.

Well, let's open the LanguageManager.cs and look at the function "OpenWebXML".

```
//Switch for the "Language" (as parameter), foreach language present in the game we have a different link for each file.
switch (Language)
{
    case "English":
        wwwXML = new WWW("http://yourdomainname.dx.am/MLS_Languages/ENG.xml");
        break;
    case "Espanol":
        wwwXML = new WWW("http://yourdomainname.dx.am/MLS_Languages/ESP.xml");
        break;
    case "Italian":
        wwwXML = new WWW("http://yourdomainname.dx.am/MLS_Languages/ITA.xml");
        break;
    default:
        wwwXML = new WWW("http://yourdomainname.dx.am/MLS_Languages/ENG.xml");
        break;
}
yield return wwwXML; //we wait for the reading
```

In this case we get the file from an URL thanks to the WWW class. So there is no need to have the file stored on the computer.

Using the Web solution can be very usefully in multiplayer games where you have files for the configuration of the world, those files should be untouchable and having them locally (with the possibility to crack and modify them) can cause serious troubles. Vice-versa, having files that contains dialogues or local settings on the web is useless and we constrain the player to wait some time to load basic texts.

Chapter 3: Outputting the XML to Unity

The last and still important process is to understand how we output what we wrote in the XMLs to Unity. Once we've selected a language, readed and saved everything in the hash-table present in the "LanguageReader.cs" (that is instantiated in the "LanguageManager.cs") we only have to take the value of the indexes and modify the "Text" or the "TextMesh" component.

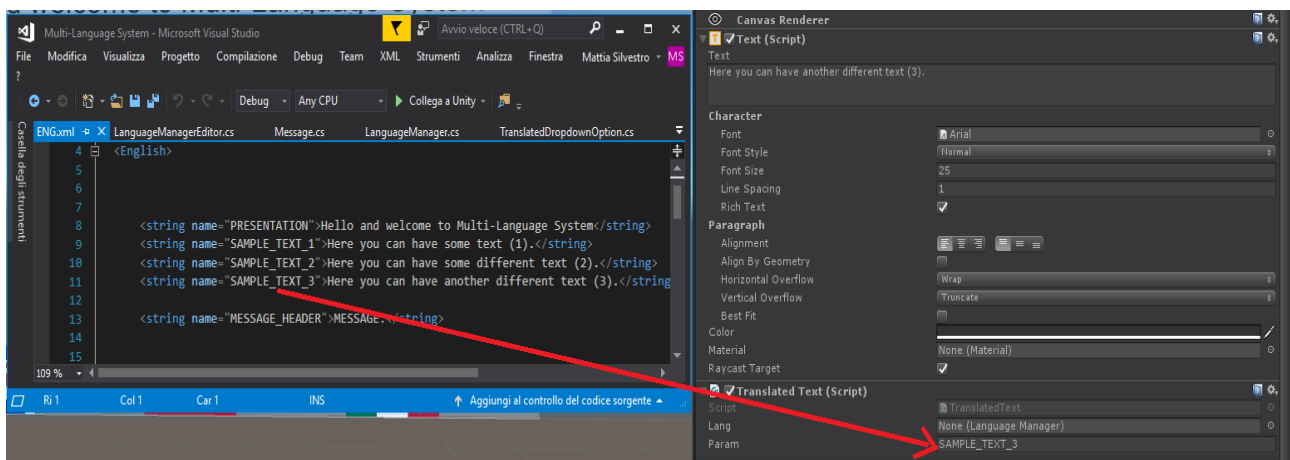
For that we use two scripts:

"TranslatedText.cs"

"TranslatedDropDown.cs"

Let's examine the first:

TranslatedText.cs will have a public string named "param" that must be the Index of the string that we want.



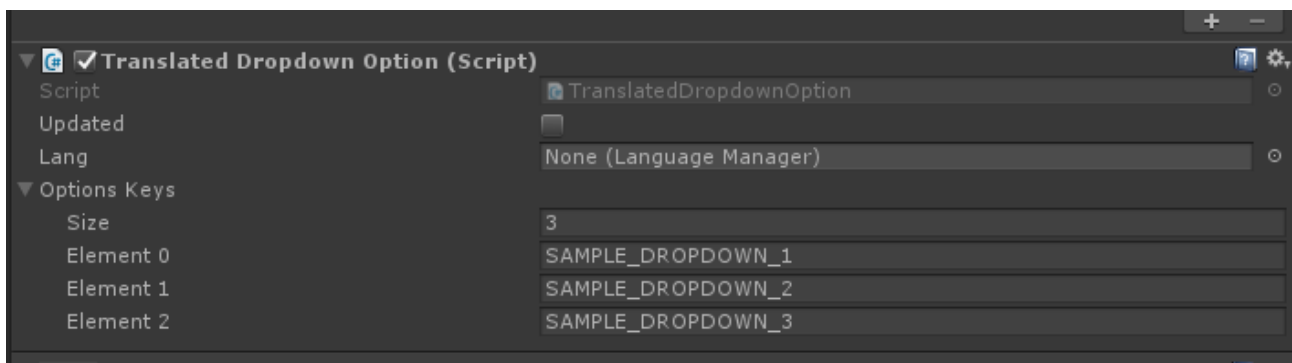
General rule: TranslatedText.cs must ALWAYS be in the same gameObject that contains the "Text" or the "TextMesh"(for 3d texts).

In the image I want that this Text have the value of the string with the index "SAMPLE_TEXT_3". That's all!

You do not have to consider the languages, because every XML files share the same Index for the strings, so every strings will be in the language that you choosed.

The same is for a 3D text, TranslatedText.cs is “smart” to understand if we’re dealing with a 2D text or a 3D text, so also for the 3D texts the process is the same.

A drop-down element is more complex than a simple text, for that you have in your arsenal “TranslatedDropDown.cs” that is a script that automatically creates the options with the translated text in your dropdown. All you have to do is to add the script to your gameObject that contains the “Dropdown” element and fulfil the array “OptionsKeys” with how many elements you want in your dropdown:



Every OptionsKeys element must have the Index name of the string that is present in the .XML, in this way you’ll have your texts translated for every language as drop-down elements.

That’s all! The options will be automatically created and the texts translated by the script!

Extra Chapter: Adding new functionality

In this Chapter we'll see together how to add new strings and how to create a new language.

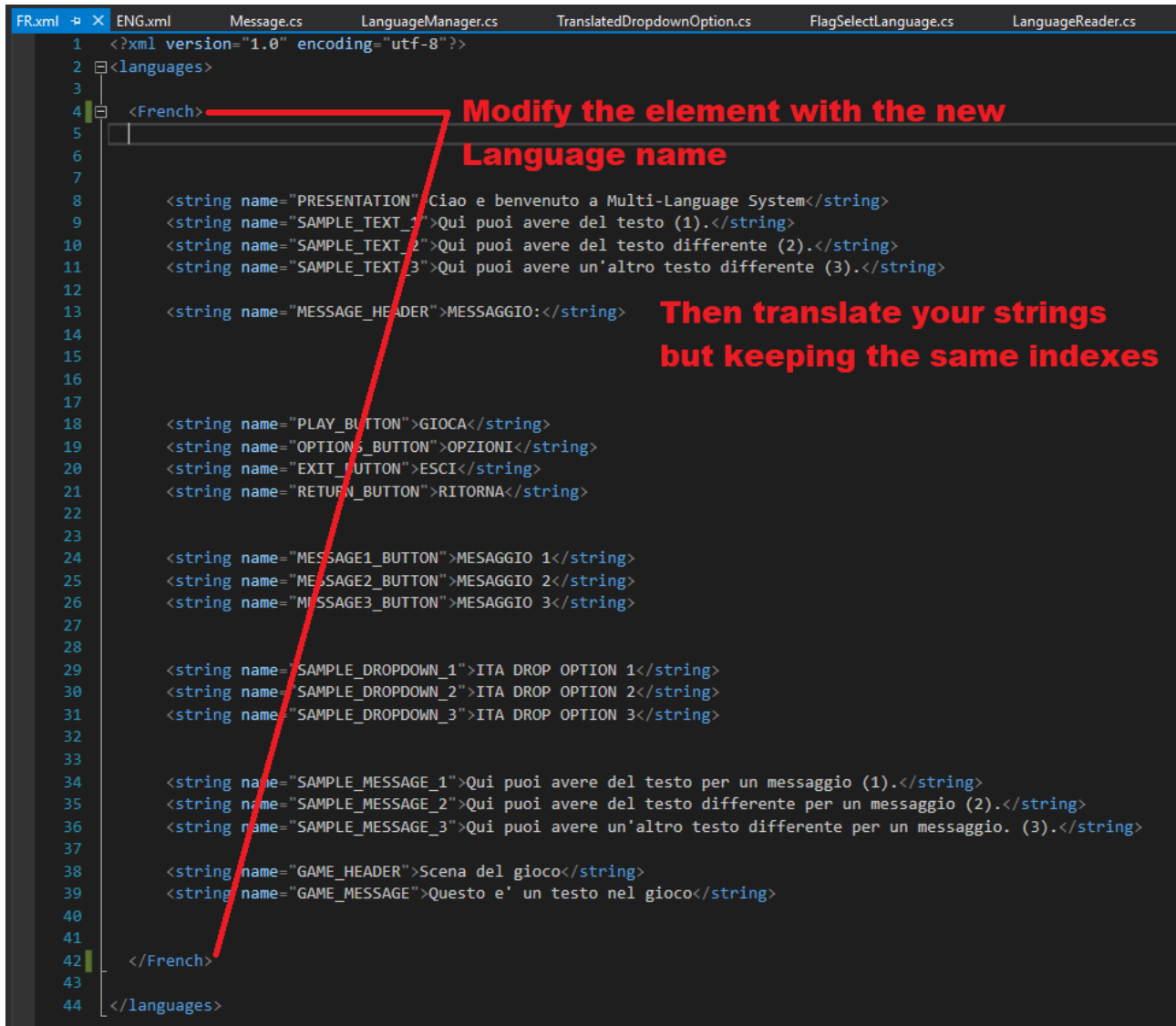
Adding a new string:

Just copy and paste any string you want, modify the index and the value. Notice that every .XML file MUST have the new string in the file.

Modify a string value is really easy, just open a XML file and modify the value of a string.

Add a new language:

The first thing that we have to do is to duplicate a XML file present in the “Lang” folder, rename it with the name you want and open it:



```
1 <?xml version="1.0" encoding="utf-8"?>
2 <languages>
3
4 <French>
5
6
7
8 <string name="PRESENTATION">Ciao e benvenuto a Multi-Language System</string>
9 <string name="SAMPLE_TEXT_1">Qui puoi avere del testo (1).</string>
10 <string name="SAMPLE_TEXT_2">Qui puoi avere del testo differente (2).</string>
11 <string name="SAMPLE_TEXT_3">Qui puoi avere un'altro testo differente (3).</string>
12
13 <string name="MESSAGE_HEADER">MESSAGGIO:</string>
14
15
16
17
18 <string name="PLAY_BUTTON">GIOCA</string>
19 <string name="OPTIONS_BUTTON">OPZIONI</string>
20 <string name="EXIT_BUTTON">ESCI</string>
21 <string name="RETURN_BUTTON">RITORNA</string>
22
23
24 <string name="MESSAGE1_BUTTON">MESAGGIO 1</string>
25 <string name="MESSAGE2_BUTTON">MESAGGIO 2</string>
26 <string name="MESSAGE3_BUTTON">MESAGGIO 3</string>
27
28
29 <string name="SAMPLE_DROPDOWN_1">ITA DROP OPTION 1</string>
30 <string name="SAMPLE_DROPDOWN_2">ITA DROP OPTION 2</string>
31 <string name="SAMPLE_DROPDOWN_3">ITA DROP OPTION 3</string>
32
33
34 <string name="SAMPLE_MESSAGE_1">Qui puoi avere del testo per un messaggio (1).</string>
35 <string name="SAMPLE_MESSAGE_2">Qui puoi avere del testo differente per un messaggio (2).</string>
36 <string name="SAMPLE_MESSAGE_3">Qui puoi avere un'altro testo differente per un messaggio. (3).</string>
37
38 <string name="GAME_HEADER">Scena del gioco</string>
39 <string name="GAME_MESSAGE">Questo e' un testo nel gioco</string>
40
41
42 </French>
43
44 </languages>
```

Modify the element with the new Language name

Then translate your strings but keeping the same indexes

Then, open the “LanguageManager.cs”, here we must add the case that the user can also select the new language, so open the script and go to the function “OpenLocalXML”:

```
//Switch for the "Language" (as parameter), foreach language present in the game we have a different name file, but the location of those is the same.
//Despite from the Web opening, here we instantiate the LanguageReader instantaniely, because the file must be not loaded from the web cause we've got it on the hard-disk.
switch (Language)
{
    case "English":
        langReader = new LanguageReader(Path.Combine(Application.dataPath, "Lang/ENG.xml"), "English", true);
        break;
    case "Espanol":
        langReader = new LanguageReader(Path.Combine(Application.dataPath, "Lang/ESP.xml"), "Espanol", true);
        break;
    case "Italian":
        langReader = new LanguageReader(Path.Combine(Application.dataPath, "Lang/ITA.xml"), "Italian", true);
        break;
    case "French":
        langReader = new LanguageReader(Path.Combine(Application.dataPath, "Lang/FR.xml"), "French", true);
        break;
    default:
        langReader = new LanguageReader(Path.Combine(Application.dataPath, "Lang/ENG.xml"), "English", true);
        break;
}
```

Here you must insert a new case in the switch with the name of your language.

If the case will be “French” so if we’ve selected the French languages we will load from the “Data/Lang” folder the file “FR.xml” that will contains all the texts translated in French.

Another thing to do is to set (if you want to use the Web path) another case in the switch of the “OpenWebXML()” method that contains the link of the FR.XML on the Web.

The last step is to add the option to select that language.

For that we use “FlagSelectLanguage.cs” that contains a string “Language”. This string must be fulfilled with the name of the language we want to select when we press that button.

That button will call the “SelectLanguage()” function present in the LanguageManager that will load a new language.

And that's all!

If you want to tell me something, please send me an email at:

silvematt@libero.it

Best Regards,

silvematt.

-silvematt