How to implement grammar processors into DFU

1 – Why to make grammar processors

We had one month ago on the discord Lysandus' Tomb the following question: how is it possible to force Daggerfall to adapt texts to the genre of objects for languages where objects have a genre? English is a language with a very simple grammar: objects have no genre, adjectives don't need to be modified depending the genre and the number. English people that never learned another language have no idea of what I'm talking about, but I'm sure that DFU team, that gathers people of multiple nationalities, knows this issue very well.

Let's take a very simple example.

In English, the sentence "the green %s" will be always valid, no matter by what %s will be replaced: a table, gauntlets... it always works fine.
Now if I take my language, French, and if I translate this string by a straight "le %s vert", this is valid only if %s is an object of genre masculine and singular: "le chapeau vert", it's OK. But if the object is of genre feminine, it should be written "la table verte", the sentence "le table vert" is incorrect with two errors. And if %s is feminine and plural, "le bottes vert" should be written "les bottes vertes". Other languages as German or Polish are even worse because some of them have more than two genres and a more complex grammar with, for example, the genitive that is a big word modifier.

Due to the simple English grammar, all games that are designed in English and that use variable expressions as %s where the word that will replace %s is unknown during the translation phase have no problem because there are few sentences where the genre may cause grammar issue. Only with humans and dual words as "his/her" or "king/queen", and for these few issues they invent a trick (for DFU, a macro %xxx) to choose one or the other word depending the genre of the concerned human. But they design nothing to be able to take into account the fact that objects have a genre and that adjectives must be modified depending the genre of the hero in other languages. And so we, translators, are naked before this problem.

The only game I know where developers decided to implement ways to take this in account, probably because they are not native English speakers but Turkish, is Mount&Blade: Warband. They have implemented special expressions that permit to change the result depending the genre, at least for humans. For example, the expression {xxx/yyy} means "write xxx if the main character is a man and yyy if she is a woman". So, a sentence in English as "Hello, you are beautiful today" said to the main character may be translated by "Bonjour, vous êtes {beau/belle} aujourd'hui", and the text parser of the game will write "vous êtes beau" or "vous êtes belle" depending the genre of the main character. This expression is the more simple one, TaleWorlds implemented in Warband other ones that permit to do things like that in other grammar cases, but not enough.
Their masterpiece in terms of respect to grammar can be found into M&B II:Bannerlord. In this game, they develop the concept of so-called "tokens", expressions written as {.xxx}, that are written into the translated files and processed by a routine called ProcessToken(). There are as much ProcessToken classes as languages implemented in their game, and each language has its own set of tokens and code to process them. The interest of the way it is done is that all the tokens are to be written into the translated files only and that the routine ProcessToken() may be externalized into a dll for languages that are not natively translated by Bannerlord through the use of the library Harmony.

Before TaleWorlds launched officially the French language into release e1.70, the class FrenchTextProcessor that contained a void routine ProcessToken() was already existing, so French translators as me that translated Bannerlord since its very first early access release created their own ProcessToken() routine with tokens we invented. This lead to a dll called GrammaireFR that was able to implement French grammar rules needed to generate correct sentences into the French translation. And because this library, from which I am the author, rely on different tokens that those used by TaleWorlds and has a totally original code, it may be used for other games. In other words, I had in hand the code of a French grammar processor that could be used for DFU and any other game that would accept to call it.

The principle would be, in the game, to read a translated string from translation files, to store it into a string named for example *stringToDisplay* and to add a call *ProcessGrammar(StringToDisplay)* before to display it. The grammar processor would interpret the grammar tokens contained into *stringToDisplay* and send back a new string where the grammar tokens would have been replaced by a correct text.

To be clearer, let's take an example.

English string: "I want the green %s."
Direct French translation: "Je veux le %s vert." -> very bad if %s is feminine and/or plural: display of "Je veux le bottes vert." instead of "Je veux les bottes vertes."
French with tokens in translation files: "Je veux {.le}%s {#vert#verts#verte#vertes}"
French processed by ProcessGrammar(): "Je veux l'arbre vert." or "Je veux les bottes vertes.", OK!

The interest is that you don't have to implement in the game any specific code dedicated to the grammar rules of any language other than English, you just have to add one call to ProcessGrammar() with the string read into the translation file, and the grammar processor will send back the correct string to be displayed.

You may watch a small video that shows the result of ProcessGrammar() into a test software of my own called "Test GrammaireFR" upon the above example with that link (download it before watching, the Google Drive default video reader is quite bad):
https://drive.google.com/file/d/15WsM8HbXBn25RJh8VQ40vZx7uy1m-ZGS/view?usp=drive_link

No need to say that it could be possible to code grammar processors for other languages, with their own tokens and rules, as it is done into Bannerlord that will stay a very good example for that as this game contains grammar processors for English, French, German, Italian, Polish, Russian, Spanish and Turkish (to be found into TaleWorlds.Localization.dll).

Once again, my code and the token I use are different from those contained into the FrenchTextProcessor of TaleWorlds Bannerlord, so I'm not breaking any legal property, I just retained the brilliant idea of using and process grammar tokens into the translated text. My code shall be public and translators in other languages may try to start from it to build their own grammar processor.

2 – How DFU must be modified

2.1 – The script Grammar.cs

First, we must introduce a new abstract class GrammarRules that shall contains the two abstract methods ProcessGrammar() and SetGenreHero().
Second, we must create a static class named GrammarManager that will permit to store the current grammar processor to use.

Both classes may be grouped into a single script named Grammar.cs that I propose to deposit into Assets\Scripts\Localization.

Here is the content of the script Grammar.cs:

```
using GrammarModule.LanguageRules;

namespace GrammarModule
{
   public abstract class GrammarRules
   {
      public abstract string ProcessGrammar(string text);
      public abstract void SetGenreHero(string Genre);
   }

   public static class GrammarManager
   {
      public static GrammarRules grammarProcessor = (GrammarRules) new
DefaultGrammarRules();
   }
}
```

2.2 – The default grammar processor

The default grammar processor does almost nothing. It exists just to permit the initialization of GrammarManager.grammarProcessor and must be considered like a stub that will react to the call to the two methods of the class GrammarRules that will be put into the code of DFU.

```
namespace GrammarModule.LanguageRules
{
        public class DefaultGrammarRules : GrammarRules
        {
                public override string ProcessGrammar(string text)
                {
                        // Process the grammar tokens
                        return text;
                }
                public override void SetGenreHero(string Genre)
                {
                        // Store the genre of the hero for the grammar tokens
                }
        }
}
```

As you can see, ProcessGrammar() is a function that just send back the text received as parameter, and SetGenreHero() does nothing.

This script is called *DefaultGrammarRules.cs* that we can also deposit inside *Assets\Scripts\Localization*.
Its structure shall be the one of the grammar rules for foreign languages.

2.3 – Calls to ProcessGrammar() from inside DFU code

Up to now, we did not change the DFU code, we just added two scripts that define classes.
With the two scripts *Grammar.cs* and *DefaultGrammarRules.cs* put into *Assets\Scripts\Localization*, it is now possible to place calls to the two grammar methods into the code.

Let's take as example the script *TextManager.cs*.

First, let's add at the beginning a `using GrammarModule;`

Then, in the method GetLocalizedText() (line 356), we can found:

```
if (TryGetLocalizedText(GetRuntimeCollectionName(collection), key, out localizedText))
            return localizedText;
```

The variable *localizedText* contain the text read into the translated files.
To have it grammar processed, we just have to write:

```
if (TryGetLocalizedText(GetRuntimeCollectionName(collection), key, out localizedText))
            return GrammarManager.grammarProcessor.ProcessGrammar(localizedText);
```

And that's it !
The static class GrammarManager exists, the grammarProcessor is initialized with the class DefaultGrammarRules, ProcessGrammar() is a method of this class and the code works. With the default grammar processor, ProcessGrammar() just send back the same string, nothing changed into the way DFU works.

So you understand the concept: in every place of DFU code where localized text is to be grammar processed before to be displayed on screen, we just have to add a call to `GrammarManager.grammarProcessor.ProcessGrammar(Text)`, and in return we will receive the processed string.

After many trials, I found four scripts where one or several calls to ProcessGrammar() are to be added:
- Game\TextManager.cs (1 call)
- Game\UserInterface\MultiFormatTextLabel.cs (1 call)
- Game\UserInterfaceWindow\DaggerfallPlayerHistoryWindow.cs (1 call)
- Game\UserInterfaceWindow\DaggerfallTalkWindow.cs (6 calls)

2.4 – Calls to SetGenreHero() from inside DFU code

This shall permit to process tokens as {male/female} related to the genre of the main character.

The genre of the current main character is defined in two different places: when we create a new game and we are asked to choose the genre of the character, and when we load a saved game.

The first case is the simpler. The question is asked into the script
Game\UserInterfaceWindow\CreateCharGenderSelect.cs

There are two callbacks MaleButton_OnMouseClick() and FemaleButton_OnMouseClick() where we just have to add one call in each:

```
GrammarManager.grammarProcessor.SetGenreHero("M");
```
and
```
GrammarManager.grammarProcessor.SetGenreHero("F");
```

The second case is more difficult to me because there are in fact two different windows where it is possible to load saves, depending you want to load a DFU save or a classic game save. And frankly it is not evident to find the right place where to find the genre of the character after loading and where to insert a call to SetGenreHero().
I must admit that this part is not yet achieved, I'm a little lost in the load call of the two windows and I must keep on searching how I can do that. Let's hope it will be done when you will send me back your acceptation of the above DFU code modification.

3 – What to do on the translator side

This document is not done to explain how to code the grammar processor, this will be done in another document dedicated to the other translators… provided the code modification presented in chapter 2 is accepted by the team.

Three things have to be done almost simultaneously:

- Code the grammar processor.
- Build a dfmod that will permit to connect the grammar processor to DFU.
- Tokenize the translated files.

Each language must develop its own grammar processor, based on the model of the default processor. For the French, I've coded the script *FrenchGrammarRules.cs* that has the following structure, almost similar to the one of DefaultGrammarRules:

```
namespace GrammarModule.LanguageRules
{
   public class FrenchGrammarRules : GrammarRules
   {
      […]
      public override string ProcessGrammar(string sourceText)
      {
         […]
         return processedText;
      }

      public override void SetGenreHero(string Genre)
      {
         // Store the genre of the hero
         […]
      }

      Private static… (internal routines)

   }
}
```

More interesting is the way to connect the grammar processor to DFU.

For that, you must create a script with a class of MonoBehaviour type in order to be called via the method Awake().
And in this script, only three lines are needed for the grammar processor:

```
using GrammarModule;
using GrammarModule.LanguageRules;

namespace MyMod
{
   public class MyModMain : MonoBehaviour
   {
      private void Awake()
       {
          GrammarManager.grammarProcessor = (GrammarRules) new FrenchGrammarRules();
```

And that's it! With this line, the default grammar processor is replaced into the GrammarManager by the grammar processor of the wanted language (here the French one).

So you just have to put into the mod both scripts, the Main one and MyLanguageGrammarRules.cs, and it's done, the grammar rules shall be applied on the grammar tokens inserted into the translated files.

Please note that, as in French we already have a French mod with scripts in order to produce the names of the shops in towns that are generated at the beginning of each new game, I just had to add the two using and the line to initialize the GrammarManager with FrenchGrammarRules to the existing script PFDMain.cs, and add the script *FrenchGrammarRules.cs* to the content of the mod so that everything works.

To tokenize the translated files is a quite tedious phase: the translator must add all needed tokens into the translated files so that they are correctly processed by the grammar processor. But all this in not the concern of DFU devs, it is just the translator's work.

4 – Conclusion

What I'm asking to the DFU team is:

- To add, into Assets\Scripts\Localization, the two declarative scripts that I will provide, *Grammar.cs* and *DefaultGrammarRules.cs*.
- To accept (or produce by themselves if they are not confident) the PR that will add the needed calls to GrammarManager.grammarProcessor.ProcessGrammar() into the 4 DFU scripts listed in §2.3. I will provide the modified scripts.
- To accept the PR that will add calls to GrammarManager.grammarProcessor.SetGenreHero() into *Game\UserInterfaceWindow\CreateCharGenderSelect.cs* and one or two scripts (still to be found) that modify the genre of the main character after loading of a save.

When this will be done, **DFU will work exactly as before** whatever is the current language because the call to the default grammar processor shall be totally transparent.

Only when a mod that contain a grammar processor for a certain language shall be enabled, something will change for the user of DFU in that language: they will discover that the text displayed in their language contains less grammar errors than before if the translator did a good job through the grammar processor and the grammar tokens added into the translated files.

I hope the French shall be the first language to benefit of these changes. My French grammar processor works, I see the changes into the windows and dialog boxes of DFU. I still have to modify some small things, polish the code and comment for possible other grammar processor producers, but the main job in front of me is to keep on tokenizing the translated files. This is going to take some time but this is only on my side, it is not a concern for DFU. Since then, the main thing to do is to have a deal on the modifications to be done into DFU, the French mod with the grammar processor shall come after they are in… or never if you decide not to accept to introduce the possibility to call a grammar processor into DFU.

Thanks for reading this (too) long document,

Best regards,

Daneel53
May 19, 2024