Overview

CRYENGINE is able to localize text and sound for different languages. All of the necessary localization data is stored inside pak files, stored in the `<root>/Localization` folder (example: `CRYENGINE_3.5/Localization/English_XML.pak`).

Inside of the pak files all files contained in the Languages folder can be directly translated except *dialog_recording_list.xml* and *ai_dialog_recording_list.xml*. These two files are used by the dialog system and need further explanation.

The **g_language** console variable can be used to load a specific language, which is addressed by the name of the language pak file (example: **g_language = german** for German.pak).

The dialog recording lists as well as translated text are loaded and handled by the LocalizationManager class. Make sure that new files are correctly loaded in the *CSystem::OpenLanguagePak()* function.

Backwards Compatibility

From CRYENGINE 3.4.3 and on the localization files have moved from `<root>/Game/Localized` to `<root>/Localization`. Also note the PAK layout has changed. The dialog folder with the audio files is located in the English.pak and the XML and gfx font files are located in the English_xml.pak.

You can set the **sys_localization_folder** CVar to "Languages" to use the old system. In 3.4.3 and onwards, it is set to "Localization" by default which will make it use the new structure.

Usage

Inside certain scripts shipped with the SDK, you'll find references to localization strings. One example can be found in the InteractiveEntity object: `GameSDK/Scripts/Entities/Others/InteractiveEntity.lua`

```
UseMessage = "@use_object",
```

The localization string for this reference is located inside: `<sdk_root>\Localization\English_XML.pak\text_ui_ingame.xml`

Key	Original Text	Translated Text
use_object	Use object	Use object

This table is used to define strings of text which are then displayed in game. When the player walks up to the InteractiveEntity Entity, "Use object" will be displayed on the screen.

If the string is defined in alternate languages and the player use using an alternate language, it will be displayed as such.

PAK/Folder structure

Using the localization files inside pak files allows you to create multiple language packs. For example, CRYENGINE SDK ships with 3 language packs, English (default), German and Korean. The structure of these pak files is the same for each language.

For testing purposes though, you are not required to use .pak files. You can place .xml files for example in the main Localization folder and the engine will read these by default (similar to extracting objects from pak files, the engine will prioritize external content).

<root>			
..	**Localization**		
..	..	**English.pak / English_XML.pak**	
..	..	..	text_ui_warnings.xml
..	..	..	dialog folder
..	..	..	etc
..	..	**German.pak / German_XML.pak**	
..	..	..	text_ui_warnings.xml
..	..	..	dialog folder
..	..	..	etc
..	..	**Korean.pak / Korean_XML.pak**	
..	..	..	text_ui_warnings.xml
..	..	..	dialog folder
..	..	..	etc

..	..	text_ui_warnings.xml
..	..	dialog folder
..	..	etc

When you don't use pak files but rather xml files in a folder structure, you need to copy the files of the language of choice to the root folder (not `Localization/<language>`), example: <root>/Localization/text_ui_warnings.xml

Translating Spoken Text

As the XML files mentioned above contain all the text that is spoken in a game, they can be very large and are best viewed inside spreadsheet software, such as Excel. Inside the files, each line represents sentences spoken by a character. A detailed description of single parameters of these lines can be found in The Dialog System. Sentences spoken by different characters can be composed to a dialog by the Dialog Editor as an additional step, for further use in Track View scenes or Flow Graphs.

The *AUDIO_FILENAME* row, which is also called *sound-key*, is a unique identifier for spoken sentences. It directly references the audio file in the dialog folder, with the Languages folder and the file extension being skipped. This sound-key can be copied into the Dialog Editor or any dialog sound field, because it is a valid sound name. There are two different files needed for each line specified in the XML table, which are automatically loaded by the engine; one optional and one mandatory.

- **.FSQ** - This file contains the facial animation data, applied to the character that is speaking. It must be generated in the Facial Editor, depending on the audio source. To speed up the translation process, these files may be taken as a basis for lip synchronization and taken over without any change. Dialog lines that are spoken offscreen do not need the FSQ file.
- **.WAV**, **.MP2**, or **.MP3** - This file contains the audio source. Its extension must be consistently of only one format as the engine allows for setting up the decoding mechanism globally (by the **s_Compression** console variable). For example, even if a sound is referenced to use the WAV extension internally, the MP2 file will be loaded if **s_Compression** is set to 2. Ideally, all dialog references should point to WAV files. If so, the compression of choice can be selected and easily changed.

In the simplest case, translating the audio data is just a matter of translating the audio files contained in the structure described above.

Texture Localization

For detailed information on localizing textures, refer to the Texture Localization article.