Tutorial for Globalization Support

- Introduction
- SceneComposer Translator Plugin
- Managing Language Packs using AssetTool
- Using Globalization in Applications
- Custom Localizer Support

Introduction

Multi-Language Support

Preparing an application to perform in multiple locales requires the CGI Studio license "Globalization" and in terms of Candera build settings to enable the CMake feature switch "CANDERA_GLOBALIZATION_ENABLED".

Preparing an application to perform in a specific locale requires the following workflow:

- **Application Development**
  - Create Master Language Pack Source (either directly in SceneComposer or by importing from an external tool)
  - Will be automatically added as binary to the asset library on asset export from SceneComposer
- **Production**
  - Remove Master Language Pack Source from asset library
  - Create Language Pack Source for each language
  - Append Language Pack for each language to the asset library

Language Pack Source Format

In general, a Language Pack Source file

- has the file extension '*.lps*'
- is a proprietary file format: Zipped XML files in a predefined structure
- contains texts associated to a specific culture

Master Language Pack Source

The Master Language Pack Source file is a design-time artifact not to be kept at production time. It defines all available translatable texts in an artificial Master Language (locale 00).

Extracting the content of a Master Language Pack Source .lps file shows following content:

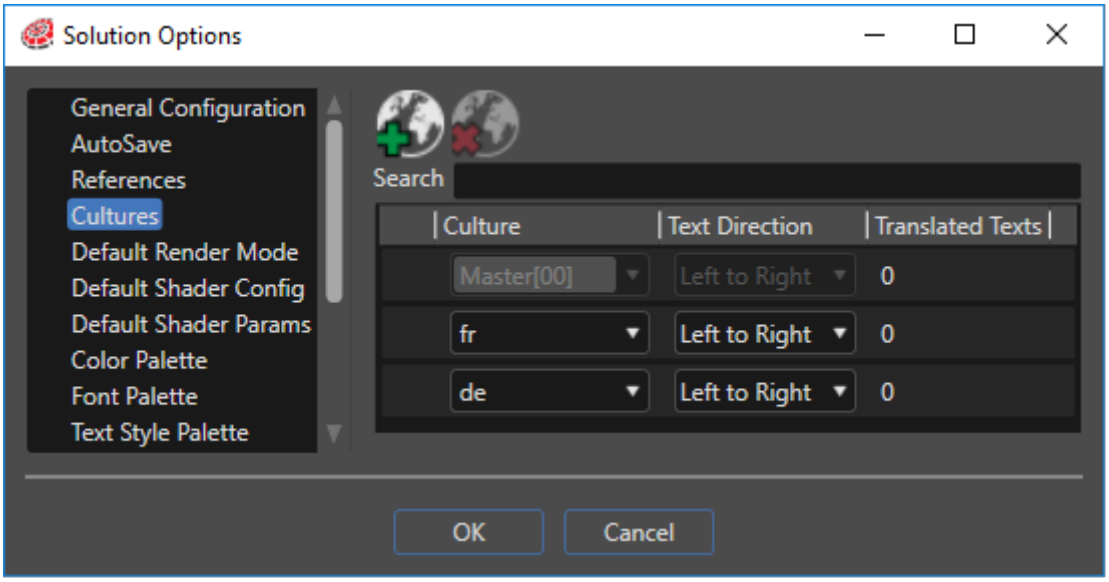
- Cultures.xml

```
<lps:Culture Name="Master Language" Locale="00" TextDirection="LeftToRight" />
```

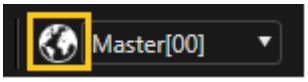
- LanguagePack_00.xml: Lists all unique Text-IDs with related default text representation, a translation hint and a maximum text length for the Locale="00"
- Partitions.xml
- Preamble.xml

Cultures

SceneComposer cultures are defined in the "Cultures" panel accesible either via *File->Solution Options* menu:



either via toolbar:



The default culture for a solution is defined in the 'General Configuration' panel as explained here.

Multi Language Pack Source

A Multi Language Pack Source provides translations for all translatable texts as defined in the Master Language Pack Source. All supported languages can be combined into a single .lps file, or be separated in single .lps files per language.

The culture definitions in language pack source files should syntactically match the culture definitions in SceneComposer to make the default culture setting of a solution work at application runtime. Example matching to the SceneComposer culture definitions above:

```
<lps:Culture Name="German" Locale="de_DE" TextDirection="LeftToRight" />
<lps:Culture Name="French" Locale="fr_FR" TextDirection="LeftToRight" />
```

In case the default language is not set or not present in the asset, the first culture available in the asset will be used as fallback.

Appending Language Packs to Assets

Language Pack Source files are compiled into a binary format and appended to asset libraries

- by means of SceneComposer Translator Plugin after asset export (Note: Enable "Plugin Notification" to make this work!)
- by means of Managing Language Packs using AssetTool to remove the master language pack from and to append multi language packs to an asset.

More Details

More details on the globalization workflow are provided here:

- SceneComposer Translator Plugin
- Managing Language Packs using AssetTool
- Using Globalization in Applications

SceneComposer Translator Plugin

Overview

CGI Studio SceneComposer Translator Plugin provides the following features:

- Create, edit and delete translatable texts in the *Language Table* dialog
- Set and manage translatable texts in the *Text property editor* of text nodes or controls and behaviors that expose Text properties.
- Alternatively import translatable texts from an existing CGI Language Pack Source or Excel file.
- Export translatable texts to a CGI Language Pack Source or Excel file.
- Append a compiled CGI Language Pack file to an asset after asset export.

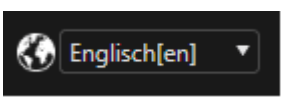
Globalization License

If the Translator Plugin files are present but no translation features are accessible in Scene Composer, please ensure that the *Globalization* license is acquired. Refer to the menu *Help* > *Licenses*.

Create and Manage Cultures and Translatable Texts

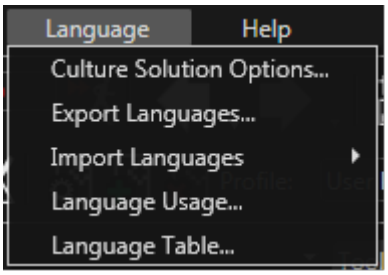
Accessing Translator Plugin in SceneComposer

The translator plugin is fully integrated into SceneComposer. The SceneComposer toolbar provides an icon for opening the *Solution Options: Cultures* dialog as well as a drop-down menu for selecting an already created culture.



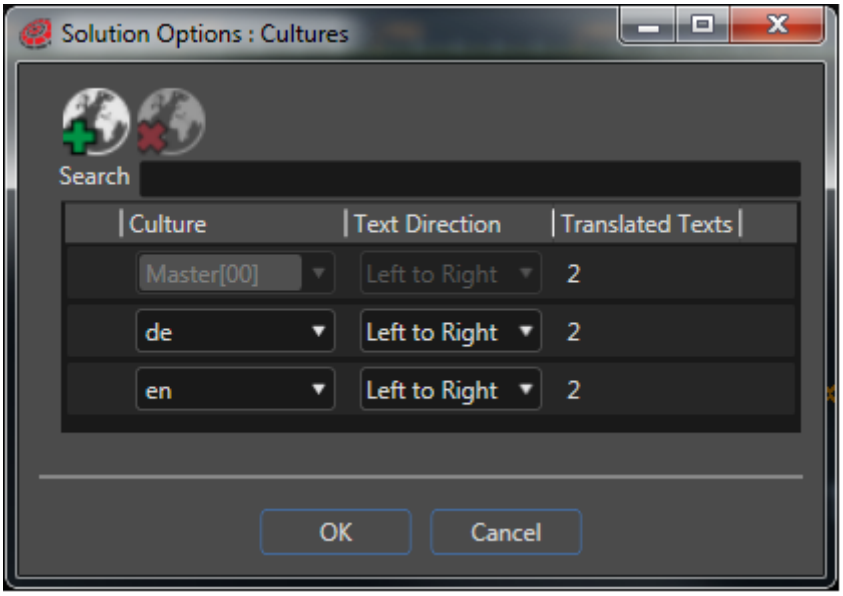
There are five additional menu entries under *Language*:

- Culture Solution Option: Open the Culture Solution Option Dialog, where the user can add, edit or remove Culture as the multiple languages setting. By selecting a Culture in SceneComposer, you can switch between the displayed language of the solution. The translated texts displayed when selecting a Culture are configured in the Language Table.
- Export Languages: Opens the *Select Cultures* dialog where you can select the languages you want to export and - after browsing to an export file path - you can export your selected languages in the format you prefer to the location you selected by pressing *Export*.
- Import Languages: Opens the Import Languages Dialog where Languages can be imported into a solution either as Language Pack Source or as Excel file, by going to *Language* > *Import Languages* and selecting the desired flavor.
- Language Usage: Opens the Language Usage Dialog, where all Text Properties of the solution are listed. For those Text Properties that have a translatable text assigned, the *Text Id* and the *Text* value for the current culture are displayed. It is also possible to assign or unassign a translatable text for a Text Property.
- Language Table: Opens the Language Table Dialog, where the user can add, edit or remove translatable texts. The Id of each translatable text is unique. The Text value for a certain culture can be accessed by selecting the respective culture in the Current Culture drop-down.



Creating Cultures

In order to have texts in different languages, cultures have to be created. This is done by clicking on the button *Culture Solution Options* in the toolbar.

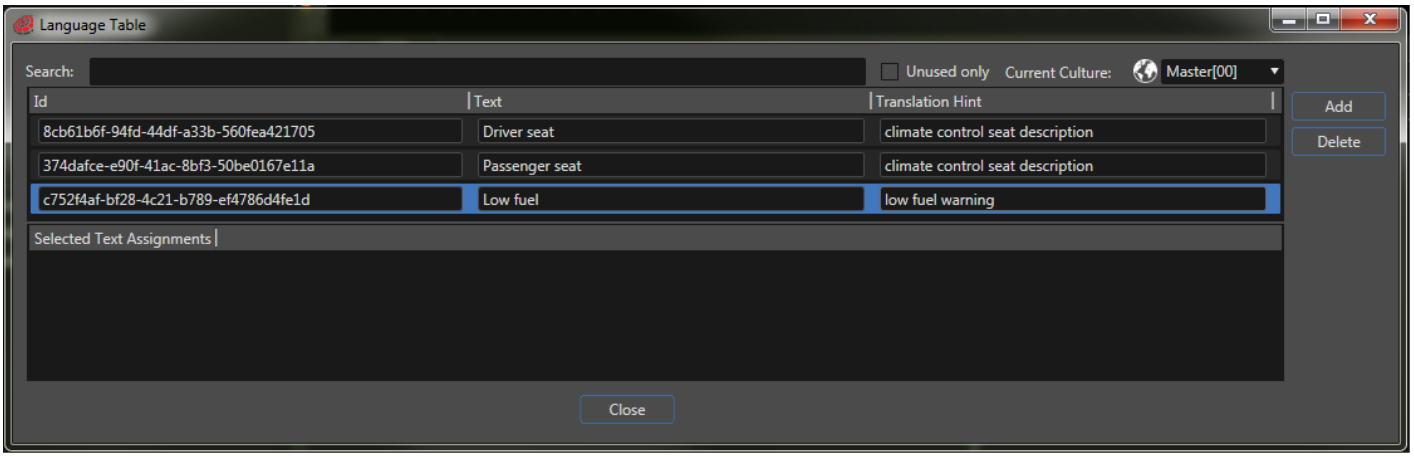


With the icons on top, cultures can be added (using the green plus sign) and deleted (with the red "x"):

- to add a new culture: use the button with the green plus sign
- to delete a culture: use the button with the red "x" For each entry the culture name can be selected from the drop-down and the text direction can be set. Once there are translated texts available for a certain culture, the number of available text for this culture is displayed in the Translated Texts column. In the above example there are two translatable texts for culture *de* and two translatable texts for *en*.

Translate Texts

Texts can be translated in the Language Table Dialog, which is opened via the menu *Language* > *Language Table...*.



The drop-down on the top right allows the Current Culture selection. The texts of the Master culture or another culture in the solution can be accessed by changing the Current Culture. If the Master culture is selected, the Add and Delete buttons are enabled, otherwise they are disabled.

A search field on top helps to easily find specific texts once the number of texts increases. With the checkbox *Unused only* it is possible to identify texts that have not yet been assigned to any Text Property of the solution.

Clicking on the globe icon on the top right opens the *Solution Options: Cultures* dialog.

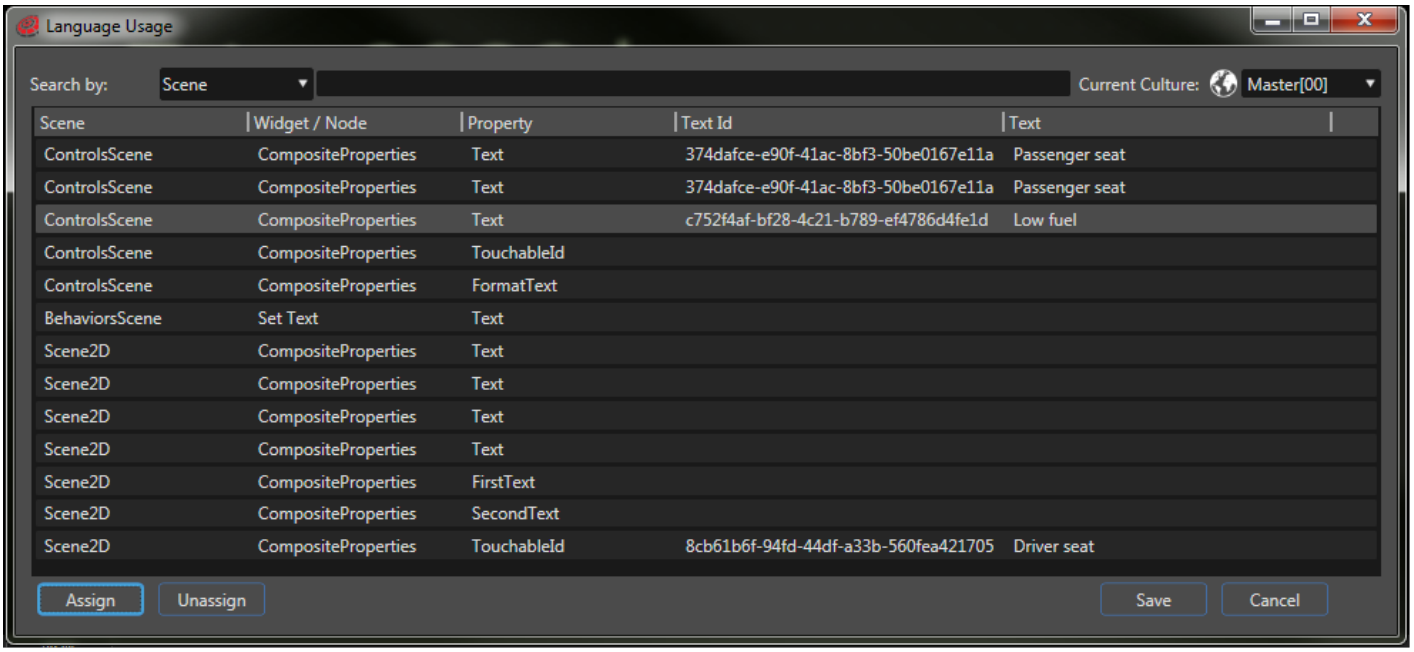
Translatable Text

A translatable text is a text which supports translation into other languages and which provides information that facilitates its translation to other languages. The translatable text has the following properties:

Id:	Unique identifier of the translatable text: • *Computed TextId*: An alphanumerical string, which will be exported into a computed hash value as TextId. • *Direct Id*: When the format HEX_VALUE is used, the hex value is directly exported into the asset without computing a hash value.
Text:	Text in the master language
Translation hint:	Hint for the process of translation

Editing Text Properties

For editing the texts' properties, Scene Composer provides the Text Properties Dialog, which can be opened via the menu *Language* > *Language Usage...*.



This dialog lists all Text Properties of the solution. For those Text Properties that have a translatable text assigned, the *Text Id* and the *Text* value for the current culture are displayed. The current culture can be changed in the drop-down menu on the right. If necessary, the *Solution Options: Cultures* dialog can be opened with the globe icon on the left of the drop-down menu.

When pressing *Save*, the changes made to the Text Properties will be applied to the solution.

Assigning Texts to TextNodes

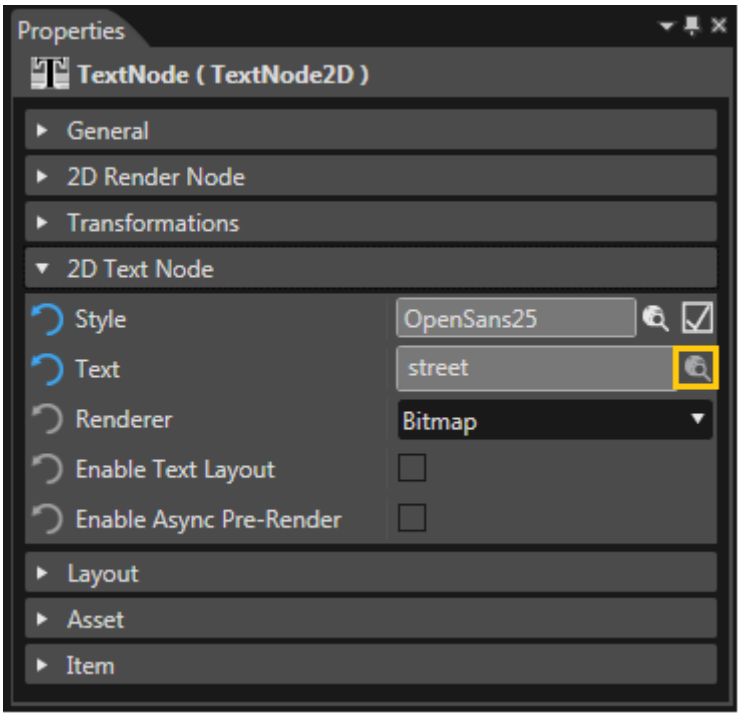
Assigning texts to Text Properties can either be done in the *Language Usage Dialog* or in the *Text Properties Panel*.

Assigning Texts using the Language Usage Dialog

In the Language Usage Dialog a Text property can be selected. When pressing the *Assign* button on the bottom of the dialog, the *Language Table Selection* dialog is opened where a text can be chosen from the list.

Assigning Texts using the Text Properties Panel

The *Language Table Selection* dialog can also be opened when editing a Text property by clicking on the icon *Select template for item*.



Switching Cultures

If cultures and texts have been created and the texts are assigned to Text properties, the culture can be switched using the drop-down in the toolbar. As soon as another culture is chosen, the solution will be displayed in the other language.

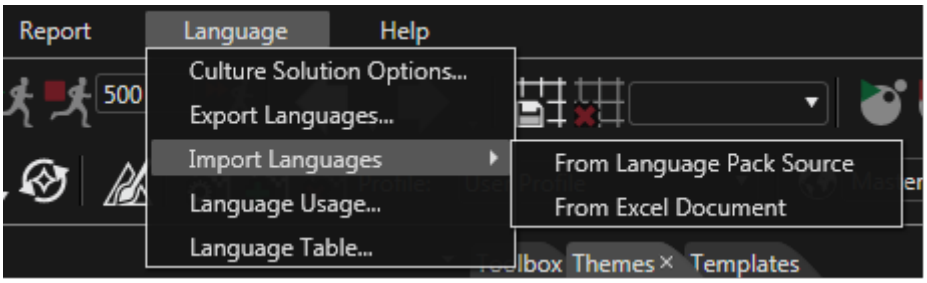
> If there is no text configured for a certain language, the *Text Id* will be displayed instead. Using this id, the missing text can easily be identified and found in the Language Table using the search function.



Import and Export of Language Packs

Import of Language Pack Sources

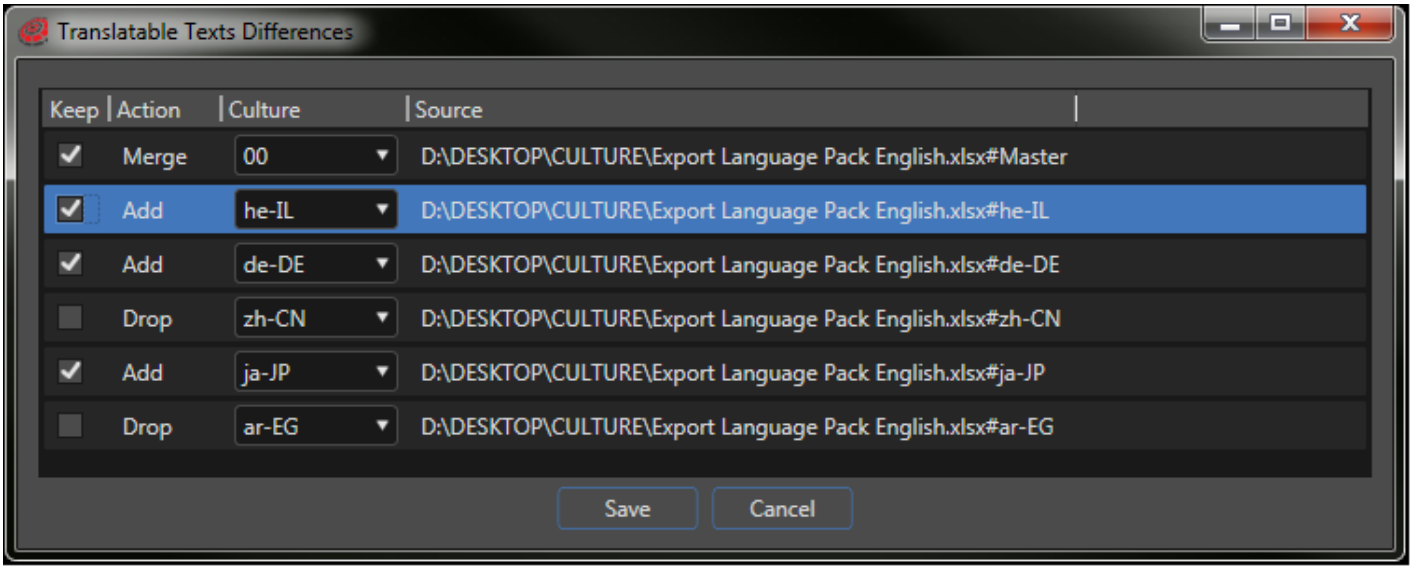
Languages can be imported into a solution either as CGI Language Pack Source or as Excel file, by going to *Language* > *Import Languages* and selecting the desired flavor.



The final step of the language import process is to show the differences between the texts in the import file and the texts in solution. There are three possible actions:

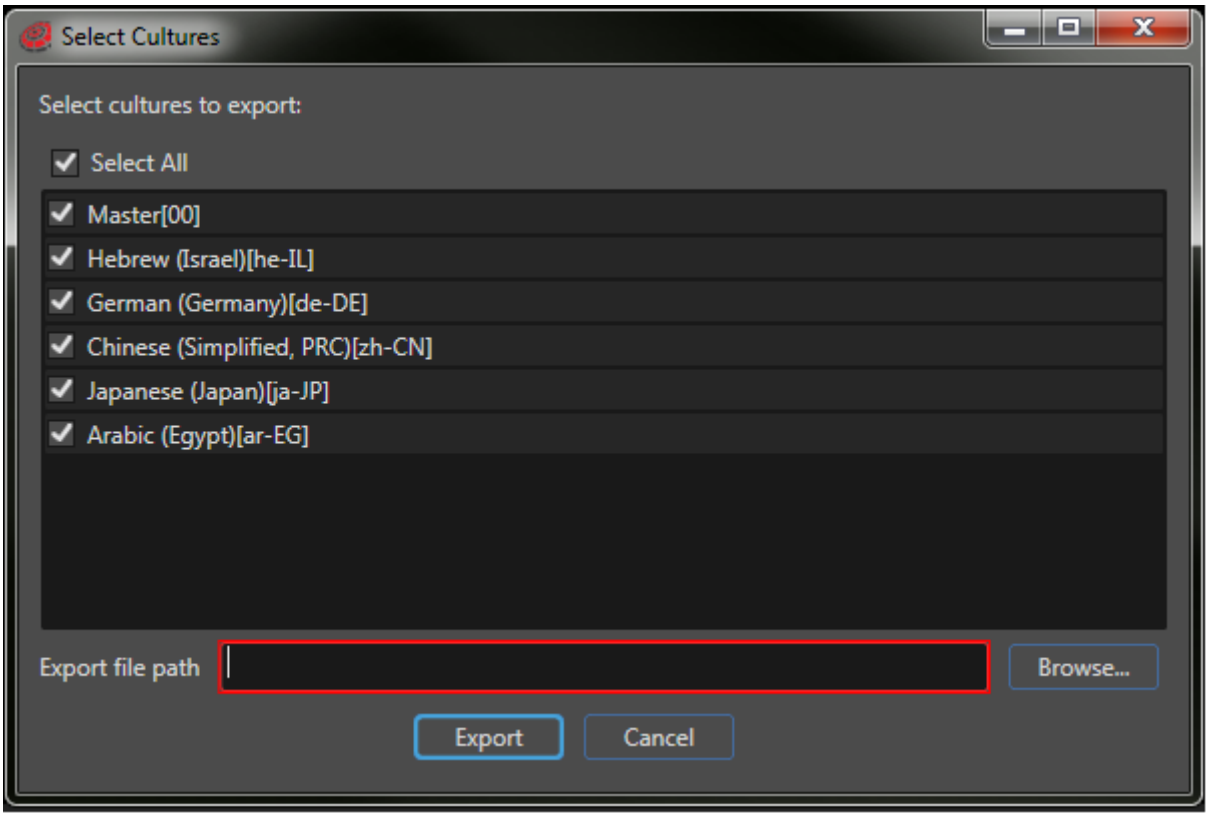
- **Add** : Indicates that the language does not already exist in the solution and will be imported from the file.
- **Merge**: Means that the language already exists in the solution and only the content differs. New texts will be added and already existing texts will be updated.
- **Drop** : When unchecking a language entry in the Translatable Text Differences dialogue, the Action changes automatically from Add/Merge to Drop. If the initial action was *Add*, this means that the language will not be added to the solution. If the initial action was *Merge*, this means its texts will not be updated and new texts from the import files will not be added to this language in the solution.

> The *Drop* action does not mean that the language will be removed from the solution. The removal of a language can be done from *File* > *Solution Options* in the *Cultures* tab.



Export of Language Pack Sources

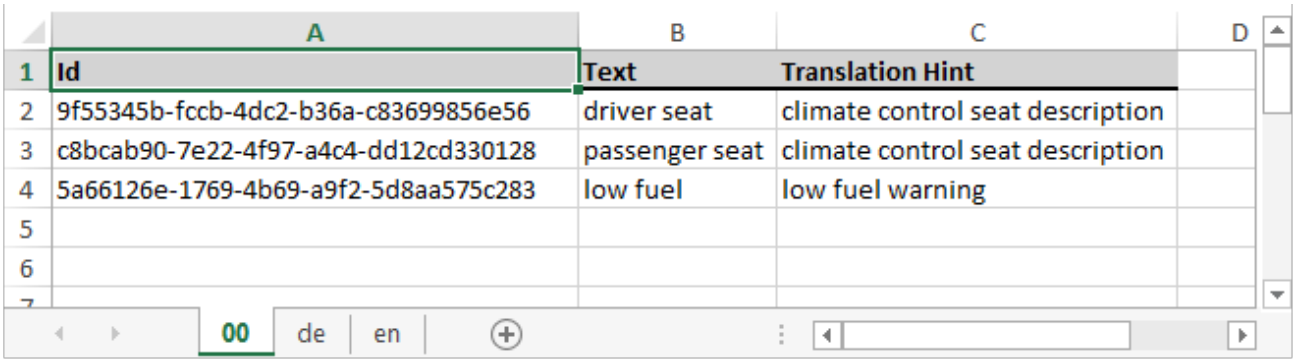
The translatable texts of the current solution can be exported by going to *Language* > *Export Languages...*, which opens the *Select Cultures* dialog.



The *Select Cultures* dialog allows the choice of which cultures to be exported or not. Languages can be exported in Excel or CGI Language Pack Source (.LPS). Click the *Browse* button to select the *Export file path*, choose the format in the *Save as type* drop down and click *Save*. Click the *Export* button to finalize the languages export.

Please note that only the texts that have a correspondent in the master language texts, considering their Id, are included in the export, all the others being ignored. In this case warnings will be added to the Output tab in Scene Composer.

- Exporting languages as Language Pack Source file creates a .LPS file that contains the master language and all the other languages available in the solution. The exported .LPS file can be compiled and appended using the CGI Studio Language Appender.
- Exporting languages as Excel document creates an Excel file that contains one sheet for the master language (named Master) and a sheet for each of the other languages in the solution having an identical name with the respective culture (e.g. de-DE, en-US, fr-FR). Each sheet has a header on the first row, with 3 columns: *Id*, *Text*, and *Translation Hint*. The following rows contain the translatable texts.



	A	B	C	D
1	**Id**	**Text**	**Translation Hint**	
2	9f55345b-fccb-4dc2-b36a-c83699856e56	driver seat	climate control seat description	
3	c8bcab90-7e22-4f97-a4c4-dd12cd330128	passenger seat	climate control seat description	
4	5a66126e-1769-4b69-a9f2-5d8aa575c283	low fuel	low fuel warning	
5				
6				

00 de en

Managing Language Packs using AssetTool

Overview

AssetTool application can be used to manage language packs for a given asset. *AssetTool* application is located in */cgi_studio_bin/AssetTool/* folder and provides the following functionality:

- Appending a new language pack to an asset
- Removing an existing language pack from an asset

A usage example of *AssetTool* application for managing language packs can be seen in */cgi_studio_player/content/Tutorials/05_Globalization/AppendMultiLanguagePack.bat* script.

Appending a New Language Pack to an Asset

For the example solution, the following command can be called for appending language pack *MultiLanguagePack_Source.lps* to the existing asset *MasterLanguagePack_Asset_Host.bin*:

```
AssetTool.exe -append -a MasterLanguagePack_Asset_Host.bin MultiLanguagePack_Asset_Host.bin Mult
```

This will result in a new generated asset called *MultiLanguagePack_Asset_Host.bin*, which will contain the translation for all languages available in the language pack.

Removing an Existing Language Pack from an Asset

For the example solution, the following command can be called for removing master language, with code "00", from the existing asset *MasterLanguagePack_Asset_Host.bin*:

```
AssetTool.exe -append -r MasterLanguagePack_Asset_Host.bin temp.bin 00
```

This will result in a new generated asset called *temp.bin*, which will no longer contain the translation for the master language pack.

For more details on *AssetTool* application, see *Tool Chain/AssetTool* chapter.

Using Globalization in Applications

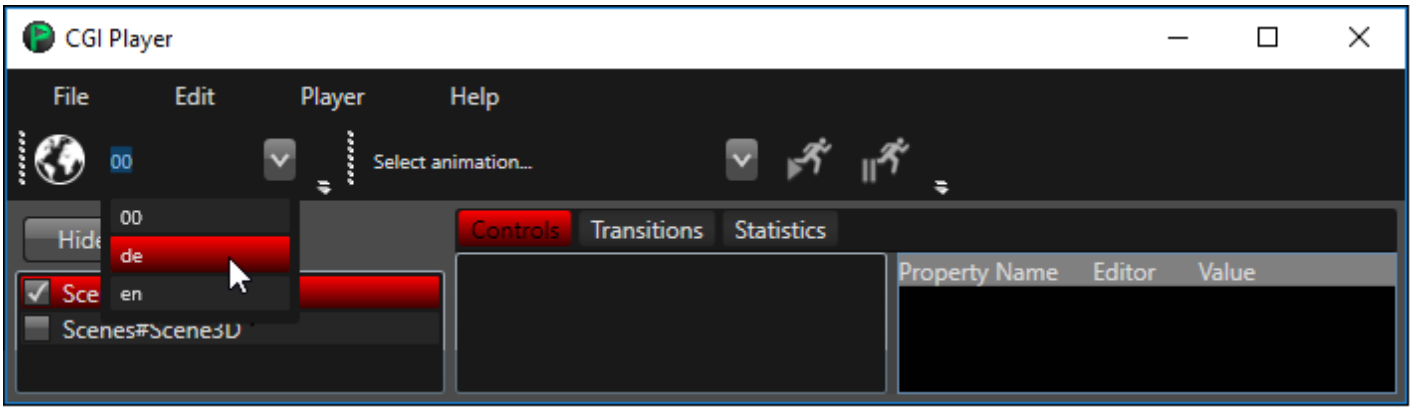
Requirements for Multi Language Support

- CMake setting CANDERA_GLOBALIZATION_ENABLED is enabled
- Sources of CanderaGlobalization available

Example: Set Culture in the Player

The Player provides, if Globalization is enabled in Candera, a checkbox to select all cultures, which are available as language packs in the loaded asset.

1. Open the asset containing language packs (MultiLanguagePack_Asset_Host.bin) and load the scene in the Player.
2. Select the desired language in the drop down list next to the globe icon in the top left corner



Candera Globalization API

The following table explains the usage of Globalization interfaces and how they are used in application code.

Candera::Globalization::Culture	Represents a language, region, and text direction • Locale: ASCII encoded character array, e.g. de_AT. Use Candera::Globalization::Culture::GetLocale() • Display name: Platform-encoded character array (usually UTF-8), e.g. "German (Austria)". Use Candera::Globalization::Culture::GetDisplayName() • Text direction: Enum describing text direction for culture. Use Candera::Globalization::Culture::GetTextDirection()

Candera::Globalization::CultureManager	Access to cultures and changing of current culture
	Singleton:
	Candera::Globalization::CultureManager::GetInstance()
	Retrieve available cultures:
	`return CultureManager::GetInstance().GetCu`
	Get current culture:
	`Culture::SharedPointer culture = CultureMa`
	Get culture name:
	`Culture::SharedPointer culture = CultureMa`
	`return culture->GetDisplayName();`
	Set new culture:
	`Culture::SharedPointer culture = CultureMa`
	`FEATSTD_LOG_INFO(" -> Select culture '%s'"`
	`rc = CultureManager::GetInstance().SetCurr`
	• Candera::Globalization::CultureManager::GetCulture(const Char* locale) const - Retrieval of specific culture by locale

Candera::Globalization::CultureChangeListener	Base class for listeners to culture change events
	Notification of CultureChangeEvents
	• Listener has to derive from Candera::Globalization::CultureChangeListener
	  ◦ override Candera::Globalization::CultureChangeListener::OnCultureChanged(const Culture& culture) {...} for handler
	• Registration as listener
	  ◦ Candera::Globalization::CultureManager::AddCultureChangeListener(CultureChangeListener* listener)
	• Deregistration of listener
	  ◦ Candera::Globalization::CultureManager::RemoveCultureChangeListener(CultureChangeListener* listener)

FeatStd::String and FeatStd::TextId	Point of access to a text in currently active language
	Non-translatable String
	• Will not change with language
	• Use constructor FeatStd::String::String(const TChar*);
	
	Translatable String
	• Changes with language
	• Text-Id from string value
	  ◦ Use FeatStd::TextId::TextId(const Char* id) to get the computed hash id from a given character id
	  ◦ Use FeatStd::String::String(const TextId& id) to access the text for a given Text-Id
	`FeatStd::String text( FeatStd::TextId ("textField_1_3"));`
	• Direct Text-Id: "#\<HEX_HASH_VALUE\>"
	  ◦ Use FeatStd::String::String(UInt32 id) to access the text for a given <HEX_HASH_VALUE>
	`FeatStd::String text(0x13); // for accessing text refe`

Custom Localizer Support

Requirements for Custom Localizer

In order to use a custom Localizer, a CMake feature switch CANDERA_CUSTOM_LOCALIZER_ENABLED has to be enabled. This flag will disable the usage of DefaultLocalizer which loads LanguagePacks from an asset.

In addition, please ensure that the following are fulfilled:

- Candera Globalization sources available
- CMake setting CANDERA_GLOBALIZATION_ENABLED is enabled

Defining a Custom Localizer

A custom Localizer class (eg: CustomLocalizer) should derive from abstract class Candera::Globalization::Localizer and provide implementations for the pure virtual functions:

- Candera::Globalization::Localizer::SetCurrentCulture
- Candera::Globalization::Localizer::GetTextBaseString
- Candera::Globalization::Localizer::GetTextType

For consistency, use FEATSTD_TYPEDEF_SHARED_POINTER(CLASS) and FEATSTD_SHARED_POINTER_CREATE_DECLARATION() macros to make the class manageable by SharedPointer.

Please note that FEATSTD_SHARED_POINTER_DECLARATION() macro should not be used in this case, because some base class methods will be overwritten and SharedPointer memory management will not work correctly.

Implement corresponding Create() method like in the following snippet:

```
/*****************************************************************************
  Create
 *****************************************************************************/
CustomLocalizer::SharedPointer CustomLocalizer::Create()
{
    return CustomLocalizer::SharedPointer(FEATSTD_NEW(CustomLocalizer));
}
```

Example: Using a Custom Localizer in Applications

Define cultures

Create new cultures and add them to the list of available cultures of Candera::Globalization::GlobalizationCultureManager:

```
    m_cultureManager.RemoveAllCultures();
    for (UInt16 i = 0; i < cultureListSize; ++i) {
        Candera::Globalization::Culture::SharedPointer culture = Culture::Create();
        culture->SetLocale(cultureList[i].locale);
        culture->SetTextDirection(cultureList[i].textDirection);
        culture->SetDisplayName(cultureList[i].locale);
        m_cultureManager.AddCulture(culture);
        m_cultureVct.Add(culture);
    }
```

Set default culture to be used in case no current culture is set. This step is mandatory since the default culture will be used for initialization.

```
        m_cultureManager.SetDefaultCulture(m_cultureVct[0]);
```

Use a custom localizer

Declare a custom localizer:

```
    CustomLocalizer::SharedPointer m_localizer1;
    CustomLocalizer::SharedPointer m_localizer2;
```

Create the custom localizer:

```
    m_localizer1(CustomLocalizer::Create()),
    m_localizer2(CustomLocalizer::Create())
```

After a custom localizer has been created, the new localizer needs to be registered to the CultureManager:

```
    GlobalizationCultureManager::GetInstance().SetLocalizer(m_localizer1);
```

Initialize culture by setting current culture. If current culture is not set, default culture will be used.

```
        m_cultureManager.SetCurrentCulture(m_cultureVct[0]);
```

Switching localizer during run-time

Since only one localizer is allowed to be registered, in order to set a new localizer, the previous one has to be unregistered by calling SetLocalizer(0), like in the following snippet.

```
            GlobalizationCultureManager::GetInstance().SetLocalizer(CustomLocalizer::SharedPoint
            GlobalizationCultureManager::GetInstance().SetLocalizer((toggleLocalizer1) ? m_local
```

Corresponding localizer cultures have to be added to the CultureManager and default culture has to be defined if no current culture is available.