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.

Revision #7
Created 2023-02-22 01:48:31 UTC by Tsuyoshi.Kato
Updated 2026-09-25 05:21:13 UTC by Kanai Tomoaki