Features

This section focuses on the “Features” folder. The features folder contains a broad feature set of how to handle framework related content. Every feature is mainly split into two different sections. The first one is the data gathering, the second one is influencing the application.
The main focus for an application developer lies within the second part: How is it possible to change the current state of the framework – for example showing a scene.
For an application developer the first part data gathering might be interesting but not required. The Player application does that because it has no knowledge of the asset. A custom made application does have this knowledge. So for example finding out which scenes exist is not required. Also reading out all behaviors or controls is not very likely to be required. The data gathering is an expensive generic approach and should be avoided.
Out of these features, the interface for external access is generated. For example, the CGI Panel or command line access will use this interface.

General Workflow

The implemented features have a general workflow which all of the features are using.

Controller Messages

Every feature receives controller messages. This is for example required for response, key or touch messages. Most messages sent by features like SetCultureReqMsg or ViewReqMsg are control messages. They control the framework.

Update System

Every feature is registered within the update system. Therefore, at a controlled time with in a message loop, the update function will be called synchronously to the render thread.

Render Thread Scheduler

Additionally, every feature has access to a render thread scheduler. The idea behind it is basic. The scheduler executes its tasks by the update system, too. External input comes asynchronously into the application. Most modern applications handle these by threads. The input has to be delegated into the render thread. Whilst messages are thread-safe, not all content is. Especially the “gathering data” phase is out of ordinary application behavior and therefore does not provide a thread-safe access. As this is very error prone, too. Especially in more dynamic applications as the validity of the data may get lost.

However this application is using this information excessively and brings this information into a thread-safe environment.

By using the asynchronous task mechanism functions which gather the information are scheduled. As soon as the results are available, they will be stored locally and prepared for usage.

This produces a slight overhead within the render thread. For performance reasons these calls should be limited to a minimum in real applications.

Asset Loading

Loading an asset is one of the most important steps application wise. It is possible to create a scene graph by hand but doing that nowhere near the speed, quality, fail-safety and vast content size of an asset designed and generated by Scene Composer.

A customized application has normally one approach of adding asset repositories. This may be a simple memory mapped asset, file or a partitioned asset. These are the standard ways to load an asset and are all supported by the implementation.

Display Creation

Display creation is the most complex part within the core parts of all applications. The given display feature has a very generic way to load all displays out of the asset. It has target specific functionality which will be called to further control the display creation. It is also the location to bind window specific input handling (e.g. touch) to the application.

Logging

Logging is a very powerful tool for development purposes. It has a huge performance impact and should be avoided in production but during development it is a must have. Even if logging happens within the framework and custom application, it can still be controlled. Log levels can be set on different log realms and logging can be completely disabled.

Note:

> Disabling the logging at runtime prevents the outputs. Disabling logging in CMake removes all the logging logic including the extraction of data required for the specific logging messages.

Courier

The courier feature control changes some of courier settings or makes access to views easier.

One of the courier settings is enabling or disabling the render wakeup mechanism. Switching between these options at runtime may not be required at all within a custom application. However, as a playground application this may be useful.

Scene Control

The features before are controlling or creating the core components of any application. These affect the whole framework. The next step is to go into the more specific features which have a specific purpose.

The scene control is the most important one as it brings life into an application. Mainly, it provides one possibility of how to show and hide scenes. Most of the content is related to extract the scenes from the asset and generate a scene cache for the application.

Other Feature Controls

There are more of these controls which are focusing on a small part of the framework. These are listed below:

- Animations
- Controls
- Culture
- Monitor
- Screenshots
- Scripting
- Statistics
- Transitions

Revision #1
Created 2023-02-17 05:58:48 UTC by Tsuyoshi.Kato
Updated 2023-06-30 08:35:18 UTC by Tsuyoshi.Kato