Data Acquisition Overview

The monitoring system provides several different ways to collect and analyze measured data:

- Offline collection
- Online collection
- Analysis at runtime
- Offline-Analysis of collected data

There are two different types of data which can be gathered:

- Scene Graph related data
- Performance based data (Recordings) The Scene Graph related data grabs the actual Scene Graph. This is only available during an established connection between Analyzer and target application.

The performance based data provides key performance indicators which gives information about the current performance on target application. Therefore, it is also used during an established connection. The performance based data also provides recordings which can be gathered during runtime of the application. These data are meant to be analyzed afterwards.

Performance Data Recording

In order to record performance data to be analyzed, the following means are provided:

Dumping to Local Files

Candera supports dumping of performance data to local files on the target system.
Refer to class

- <FEATSTD-ROOT> \featstd\src\FeatStd\Monitor\PerformanceRecording\FileDumper.h.

The Player part of CGI Studio Releases is preconfigured to use this class for periodically dumping performance logs to the local file system, i.e. to the same location the asset is being loaded from. It is only required to enable performance recording in the CGIApplication via the respective macro call during application initialization:

```
CANDERA_PERF_SET_ENABLED(true);
```

The generated performance log files need to be transferred to the host system running Analyzer manually.

Online Transfer of Performance Data

As an alternative to local file dumping, Analyzer can be used to fetch performance logs via a TCP/IP or Serial connection, which is also used for running online experiments on the target. This way, a performance log can be stored already on the host system running Analyzer for further analysis.

> For online transfer, there is a buffer limit of 468,75 kB of data, so only the first 468,75 kB of data will be covered in the performance log.

For setting up the online connection between Analyzer and the target application, refer to section Analysis for more details. The Candera application needs to be in Paused Mode to get a log file via the "Online" tab of Analyzer. Note that "Enable Recording" flag must be enabled.

Dumping to Memory

Analyzer supports dumping performance logs to memory (e.g. where no file system is available). Usage for recording is similar to dumping to local files.

Refer to class:

- <FEATSTD-ROOT> \featstd\src\FeatStd\Monitor\PerformanceRecording\MemoryDumper.h.

Performance Logs have to be transferred from memory on target to the host computer for usage in Analyzer. The transfer from target memory to a host file is target specific. With the INTEGRITY operating system, Multi can be used for this task.

Establishing connection

A connection can be established with selecting a communication type, configuring it and start the connection. Both, target and host, have to be connected by this communication type. It is not restricted which system (target or host) has to be started first.

Currently provided online communication types are:

- TcpIp
- Serial port

Target Side:

To use the monitor on target side the following cmake flag is required:

```
FEATSTD_MONITOR_TYPE
```

The default value is

```
 FEATSTD_COM_TYPE_NONE
```

which disables monitoring completely.

Depending on the chosen value additional settings will be added to cmake:

TcpIp:

```
FEATSTD_MONITOR_TCPIP_ADDRESS
FEATSTD_MONITOR_TCPIP_PORT
FEATSTD_MONITOR_TCPIP_STACK_ENABLED
```

These settings are used to connect to a specific ip-address and a specific port.

Serial Port:

```
FEATSTD_MONITOR_SERIALPORT_ADDRESS
FEATSTD_MONITOR_SERIALPORT_BAUDRATE
```

The baudrate supports common baudrates whereas it is not guaranteed that the target fully supports the given baudrates.

When Player is used as simulation environment, it also asks for these settings. The Player does not know which communication type is in use, so it provides both. Selecting a different communication type as the application uses leads to errors and should be avoided. Nevertheless choosing new settings for given communication type is possible.

Analyzer Side:

To connect the analyzer to a target or another simulation environment the right module has to be selected in the settings of analyzer. To reach this settings view: Ribbon-Tab: General -> Settings -> Monitoring -> Module or Ribbon-Tab: Online -> Select Com Type Select the desired communication type and as module Scene Analyzer.

To establish the connection click the Button start at Ribbon-Tab Online.

Offline Performance Data Analysis

Once a performance log is opened via Open menu of Analyzer, features in the "Performance Log" tab of Analyzer are enabled.

Please refer to section Measurements for further details.

Usage of different Candera mechanisms

Offline Performance Data Analysis

The Analyzer supports different concepts of rendering. The two main concepts are a cyclic rendering concept and the render wakeup mechanism of courier applications. Last one triggers rendering only when graphical changes happened. Both of these concepts work with Analyzer, but there are some logical restrictions. For example: A cyclic rendering mode sends updates to the Analyzer on a regular base. When these updates stop for a certain time, the Analyzer can assume that the communication has troubles. The render wakeup mechanism on the other hand produces these update stops intentionally.

Therefore, handling render wakeup mechanism differs from cyclic concept: All updates and requests are only handled when rendering is triggered. Timeouts should be disabled which disables the earlier mentioned communication trouble checking. Enabling timeouts do not necessarily disconnect the communication but disables some features(e.g. pause) until the application triggers rendering process again. This includes KPI-Updates and Pause requests. Pressing pause results in a waiting phase until the rendering process is triggered. The pause mode works as usual.

Multithreaded performance logs

Performance recording supports multithreaded recordings. To enable this feature, use the recordings/recorder equal to the single threaded variant. There is no additional code needed by default. However, there are some restrictions and helpful macros which should be kept in mind to avoid unwanted behavior. Normally, these are not required to use this feature. They should be used when displayed results in the analyzer are not expected.

Synchronization of threads

It is advisable to synchronize the threads when writing the data from the performance log buffer to any destination (e.g. filedump, memorydump, data streaming). This prevents unstable data within the buffer as the other threads continue running even in the paused mode of an online connection with the analyzer tool. There are two solutions to synchronize the threads.

```
#include <Candera/System/Monitor/PerfMonPublicIF.h>
…
#define CANDERA_PERF_COLLECT_RECORDING_THREAD_SYNC_BARRIER
#define CANDERA_PERF_PRODUCE_RECORDING_THREAD_SYNC_BARRIER
```

The first macro should be used in the thread which dumps data. The macro MONITOR_HANDLE_PAUSED does that internally, so using the analyzer tool for a live connection

does not require the call of the first macro. The second macro should be called in all the other threads which are using performance recordings. The second solution is to use a custom synchronization method of all threads.

> It is also advised to add the synchronization barrier outside of the top level recording of a thread. This prevents influences of nested recordings on the current dump but also on an additional dump – as the buffer will be cleared even if there are recordings which are not done recording yet.

Restrictions

Some of the recorder types have to store information of a recording temporary until the recording itself has been completed. It is advisable to use own recorder (own ID and/or name) for each thread. Typical issues can occur with AsyncTimingRecorder and cumulative recorders as their data may be written and corrupted by multiple threads.

> In most cases using one recorder in multiple threads will not produce issues but it is not guaranteed to do so.

Revision #6
Created 2023-02-16 02:24:55 UTC by Tsuyoshi.Kato
Updated 2025-01-10 08:20:46 UTC by Elisabeth Zauner