Scene Analyzer Overview

Scene Analyzer feature list

The Analyzer analyzes a Candera Scene Graph and its performance of a 2D or 3D Candera project. It supports:

- Scene Graph View [1]
- Detailed information about the Scene Graph [1]
- Benchmarks related to Scene Graph [1]
- Measurement and data presentation of render processes [2]
- Online – Experiments [1]
- Measurement of key performance indicators [1]
- Storage of measured data

[1] Features available during an online connection with appropriate targets
[2] Features traced during an online connection with appropriate targets and analyzed afterwards

Measurement Concepts

Introduction

Measurement Concepts

The Analyzer measurement module allows the user to analyze measurements performed and recorded on the host simulation or target.

It is important to differentiate between the concept of a recorder and the concept of a recording. A recorder is a means to record something, as a DVD recorder is a means to record a DVD.

Recorders

The concept of recorders lies at the heart of the Analyzer performance log recording and analysis.

Examples are recorders to record

- Every point in time, a new frame starts (an Event Recorder)
- How long it takes an OpenGL call (e.g. glDrawArray) to return to the caller (a Timing Recorder)
- The current CPU or GPU load (a Value Recorder)

In order to cover these three main areas of interest (events, durations, values), the following six recorder classes were defined:

- Event Recorder
- Timing Recorder
- Asynchronous Timing Recorder
- Cumulative Timing Recorder
- Value Recorder
- Cumulative Value Recorder

Every recorder has the following properties:

- Name: A name unique in the recorders class
- Class: The class to which he belongs

Recorder Classes

Event Recorder

Event Recorders are the simplest recorders. They just record an event at a given point in time.

Timing Recorder

Timing Recorders are used to measure durations. Therefore, begin- and end time of a timing recording may differ.

Asynchronous Timing Recorder

An Asynchronous Timing Recorder is a timing recorder that starts recording in one execution block and may end in a different one.

Cumulative Timing Recorder

A Cumulative Timing Recorder does not keep track of every single timing it was used to record. Instead, it sums up all durations recorded and also keeps track of the number of recordings.

Value Recorder

A Value Recorder is used to record a value at a point in time. These values may be signed or unsigned long values.

Values may have the following relationship:

- Unrelated. Two recorded values in succession have no relationship.
- Differential. Only the difference after the last recording is recorded.
- Absolute. The full value is always recorded.

Example for value relationships:

An application records memory consumption and uses a recorder for this task. The application allocates first 1000kb and then 500kb. At the end, the total (absolute) consumption is 1500kb.

- When using a differential Value Recorder, the application would first record the value 1000 and then the value 500.
- When using an absolute Value Recorder, the application would have to first record 1000 and then 1500 to achieve the same effect.

Cumulative Value Recorder

A Cumulative Value Recorder, similar to a Cumulative Timing Recorder, does not keep track of every value it was used to record, but sums up all values and counts the times something was recorded.

Recordings

A recording represents actual data recorded, e.g. one new frame event recorded at 1000 ms after system startup. Depending on the recorder used, the following is recorded:

- A signed or unsigned long value
- An Event
- A Duration

Separation/Interpretation of Data

Separation of Data and Interpretation Information

In order to gain flexibility, performance log data and the information about how they have to be interpreted and displayed are separated. This separation is reflected on the file system level: performance log data is stored in .perflog files and the meta-information for interpretation is stored in an XML file (PerfDataConfig.xml). The meta-information comprises the following:

- The name to use for display
- The foreground color to use for display
- The background color to use for display
- For value recorders:
  > If the values have to interpreted as signed or unsigned
  > The relationship of the values (Absolute, Differential, Unrelated)
- For cumulative value recorders:
  > If the values have to interpreted as signed or unsigned

The meta-information can be edited using the View Configuration Dialog (see Section View Configuration Dialog).

Views - Filtering - Sampling

Views

A view shows certain aspects of the overall recorded information and thus helps the user to focus on the information he needs to see.

See also:

- Measurement Views

Filtering

Each view except Dashboard View and Treemap View allow filtering to display the desired recordings only. Every filtering-enabled view supports filtering by a means of its own and filtering will be described in the corresponding sections. Filtering is always done before sampling.

Sampling

Sampling is, similar to filtering, a means to reduce the amount of data displayed. A filtering expression can be arbitrary, whereas a sample always contains all (not-filtered) recordings between two given points in time.

What is or is not contained in a sample is defined by the sampling strategy in use. There are currently two sampling strategies available:

- Event Sampling Strategy
- Duration Sampling Strategy

If an event sampling strategy is used, all recordings between two events of the same type belong to the same sample (e.g. all recordings between two occurrences of the "New Frame" event). The duration sampling strategy combines all recordings that are e.g. in the first 1000 ms after recording started into one sample.

See also:

- Sampling

File Filter

File Filter Overview

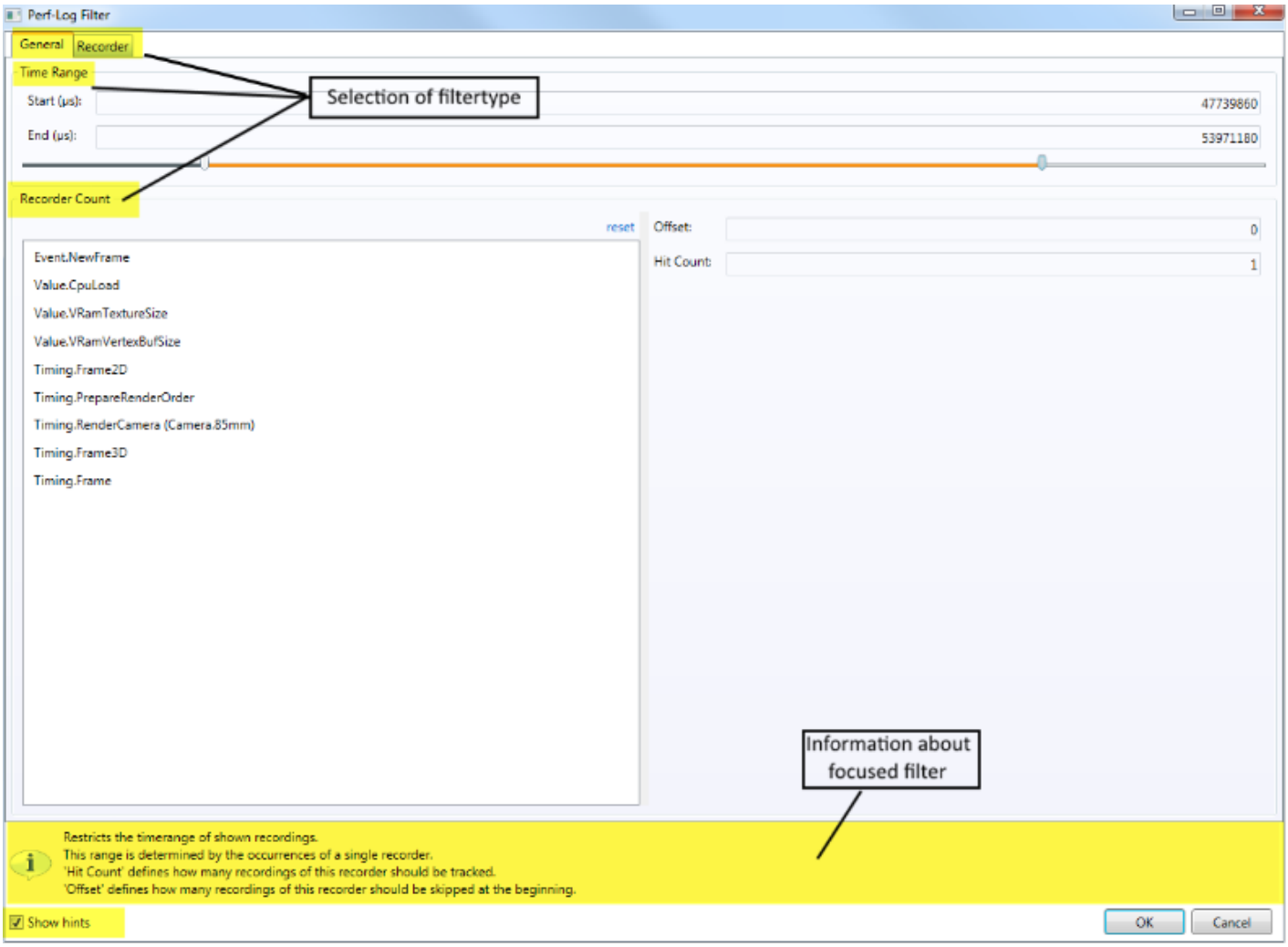
A performance log can contain a huge amount of recordings. More recordings lead to a more detailed information about the own system.
However loading millions of recordings into the Analyzer also leads to a dull application with a barely comprehensible dataset.
A good way to reduce the amount of data is to filter recordings away which are not useful in the current analysis.
This step is done during the loading process of a file and is an optional feature.
In general the loading speed will not be reduced due to the additional filtering effort but the Analyzer will work more smoothly and the data will be concreted to the requested analysis.

Open Filtered File

GUI Overview

The GUI has two tabs which both contain filter. The first tab (General) contains timespan changing filter.
The second tab (Recorder) contains a filter to show or hide recordings.

At the bottom of the window is a information area. It shows a brief overview how a filter is meant to be used.
This information area can be toggled off, so it will not show any more.



Time Range Filter

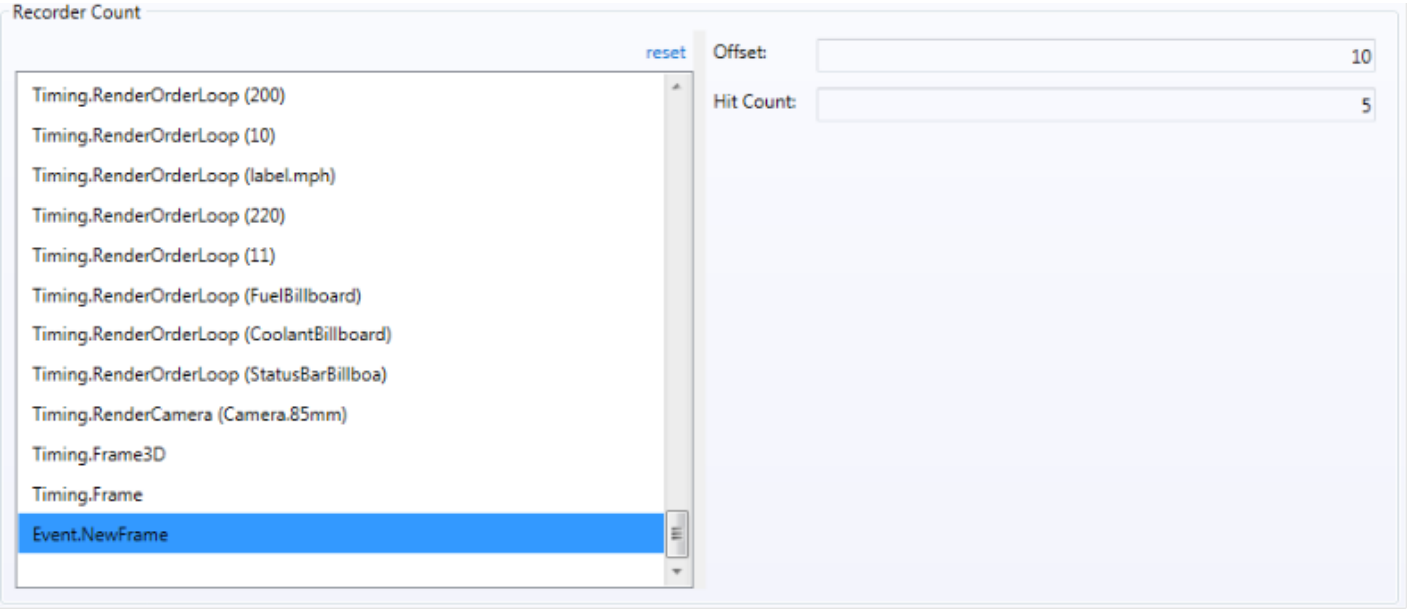
The time range filter is the most basic filter. You select the exact time range which will be loaded. There is a start and an end time. Everything before the start and after the end time will be truncated during loading. This filter applies to all other filter.
Other filter will operate on the remaining subset of recordings.

Recorder Count Filter

The recorder count is a filter which indirectly influences the time range. The concept of this filter is to only show data starting from a specific recording and ending after this recorder type appeared a defined amount of times.
The filter is split a recorder list and two options.
Within the recorder list a recorder can be selected. This recorder is used for the occurrence count. The recorders are listed as annotated recorders. So the selection is limited to a recorder subtype.
The option Offset defines the amount of skipped recordings of the selected recorder at the beginning.

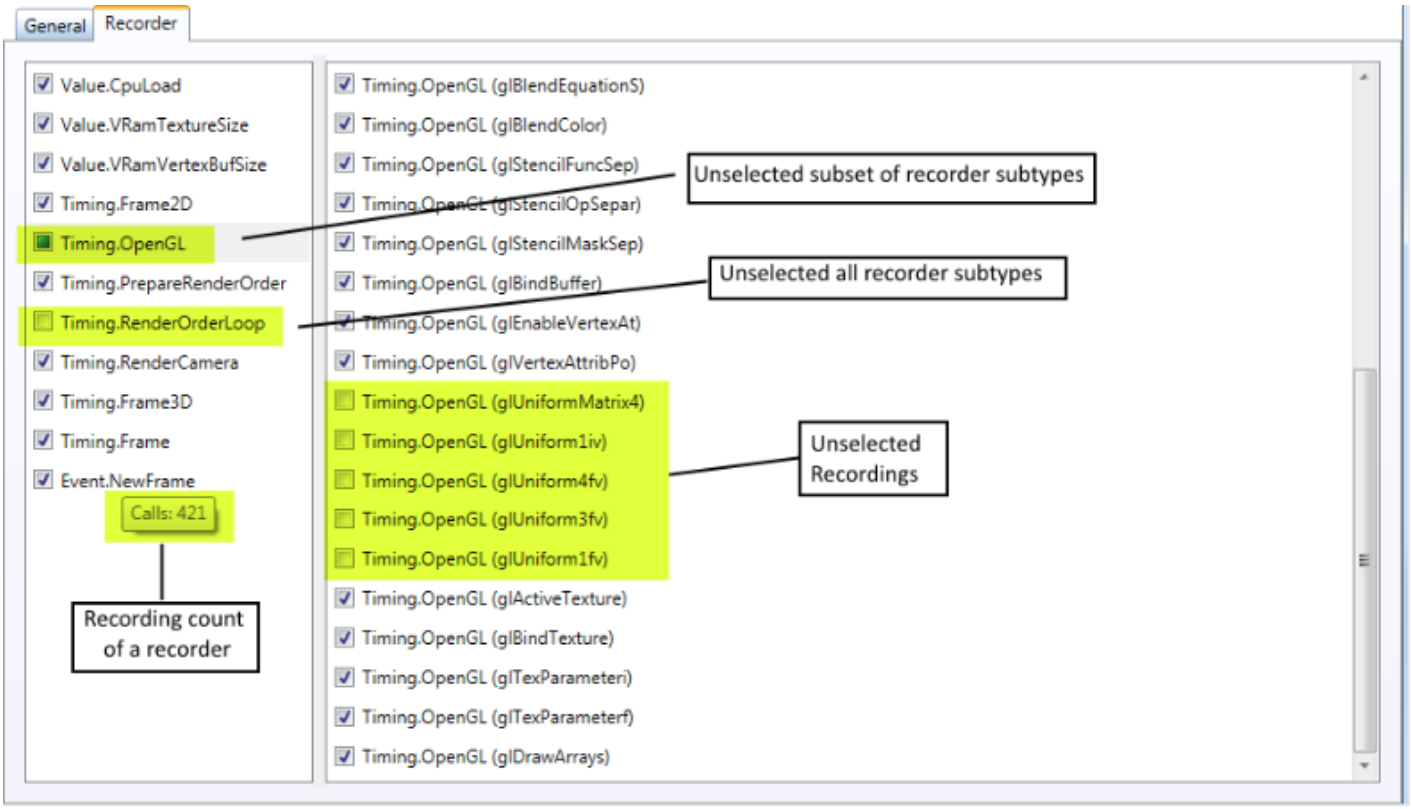
The option Hit Count defines how often the recorder occurs. Afterwards the loading of file stops. When the hit count is reached this last recording will still be loaded.

Typical use case:
Select Event: New Frame
Offset: 10
Hit Count: 5
Result: Skips the first 10 frames; shows 4 frames and the occurrence of the 5th new frame event.



Recorder Count Filter

The recorder selection filter is used to hide specific recorder (sub-)types.
A type is based on recorder type and recorder ID. The subtype of such a recorder takes the annotations into account. The different subtypes or a whole type can be (un-)selected. An unselected recorder will not be loaded. Its nested recordings will be moved to the recording above.
Brief example:
A duration recording (A) contains two occurrences of another duration recording (B).
Recording B contains additional recordings (C). Unselecting the recorder of recording B will hide all occurrences of B.
The additional recordings C will be moved to A as its parent.


General
Recorder
Value.CpuLoad
Value.VRamTextureSize
Value.VRamVertexBufSize
Timing.Frame2D
Timing.OpenGL
Timing.PrepareRenderOrder
Timing.RenderOrderLoop
Timing.RenderCamera
Timing.Frame3D
Timing.Frame
Event.NewFrame
Calls: 421
Recording count of a recorder
Timing.OpenGL (glBlendEquationS)
Timing.OpenGL (glBlendColor)
Timing.OpenGL (glStencilFuncSep)
Timing.OpenGL (glStencilOpSepar)
Timing.OpenGL (glStencilMaskSep)
Timing.OpenGL (glBindBuffer)
Timing.OpenGL (glEnableVertexAt)
Timing.OpenGL (glVertexAttribPo)
Timing.OpenGL (glUniformMatrix4)
Timing.OpenGL (glUniform1iv)
Timing.OpenGL (glUniform4fv)
Timing.OpenGL (glUniform3fv)
Timing.OpenGL (glUniform1fv)
Timing.OpenGL (glActiveTexture)
Timing.OpenGL (glBindTexture)
Timing.OpenGL (glTexParameteri)
Timing.OpenGL (glTexParameterf)
Timing.OpenGL (glDrawArrays)
Unselected subset of recorder subtypes
Unselected all recorder subtypes
Unselected Recordings

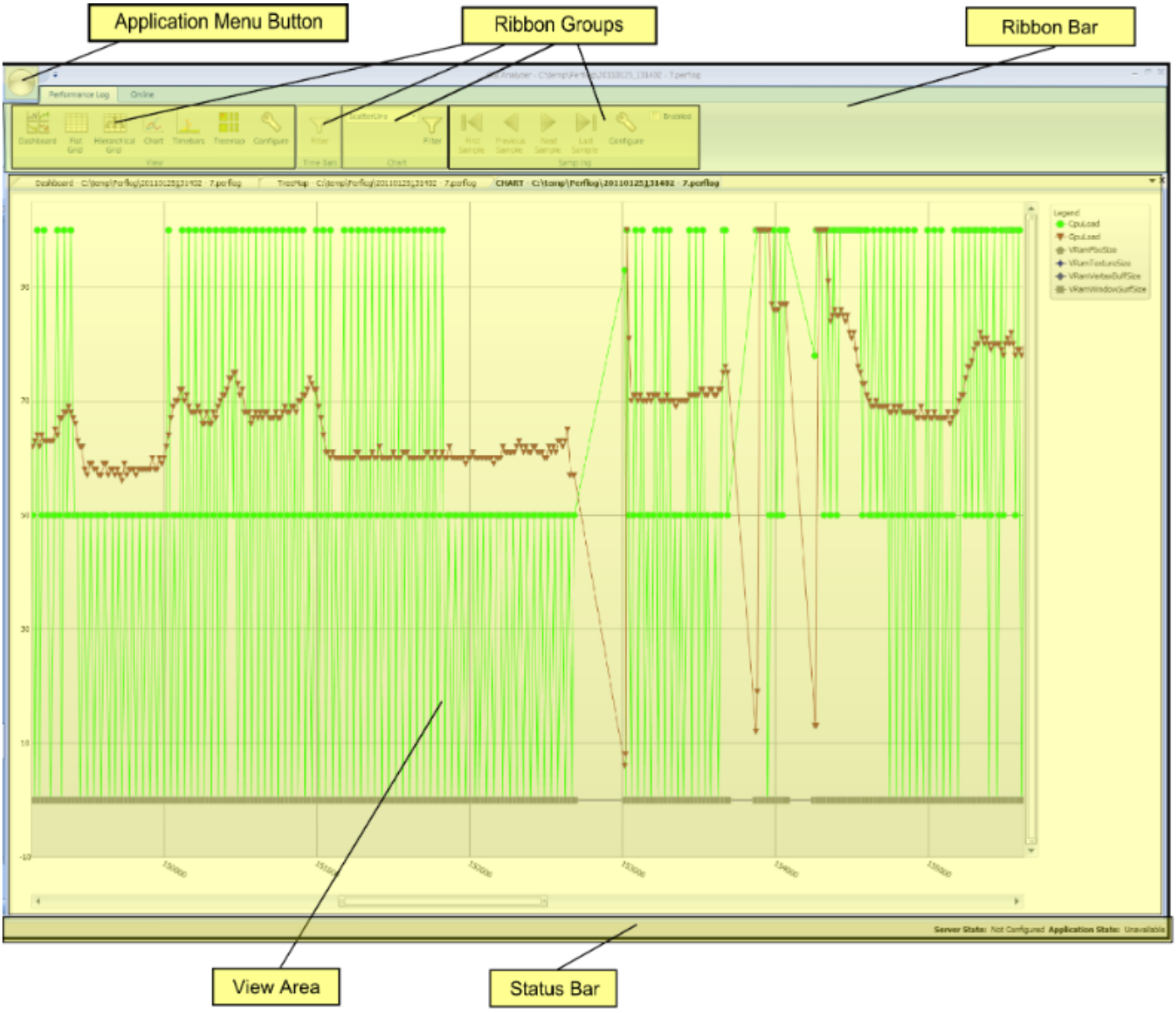
Measurement User Interface

User Interface

The User Interface is divided in 3 areas:

- Ribbon bar
- View area
- Status bar

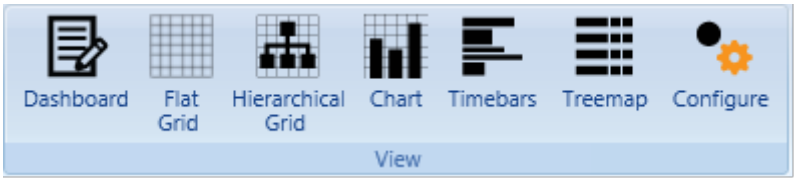
The ribbon bar is subdivided into groups of controls dedicated to a specific task - ribbon groups. The View Ribbon Group for example allows configuring the display of, opening of, and switching between views.



Analyzer allows Performance Logs to be viewed from different vantage points (views). Each view abstracts from the total amount of data and information to allow the user to easier concentrate on and find the information he searches.

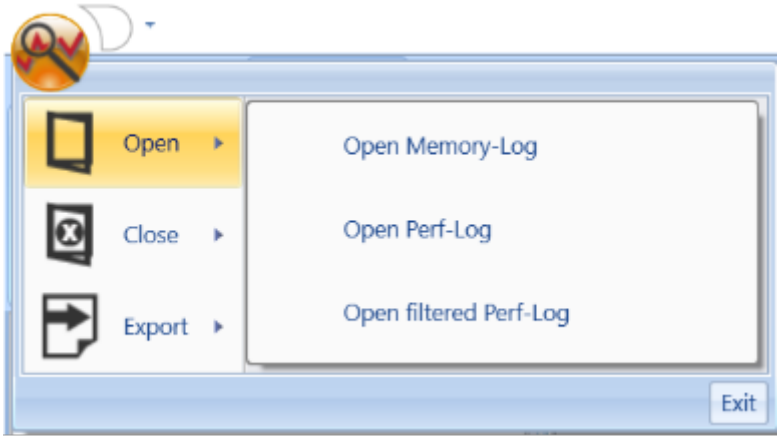
The following views are supported and described in the subsequent sections:

- Performance Dashboard View: An overview of key performance indicators
- Chart View: A view of the trajectories of all value recordings
- Time Bars View: A chronological as well as hierarchical view of all recordings
- Grid Views: Views in table form
  > Flat View: A flat view on all recordings
  > Hierarchical View: A hierarchical (nested) view on all recordings
- Treemap View: A nested view of the timing recorders Access to opening and switching between views, as well as configuration of views is provided by the View Ribbon Group.



Application Menu

The application menu holds commands for opening and closing a performance log, as well as exporting performance log data.



Configuration

The display of performance logs can be configured using the Performance Log Configuration dialog. The dialog can be opened from the "View" Ribbon Group.

The configuration dialog (depicted in Figure 4) allows the configuration of recording appearances in views as well as how the recordings are interpreted.

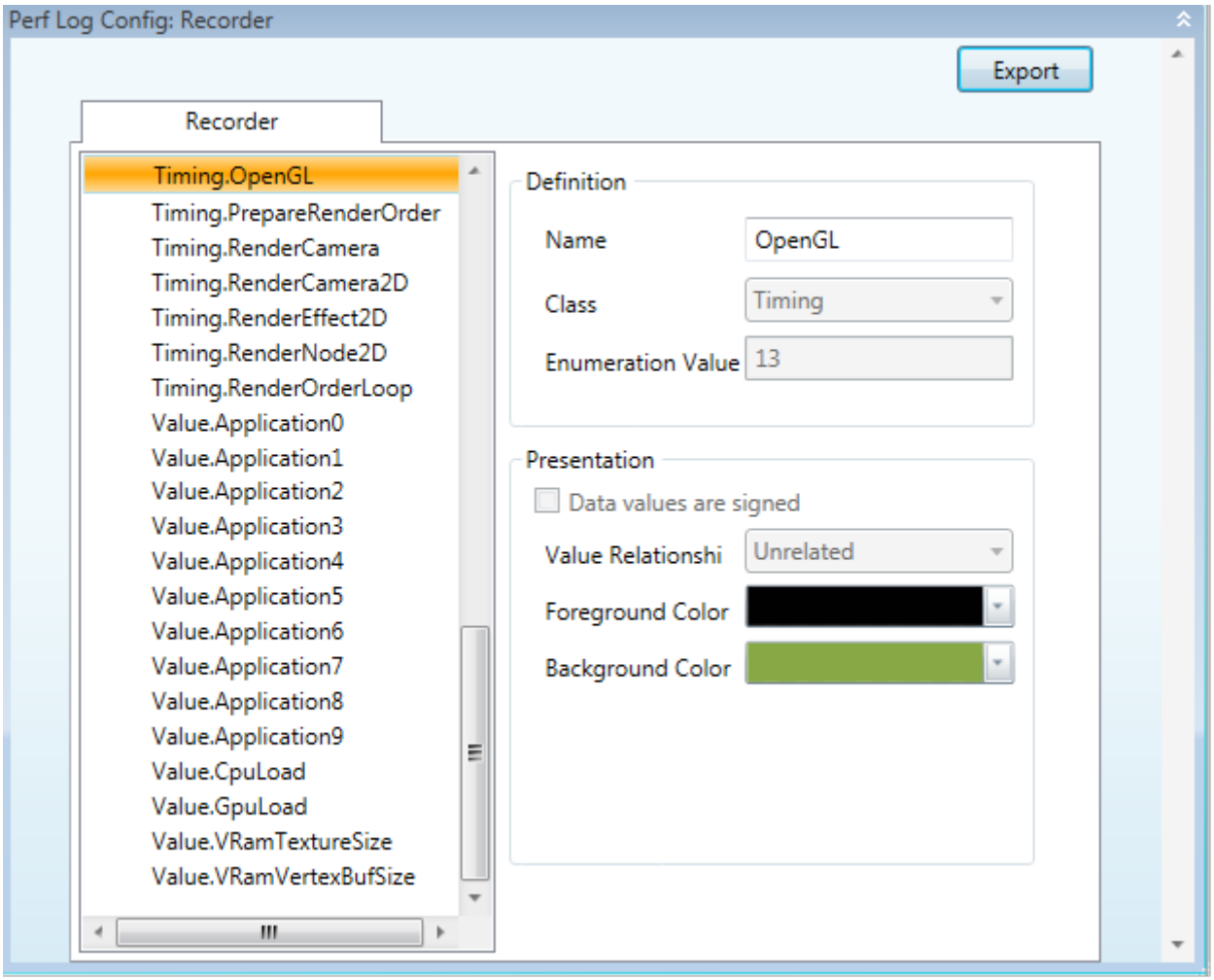
On the left side of the dialog, there is a list of all recorder names used. The recorder names displayed are fully qualified, consisting of the recorder class as prefix and the display name of the recorder. On the right side, the properties of the currently selected recorder are displayed and can be changed.

Changeable for all recorders are the following properties:

- Name: The name of the recorder displayed in all views
- Foreground color: Text color for recordings in Time Bars View
- Background color: Background color for recordings in Time Bars View, Color for trajectory of recordings in Chart View

Changeable properties for value recorders are:

- Data values are signed: The displayed values are treated as signed values
- Value relationship:
  > Unrelated: Recorded values are unrelated
  > Differential: The recorded values are only the difference from the last recorded value, not the total (absolute) value.
  > Absolute: The values recorded are always equal to the absolute values.



Performance Log Export

Export of Performance Log

The export of the currently open performance log is available via the application menu.

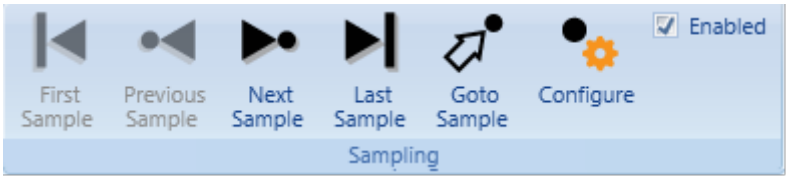
Currently, exporting of performance log data is supported in the following formats:

- Microsoft Excel 2007 (XLSX)
- Comma Separated Values (CSV)

Since Excel worksheets allow only a limited number of rows, data is distributed over several worksheets. Using this approach, a theoretical total number of 65000x255 rows can be stored in one Excel file. The export of data takes some time, so please be patient.

Sampling

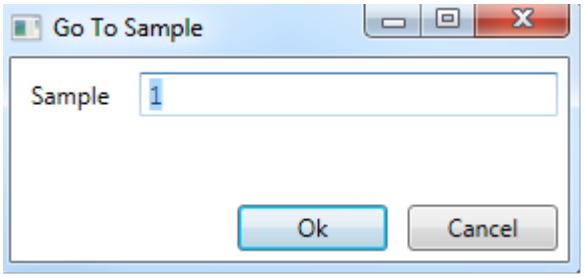
The figure shows the Ribbon Group for sampling. All sampling functionality is available via these controls.



Usage of sampling requires two steps:

- Configuration of sampling for a given view. The sampling configuration dialog can be opened via the "Configure" button.
- Enabling of sampling for the view. Sampling can be enabled via the "enable" checkbox

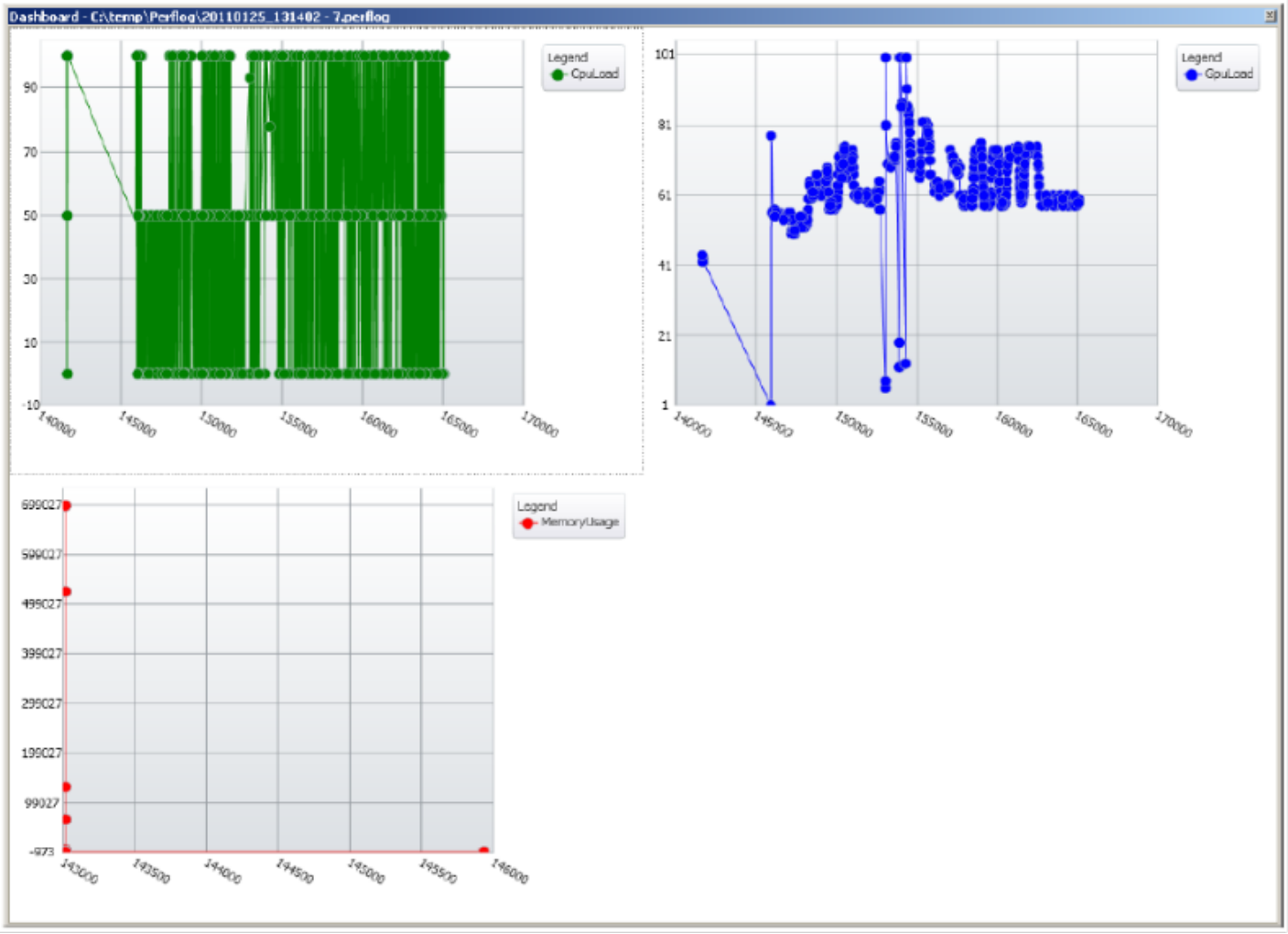
Navigation between samples is supported by the navigation buttons in the Sampling Ribbon Group or keyboard shortcuts (see the end of this document for a list).
Also, the user can directly jump to a desired sample by pressing the "Goto Sample" button and specifying the desired sample in the provided dialog.



Measurement Views

Performance Dashboard View

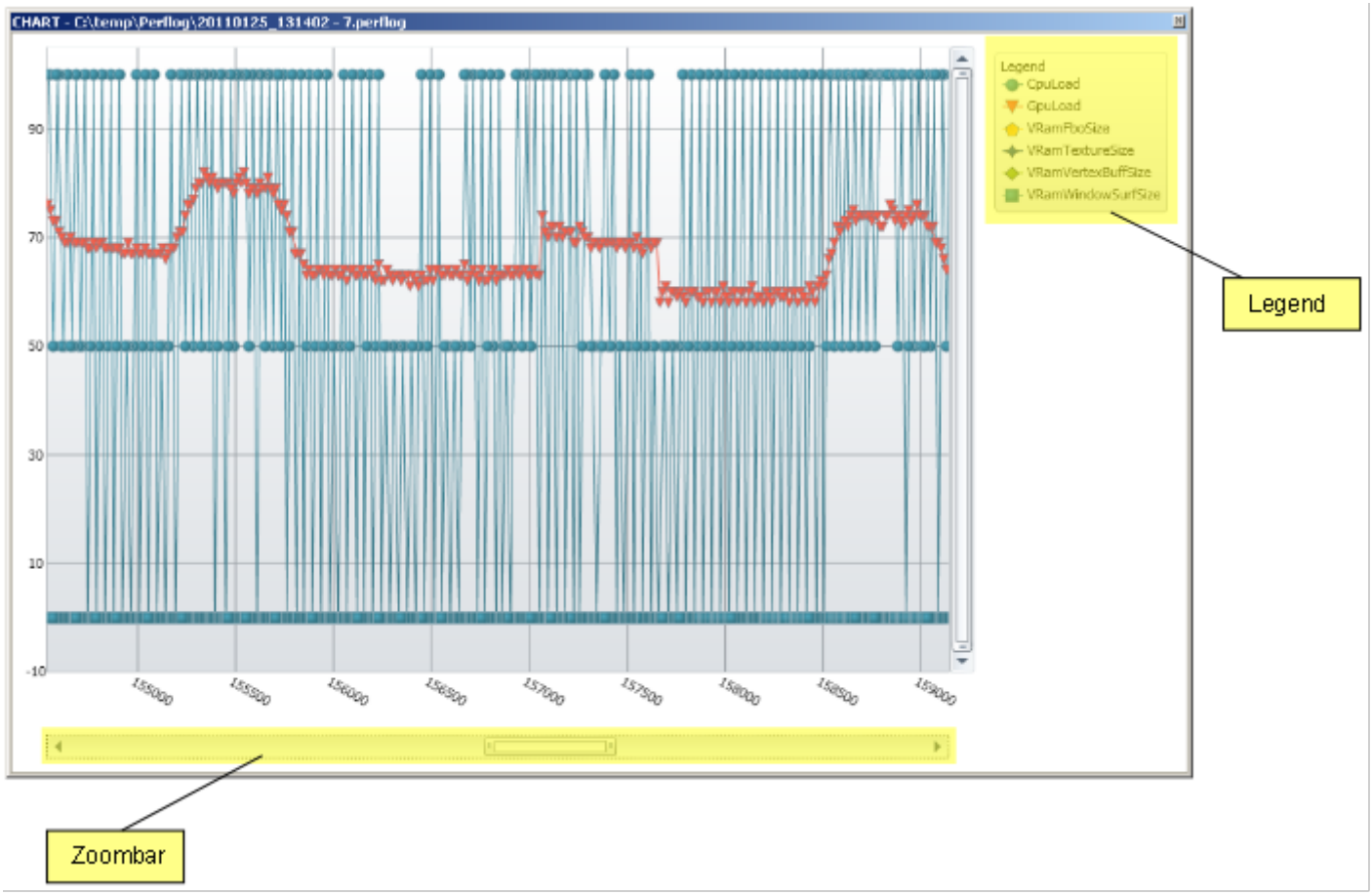
The Performance Dashboard View is the first view, automatically displayed when opening a performance log. It displays recordings of up to four Value Recorders. The actual Value Recorders used here can be customized. After installation, however, CPU load, GPU load and memory allocations are displayed (if available). The figure shows a typical Dashboard View.



For every chart, the x-axis represents the elapsed time and the y-axis a value scale. To the right of each chart, there is a legend for the values displayed in this chart. There are no interactions possible with this view type. Only the values are displayed upon hovering over graphs.

Chart View

A Chart View, as shown in the figure displays only Value Recordings.



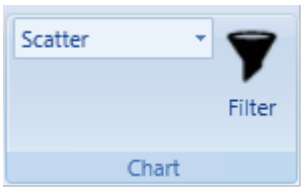
On the x-axis, a time scale is displayed, on the y-axis a common scale for all values. To the right there is a legend for the values currently displayed.

Below the x-axis and to the right of the y-axis, there are so-called zoom bars. They allow the user to zoom-in to as well as pan the chart. Panning is also allowed by left-clicking the chart and dragging it to the left or the right.

By pressing the left shift button, clicking the left mouse button, and dragging the mouse, a part of the chart can be selected for zooming in.

Hovering over a trajectory or a marker displays a tooltip with information about the corresponding value.

Associated Ribbon Group



The Chart Ribbon Group comprises UI elements for configuring the currently active Chart View:

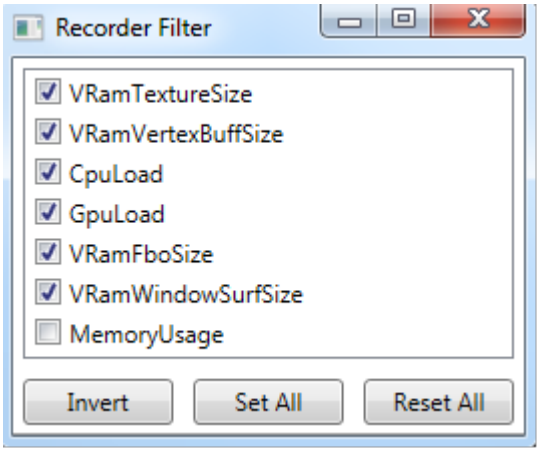
- Filtering is available via the "Filter" button
- Chart interpolation settings are available from the combo box.

The following interpolations are supported:

- Scatter: No interpolation is performed; only markers are displayed
- Scatter Line: Simple line interpolation is performed by connection successive markers
- Scatter Spline: Spline interpolation is used.

Filtering

Filtering allows the user to reduce the set of displayed Value Recorders to a desired subset. The shows the recorder filter editor opened via the "Filter" button in the Chart Ribbon Group.



En- and disabling of filters is immediately reflected in the chart.

- "Invert" inverts the current selection of recorders,
- "Set All" enables all recorders,
- "Reset All" disables all recorders in the current Chart View.

Sampling

Sampling is available via the Sampling Ribbon Group. See section "Filtering" on page Views - Filtering - Sampling for details.

Grid View

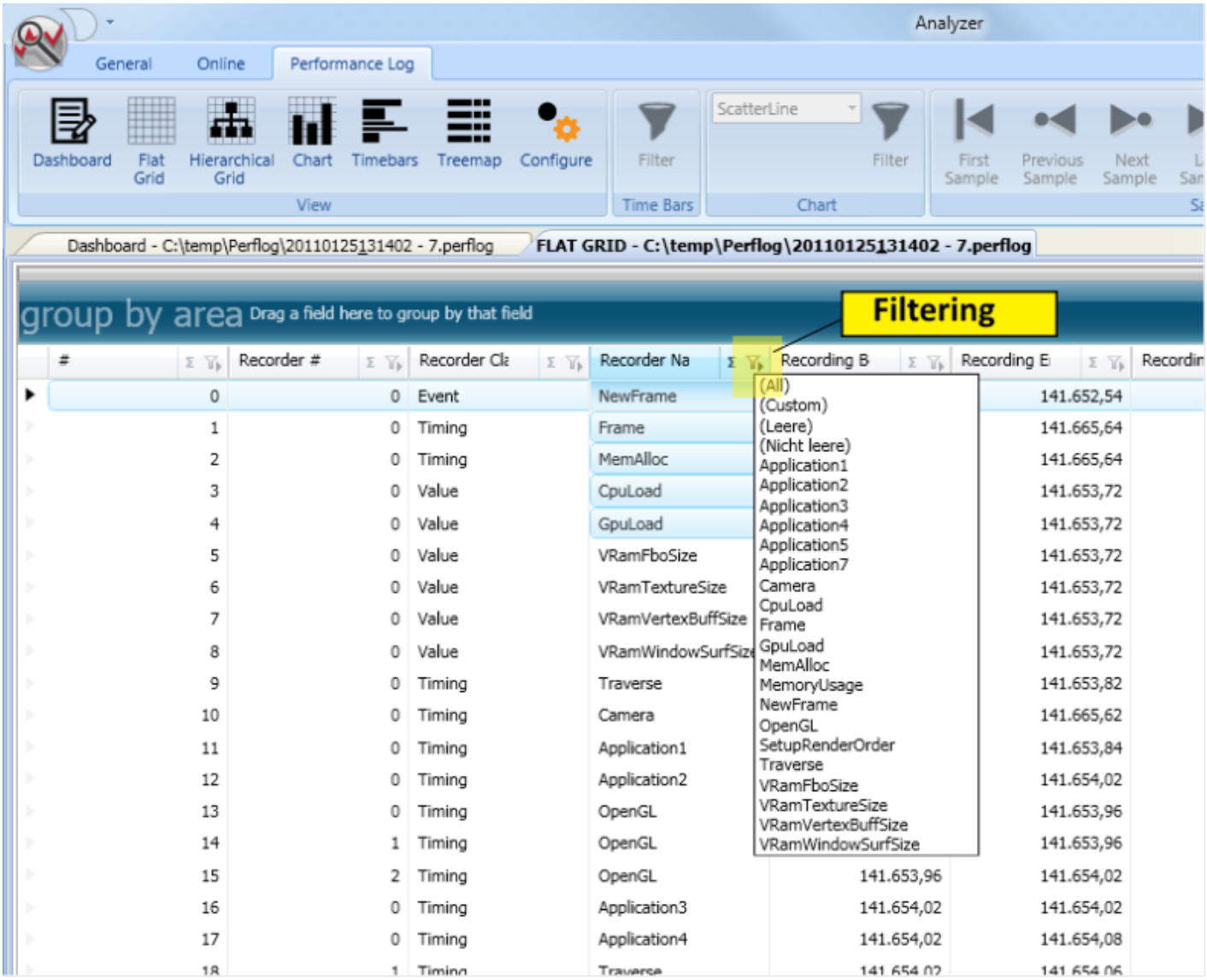
Flat and Hierarchical Grid Views allow display of all recordings in a grid similar to Microsoft Excel. Table 1 lists and explains the displayed columns.

Grid View Columns

Column Header	Title Description
#	Sequence number of the recording. Numbering starts at zero.
Recorder #	Sequence number of the recorder. This is the number of the recorder in the sequence of recorders with same class and name. Numbering starts at zero.
Recorder Class	The class of the recorder. See section 4.1.1 for a list of available recorder classes.
Recorder Name	The unique name of the recorder within its class
Recording Begin	The time, this recording started. Unit is milliseconds.
Recording End	The time, this recording ended. Unit is milliseconds.
Recording Duration	The difference between Recording Begin and Recording End. Unit is milliseconds.
Self Duration	The amount of time, this recording contributed to the recorded duration (if nested). Unit is milliseconds.
Effective Duration	Depending on the recorder class, the effective duration yields the following: • Cumulative Timing Recording: The accumulated duration • Asynchronous Timing Recording: The asynchronous duration • Other recorder classes: The self duration Unit is milliseconds.
Time To Next Recording	Unit is milliseconds.
Annotation	A string enabling further differentiation between Recordings with same class and same name. The OpenGL timing recorder uses annotations e.g. to differentiate between calls of different functions.
Call Count	Depending on the recorder class, the call count yields the following: • Cumulative Timing Recorder: The number of recordings of this recorder • Cumulative Value Recorder: The number of recordings of this recorder • Other recorder classes: No value
Value	Depending on the recorder class, the value yields the following: • Value Recorder The value • Otherwise no value
Absolute Value	Depending on the recorder class, the absolute value yields the following: • Cumulative Value Recorder The absolute (up until then cumulated) value • Other recorder classes no value

Filtering

Filtering in the grid views is available via the grid column headers as depicted in Figure 11. The filtering options are:



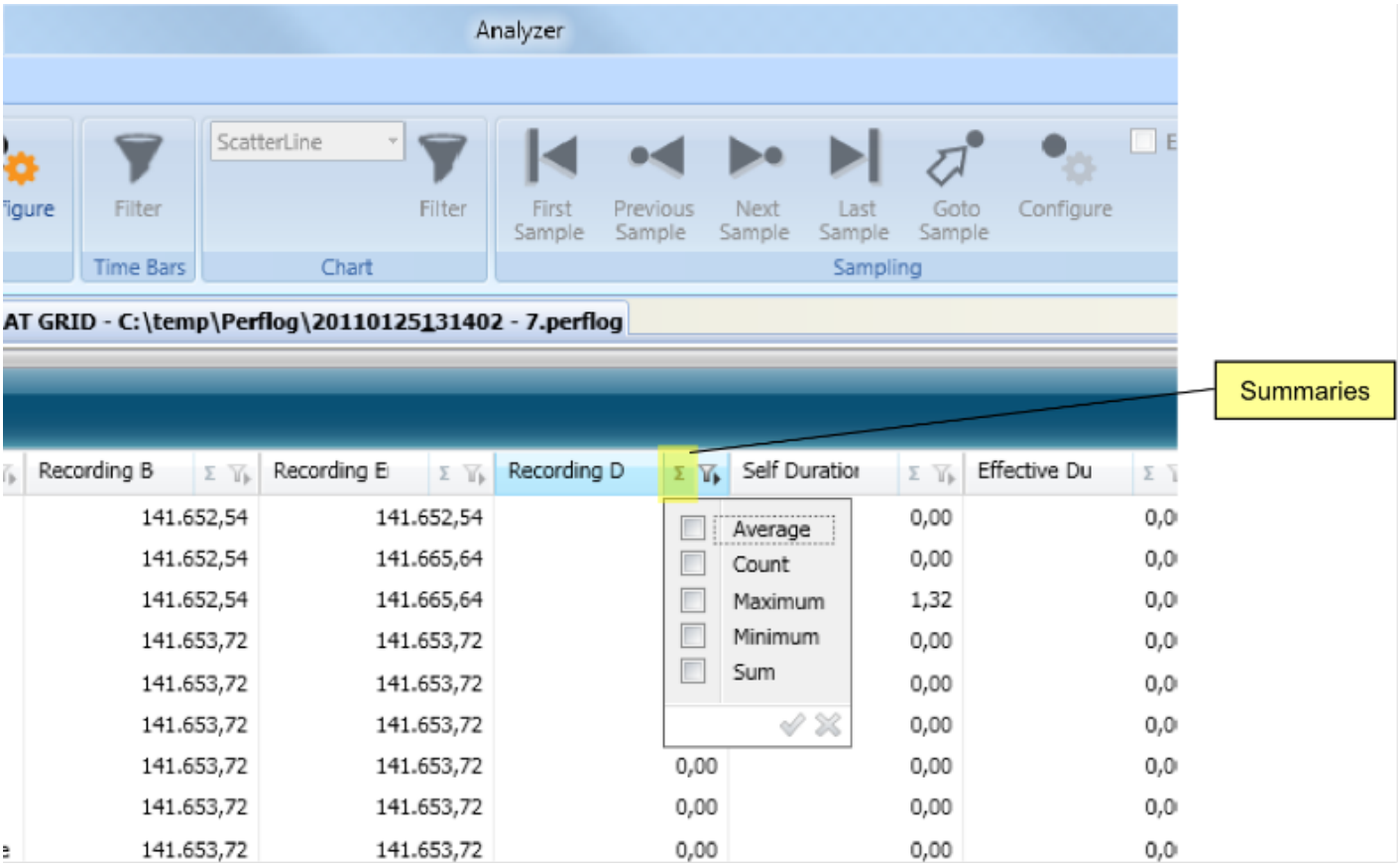
Grid Filtering Options

Filter Option	Description	Only Numeric
All	Displays all recordings	No
Custom	Displays a dialog for constructing a custom filter expression	No
Empty	Display all recordings that contain no value in this column	No
Not Empty	Display all recordings that contain a value	No
Below Average	All recordings that have values below average in this column	Yes
Above Average	All recordings that have values above average in this column	Yes
Lower 10	The 10 lowest values in this column	Yes
Upper 10	The ten highest values in this column	Yes

Lower 10-Quantile	In descriptive statistics, a 10-quantile (also known as decile) is any of the nine values that divide the sorted data into ten equal parts, so that each part represents 1/10 of the sample or population. Thus, the lower 10-Quantile is the 1st decile and contains the lowest 10% of data, i.e. the 10th percentile.	Yes
Upper 10-Quantile	The upper 10-Quantile is the 10th decile and contains the highest 10% of data.	Yes

Summaries

Summaries are small evaluations over values in one column. Summaries can be en- and disabled via the grid column headers.



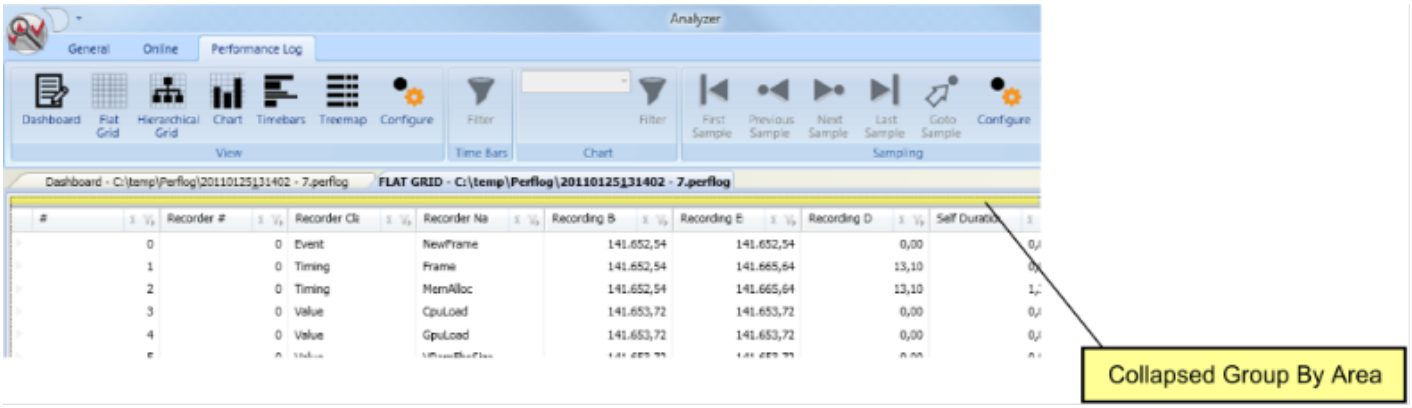
Grid View Summary Options

Summary Option	Description	Only Numeric
Count	Number of rows displayed	No
Average	The average over all displayed values	Yes
Maximum	The highest value	No
Minimum	The lowest value	No

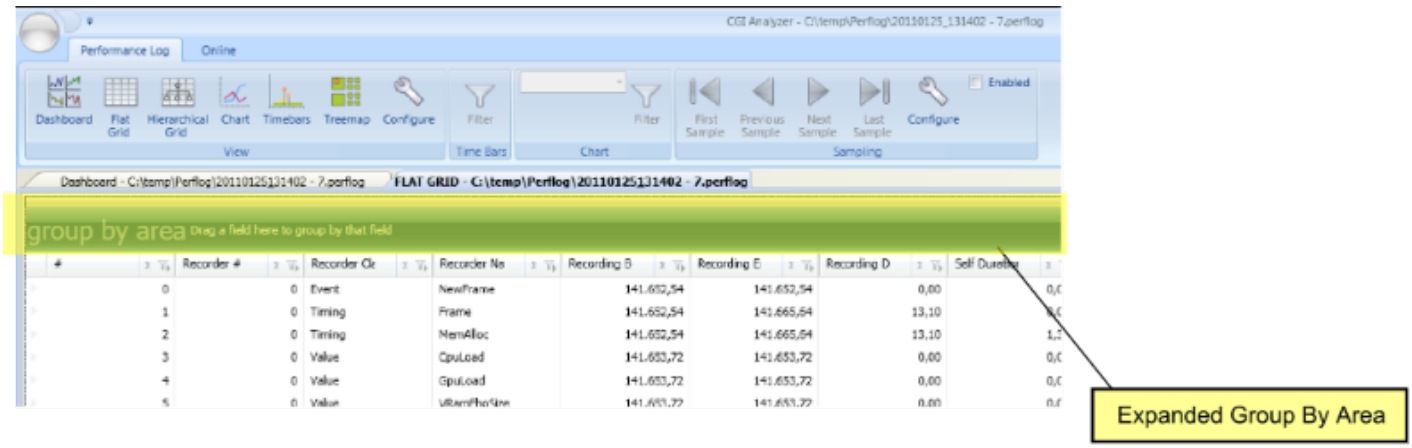
Sum	The sum over all values	Yes

Grouping

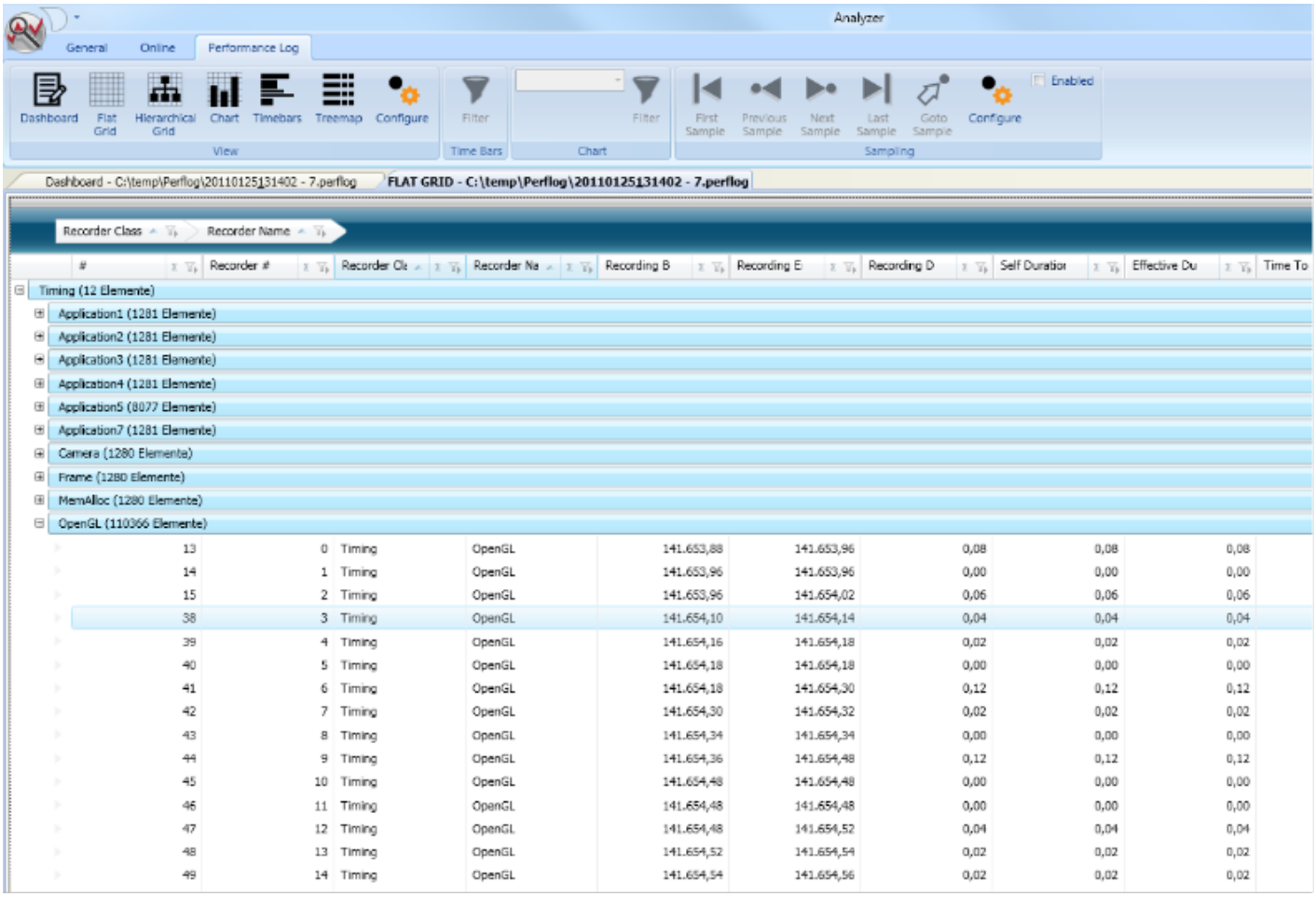
A grouping is a planned arrangement of all recordings by values in one column. Using this feature, it is possible to impose a hierarchical structure on the recordings. Grouping is accomplished by dragging a column header to the group-by-area. By default the group by area is collapsed.



In order to restore the group by area, left-click on it or drag a column header and drop in on the collapsed group by area.



Groupings can be chained, i.e. the recordings can first be grouped by values of one column, than by values of another column, and so on. The next figure shows recordings first grouped by recorder class and then by recorder name in order to collect all OpenGL recorders in one group.



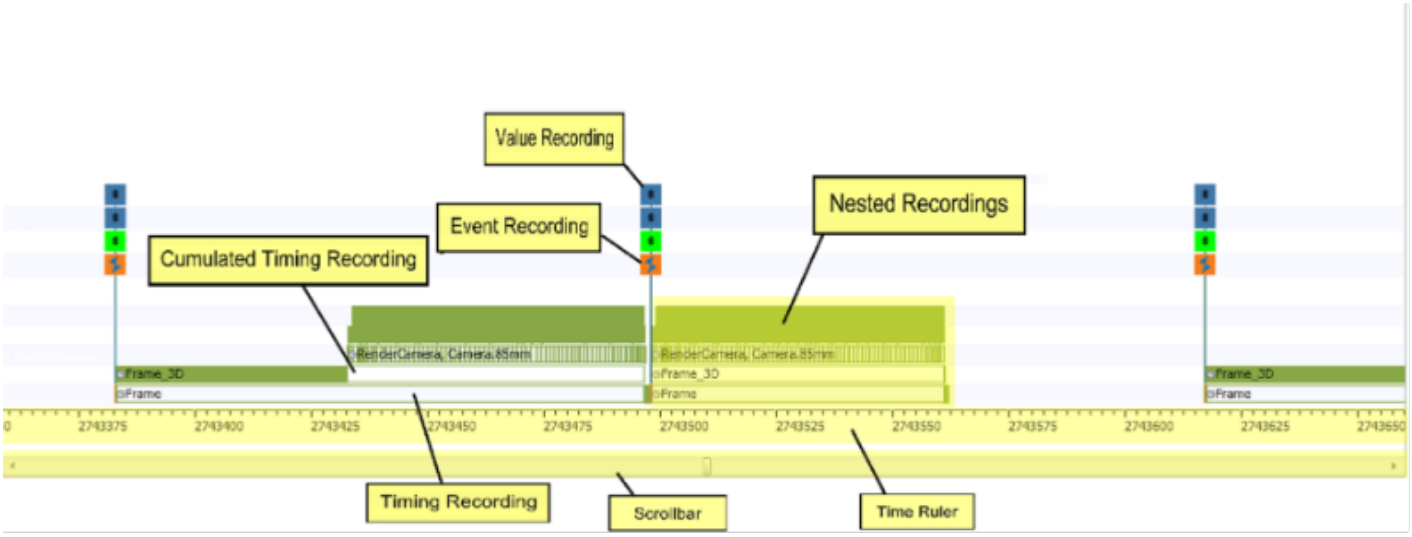
Time Bars View

Time Bars View displays all recordings arranged along a timeline. Below the actual time bars, there is a scrollbar for panning the displayed time frame. The time ruler is situated between time bars and the scrollbar. Here, time values in milliseconds and a cursor with the current time under the mouse pointer are displayed.

These icons are used to distinguish the different recorder classes:

	Time Recording
	Value Recording
	Event Recording
	Asynchronous Recording

	Cumulative Recording (timing or value)



Associated Ribbon Group

The next figure shows the ribbon group associated with the Time Bars View. It contains only one button for displaying the recorder filtering dialog.

Filtering,

Filtering here works analogous to filtering in the Chart View. Via the "Filter" button on the associated ribbon group the filter editor dialog is opened. En- and disabling of filters is immediately reflected in the chart.

- "Invert" inverts the current selection of recorders,
- "Set All" enables all recorders,
- "Reset All" disables all recorders in the current Chart View.

Sampling

Sampling is not supported for the Time Bars View.

Zooming

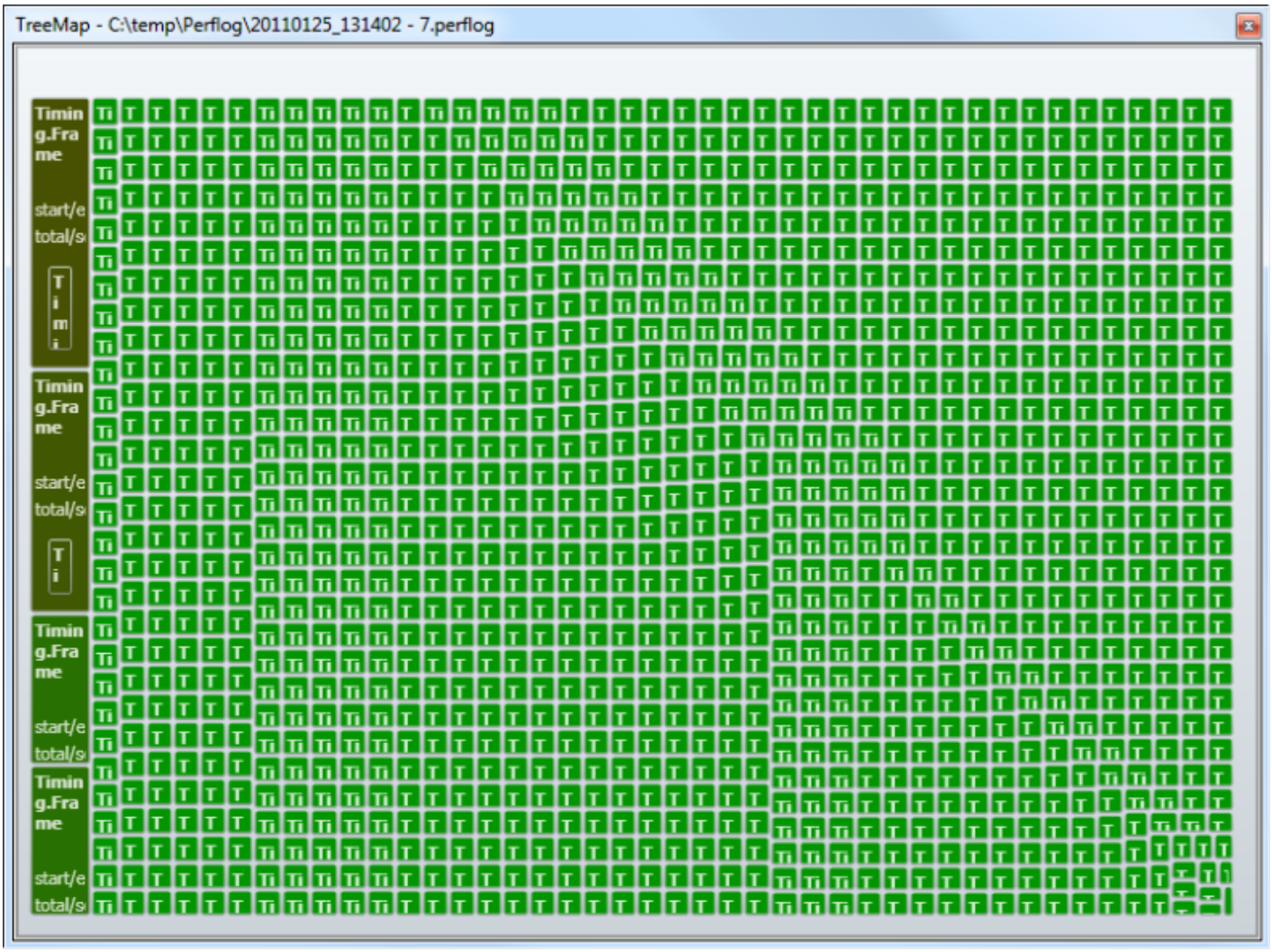
Zooming is supported via two mechanisms:

- Mouse wheel
- Keyboard shortcuts

Point the mouse pointer to the time you wish to zoom into. By using the mouse wheel or pressing Ctrl and the "+" or "-" button, you can zoom in to or out of this point in time. See section Keyboard Shortcuts for a list of all available keyboard shortcuts.

Treemap View

The Treemap View only displays timing recordings along with their nesting. The following figure depicts the Treemap View of a performance log.



This view shows all timing recordings ordered by decreasing recording duration. The timing recording with the highest performance impact on one nesting level is drawn in the upper left corner, the recording with the lowest impact in the lower right corner.

Left-clicking an item in the treemap drills into this item. Right-clicking an item returns to its parent.

Filtering

Filtering is not available for the Treemap View.

Sampling

Sampling is available via the Sampling Ribbon Group.

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 recordingof 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.

Analysis

Concept

Analysis Concept

The analysis module allows the user to conduct experiments and thereby discover performance bottlenecks.

These experiments can be performed on the host in a simulation or on the target.
During an experiment, the host and target have to be connected. Currently, Analyzer only supports a connection via TCP/IP and Serial Port.

Preparations for Analysis

Build Candera with Monitor enabled

Candera, as well as the application you are writing, have to be configured via CMake. Point CMake to the source directory of your application and indicate a directory for binary generation. Then fire up configuration. For further instructions see Establishing connection.

For configuration of CGI Monitor for a binary Candera distribution, see section 0.

Provide CPU and/or GPU load

In order to display a trend with key performance indicators (CPU load, GPU load, frame rate) on Linux, the proc file system has to be mounted.

This can be accomplished (if not already mounted) by adjusting / adding the corresponding entry under /etc/mtab or mounting the proc file system by hand with the following command:

```
mount /proc
```

CPU and GPU information have to be provided by drivers / kernel modules and are thus not available on every (Linux) platform.

Provide communication infrastructure

Candera is only pre-configured for communication with Analyzer, if you call the function Renderer::RenderAllCameras or Renderer2D::RenderAllCameras (or both).

Otherwise, you have to ensure, that

- Connection parameters are set correctly
- A connection is established
- A Candera application can be paused by Analyzer

Setting connection parameters

Communication parameters can be set, if necessary, using the following macro:

```
#include <Candera/System/Monitor/MonitorPublicIf.h>
MONITOR_PORT_CONFIGURE(address, port)
```

> This macro name has changed with v3.2. Previous version were named MONITOR_TCPIP_CONFIGURE.

The standard Player for use with the tunnel application under Windows presents a dialog for setting communication parameters after the asset library to load has been chosen. The tunnel sample application delivered with Candera now takes two additional parameters for TCP/IP address and port or for Serial Communication baudrate and com port. This macro has to be used before either MONITOR_TRYCONNECT() or MONITOR_HANDLE_PAUSED(), otherwise the standard configuration will still remain.

Establishing a connection

A connection can be established explicitly, if necessary, from application code, by using the following macro

```
#include <Candera/System/Monitor/MonitorPublicIf.h>
MONITOR_TRYCONNECT()
```

Enabling the pausing of a Candera application

For Analyzer to be able to pause your application and the key performance indicators to be updated, the following macro has to be called every time, the drawing of a new frame begins:

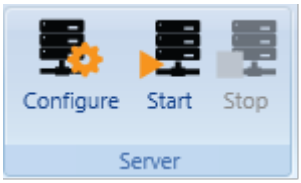
```
#include <Candera/System/Monitor/MonitorPublicIf.h>
MONITOR_HANDLE_PAUSED()
```

Connecting with a Candera application

The procedure for setting up a connection between host and target is as follows:

1. Start Analyzer
2. Configure the communication server
3. Start the server via the Server Ribbon Group on the Online Ribbon Tab
4. Start the Candera application, it will try to connect to the pre-configured port and IP-address

Configuration, starting, and stopping of the analysis server are triggered from the server ribbon group as depicted in the following figure.

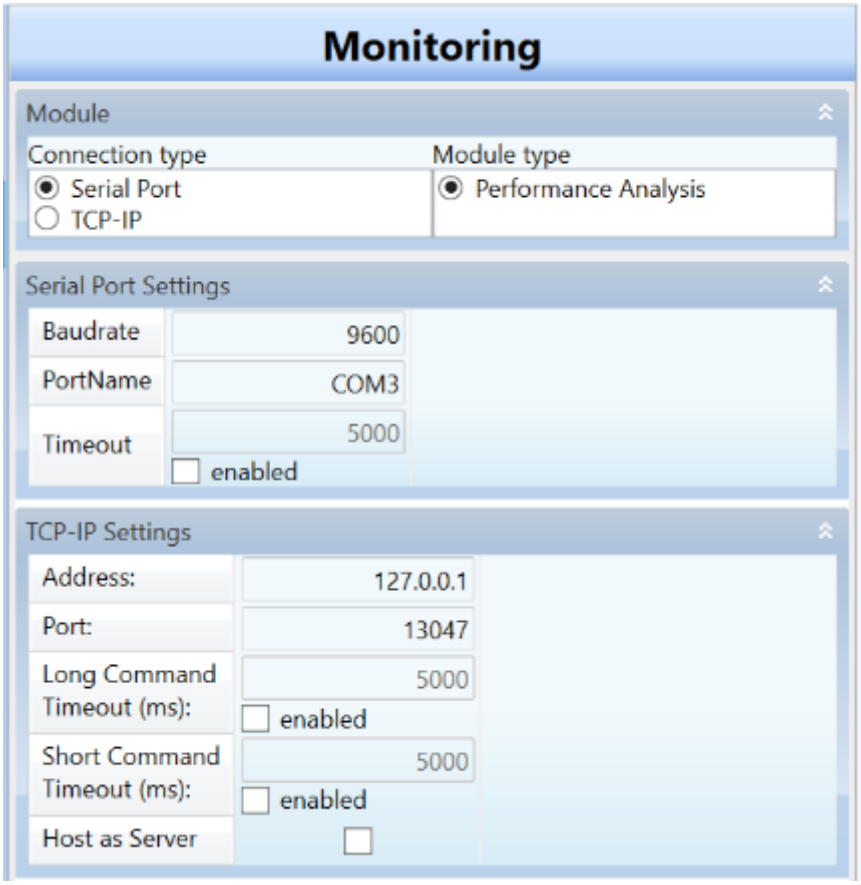


Upon successful connection between Analyzer and the Candera application, the Candera application is in Running Mode. A successful connection is indicated by the status bar. The server state is displayed as Connected and the application state as "Running".

Configuration

The communication server is configured using a configuration dialog accessed through the "Configure" button on the Server Ribbon Group. Figure 20 shows the configuration dialog for TCP/IP connections.

> IP-address and port, where Analyzer should listen for incoming connection requests have to be set.



Timeouts can be set in order to prevent the user interface from freezing indefinitely if the communication fails. Disabling these timeouts completely can lead to freezes and communication failures depending on application type, communication type and stability of connection.

Analysis Views

Trend View

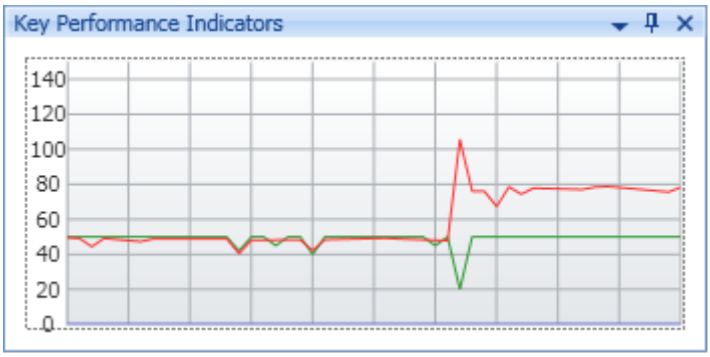
The Trend View shows successively recorded key performance indicators such as

- CPU load (if available)
- GPU load (if available)
- Frame Rate

It can be displayed via the View Ribbon Group of the Online Ribbon Tab.

The following shows a Trend View for a host simulation. The color assignment for the performance indicators is currently fixed and as follows:

- Frame Rate (red)
- CPU load (green)
- GPU load (blue)



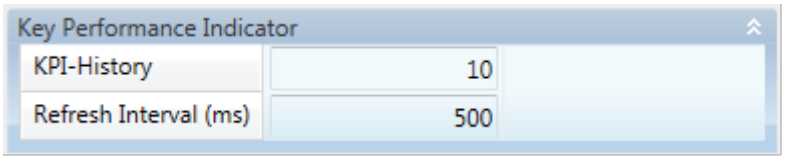
Mouse over the Trend View shows a clear button which resets the view.

Right-clicking the Trend View opens a context menu for

- closing it and
- displaying it always on top of every other window.

> GPU load availability is dependent on device capabilities.

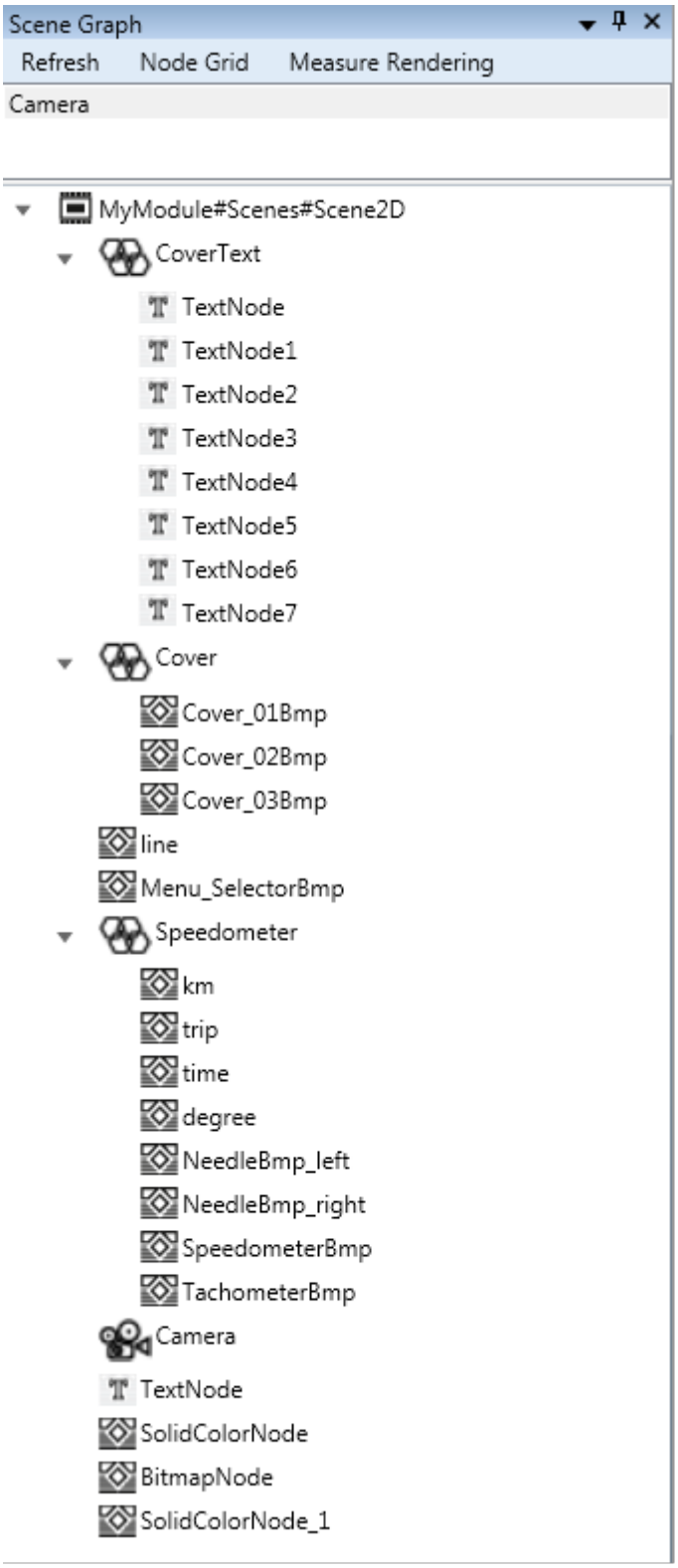
The settings menu also provides settings for trends:



These settings allow to set the update rate and the history length of trends. IT is possible to change these values on the fly without restarting, pausing or changing the communication

Scene Graph View

The Scene Graph View allows analysis of the scene graph during paused mode.



This view is divided vertically by a splitter into two sections:

- A camera list on top and
- A tree view of a scene graph below.

The tree view shows the scene graph associated with the selected camera in the camera list.

The following commands are available via a toolbar on top of the Scene Graph View:

Refresh:

- This command is always available in paused mode.
- Fetches the scene graphs from the Candera application and displays it. Node expansion and selected nodes are preserved.

Node Grid:

- This command is available in paused mode, if a camera was selected from the camera list.
- Displays all information transferred for the scene graph of the selected camera in a grid.

Measure Rendering:

- This command is available in paused mode, if a camera was selected from the camera list.
- Measures the rendering times of individual nodes and displays the render times along with static scene graph information in a grid

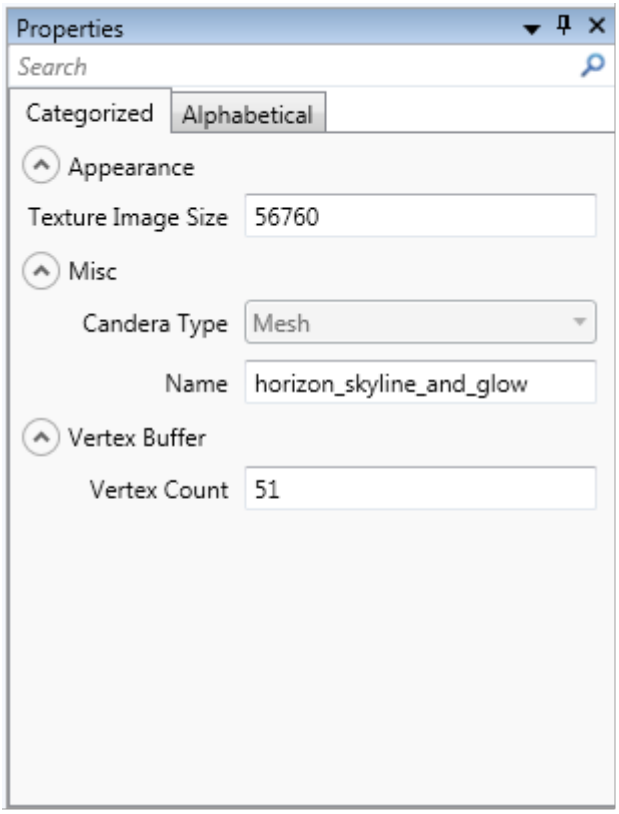
The properties of a selected node are displayed in the Properties View.

Properties View

Depending on the scene graph node selected in the Scene Graph View, different properties are displayed.

For each node, the following properties are displayed:

- **Name:** The name of the node, if available, a blank field otherwise
- **Candera Type:** The type of the node (e.g. Mesh or Group)



Guards

Guards are a means to switch a Candera application from running to paused mode.

We differentiate between two flavors of guards:

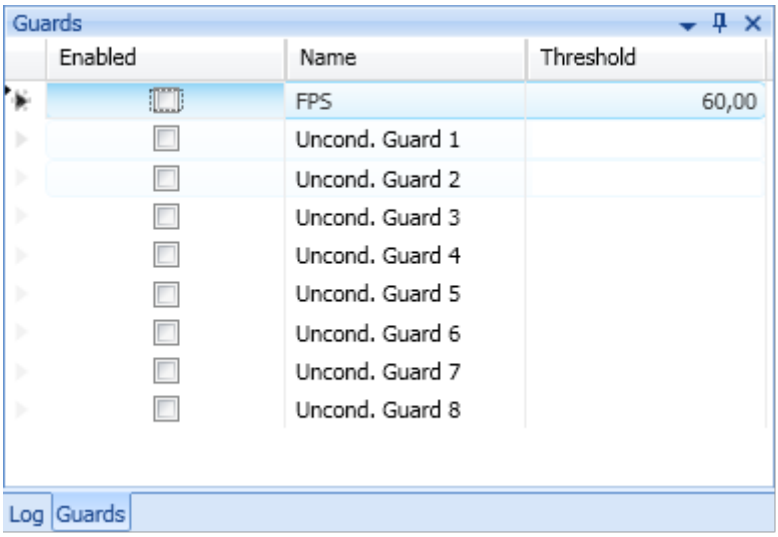
- Conditional Guards
- Unconditional Guards

Conditional guards are triggered, when a condition is met. Currently, only one conditional guard (based on measured frames per second) is supported. If the measured frames per second fall below a specified value, the guard is triggered, if it has been enabled, and the Candera application enters the paused mode.

Unconditional guards can be set in application code via the use of the following macros:

```
#include <Candera/System/Monitor/MonitorPublicIf.h>
MONITOR_FORCE_PAUSED(groupId)
```

Guards are combined in guard groups. Each guard group has a unique id (the guard group ids are defined in the include file MonitorTypes.h).



All guards can be en- and disabled. Additionally, the number of frames per second can be set for a FPS guard.

Performance Log Transfer

During an active connection to a Candera application, a performance log can be transferred from the target to the host via the communication channel used.

There are two flavors of performance log transfer:

- Recording until the recording buffer is full and explicit transfer of the buffer content
- Continuous streaming of performance log data to a specified file

Single Buffer Recording and Transfer

Caution has to be taken to use only one of the following means of performance log recording

- Record to file on device
- Record to memory buffer on device

If both flavors of recording are enabled, unpredictable effects may occur.

Recording to memory buffer is enabled via the Enable Recording checkbox of the Performance Log Ribbon Group.

After the memory buffer has been filled (approx. 1 sec), the buffer contents can be transferred to the host and stored in a file by clicking the "Get Log" button, also located in the Performance Log Ribbon Group.

Performance Log Streaming

Performance log streaming is the continuous recording and transfer for performance log data from a Candera application to Analyzer.
Streaming is enabled by first enabling recording via the Performance Log Ribbon Group and then enabling streaming via the "Enable Streaming" checkbox. A storage location for the performance log data (file) is asked.

After provision of this information, performance log data is continuously streamed from target to host and appended to the specified file.

Global Experiments

Experiments allow the user to locate performance bottlenecks by intervening in the rendering process. By targeted change of rendering parameters, conclusions about the origins of performance issues can be drawn.

There are three different groups of experiments:

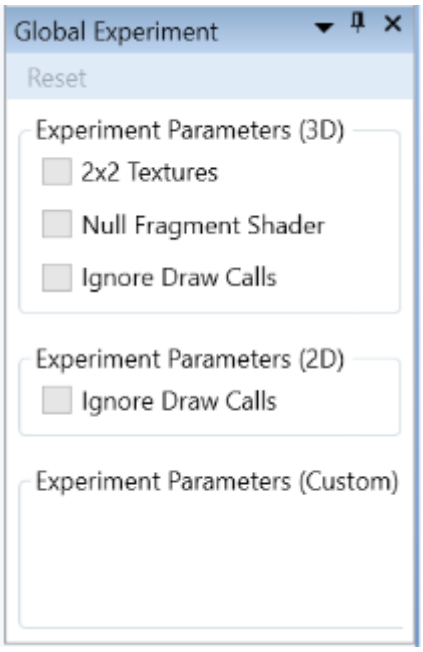
- 3D experiments
- 2D experiments
- Custom experiments

2D/3D Experiments

The following experiments are available:

- 2x2 textures (3D only)
- Null Fragment Shader (3D only)
- Ignore Draw Calls

The following figure shows the dialog for setting experiments. Currently, experiments can only be en- or disabled in Paused Mode.



Using 2x2 textures

All textures are replaces with a 2x2 texture using a chess pattern.
If the frame rate rises significantly after enabling 2x2 textures, there are likely too big textures in use.

Null Fragment Shader Program

All shader programs are replaces with a default shader program whose fragment shader always produces the color red.
If the frame rate rises significantly after enabling the null fragment shader, fragment shader may be to complex.

Ignore Draw Calls

All draw calls are ignored, thereby simulating an infinitely fast GPU. If the frame rate does not rise significantly, the problem may be CPU-bound.

Custom Experiments

Custom experiments are universally usable experiments. In general it is a table of functions which have a parameter (enable/disable). The analyzer can trigger function calls on these specified functions. An experiment state is false/disabled per default.

Custom Experiments - Target side:

To use this feature, the following flag has to be set:

```
// Enables custom experiments
#define CANDERA_MONITOR_CUSTOM_EXPERIMENTS
```

This flag can be set within code when it were not enabled by cmake configuration.

> Using this flag requires init-Call (next step). Compiler produces error without this init-Call.

The following sample code snippet has to be used in order to register a function as experiment:

```
//Adds macros for global experiments
#include <Candera/System/Monitor/GlobalExperimentPublicIF.h>
…
// Function which can be triggered by analyzer
void SampleFunction(bool flag);

// Initializes list – has to be outside of any namespace
//First param is amount of function pointer
//following parameters are function pointers
GLOBAL_EXPERIMENT_INIT_CUSTOM_LIST(1, SampleFunction)
```

The include adds the appropriate macros. To enable these custom experiments previously mentioned define has to be set.

> The Init-Call has to be placed outside of any namespace and function.

A callable function’s declaration is defined as:

```
void FunctionName(bool flag);
```

In general: A static function with no return value. The flag is set by analyzer. A function within namespaces and classes can be added but need full declaration. This function should never be called by custom code - otherwise undefined behavior. The next point explains a way to indirectly call such a function.

Calling a function pointer by target application
If it is required to call a registered function pointer, use the following macro:

```
GLOBAL_EXPERIMENT_CALL_CUSTOM_EXPERIMENT(fct_ptr, value);
```

Global Experiments - Analyzer:

The analyzer provides a view for global experiments. It contains three different areas for 2D, 3D and custom experiments. To enable or disable an experiment toggle the checkbox. Custom experiments are listed in the same order as they were added to list on target side (see Custom

Experiments - Target side: GLOBAL_EXPERIMENT_INIT_CUSTOM_LIST). If the list is wrong or not present, pause the application to update the experiments list.

Additionally it provides a reset button which is used to disable all experiments.