Player

The Player is an application which provides insides into a generic CGI Studio application driven by the courier messaging framework.

It also provides samples of several features. These features are core components like showing or hiding views or trigger animations but also sample implementation of models with databinding.

On top of that, the player shall be an application to test and play around with an asset and the functionality of the framework itself. The focus is not on the customer specific logic but on the core logic of the customer’s specific asset.

Therefore, some of the content within the application is not required for customer made applications. Especially the Features and Sample Components folder.

- Panel
  - Overview
  - File Menu
  - Edit Menu
  - Player Menu
  - Recorder Menu
  - Help Menu
  - Scene View
  - Configuration View
  - Snapshots View
  - Logging View
  - Controls View
  - Transitions View
  - Statistics View
  - State Machine View
  - Value Generator
  - Zooming
  - Rotate

    - Data Bindings

- Features
- Command Line Usage
- Application Structure
- Layout Monitor

Panel

Overview

Description

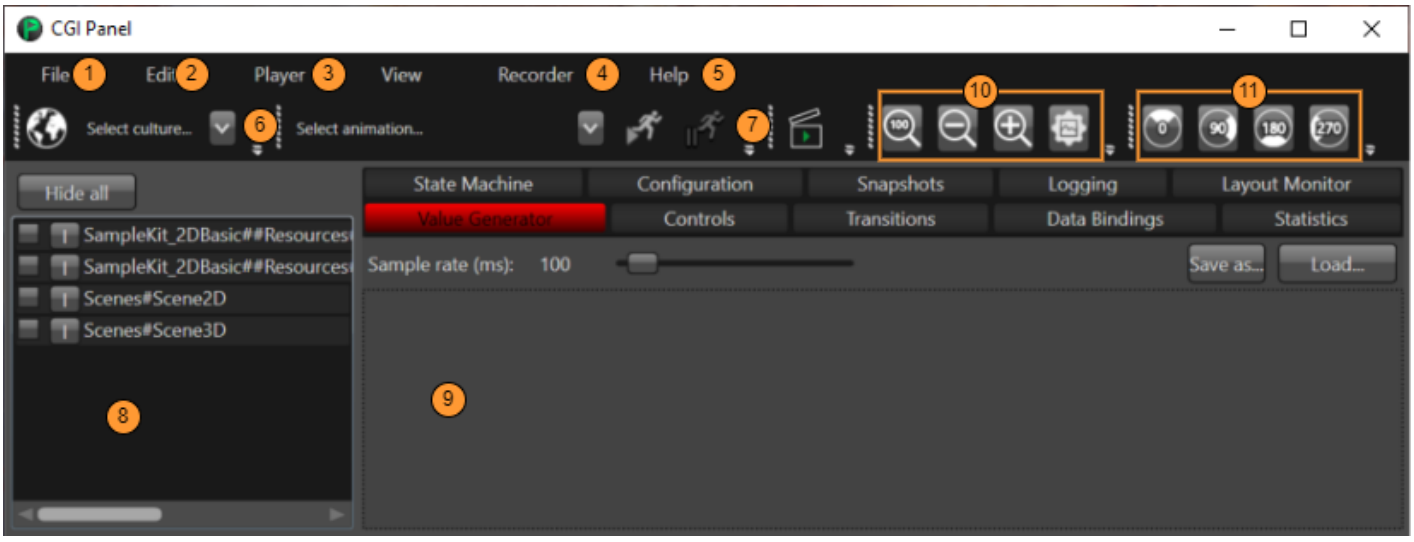
The Panel is a control tool for the Player application. It visualizes all the mentioned feature blocks in the Features section.

Hotkeys

Following shortcuts are available for CGI Panel:

Hotkeys	Description
Ctrl+O	Open asset
Ctrl+S	Capture screenshot
Ctrl+Shift+S	Capture screenshot (select file)
Ctrl+P	Open settings
Ctrl+I	Single invalidation
F1	Documentation

Interface

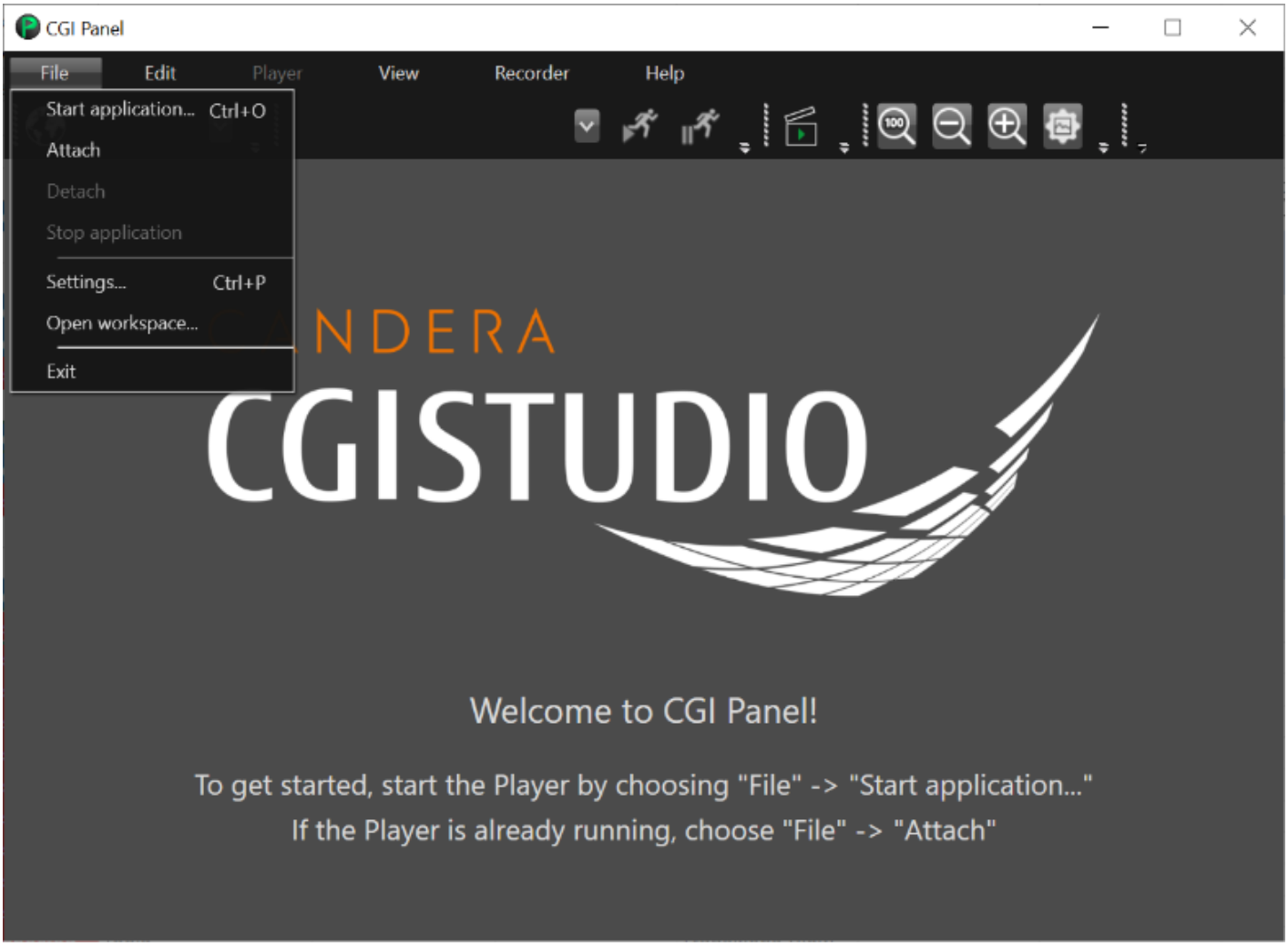


1. File menu allows user to run and configure the application.

2. Edit menu offers access to editing operations and undo-redo actions.
3. Player menu allows to control the rendering and scripting system.
4. Recorder menu gives control to the recording of application handling.
5. Help menu offers information about CGI Panel.
6. Culture toolbar allows to switch between languages. In order to use this, "Globalization" needs to be enabled.
7. Animation toolbar shows a list of all animations available. Select an animation from dropdown box and press Start to run it.
8. Scene View panel shows a list of all scenes. In order to activate one scene, click the corresponding checkbox.
9. Miscellaneous Panel embeds the following views:
    - Configuration view presents information about the features enabled in the Player.
    - Snapshots view allows to take snapshots (on host) and shows the last snapshots taken.
    - Logging view allows to enable or disable the logging and also to set the log level for the main realms.
    - Controls view shows all controls of a selected scene.
    - Transitions view is available to trigger transitions.
    - Statistics view shows memory consumption statistics.
    - State Machine view shows the state for all state machines.
    - Value Generator provides the possibility to simulate the change of values on view elements.
10. Zooming toolbar
11. Rotate display toolbar

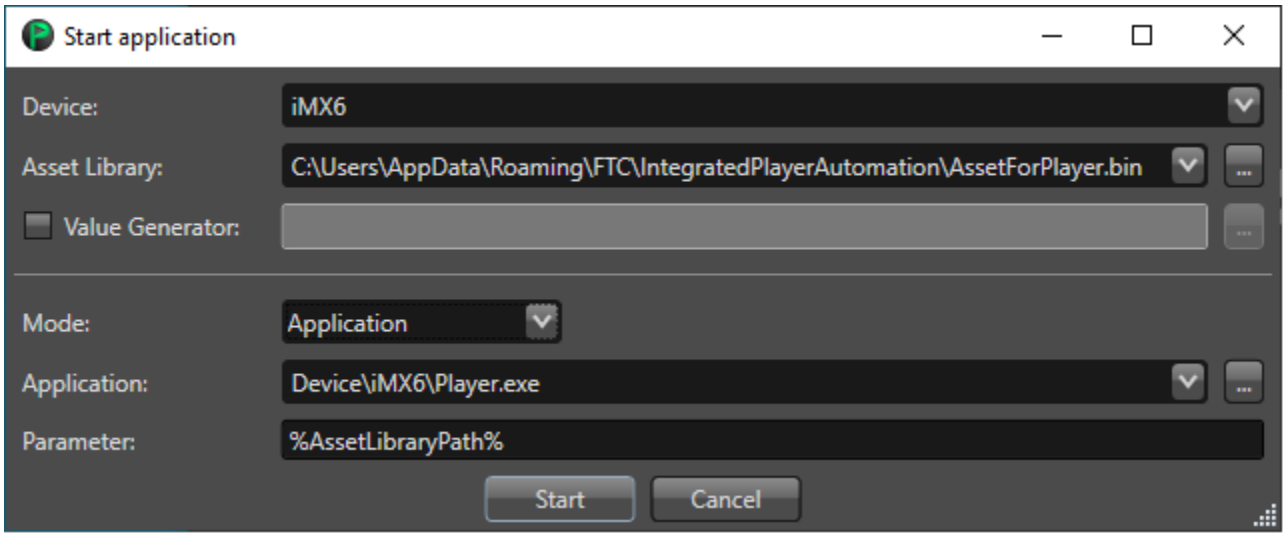
File Menu

offer commands that allows to control how the application runs.

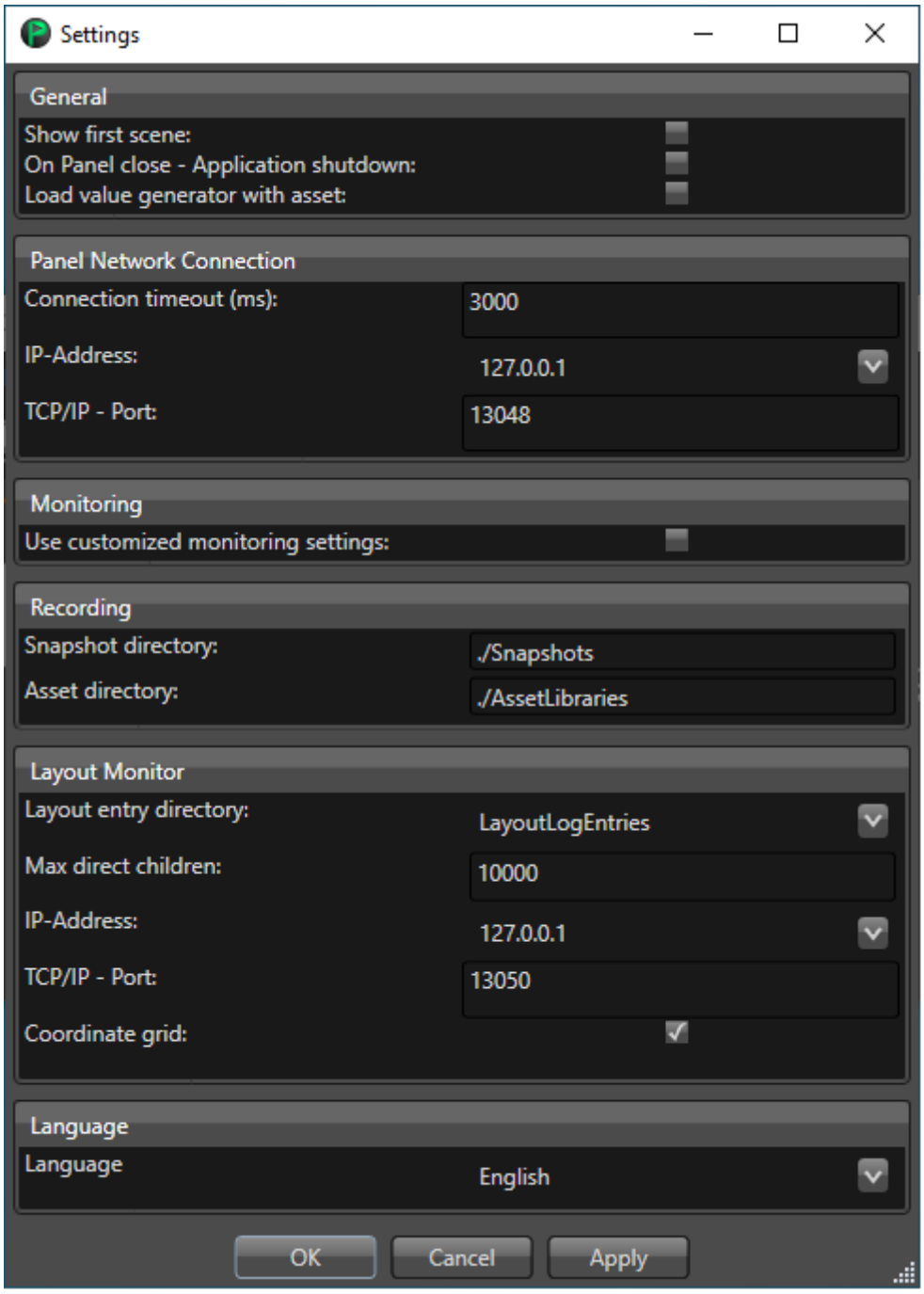


Options:

- **Start application ...** - opens a dialog window which allows to configure and start the application.



- The configuration options are:
  - *Device*: allows to choose the device from a dropdown list.
  - *Asset Library Path*: allows to set the full path to the asset.
  - *Mode*: allows to decide if the Player will be started as an application or by a script.
  - Application Mode
    - *Application*: allows to set the path to the Player executable.
    - *Parameter*: allows to set a parameter which shall be passed to the application.
  - Script Mode
    - *Script*: allows to set the path to the script.
    - *Parameter*: allows to set a parameter which shall be passed to the script.
    - *Wait for script to finish*: configures whether the panel should wait or not until the script has finished.
- **Attach** - attaches the panel to a running Player.
- **Settings...** - opens a dialog window which allows to configure the Cgi Panel.



The configuration options are:

- General
  - *Show first scene*: enables that the first scene is automatically loaded at start time.
  - *On Panel close - Application shutdown*: if enabled the applications started by the Player will be shutdown automatically.
- Panel Network Connection
  - *Connection timeout (ms)*: sets a timeout for network connection.
  - *IP-Address*: sets a custom IP address of the network control.
  - *TCP/IP - Port*: sets a custom TCP/IP port of the network control.
- Monitoring
  - *Use customized monitoring settings*: allows to set custom setting for the monitoring (*TCP/IP-Port*, *Serial-Port* and *Serial-Baudrate*).

    - Recording
        - *Snapshot directory*: defines the location where the snapshots shall be stored.
        - *Asset directory*: defines the location of assets which will be loaded.
- **Open workspace**: opens in File Explorer the directory were the CGIPanel.exe resides.
- **Exit**: closes CGI Panel.

Edit Menu

Edit menu offers access to editing operations and undo-redo actions.

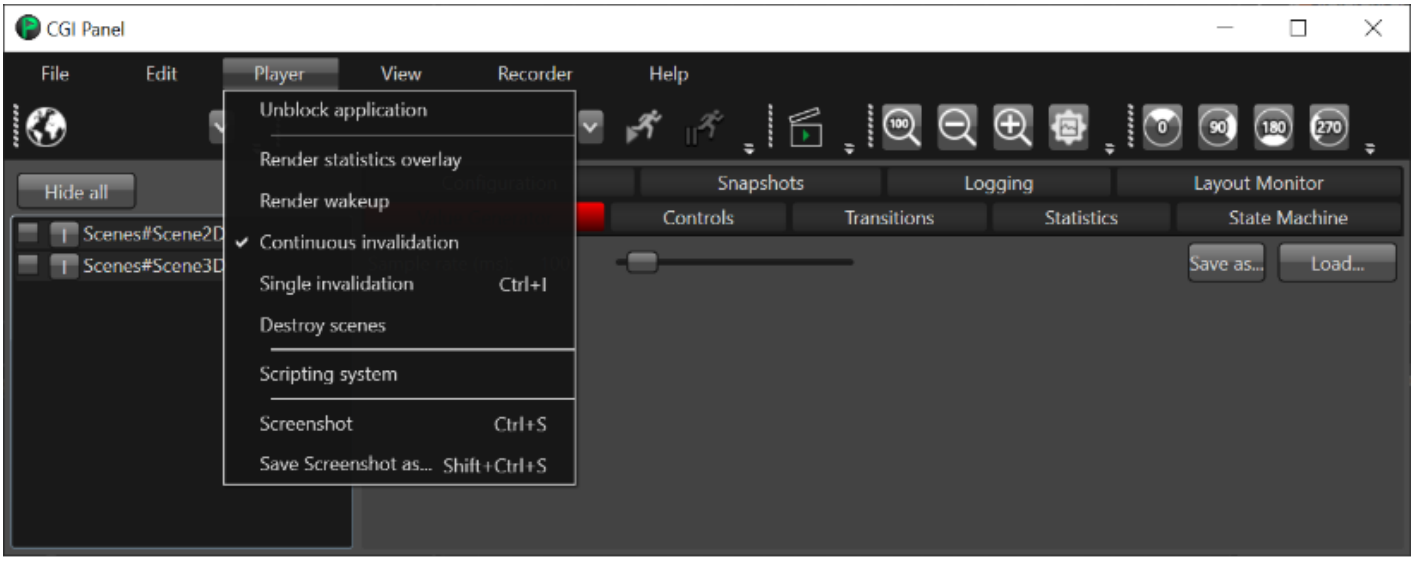


Options:

- *Undo*: undo last action.
- *Redo*: redo last undone action.
- *Cut*: cut text.
- *Copy*: copy text.
- *Paste*: paste text.
- *Delete*: delete text.

Player Menu

Player menu allows to control the rendering and scripting system



Options:

- *Unblock application*: unblocks an application which is currently in the waiting state.
- *Render statistics overlay*: enables or disables the render statistics overlay within the scene.
- *Render wakeup*: enables or disables the render wake up mechanism.
- *Continuous invalidation*: enables or disables continuous invalidation. (Please note that it should be disabled when Dirty Area Management feature is used)
- *Single invalidation*: sends a single 'invalidate all' message.
- *Destroy scenes*: enables or disables the destroying of the scenes as soon they are hidden.
- *Scripting system*: enables or disables the script system.
- *Screenshot*: saves a PNG file for each display in the folder of the loaded asset.
- *Save Screenshot as...*: saves a PNG file for each display but the location can be set via "Save as ..." dialog.

Recorder Menu

Recorder menu gives control to the recording of application handling.



Options

- *Play...*: shows the Open dialog to give the path to a recorded file that will be played.
- *Record*: enables or disables the recording of application handling.
- *Record Touch Hover*: enables or disables the recording of touch hover messages.
- *Restrict action interval to 500ms*: restricts the action interval to 500 ms.

Help Menu

Help menu offers information about the application.

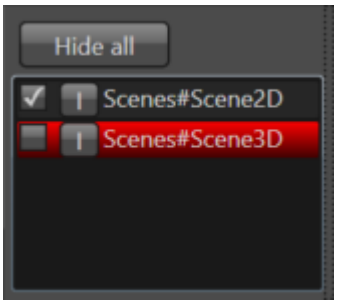


Options:

- *Documentation:* opens the CGI Studio Documentation
- *About...*: opens a window which shows information about the application.

Scene View

The scene view is a central component of the CGI Panel. A scene can be shown or hidden by checking or unchecking the checkbox of a scene. Based on the selected most of functionality chooses its content. This includes for example animations and controls.



Panel

Configuration View

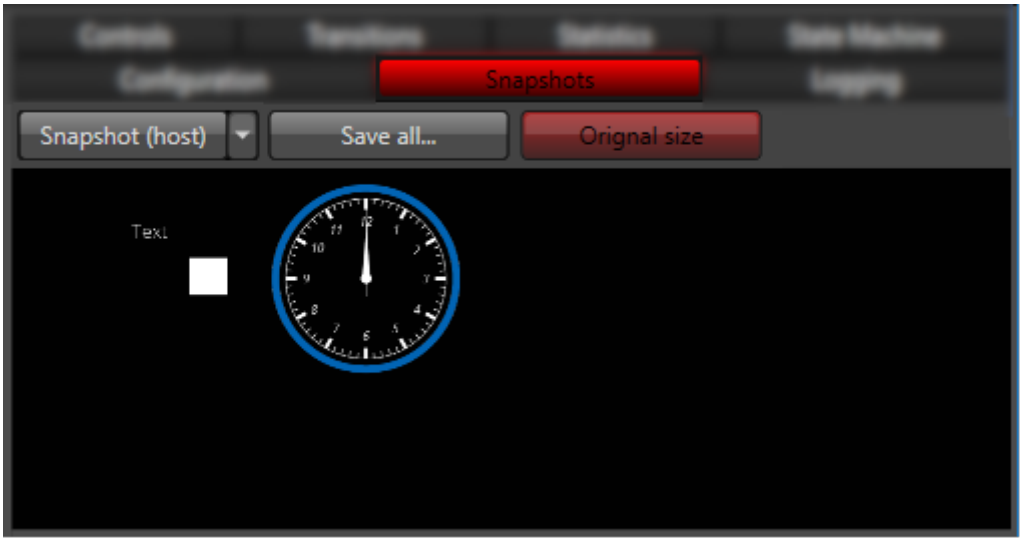
The Configuration view presents information about the features enabled in the Player.



Panel

Snapshots View

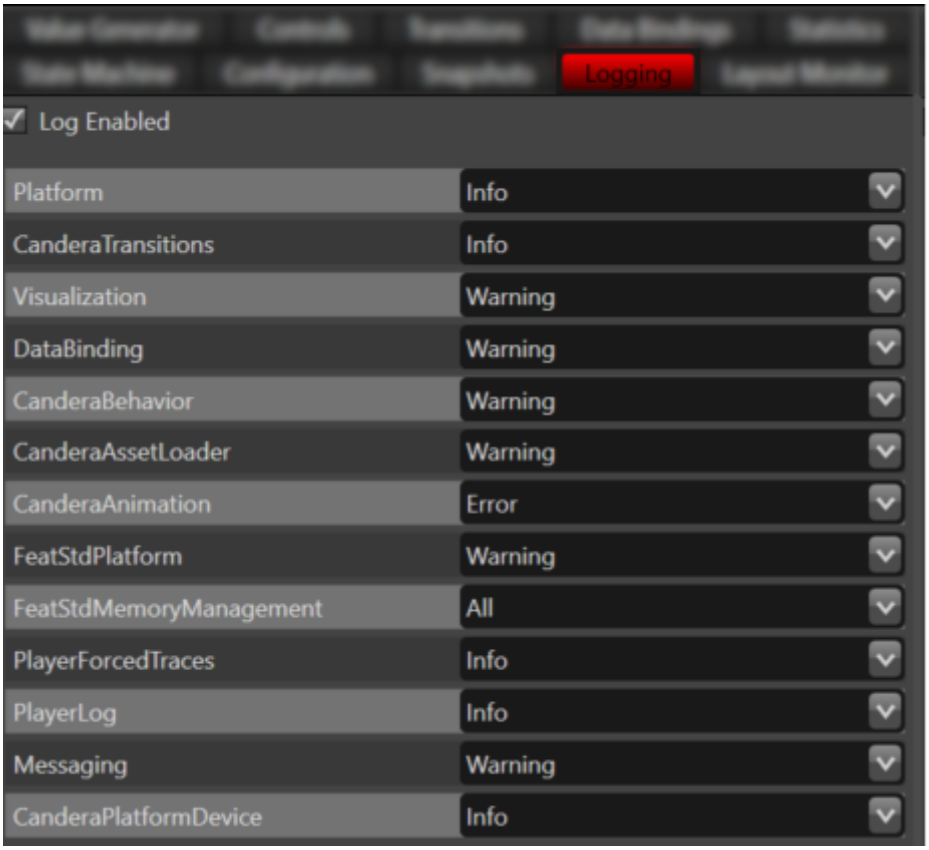
The Snapshots view allows to take snapshots and also shows the last snapshots taken.



Panel

Logging View

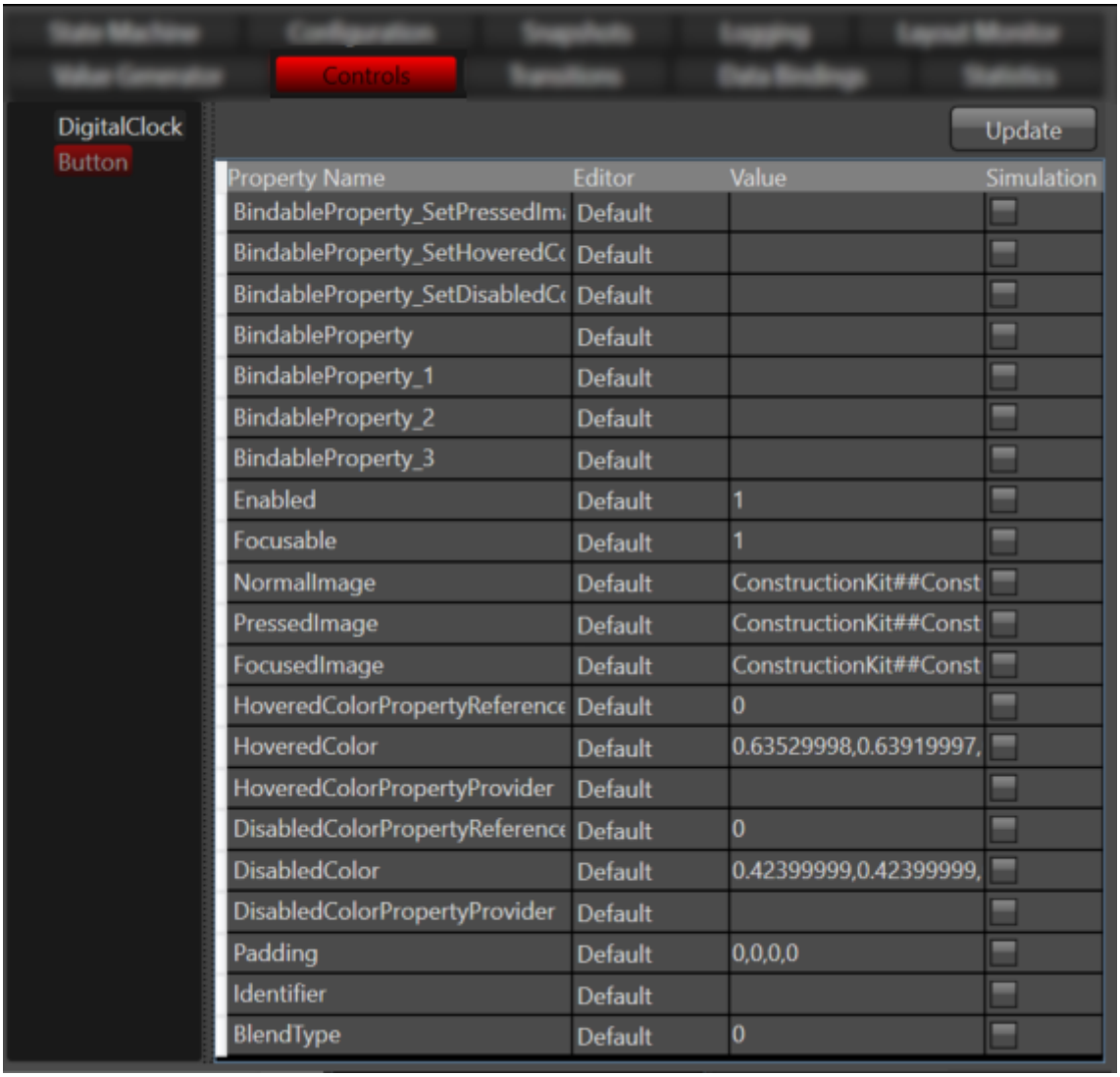
The Logging view allows to enable or disable the logging and also to set the log level for the main realms.



Panel

Controls View

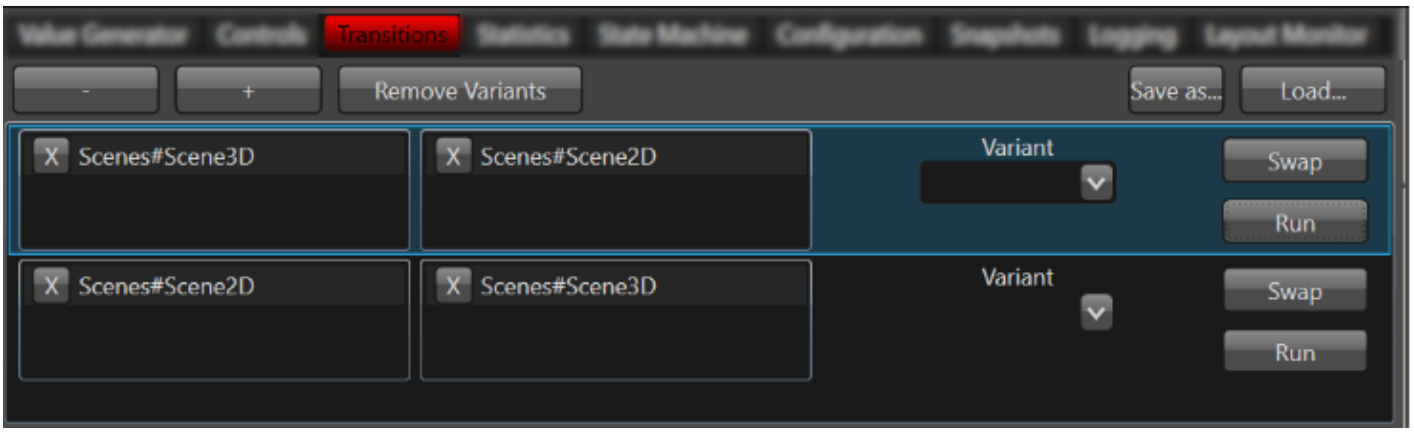
The controls view shows all controls of a selected scene. Selecting a control opens a table of all the properties and its values. Changing the value will also change the value in the application.



Panel

Transitions View

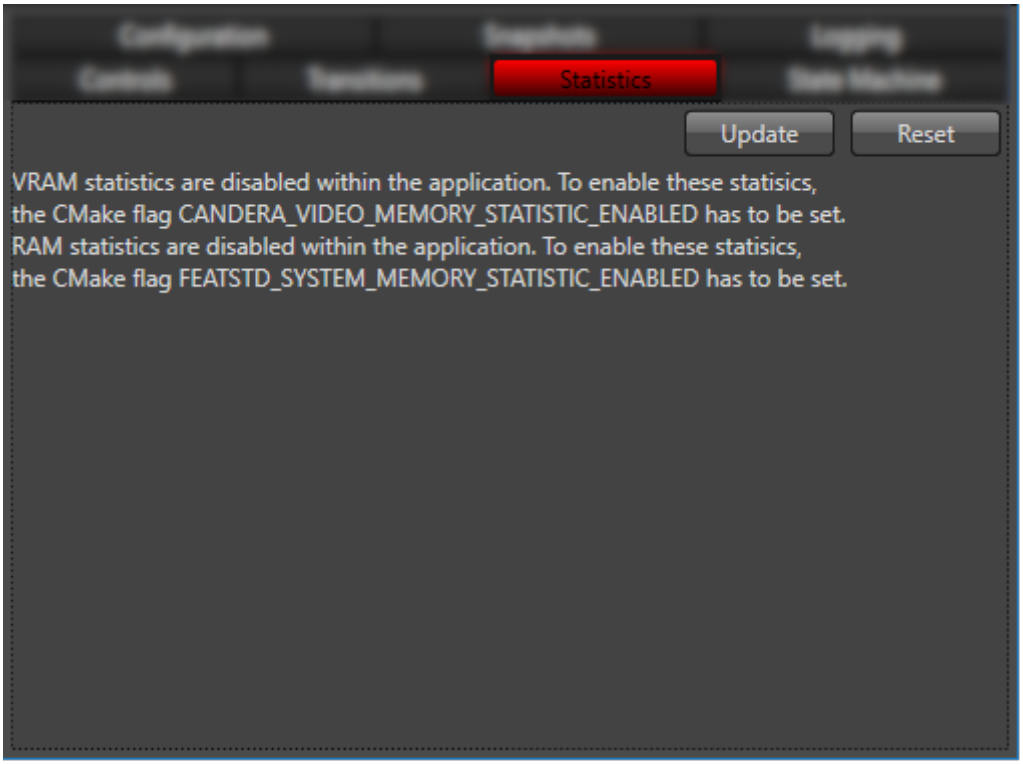
The transitions view is available to trigger transitions. This happens by drag and drop from the scene list into the deactivation or activation area. A deactivation scene will be hidden and an activation scene will be shown at the end of a transition. More than a single scene can be added to the lists. The transition will be executed by pressing the Run button. The swap button swaps the list. So it will revert the execution. More of the transition instances can be created to keep one but create a second one. These transitions are kept only for a single session.



Panel

Statistics View

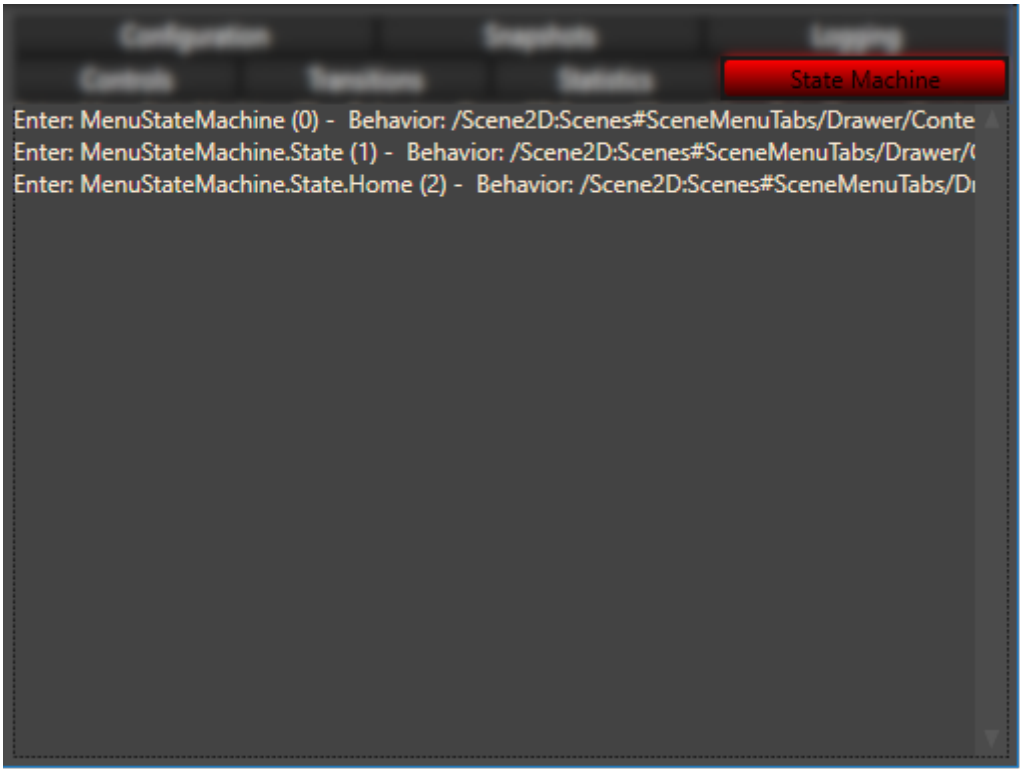
The statistics view shows memory consumption statistics and it is only available when the CMake flag "CANDERA_VIDEO_MEMORY_STATISTIC_ENABLED" is set.



Panel

State Machine View

This view shows a log which describes the states for all state machines.

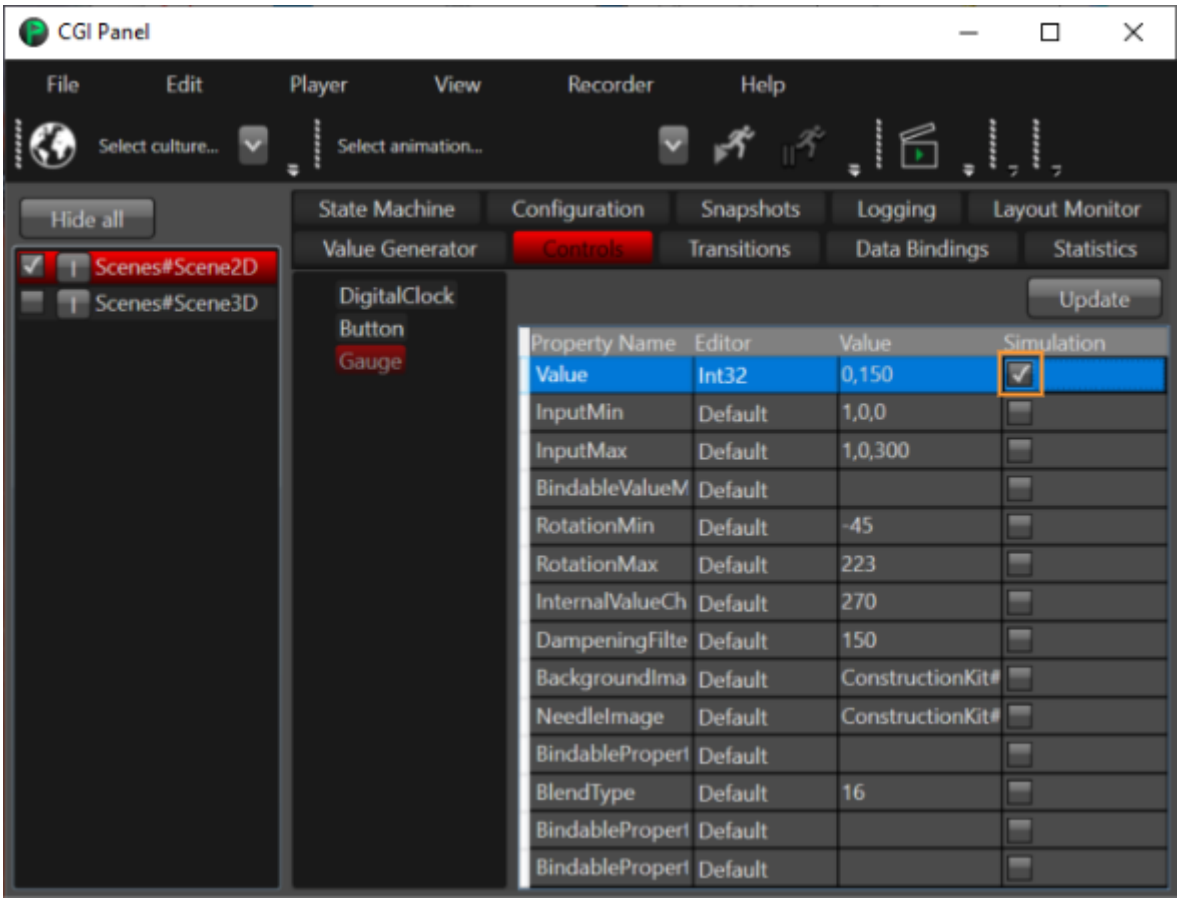


Value Generator

This feature provides the possibility to simulate the change of values on view elements. With this, view elements can be simulated without logic implementation.

Selecting a property to simulate

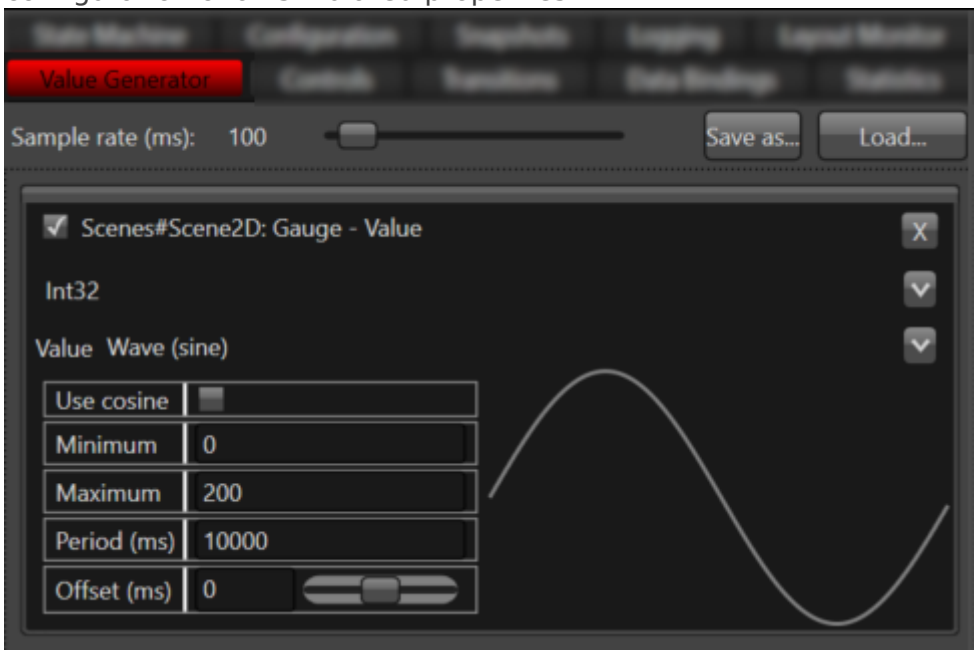
In the CGI Panel, select a scene and click on the Controls tab. This will display all controls and their properties.
Checking the Simulation checkbox of a property will create a simulation entry inside the Value Generator tab.
The Editor column defines the type for the property. Based on the Editor, the simulated value will be calculated.



Modifying simulation properties (of a selected property)

The properties which have been selected in the Controls tab, are created in the Value Generator tab.

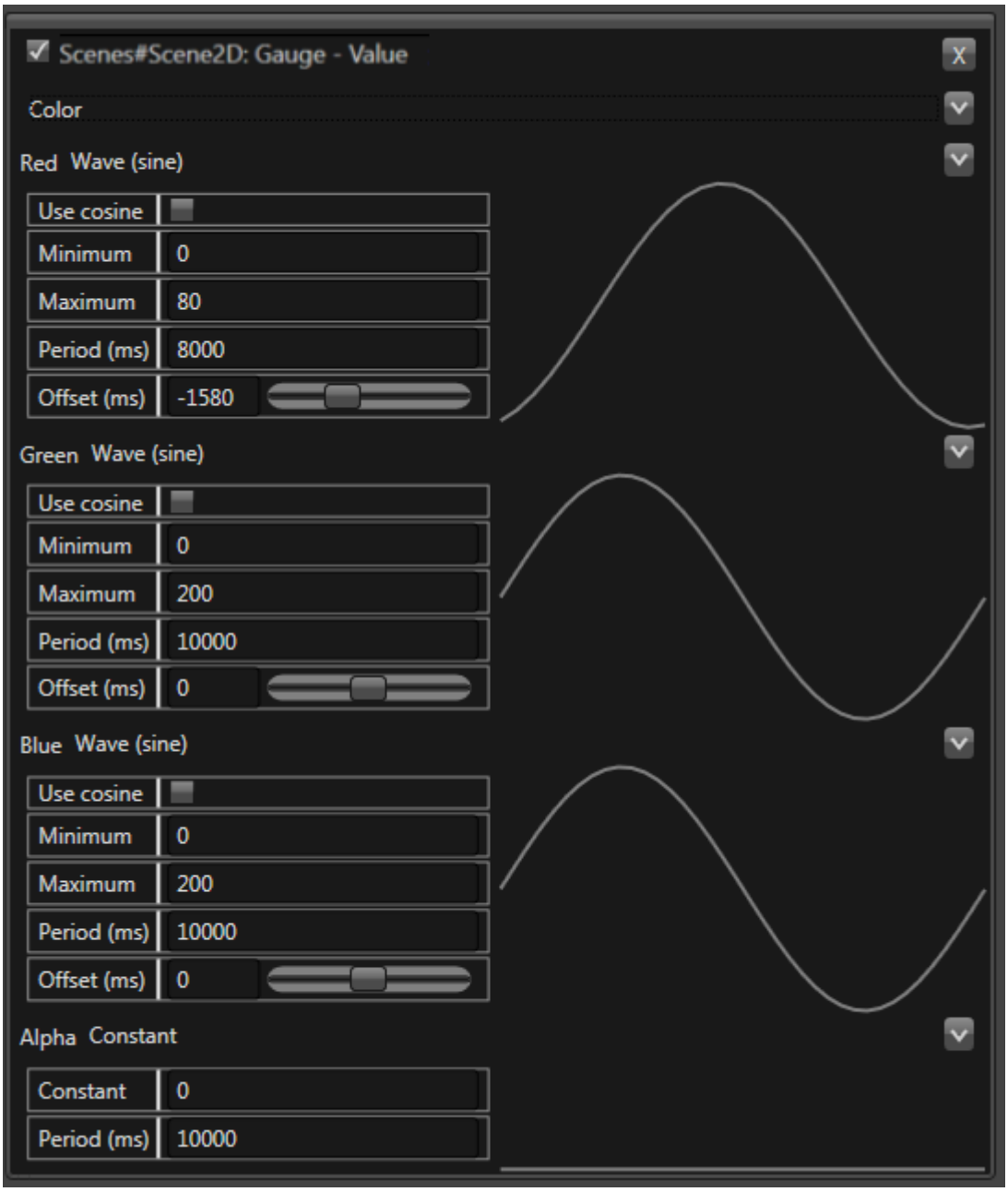
- The Value Generator tab contains general configurations, serialization and the configuration of all simulated properties.



  - The **Sample Rate** defines the update rate of the simulated properties. This sample rate is applied to all simulated property entries.
  - Each simulated property entry has a predefined set of configurations.

    - **Data type**
      The simulation supports different datatypes. This includes double, float, all int types and special datatypes like size/coordinates and color.
      If default editor is selected, it will try to guess the datatype based on the property name. Each value of a complex datatype can be set individually.

      The sample below simulates a color:

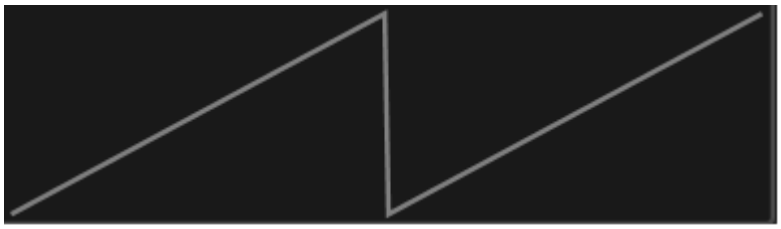


- **Value Simulation**

  This represents the simulation algorithm that will be used to calculate the value.
  The following list shows the different graph types and and some variants when modifying its configurations.

  1. **Constant (Updates the value with the same value)**

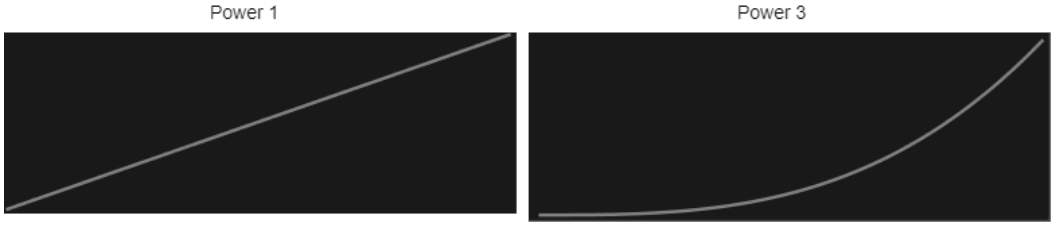
2. **Saw Wave**



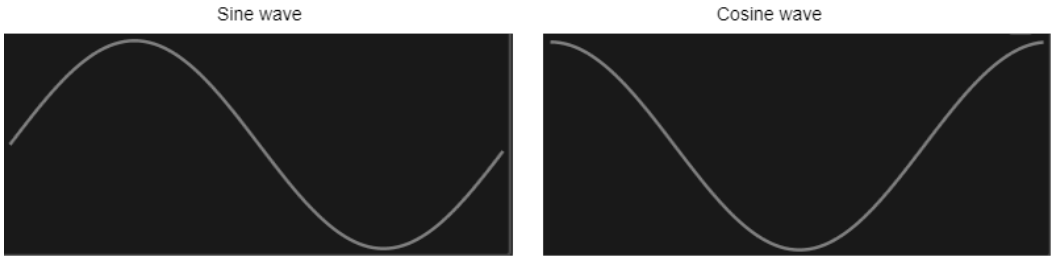
3. **Triangle**



4. **Power (Power of 1 - linear, Power of 3)**



5. **Sine Wave (Sine, Cosine)**

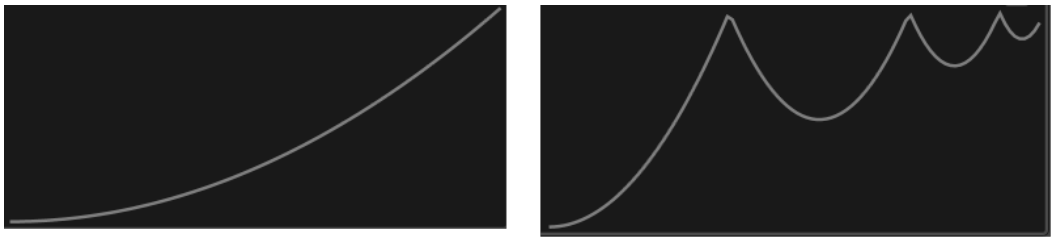


6. **Random (No Visualization available)**

7. **Toggle (Toggles between 0 and 1)**



8. **Bounce (Bounces 0, Bounciness 0.5; Bounces 3, Bounciness 0.20)**



- **Minimum and Maximum range**

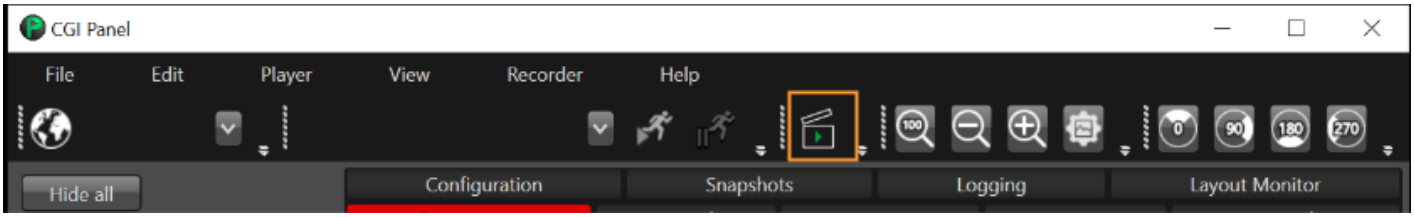
  The minimum and maximum range of values.

- **Period and Offset (ms)**

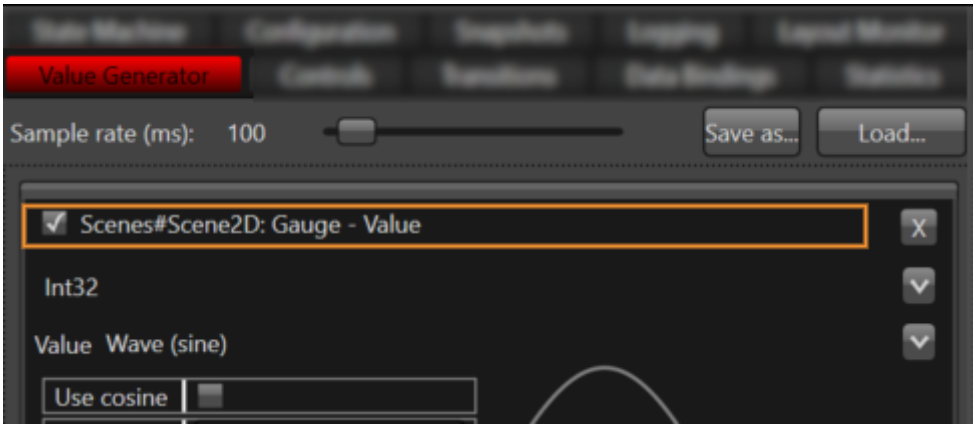
  The x value into the function is shifted by the offset. It shifts the graph, therefore the output value is also shifted. However, currently, offset is only supported in Sine wave. (limitation)

Starting the Value Generator simulation

Pressing the clapperboard icon will start and stop the simulation.

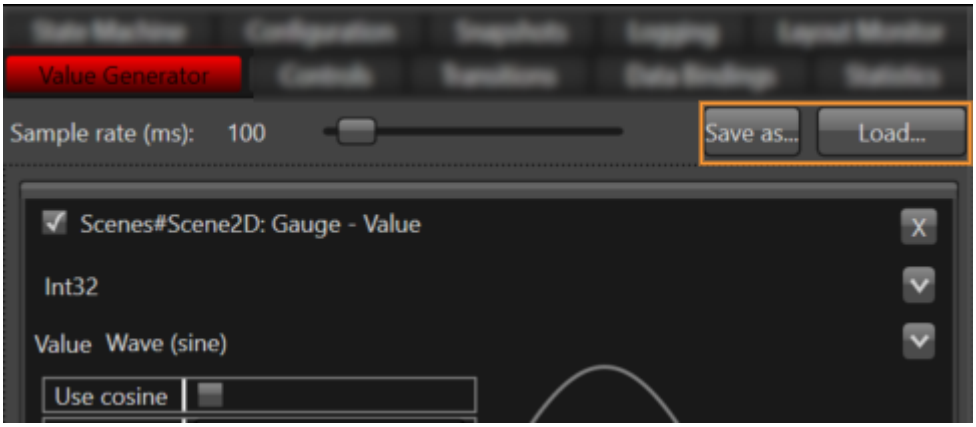


The simulation of a single property can be paused by the checkbox next to the property name. It can be completely removed by pressing the X button on the right side of the name.



Serialization

The simulation configuration can be saved as an xml file and loaded again.
It is possible to load a simulation file when loading an asset.



It can also be loaded at Panel startup by using the command line:
`--sim <filename>`
In this case, the simulation will be directly started.

Zooming

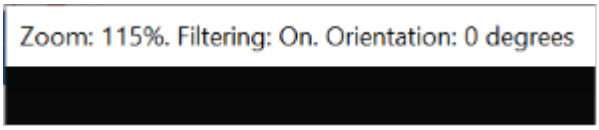
This feature was created to assist in developing high- or low-resolution HMIs. Developers of low-resolution HMIs can zoom in, developers of high-resolution HMIs can zoom out.

It may also be convenient if you want to take a closer look and examine the HMI in detail.



- The first icon of the toolbar resets the zoom level of every host application display.
- The second and third icon are for zooming out and zooming in. The zoom level is adjusted by 5% on every button click.
- The fourth option of the zooming toolbar is a filter that can be enabled to improve the visual output when zoomed in.

The current zoom factor and whether the filter is enabled is displayed in the Player's titlebar.



> Please be aware, that while using zoom, the output is not pixel exact anymore.

Panel

Rotate

This feature simulates a physical rotation of the display. The toolbar allows to rotate the Player's display clockwise. You can choose to rotate it by 0° (default), 90°, 180°, or 270°.



Panel

Data Bindings

In Data Bindings tab, you can load a predefined data binding file and perform various customisations and simulations using the Value Generator.



Example of data binding file (.xhcdl)

```
<bindingSources>
    <bindingSource name="PredefinedBindingSource1" access="asynchronous" uid="{491
        <item name="Item1" type="FeatStd::Variant" uid="{AF9A4C07-C3CE-44CC-9195-8
        <item name="Item2" type="FeatStd::Variant" uid="{7D6765D8-9117-4D63-A41A-A
        <item name="Item3" type="FeatStd::Variant" uid="{C5EE2EB6-B2E4-4123-B43F-B
        <item name="Item4" type="FeatStd::Variant" uid="{A00D262E-0E4C-4A49-8344-4
        <item name="Item5" type="FeatStd::Variant" uid="{4308D1C9-F458-4039-9C25-B
        <item name="Item6" type="FeatStd::Variant" uid="{0B6D6559-156E-431E-A5D2-B
        <item name="Item7" type="FeatStd::Variant" uid="{CED67FF4-061A-4643-9402-C
        <item name="Item8" type="FeatStd::Variant" uid="{E69CEF9F-6744-40C6-8A87-D
        <item name="Item9" type="FeatStd::Variant" uid="{83DCF8EF-592E-456F-9E27-B
        <item name="Item10" type="FeatStd::Variant" uid="{89A7D182-0EAD-4650-8AD2-
    </bindingSource>
```

1	Load XHCDL button	Use this button to load an xml file containing predefined data bindings. The file format is ***.xhcdl** and files created with the following structure in the file can be loaded. The following naming convention inside of the xhcdl is required: • bindingSource name: [any name]+BindingSource+[number] **See Section A in the above image** • item name: Item+[number] **See section B in the above image**
2	Name	The data binding item name specified by “item name” in the loaded xhcdl file is displayed. The names specified as “bindingSource name” in the xhcdl file that has been loaded are displayed in the above image in section 7.
3	Alias name	You can set a custom name for the data binding item name. This name will persist on Panel closing.
4	Value	Set the data binding value. After entering the value, a request to update the data binding value will be sent to the player.
5	Sim	Items with this checkbox checked can be simulated on the Value Generator tab.
6	Datatype	Change the data type. Select the any data type from the pull-down menu. After setting the data type, a request to update the data binding datatype will be sent to the player.

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

Command Line Usage

Description

The Player can also be used as standalone. In this mode the application has to be controlled by a command line or serial connection. It is a controlling instance and has access to the feature API. A similar approach is also used by the CGI Panel. However, then the interaction is packed into a UI. Parameters are set as numbers as additional value. E.g.: “show scene 1”

Commands

The command line currently supports the following commands:

Short command	Long command	Param	Description
h	help		Shows information about all features.
	list scenes		Lists all scenes and their IDs which can be used to show and hide scenes.
	show scene	1	Requires the ID of a scene. Shows the scene on the screen.
	hide scene	1	Requires the ID of a scene. Hides the scene.
	hide all scenes		Hides all scenes.
	list animations		Lists all animations and their IDs which can be used to start and stop animation.
	start animation	1	Requires the ID of an animation. Starts an animation.
	stop animation	1	Requires the ID of an animation. Stops an animation.
	list cultures		Lists all cultures and their IDs which can be used to switch the culture.
c	culture	1	Requires the ID of a culture. Switches to the selected culture.
	next culture		Switches to the next culture in the list.

	list controls	1	Requires the ID of a scene. Lists all controls of the scene.
	start script		Starts or pause script system.
	stop script		Stops the script system.
o	statoverlay		Shows render statistics overlay.
m	memstats		Shows memory statistics.
q	quit/shutdown		Terminates the application.

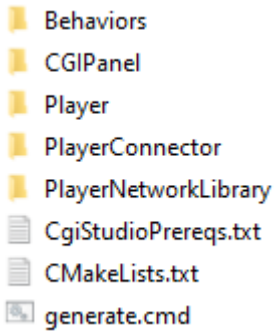
Application Structure

Introduction

This section gives a rough information of how this application is set up and where to find what.

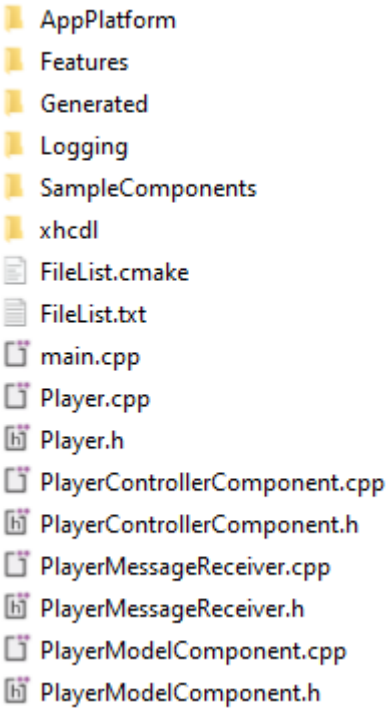
Main folder structure

The main folder shown here is <cgi-studio-root>/cgi_studio_player/src.
It contains the C++ core application, the C#-CGIPanel for a communication UI and sample behaviors.
Additionally a command exists to generate files out of associated xhcdl files. These files are used for data binding and message generation.



The Player Application

The core application has folders for the logical main parts of the application:



- **AppPlatform**: contains target/OS specific content. It defines the default input handler, asset type and location and can influence the display initialization.
- **Features**: contains all the Player specific features. They are not required for a running courier application but define the Player application.
- **Xhcdl**: contains the definitions for messages and databinding. With the generate.cmd in the main folder, these description files will be compiled into c++ header and source files.
- **Generated**: contains the generated files of the xhcdl files.
- **Logging**: contains content regarding logging. It is directly linked to the logging feature.
- **Sample Components**: contains sample implementations of different CGIStudio related features and samples.
- **Main folder**: represents the blank application. The Player related content has been reduced so that a generic courier application workflow can be derived.

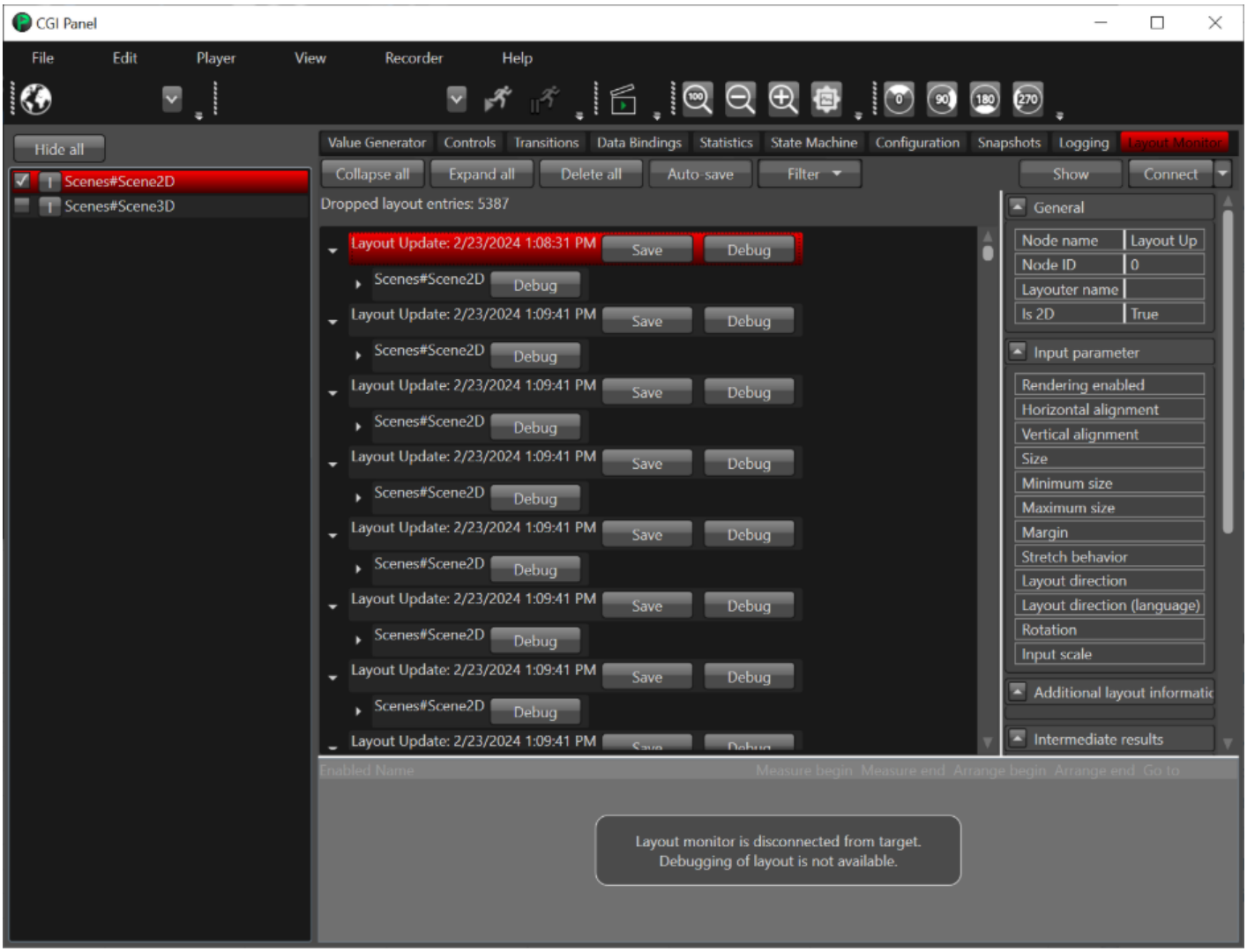
Layout Monitor

Introduction

The Layout Monitor is a feature that allows the user to see the relevant areas computed by the layouter as an overlay on top of the rendering of the application. The user can select nodes in the scene tree, the layouter rectangles are drawn automatically. The retrieval of various layout properties is available. Additionally the visualization is useful to understand how the dynamic layout is applied. The feature is available in the Player.

Layout Monitor user interface

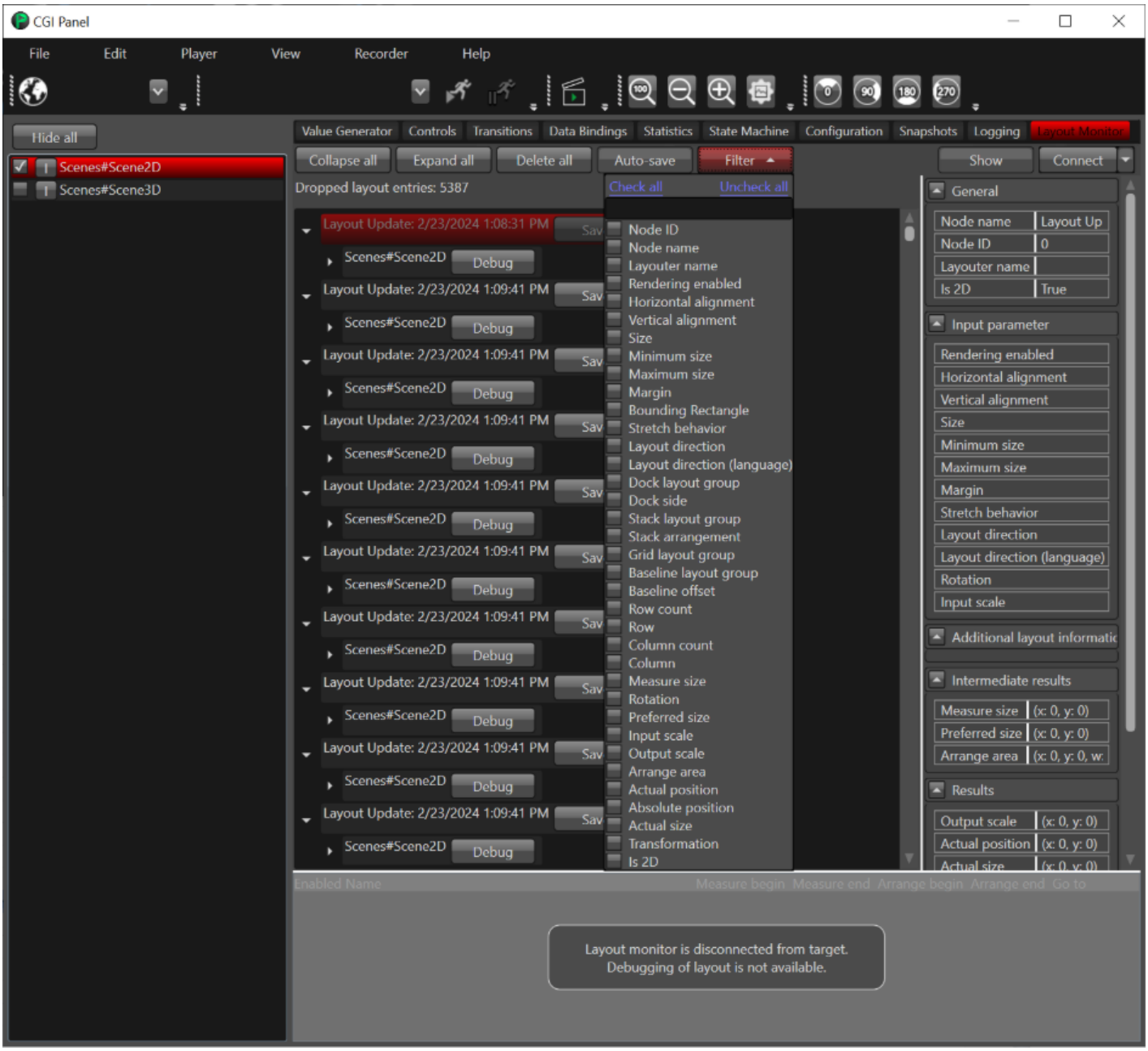
The figure below shows the Layout Monitor user interface.



In the middle of the window, all the layout trees are displayed. To customize the view, these can be individually collapsed or expanded by clicking the small arrows to the left of the layout tree. It is also possible to collapse, expand and delete all layout trees by clicking the buttons above them. By

clicking the Auto-save button, all incoming layout trees will automatically be saved to a location relative to the application workspace. The individual layout trees can be expended, collapsed and deleted by right clicking on them.

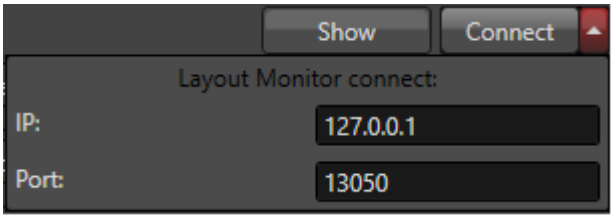
The layout trees can be filtered by clicking the Filter button as well as typing text in the textbox on top of the list. A dropdown list opens that enables choosing from various filters. These can be individually enabled or disabled. By clicking the according options in the list, all of the filters can be enabled or disabled at once as well. If a filter is enabled, it displays the according information inline with the layout tree. The figure below shows the filters available in Layout Monitor.



Clicking the Show button to the right shows the according overlay. For more information refer to Information shown in Layout Monitor. Clicking the Connect button connects to the target device. The connection is required always (also on host) and must be enabled before the Layout Monitor is used. The IP-Address and the Port used in connecting can be changed in the Settings window which can be opened by clicking File -> Settings... as well as by opening the drop down menu to the right of the Connect button. For more information refer to Connection Setup.

Connection Setup

A connection to the target or to host can be established by clicking the Connect button to the right of the window. It is possible to adjust the settings of the connection by clicking the dropdown button to the right of it. In the dropdown window the IP address and the Port can be set. After the connection is established, it can receive layout trees.



Layout Monitor specific settings

In the settings window, which can be opened by clicking File -> Settings... settings specific to the Layout Monitor can be changed. These settings are:

- The Layout entry directory is the default directory for saving logs which is used by the autosave- and recording-functions.
- The Max direct children is a threshold for maximal scene tree children, it needs to be changed only when more children are required in the scene tree.
- The IP-Adress and the Port indicate where the monitor connects to.
- The coordinate grid enables and disables the coordinate grid in the overlay.

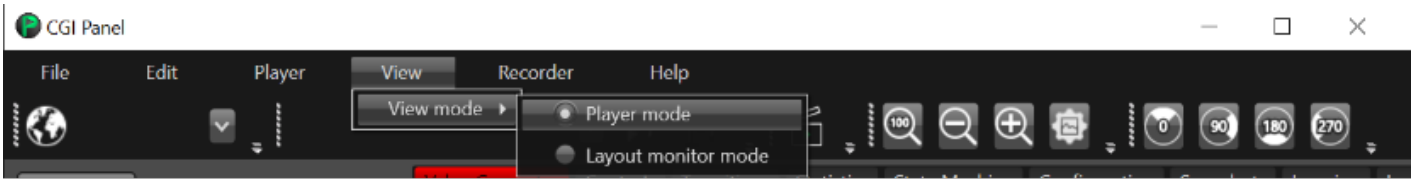
View Modes

The Layout Monitor mode can be used for applications that do not derive from the Player. These applications do not support all the other interfaces that would typically be required by the Player.

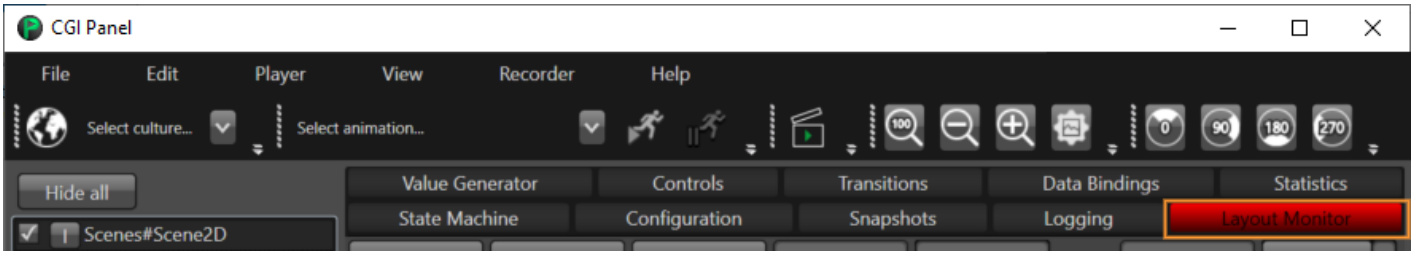
The Layout Monitor can be used in both of the CGIPanel modes:

- Player mode
- Layout Monitor mode (standalone mode)

The mode can be selected using the View -> View mode option.

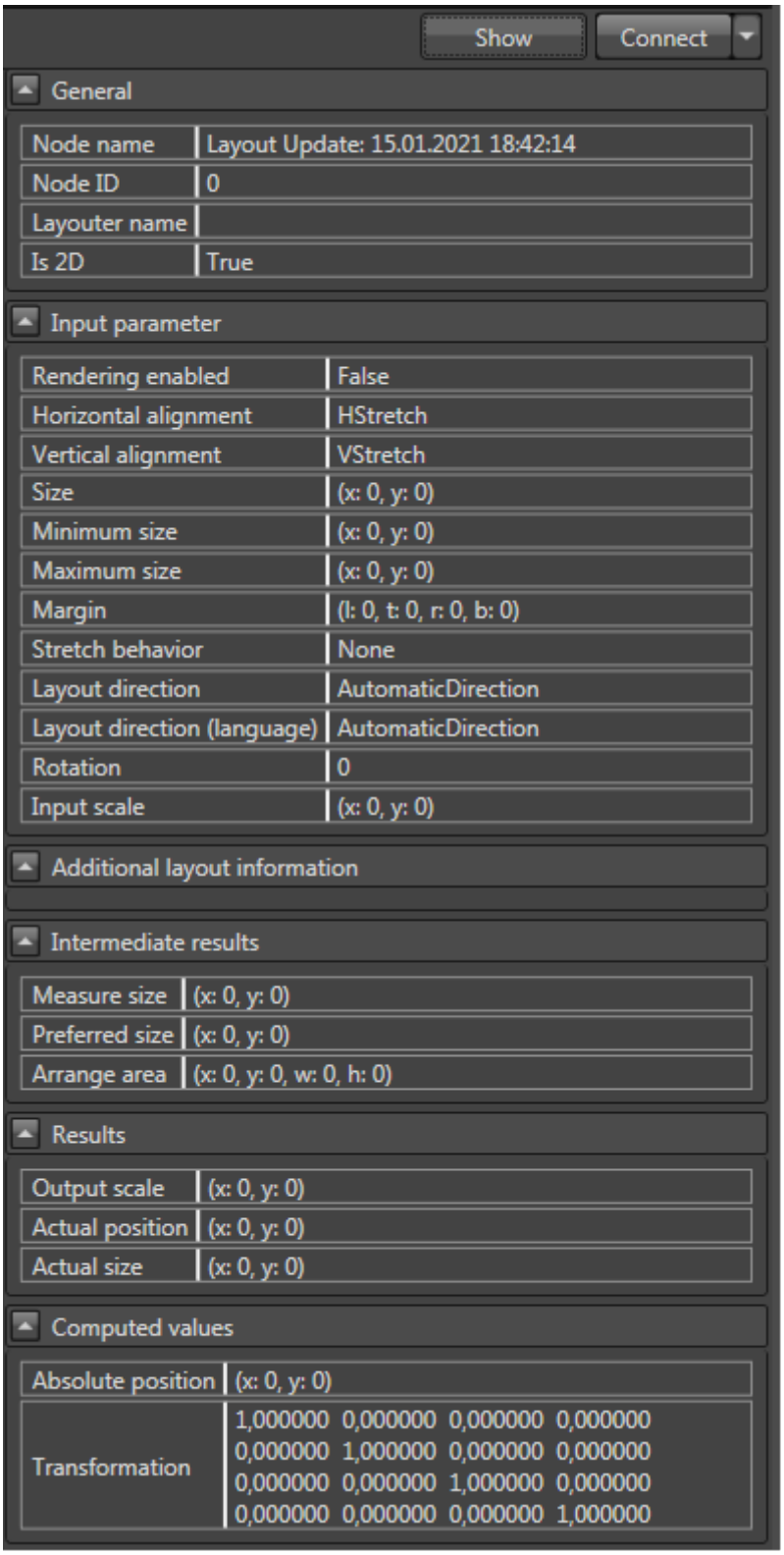


In Player mode, Layout Monitor can be accessed by pressing the "Layout Monitor" button, shown in the figure below.



Information shown in Layout Monitor

By clicking the Show button, Layout Monitor shows various information of the selected layout tree and also the overlays.



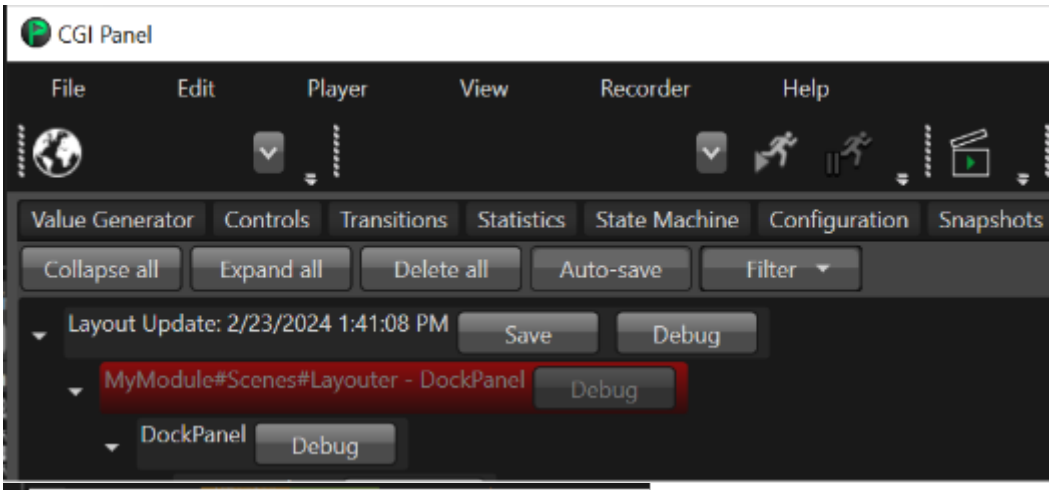
- The General drop-down field shows general information from the received layout tree.
- The Input parameter drop-down field shows information set within SceneComposer.
- The Additional layout information shows miscellaneous information that do not fit in the other categories.
- Intermediate values are values used as an intermediate result for the end results.
  - For example the Measure size defines the size a node would like to have under specific layout preconditions.
- The Results section shows the final values that define the node element.
- The values shown in the Computed values drop-down field are computed directly on the host, not on the target.

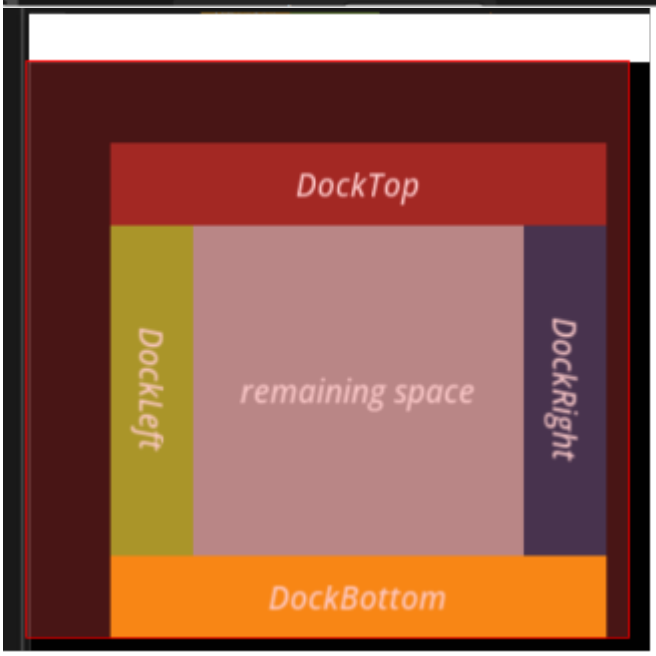
For every individual value there is also a tooltip available that explains the values in more detail.

By selecting nodes of the layout tree, differently colored rectangles are shown as an overlay on the screen.

> These rectangles are a host-only overlay. If they should be applied to an application displayed on the target, screenshots have to be used. For more information on how to make screenshots, please refer to Player. To use a screenshot with the overlay, show the screenshot and the overlay. Then move the overlay over the screenshot.

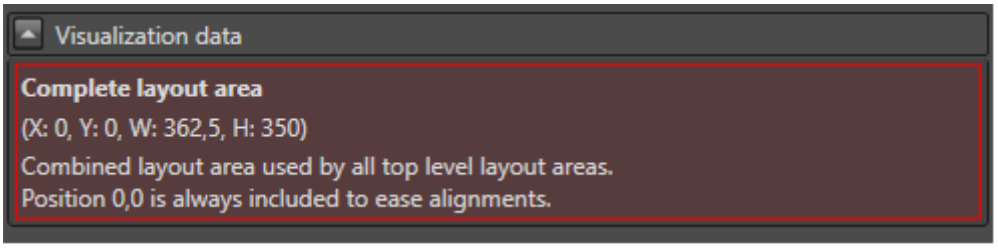
If the top node is selected, a red rectangle appears. This rectangle can be moved freely and is to be placed atop the displayed layout. The top left corner of the overlay should be placed on the top left corner of the Player screen. This is also shown in the figure below. If the mouse is hovered above the rectangle, a tooltip appears that shows its size and position.



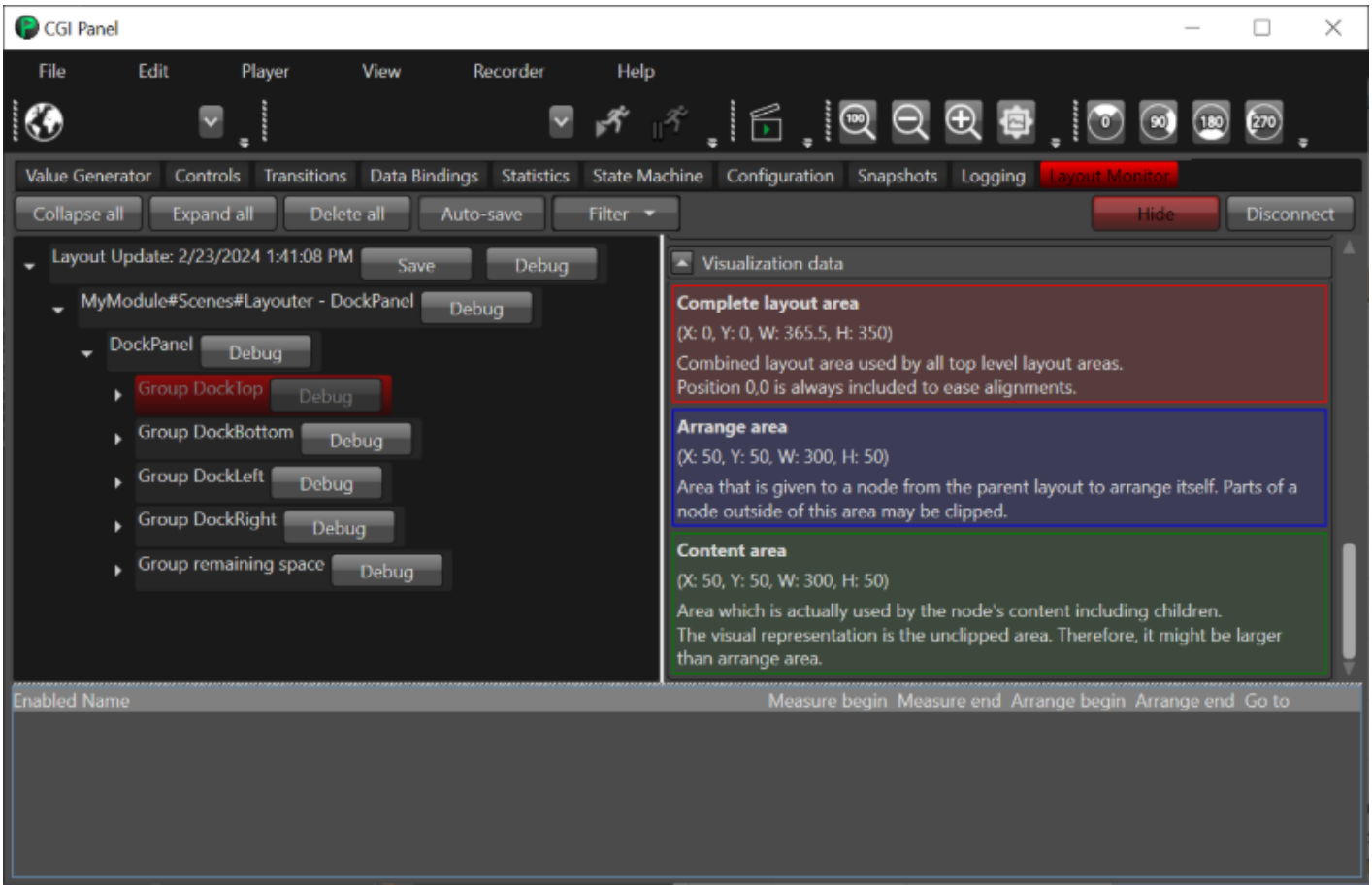


Additionally to the layout, the field Visualization data appears to the right of the user interface. This field shows the size, position and a description of the overlay. To help distinguishing between the different overlays, the different entries in the visualization data section of the information panel to

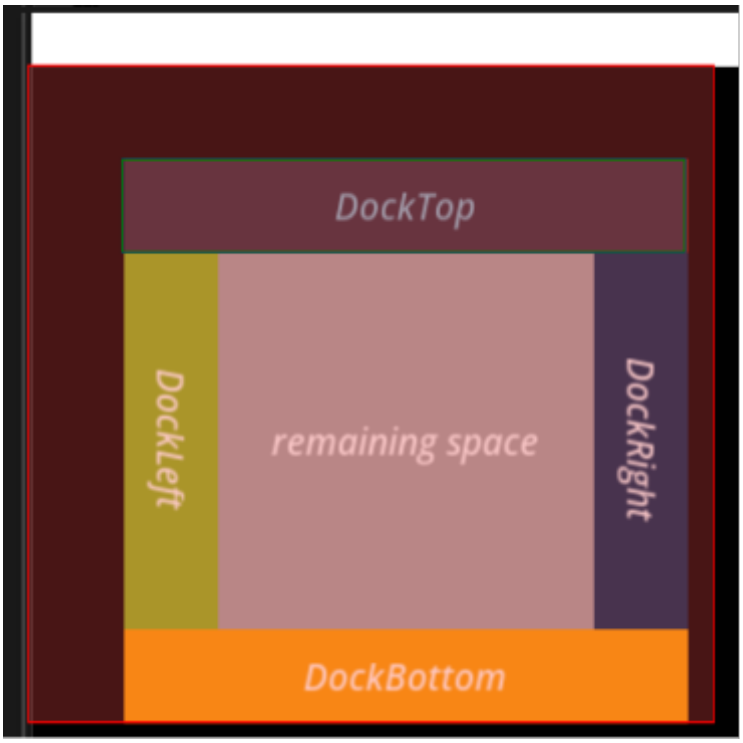
the right have the same color as the according overlay. In case of the top layer overlay, the field looks as displayed in the figure below.



If a sub-node is selected in Layout Monitor, rectangles appear that display all included parent nodes. As an example, SolidColorNode is selected. Now the Visualization data dropdown field contains three entries of the three shown overlays, their data and their description.



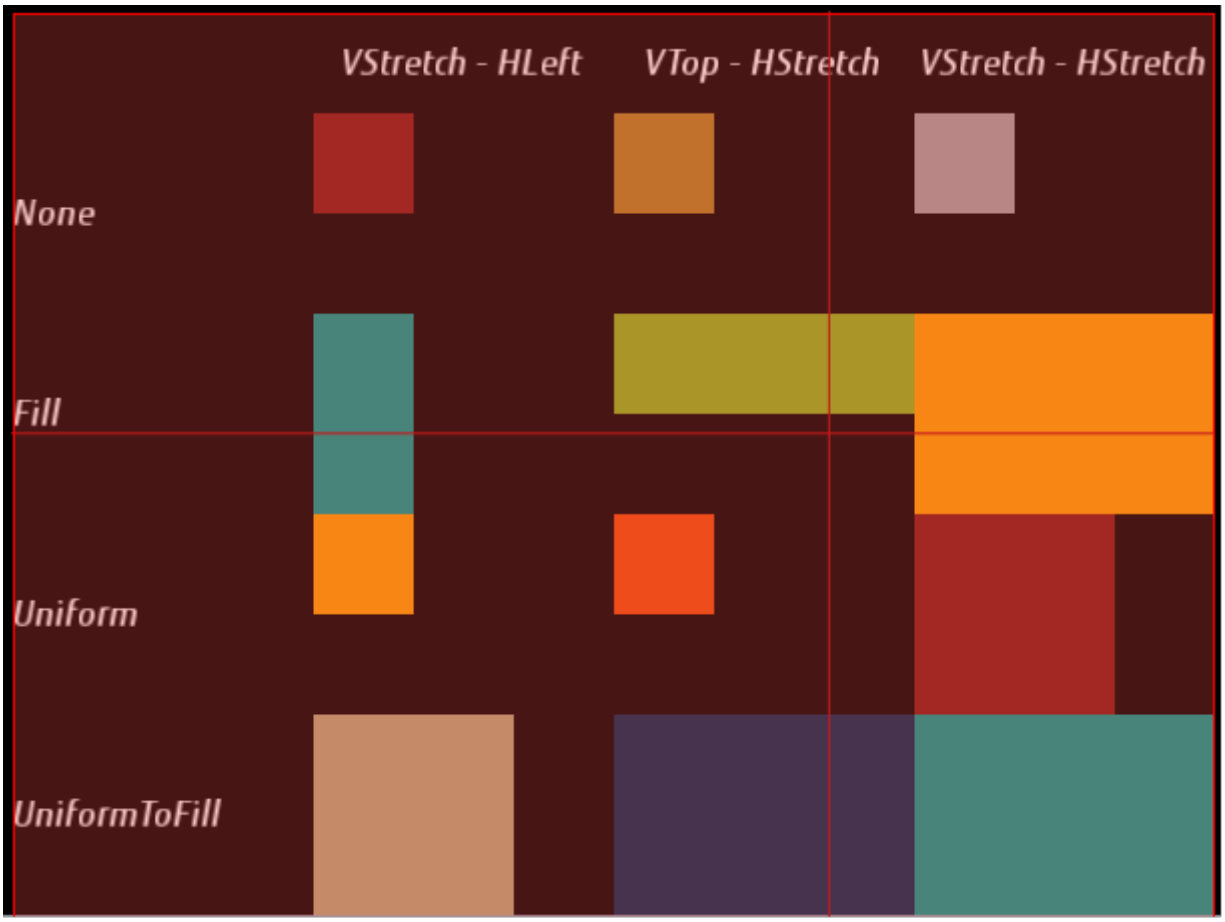
Now there are three overlays that show how the layouts are set.



It is also possible to show the axis (which is important for 3D usage). The axis serves as an aid to make the alignment of the overlays easier. The axis are always positioned to the 0,0,0 position. Using the mouse to hover over the show/hide button shows a menu which allows the user to

- enable/disable the axis
- set the axis color
- enable/disable the blinking

As is shown below, white lines appear which are the X and Y axis that indicate the 0/0 position and help aligning the overlay.



Debugging using Layout Monitor

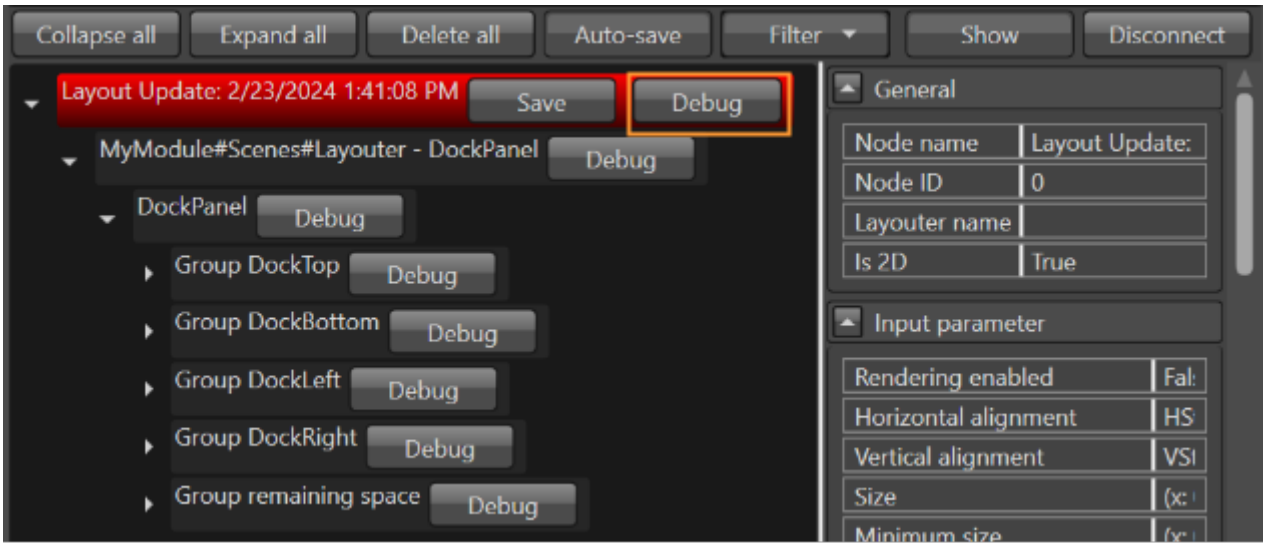
It is possible to debug either individual nodes or a parent node and its child nodes. To do this, simply press the debug button to the right of the node. If the node has child nodes, these will be debugged too.

The preconditions necessary for debugging are:

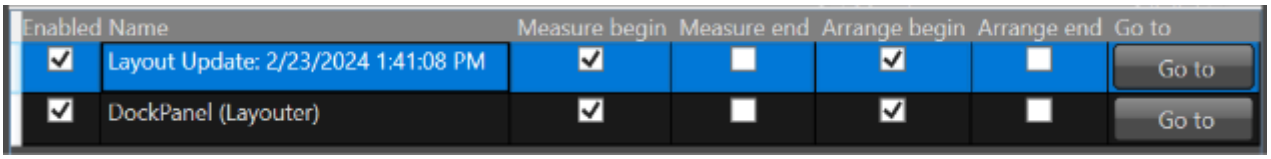
- Debug build of application
- JIT debugging must be enabled in Visual Studio
- Visual Studio must be present

Activate just-in-time-debugging in Visual Studio following these steps:

- In the Tools or Debug menu, select Options -> Debugging -> Just-in-time-debugger
- In the Enable Just-in-time debugging settings, ensure that "Native" is enabled



Clicking the debug button will cause breakpoints to appear on the bottom of the window. In this example, two breakpoints are set.



There are several options available for setting up breakpoints, which can be enabled or disabled using the checkoxes, as indicated in the Figure above. The options are:

- Enabled: breaks when node rendering is enabled or disabled
- Measure begin: breaks at the beginning of the measurement phase
- Measure end: breaks at the end of the measurement phase
- Arrange begin: breaks at the beginning of the arrange phase
- Arrange end: breaks at the end of the arrange phase

The breakpoints are actual code breakpoints so that Visual Studio is opened.

The Go to button selects the node in the layout tree.

To completely remove a breakpoint, select it so that it is marked blue and press the Delete button on the keyboard.

Saving the layout tree

It is possible to save the layout tree by clicking the Save button to the right of the main layout tree. It is then persistently available. Drag and drop can be used to import the previously saved tree into the Layout Monitor.

How to integrate Layout Monitor in existing custom applications

As a tutorial on how to integrate Layout Monitor into a custom application, please see following code sample:

LayoutMonitor.h must be included:

```
// Main include
#include <Candera/System/Diagnostics/LayoutMonitor.h>

// includes for the threading
#include <FeatStd/Platform/TcpServer.h>
#include <FeatStd/Platform/CriticalSectionLocker.h>
#include <FeatStd/Platform/Thread.h>


class LayoutMonitorListener : public FeatStd::Internal::Thread {
    public:
        static LayoutMonitorListener& GetInstance()
        {
            FEATSTD_UNSYNCED_STATIC_OBJECT(LayoutMonitorListener, s_inst);
            return s_inst;
        }

        void Stop()
        {
            {
                FeatStd::Internal::CriticalSectionLocker l(&m_memberSection);
                if (m_server != 0) {
                    m_server->Close();
                }
            }
            Candera::Diagnostics::LayoutMonitor::CloseStream();
        }

        bool Start(FeatStd::UInt32 port)
        {
            if (CreateServer(port)) {
                return Run();
            }
```

```
            return false;
        }

        LayoutMonitorListener() :m_server(0)
        {
            Candera::Diagnostics::LayoutMonitor::Register();
        }

        ~LayoutMonitorListener()
        {
            Candera::Diagnostics::LayoutMonitor::Unregister();
        }
    protected:
        bool CreateServer(FeatStd::UInt32 port)
        {
            FeatStd::Internal::CriticalSectionLocker l(&m_memberSection);
            if (m_server != 0) {
                return false;
            }
            m_server = FeatStd::Internal::TcpServer::Create(port);
            return m_server != 0;
        }

        virtual FeatStd::Int ThreadFn() override
        {
            do {
                FeatStd::Internal::IO::Stream * localStream = m_server->Accept();
                if (localStream != 0) {

Candera::Diagnostics::LayoutMonitor::Run(FeatStd::Internal::IO::SharedStream::Create(localStre
am));
                }
            } while (m_server->IsOpen());
            if (m_server != 0) {
                FeatStd::Internal::CriticalSectionLocker l(&m_memberSection);
                FEATSTD_SAFE_DELETE(m_server);
            }
            Candera::Diagnostics::LayoutMonitor::CloseStream();
            return 0;
        }
    private:
```

```
        mutable FeatStd::Internal::CriticalSection m_memberSection;
        FeatStd::Internal::TcpServer * m_server;
    };
```

To start the monitor:

> return LayoutMonitorListener::GetInstance().Start(CGIAPP_LAYOUTMONITOR_PORT); // default port: 13050

To stop the monitor:

> LayoutMonitorListener::GetInstance().Stop();
