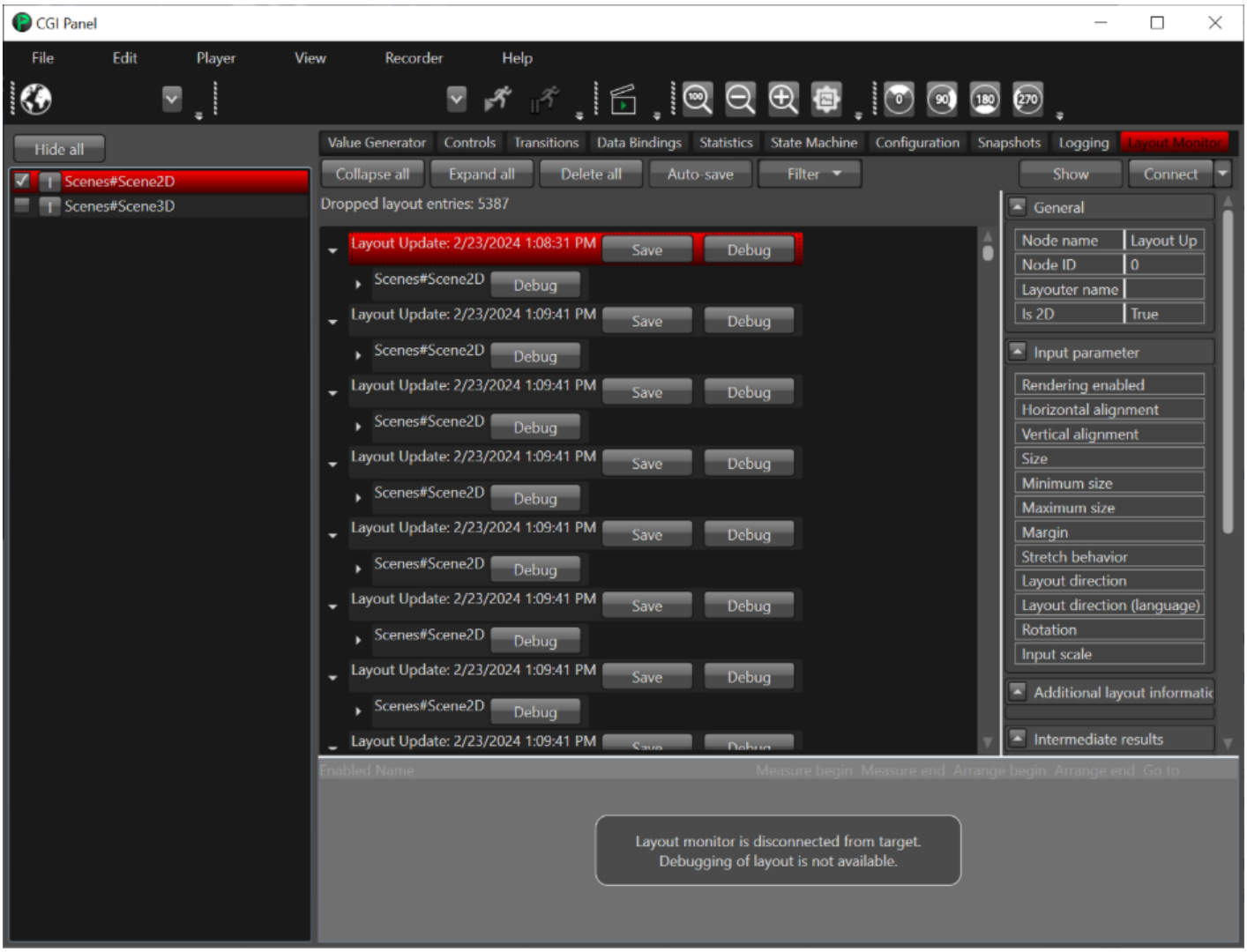
Layout Monitor

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

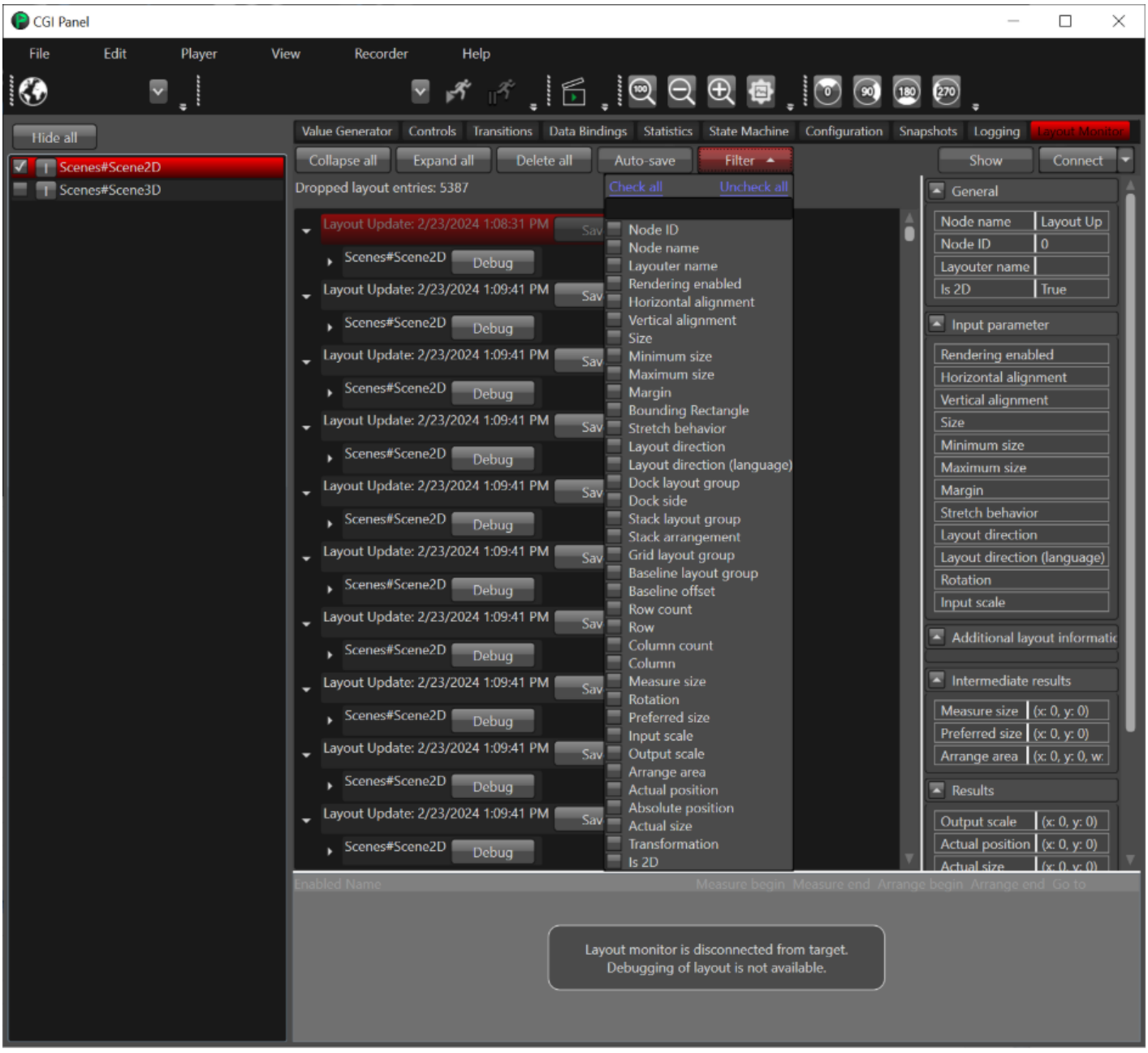
The figure below shows the Layout Monitor user interface.



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.

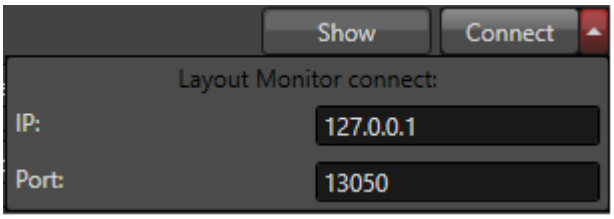
The layout trees can be filtered by clicking the Filter button as well as typing text in the textbox on top of the list. A dropdown list opens that enables choosing from various filters. These can be individually enabled or disabled. By clicking the according options in the list, all of the filters can 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.

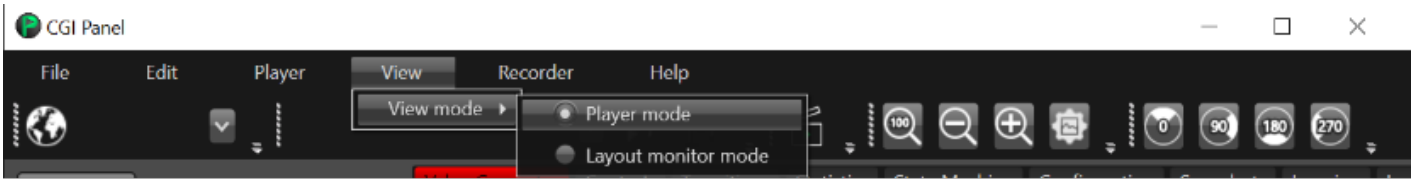
View Modes

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.

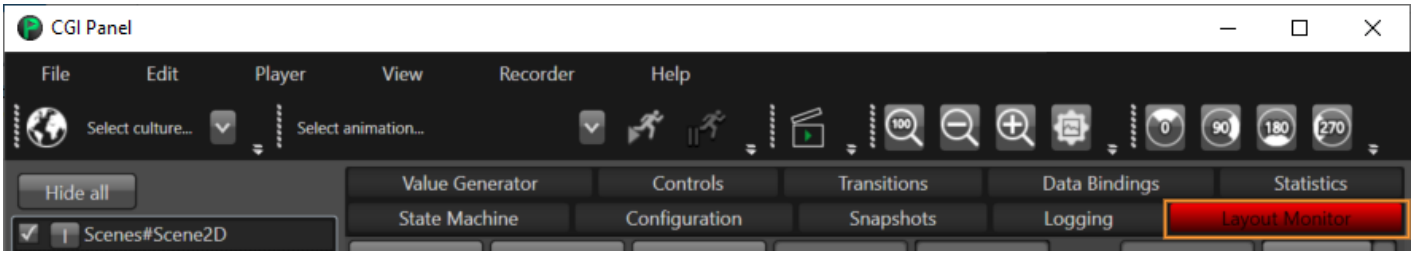
The Layout Monitor can be used in both of the CGIPanel modes:

- Player mode
- Layout Monitor mode (standalone mode)

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

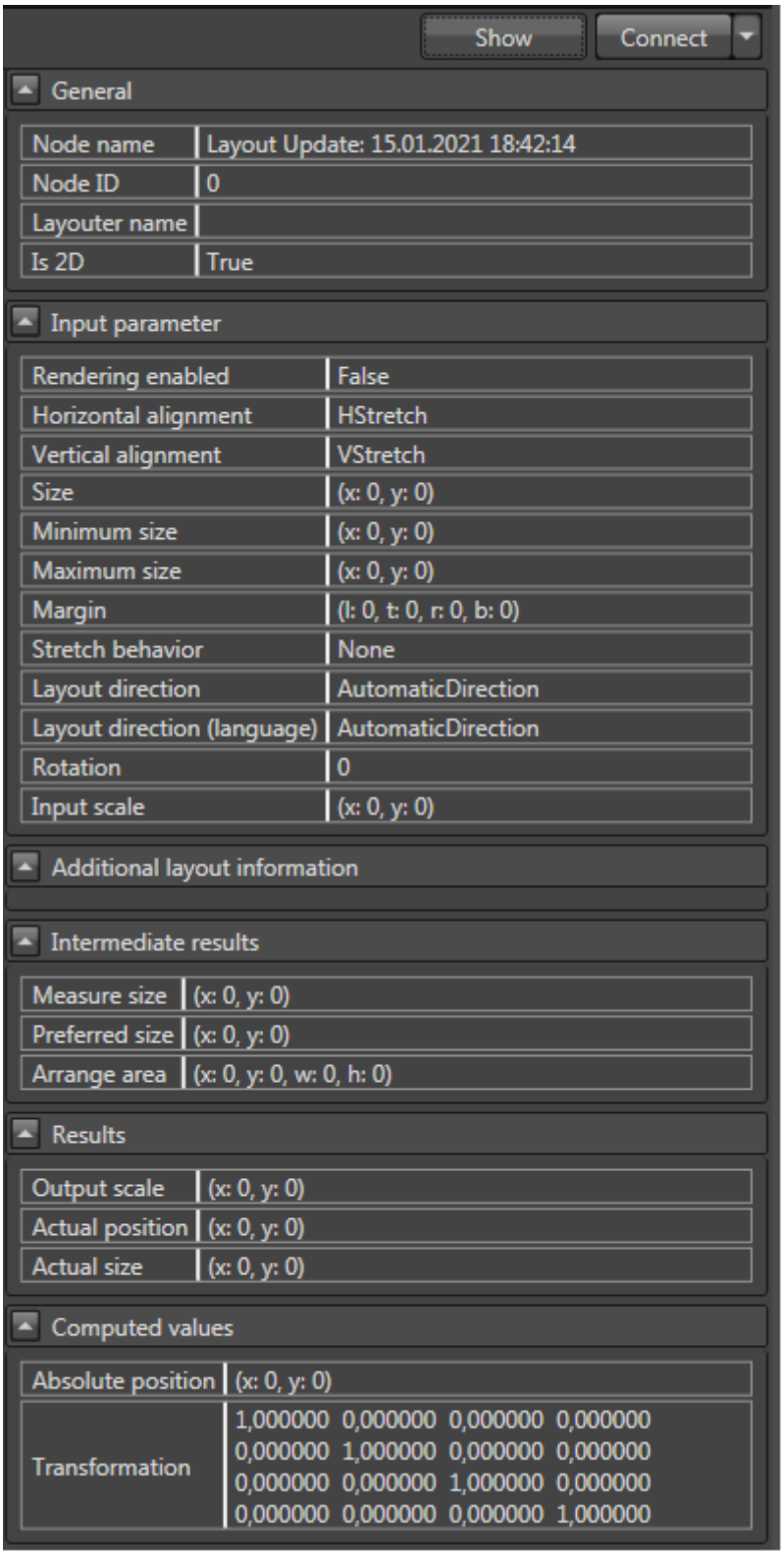


In Player mode, Layout Monitor can be accessed by pressing the "Layout Monitor" button, shown in the figure below.



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.

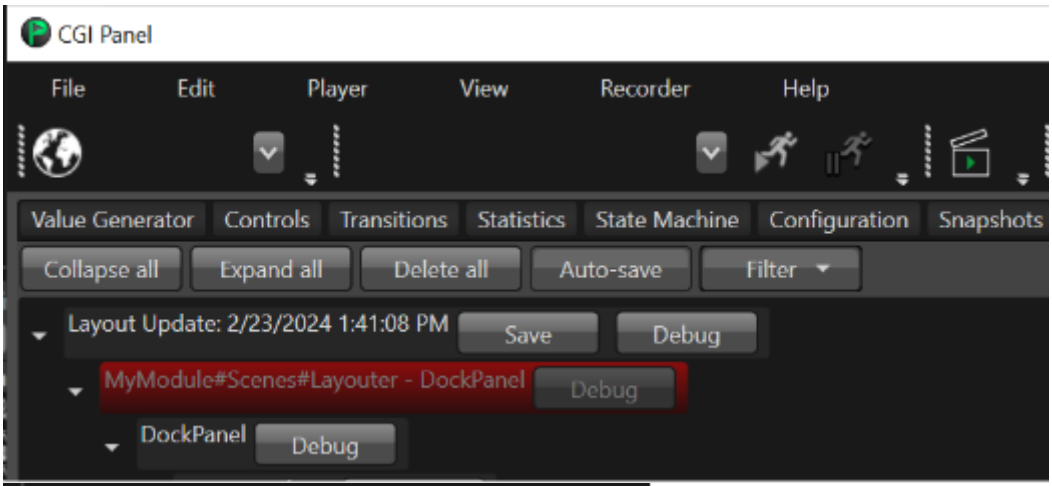
- The values shown in the Computed values drop-down field are computed directly on the host, not on the target.

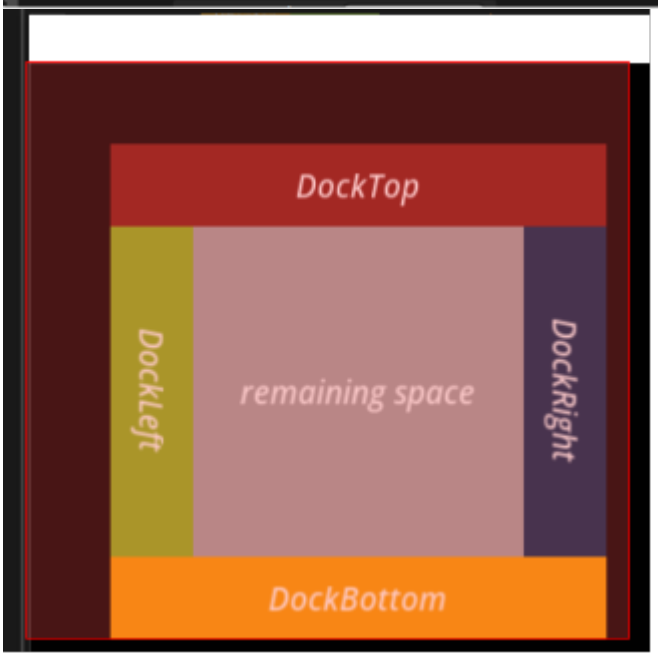
For every individual value there is also a tooltip available that explains the values in more detail.

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

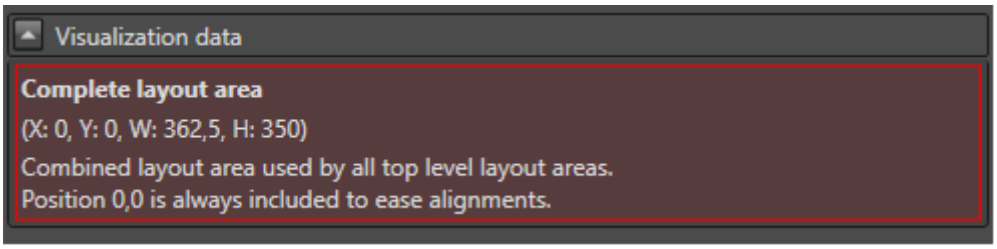
> These rectangles are a host-only overlay. If they should be applied to an application displayed on the target, screenshots have to be used. For more information on how to make screenshots, please refer to Player. To use a screenshot with the overlay, show the screenshot and the overlay. Then move the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

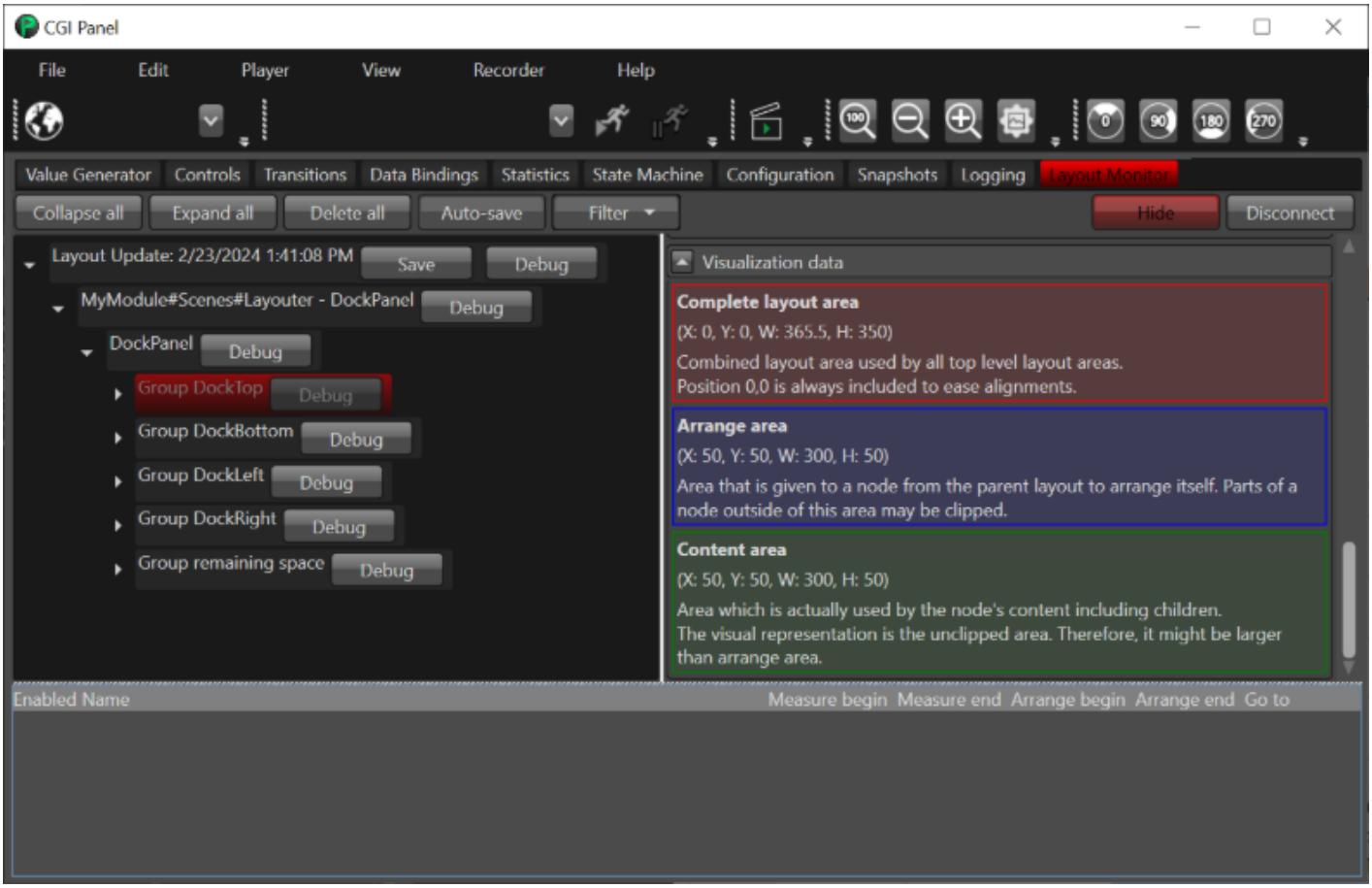




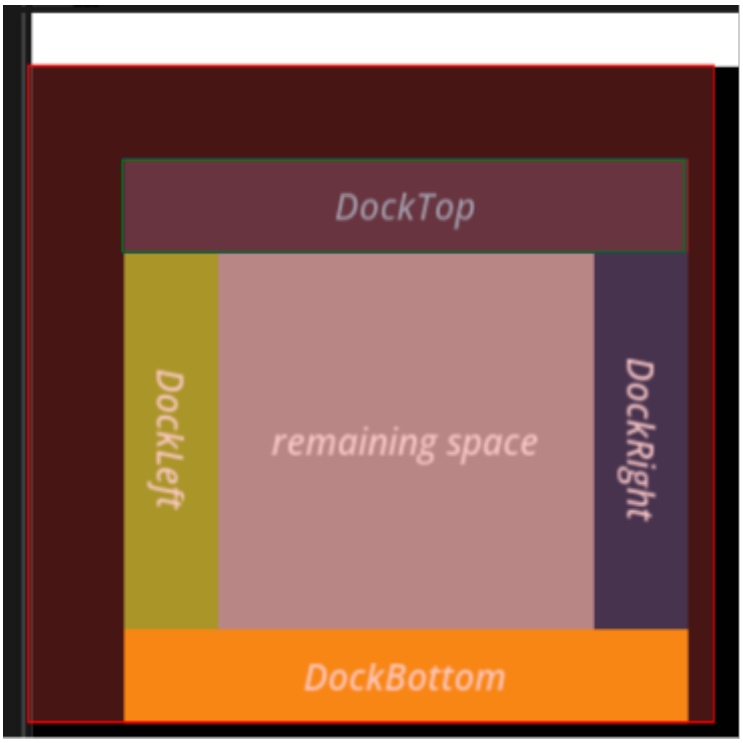
Additionally to the layout, the field Visualization data appears to the right of the user interface. This field shows the size, position and a description of the overlay. To help distinguishing between the different overlays, the different entries in the visualization data section of the information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



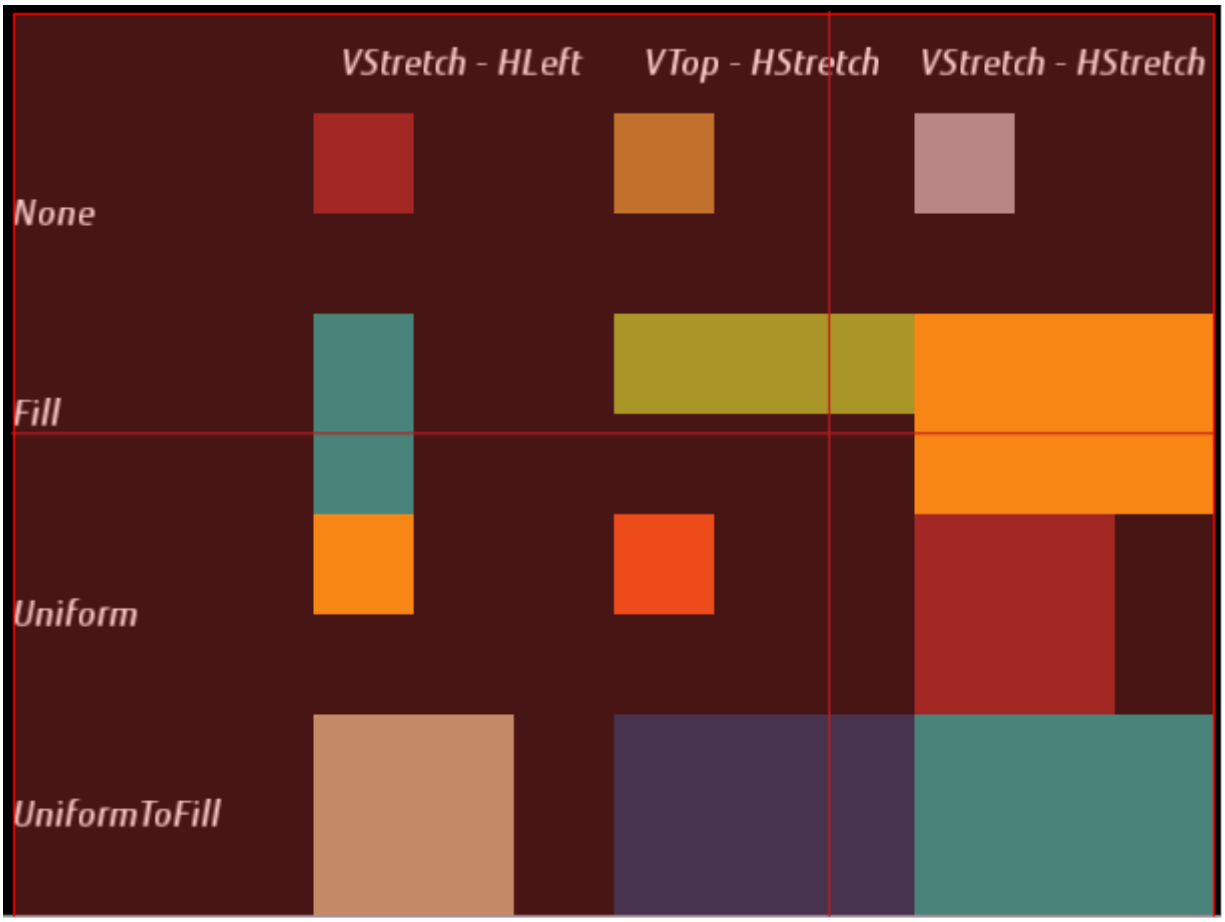
Now there are three overlays that show how the layouts are set.



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

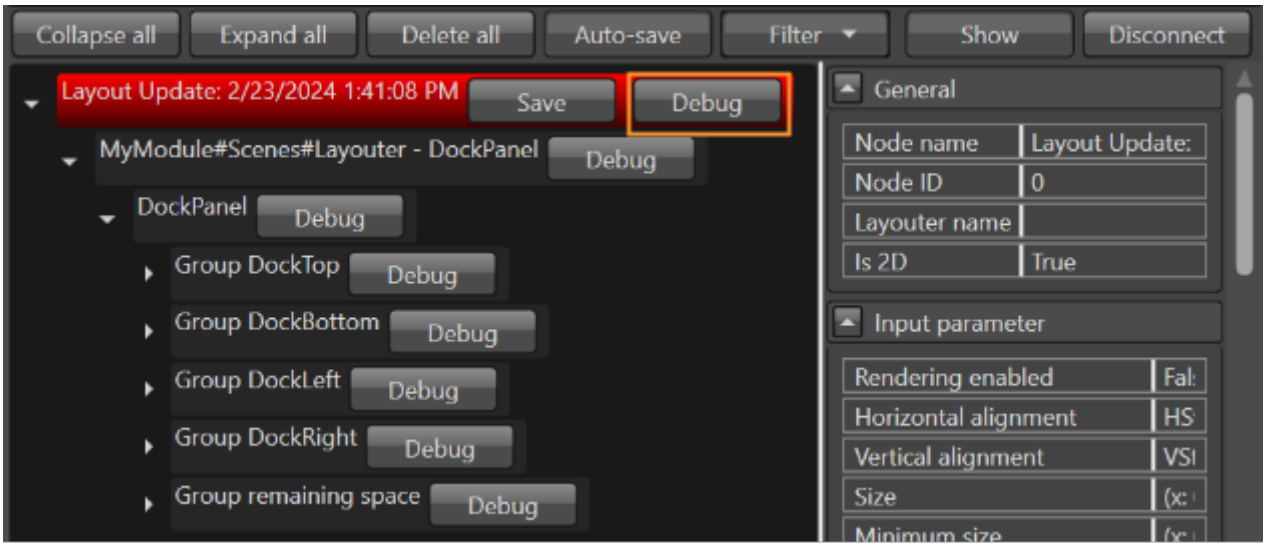
It is possible to debug either individual nodes or a parent node and its child nodes. To do this, simply press the debug button to the right of the node. If the node has child nodes, these will be debugged too.

The preconditions necessary for debugging are:

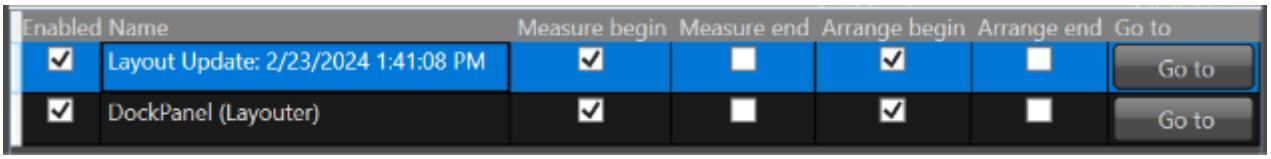
- Debug build of application
- JIT debugging must be enabled in Visual Studio
- Visual Studio must be present

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();

Revision #6
Created 2023-06-30 08:33:59 UTC by Elisabeth Zauner
Updated 2025-01-15 05:25:53 UTC by Tsuyoshi.Kato