Yamcs Studio User Guide

YAMCSSTCORE-SA-MA-001

February, 21st
2019

www.yamcs.org

Yamcs Studio User Guide

YAMCSSTCORE-SA-MA-001

This version was published February, 21st 2019
A later version of this document may be available at www.yamcs.org

© Copyright 2015 – Space Applications Services, NV

Table of Contents

1. Introduction **3**

Overview 3

Installation 4

First Steps 5

Understanding the User Interface 7

Connecting to Yamcs 10

2. Display Runner **13**

3. Processed Variables **15**

Local PVs 15

Parameters 15

Software Parameters 16

Simulated Values 16

Formulas 16

4. Widgets **18**

Catalogue of Widgets 18

Color Decorations 19

5. Inspector Windows **20**

PV List 20

PV Info 21

OPI Probe 23

6. Editing Displays **24**

Display Runner 24

Explorer 26

Editor Area 29

Outline 32

Properties 33

Palette 35

Actions 54

Rules & Scripts 56

OPI Schema 60

7. Views **63**
Archive 63
Event Log 67
Alarms 68
Command Stack 69
Command History 74
Command Queues 76
Yamcs Clients 77
Data Links 78

8. Tuning **79**
Command Options 79
Capturing Log Output 80
Preference Defaults 81

Chapter 1. Introduction

1.1. Overview

Yamcs Studio

Yamcs Studio is a TM/TC front-end for Yamcs. It contains an editor for building synoptic operator displays and supports basic telecommanding.

License

Yamcs Studio follows a similar licensing scheme as Yamcs Server. The core of Yamcs Studio is open-source and licensed under the Eclipse Public License. Mission-specific extensions can be developed on a case-by-case basis and under custom licenses.

1.2. Installation

Prerequisites

Java 8 or higher must be installed. We currently recommend the latest Oracle JDK 8.

64-bit packages are generated for Linux, Windows and Mac OS.

Install Manually

Download the latest Yamcs Studio release for your platform. Extract to your preferred location and launch it. When it asks you to choose a workspace, choose a new directory where you have write rights , e.g. under your home directory. Workspaces contain displays, scripts and user preferences. By default your workspace will be populated with a few sample projects. These projects contain displays that show simulated parameters as produced by a default-configured Yamcs Server.

Install from Repository

On Linux distributions you can install the open-source version of Yamcs Studio via a package repository. Configure the Yamcs repository appropriate to your distribution following the repository instructions.

RPM (RHEL, Fedora, CentOS)

Install via dnf (or yum on older distributions)

```
$ dnf check-update
$ sudo dnf install yamcs-studio
```

RPM (SLE, openSUSE)

```
$ sudo zypper refresh
$ sudo zypper install yamcs-studio
```

Troubleshooting

Most problems related to starting Yamcs Studio have to do with Java not being correctly detected, or by trying to launch Yamcs Studio with an old version of Java. Both of these issues are usually resolved by installing Oracle JDK 8.

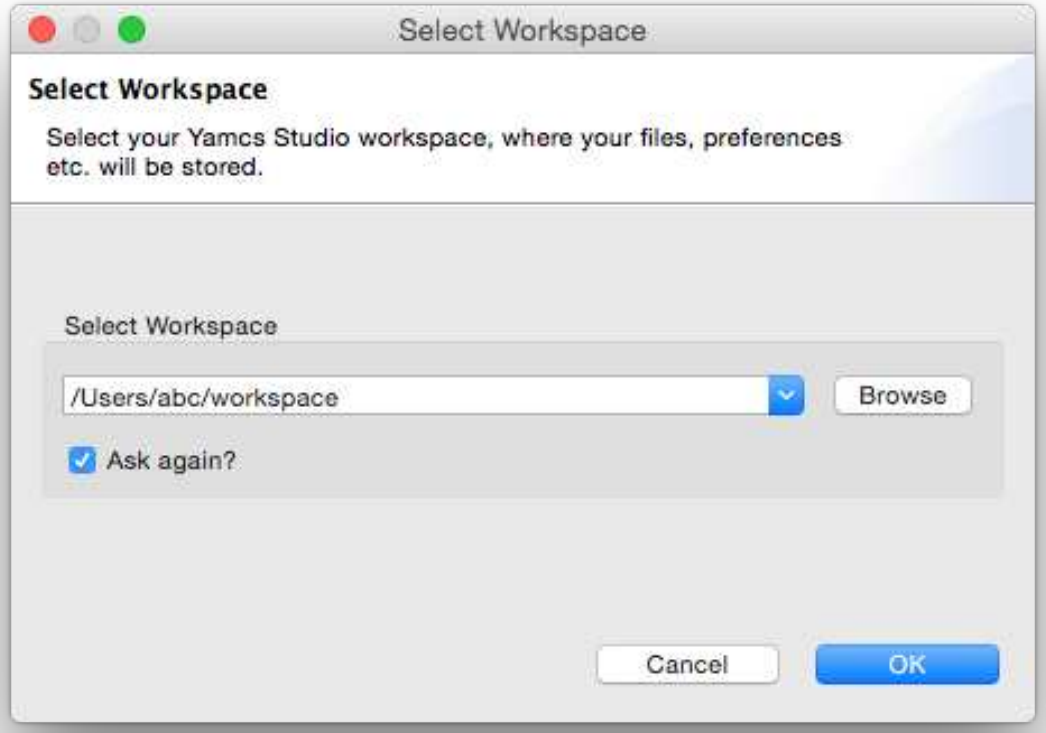
In case that didn't help, Try defining the -vm property in the root Yamcs Studio.ini file. Refer to the instructions available at https://wiki.eclipse.org/Eclipse.ini.

1.3. First Steps

Launching Yamcs Studio

When you launch Yamcs Studio for the first time it will ask you to choose a workspace. A **workspace** is where you store your resources (e.g. a display file).

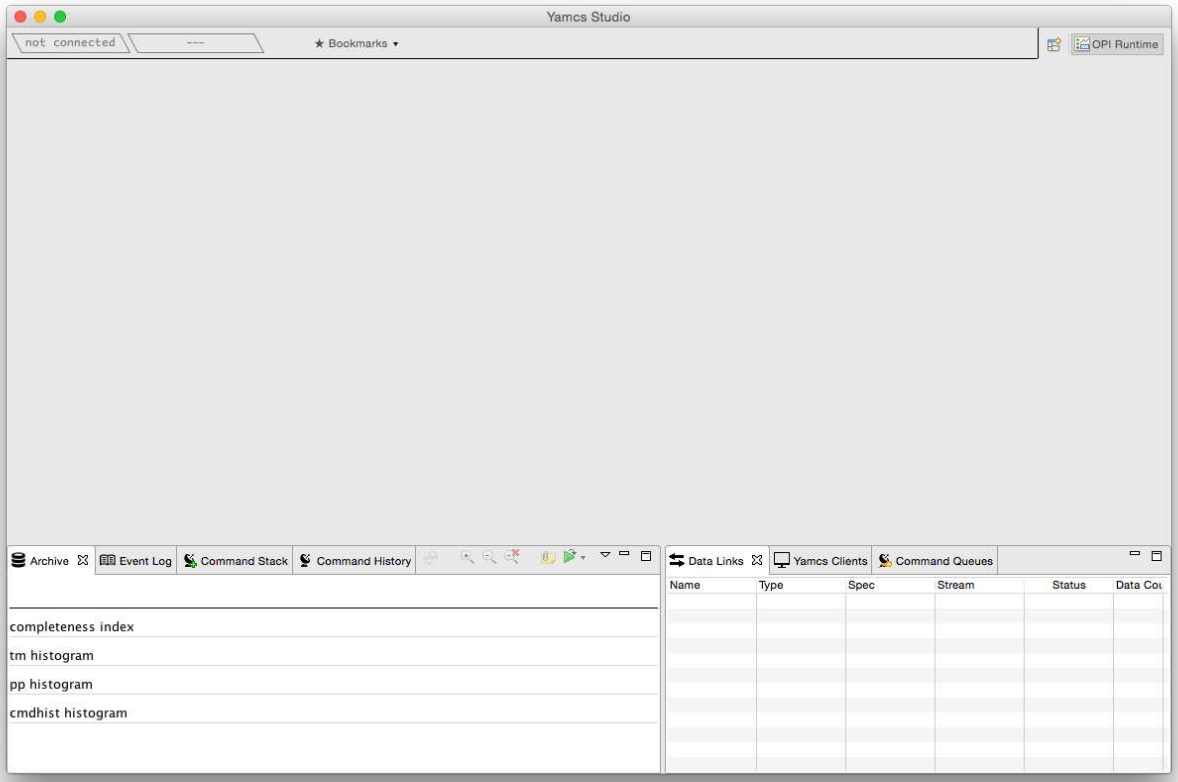
With Yamcs Studio, you are always working on one workspace at a time. Usually workspaces are fairly static, and you can often do with just one of them.



Choose your preferred location, and click **OK**.

Empty Workspace

Yamcs Studio is now launched and you should see an empty workspace with the default window arrangement:



The empty area in the middle is where displays will open.

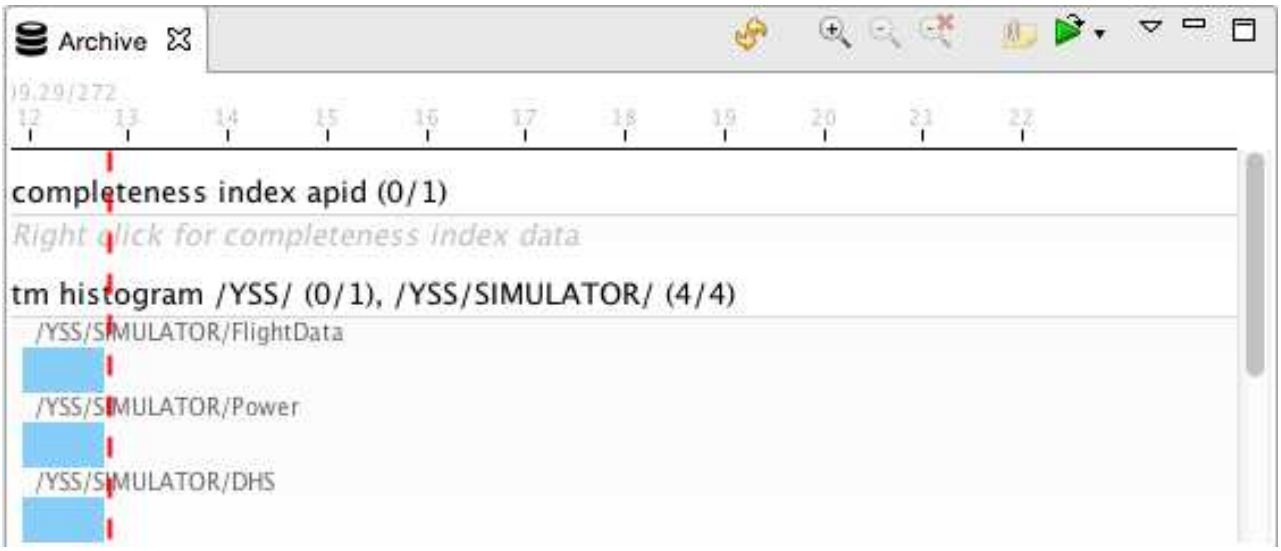
Yamcs Studio has two different windows. Display Builder and Display Runner. When Yamcs Studio is launched for the first time on a new workspace the user will be welcomed with the default Display Builder window.

1.4. Understanding the User Interface

Yamcs Studio is composed out of multiple views that are arranged together in windows. The user has great flexibility in modifying the default arrangement.

Views

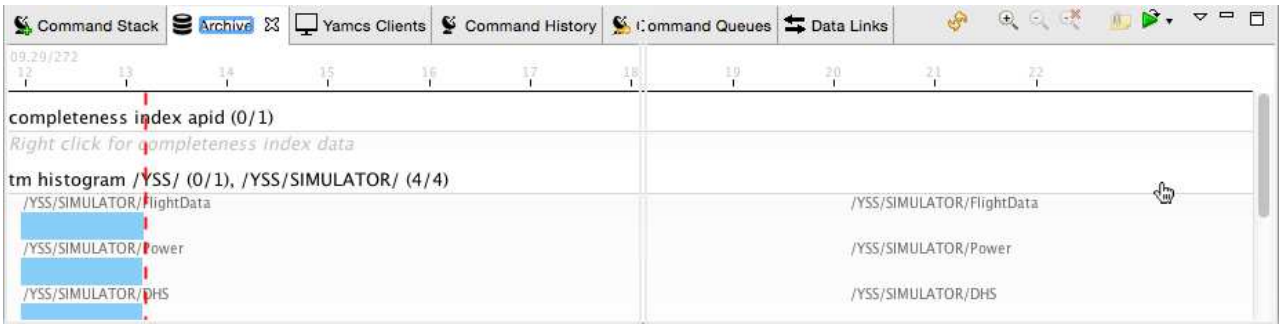
Views all share the same user interface organization. On the left you see a tab with the view icon, followed by a title, and then a close icon. On the outer right there are actions to **Minimize** or **Maximize** the view. Some views (such as the one in the screenshot) also have a third pull-down icon with view-specific actions in it. Most views, though, add dedicated colored icons next to the standard icons. The pull-down menu is used to hide less-often used actions.



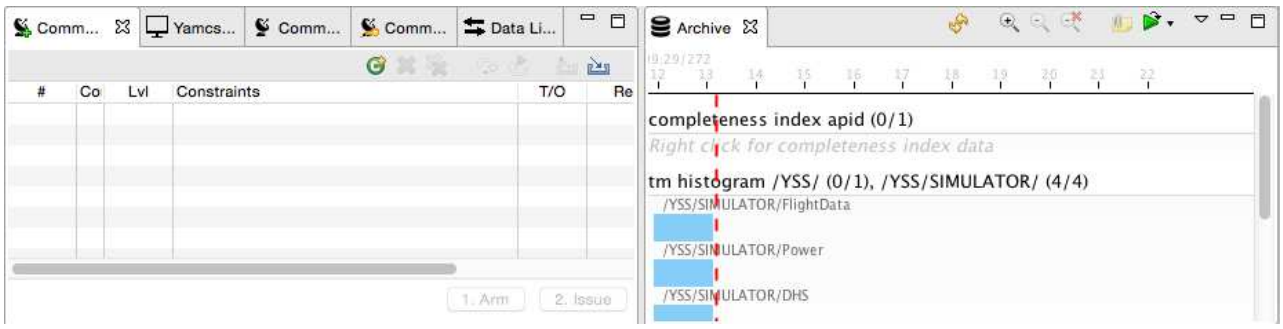
To reopen a view which you closed earlier, or to open another view choose **Window > Show View**.

Views can be resized, moved and stacked. This allows you to customize your workspace to your own personal preference.

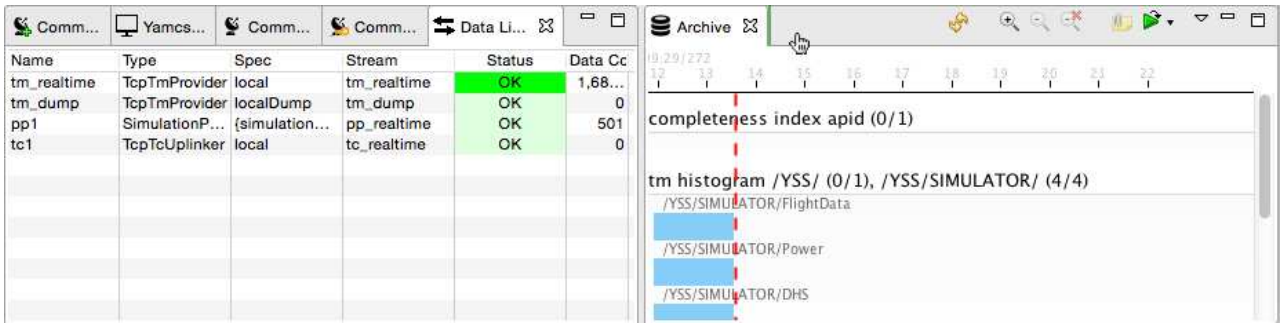
For example, let's say we want to put the Archive view in its own dedicated location. Click on the tab title, and while holding the mouse down, drag towards the right. If you move far enough, you will notice an outline suggesting the view's new position (this may look slightly different on your platform).



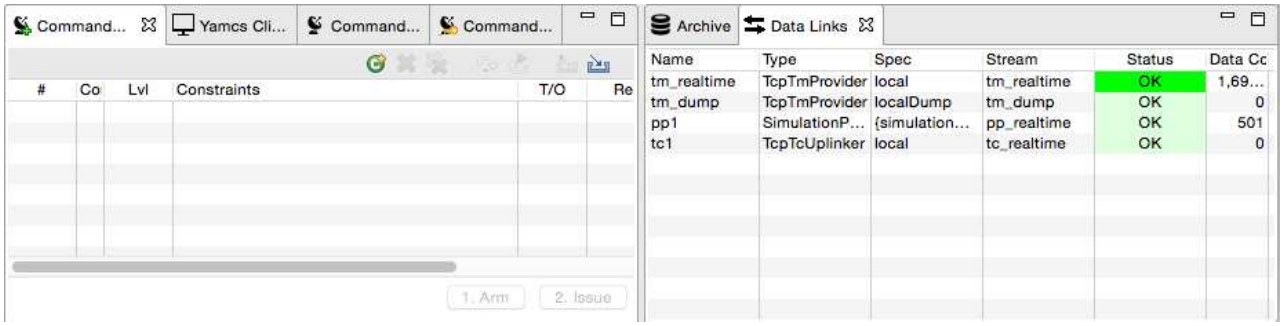
Release the mouse to confirm this view's new position.



Let's say we want to move the Data Links view to the right as well. Again, click its title and drag your mouse next to the Archive tab. You will see a green bar suggesting this tab's new placement.



Release the mouse to confirm this view's new position.



As you try dragging to different locations, you will notice that Yamcs Studio has several hot spots where you can attach your views. For example, you can detach windows by dragging them outside of your application window. This provides additional screen space if your workstation supports multiple monitors.

When you close Yamcs Studio and later reopen it, your last view and window arrangement will be restored.

> Yamcs Studio stores the information about your view arrangement in a .metadata folder inside your workspace. This is how it knows how to restore this information through restarts. If you share your workspace with other users through a version control system, you should consider *not* committing this .metadata folder. This way everybody can have his own preferred arrangement without colliding with each other.

Windows

Yamcs Studio uses two special windows that serve a different purpose:

- Display Builder
- Display Runner

On a new workspace, Yamcs Studio will by default open the Display Builder window. The Display Builder includes a default set of views that allow creating or editing operator displays.

In the top toolbar, you can see a play button which allows you to open the Display Runner window. This window includes a different set of windows that put the focus on operations. In this window, displays open in runtime mode.

In the top toolbar of the Display Runner there is a pickaxe button which allows you to open or bring to the front the Display Builder window.

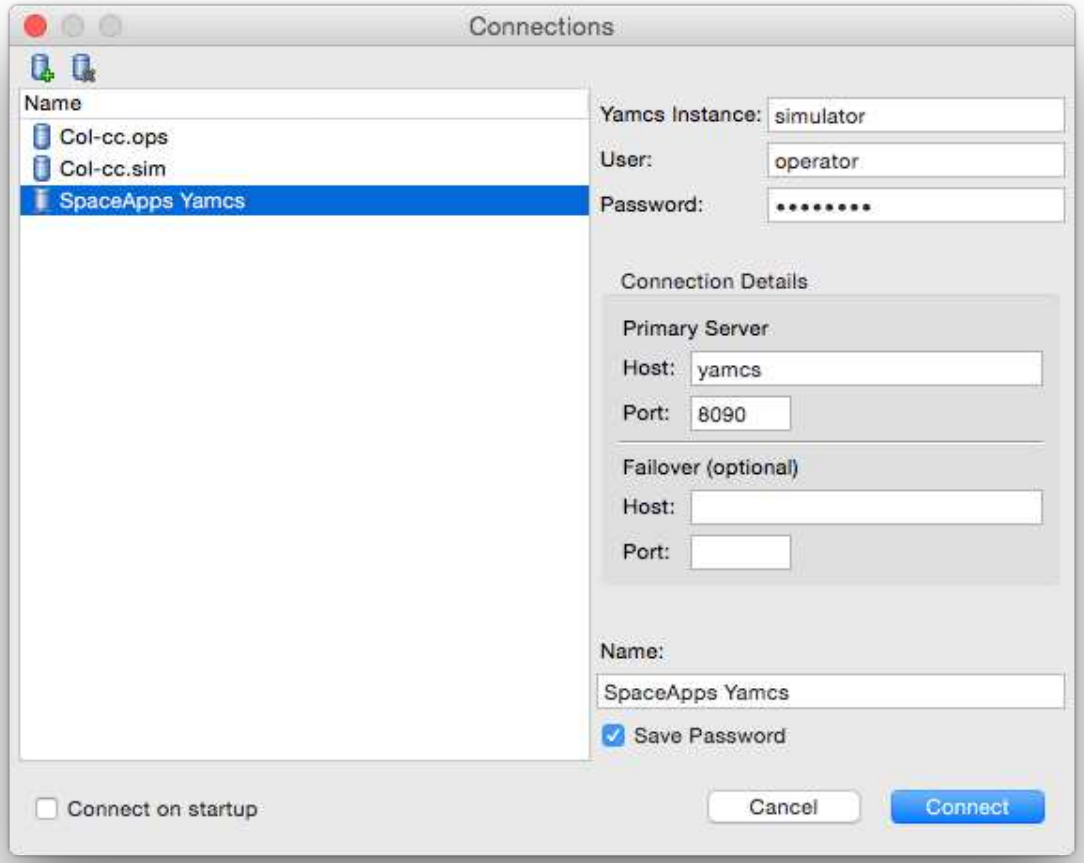
If at any time you want to reset your window to the defaults, select **Window > Reset Window Layout…**.

1.5. Connecting to Yamcs

Yamcs Studio is a client application that is meant to be connected with Yamcs Server.

Yamcs Server, or 'Yamcs', handles the processing, archiving and dispatching of telemetry data. Yamcs Studio is one of the possible Yamcs clients for receiving telemetry data.

To configure a Yamcs connection, select **File > Connect…**. This will open the Connections window where you can manage your connections. For many missions, one connection will do just fine, but depending on how Yamcs is deployed at your site, you may have multiple Yamcs instances on the same server, or even multiple Yamcs servers.



Click **Add Server** to add a server connection, or **Remove Server** to remove the selected server connection.

Connection Properties

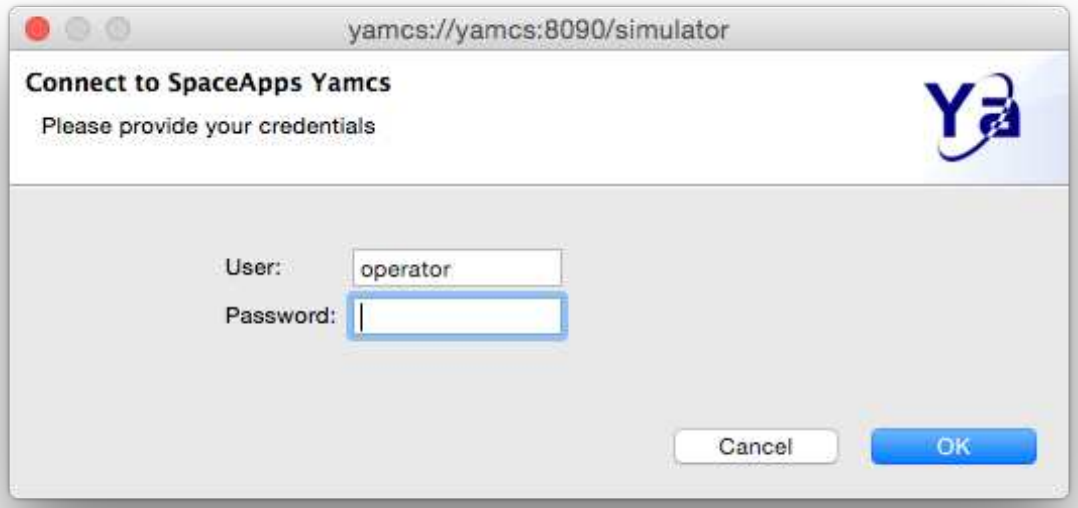
The right panel contains editable details for the selected server connection. We document the available properties below, but if you're unsure what to fill in, ask details to the person that is responsible for installing Yamcs at your site.

Yamcs Instance	Required	Yamcs can run multiple instances in parallel. You can think of instances like different environments, where every instance is completely separated from the other instance. While Yamcs Server may be running multiple instances in parallel, Yamcs Studio will always connects the user to one specific instance, which you have to configure here.
User / **Password**	Optional	If your Yamcs instance is secured, fill in your user and password here.
Primary Server	Required	Specify your actual host and port connection details here. The port is usually 8090.
Failover Server	Optional	If you specify a second host/port configuration, then Yamcs Studio will automatically failover to this second server in case connection with the primary server could not be established, or was lost. On the server-end, this setup requires two distinct Yamcs servers that are being kept in sync.
Name	Required	You can give your configuration a name of your choosing. This name will be used to represent this connection in the left panel of the Connections window.
Save Password	Optional	If you prefer not to enter your password at every occasion, tick this box to save your password to disk. Please be aware that your password will be saved in a manner that is difficult, but not impossible, for an intruder to break.

Connecting

All changes you make are automatically saved when you click **Connect**. If you want to discard your changes click **Cancel**.

Select the **Connect on startup** option, if you would like Yamcs Studio to automatically reconnect to the last used Yamcs instance during start-up. If this connection requires privileges and you chose not to save your password to disk, you will see a specialised login window everytime you start Yamcs Studio:



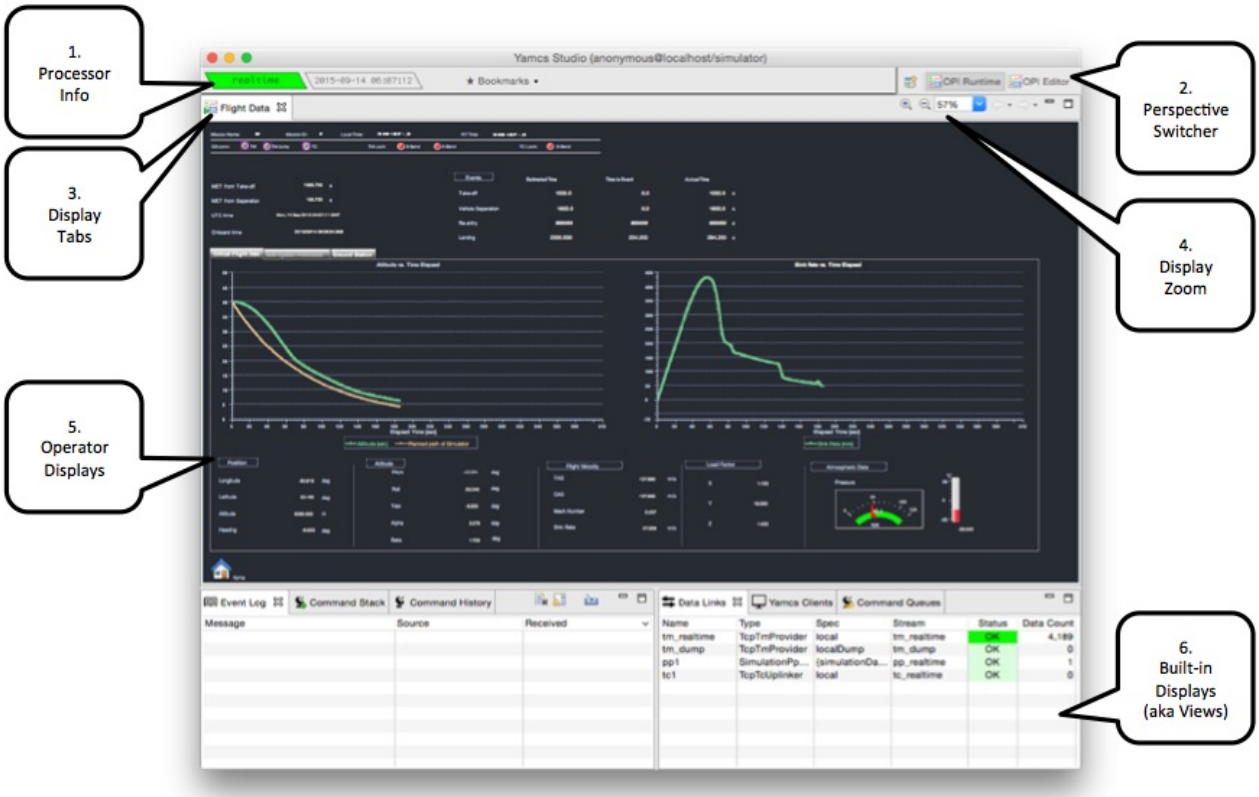
Connection preferences are stored in a hidden folder under your home directory, and will continue functioning whenever you upgrade your copy of Yamcs Studio.

You can verify that your copy of Yamcs Studio is properly connected by looking at the bottom status bar:

If it says realtime, then you've successfully connected.

Chapter 2. Display Runner

The Display Runner window is useful for realtime operations, or for testing out displays as they are being built. The default layout looks like this:



1. Processor Info This zone holds two status indicators. The first indicator light shows the processor that Yamcs Studio is currently listening to. Yamcs supports many concurrent processors (realtime, replay channels). By default Yamcs Studio will always connect to realtime.

   Next to that we see a second indicator which currently shows the processor time as provided by Yamcs. The simulator outputs generation times equal to the local system clock.

2. Display Tabs Displays open in different tabs. By clicking and dragging these tabs we can easily create split screens, or different tab stacks. We can also drag a tab out of its parent window into a new window. In fact, Yamcs Studio is optimised for multi-monitor systems. Window layouts are restored through restarts of Yamcs Studio.

3. Display Zoom The display shown in the picture was configured in such a way that it automatically stretches (while preserving aspect ratio) to fit the available screen space. This behaviour can be turned on or off by the display author. Regardless of its setting, as a display user we can always zoom in or out of the display using these controls.

4. Operator Displays This area contains displays that were authored in the Display Builder window. Displays contain any number of widgets. Most widgets can be connected to TM, which will also make them alarm-sensitive. In practice this means that they will be highlighted using different decorations depending on the alarm level. There are also things like button widgets which can

for example open other displays, or launch a telecommand, or open dialog boxes, etc. All widgets are highly customisable using scripts and/or rules.

5. Built-In Displays Yamcs Studio comes with an array of built-in displays that offer more dynamic views on different aspects of Yamcs. These built-in displays (or Views, as Yamcs Studio calls them) cover concepts like commanding, event logging, alarm overviews (upcoming) and archive insight.

Chapter 3. Processed Variables

Processed Variable or ‘PV’ is a term used by Yamcs Studio that covers the different types of data sources that a widget can be connected to. It is a more general term than parameter, which is a Yamcs Server notion.

PVs are uniquely identified by a *PV Name*. If multiple widgets have dependencies on the same PV, only one instance will be created and shared between these widgets.

The term PV is used to indicate both the name of a specific data source definition, as well as any instances of that definition. Context usually makes it apparent which of the two is meant.

A PV is considered *connected* if the data source is available, and at least one widget within Yamcs Studio is subscribing to it. As soon as no more widgets are connected to a PV, the PV gets *disconnected*.

A side effect of this last property, is that widgets with memory, such as chart widgets, lose their history when closing and reopening the display. We are aware of this, and are taking care of this shortcoming.

There are different types of PVs:

Local PVs

Local PVs are read and written entirely in a running Yamcs Studio instance. They are never communicated to Yamcs, nor to any other copies of Yamcs Studio. Local PVs are typically used by the display author as a means to store information that needs to be communicated from one widget to another. They also form a powerful building block when scripting advanced displays due to their ability to store runtime state. This makes it possible to script logic based on a historical window of values.

Local PVs are transient, and are reset when Yamcs Studio is restarted. Local PVs do not need to be specially created. They are automatically instantiated when needed.

Example PV Names:

- loc://foo
- loc://my-favourite-local-pv
- loc://anything-you-want-really

You can assign an initial value to a local PV by adding it after its name. For instance:

- loc://foo(1)
- loc://bar(“abc”)

Parameters

Parameter PVs represent a read-only value that is provided by Yamcs. Typically this denotes telemetry.

The PV Name for parameters is the fully qualified XTCE name as specified in the Yamcs Mission Database.

Example PV Names:

- para:///YSS/SIMULATOR/BatteryVoltage1
- para:///YSS/SIMULATOR/BatteryTemperature1

Or simply:

- /YSS/SIMULATOR/BatteryVoltage1
- /YSS/SIMULATOR/BatteryTemperature1

In these examples YSS is the name of the root space system. SIMULATOR is the name of the space system directly below, which defines both measurements BatteryVoltage1 and BatteryTemperature1.

Software Parameters

Same concept as a Parameter, but has additional support for writing values from the client to the server. In this regard they can be used as a means of communicating information from one client to another using Yamcs Server as the medium.

Remark that software parameters are not currently archived by Yamcs Server, and will therefore be reset when Yamcs is restarted.

Example PV Name:

- sw:///YSS/SIMULATOR/some-software-param

Simulated Values

Locally generated simulation data. Mainly useful during testing, or in combination with other PVs using formulas. Full documentation is upcoming. For now please have a look at the example operator displays in the YSS projects.

Example PV Names:

- sim://ramp(0,1,1,0.5)
- sim://const(4)
- sim://noise
- sim://sine

Formulas

PVs can be combined with mathematical expressions. Formulas always start with = followed by a formula expression. Note that any referenced PVs must be wrapped with single quotes.

Example PV Names:

- =3*‘loc://foo(2)’
- =3.14
- =log(‘loc://foo(2)’)

Supported formulas include:

- abs(a)
- acos(a)
- asin(a)
- atan(a)
- ceil(a)
- cos(a)
- cosh(a)
- exp(a)
- expm1(a)
- floor(a)
- log(a)
- log10(a)
- round(a)
- sin(a)
- sinh(a)
- sqrt(a)
- tan(a)
- tanh(a)
- toDegrees(a)
- toRadians(a)
- atan2(a, b)
- hypot(a, b)
- pow(a, b)
- min(a, b, c, d, e)
- max(a, b, c, d, e)

Chapter 4. Widgets

A display is a container for widgets.

Most widgets are backed by a PV. Some widgets (e.g. widgets used for layout) are typically not connected to a PV. Other widgets (e.g. charts) can be backed by more than one PV.

Catalogue of Widgets

The default widgets in Yamcs Studio are listed below. Their runtime behaviour should be fairly straightforward. The various properties are detailed when we address the Display Builder.

Graphics

Arc	Rectangle	Label
Polyline	Rounded Rectangle	Image
Polygon	Ellipse	

Monitors

LED	Progress Bar	XY Graph[1]
Image Boolean Indicator	Gauge	Intensity Graph
Text Update	Thermometer	Byte Monitor
Meter	Tank	

[1] Clear the view on this widget by right-clicking on it and selecting **Clear Graph**. If you want advanced controls, like zooming, activate the toolbar by right-clicking on your widget and selecting **Show/Hide Graph Toolbar**.

Controls

Action Button[2]	Knob	Image Boolean Button
Menu Button	Scrollbar	Check Box
Text Input	Thumb Wheel	Radio Box
Spinner	Boolean Switch	Choice Button
Scaled Slider	Boolean Button	Combo

[2] Action Buttons are often used to open other displays. Whether this opens in a new tab or in the same tab depends on how the display author constructed the display. Override the default by right-clicking the Action Button.

Others

Table	Grouping Container	Sash Container
Web Browser	Linking Container	Grid Layout
Array	Tabbed Container	

Color Decorations

When a widget is backed by a PV, it will be decorated according to its runtime state. The specific colors of these decorations can vary since the default colors can be overridden (or disabled) by the display author.

State	Decoration
Connected	No decorations
Connected, but no value (yet)	Dashed pink border around the widget
Disconnected	Solid pink border around the widget and the label 'Disconnected' in the top left corner (space-permitting)
Expired	Blinking solid pink border around the widget
Minor Alarm	Solid orange border around the widget
Major Alarm	Solid red border around the widget

Note that the color information for alarms is currently not as rich as it could be. Yamcs parameters support five different levels of alarms, as well as a range of special monitoring values. This information has for now been transformed using the following mapping:

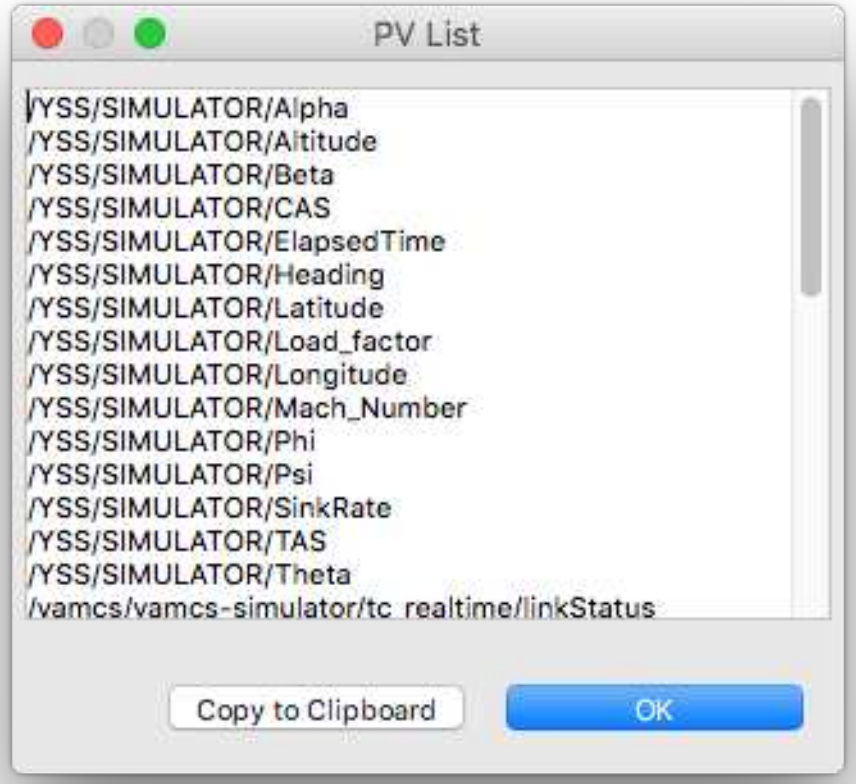
Yamcs Server	Yamcs Studio
WATCH	MINOR
WARNING	
DISTRESS	
CRITICAL	MAJOR
SEVERE	

Chapter 5. Inspector Windows

There are a few standalone windows that can be opened for inspecting widgets.

5.1. PV List

Right-click anywhere in a display, and choose **Dump PV List**, you will see a window listing the unique PVs that are defined inside any widget of that display. This can be useful for quick-fixing runtime issues.

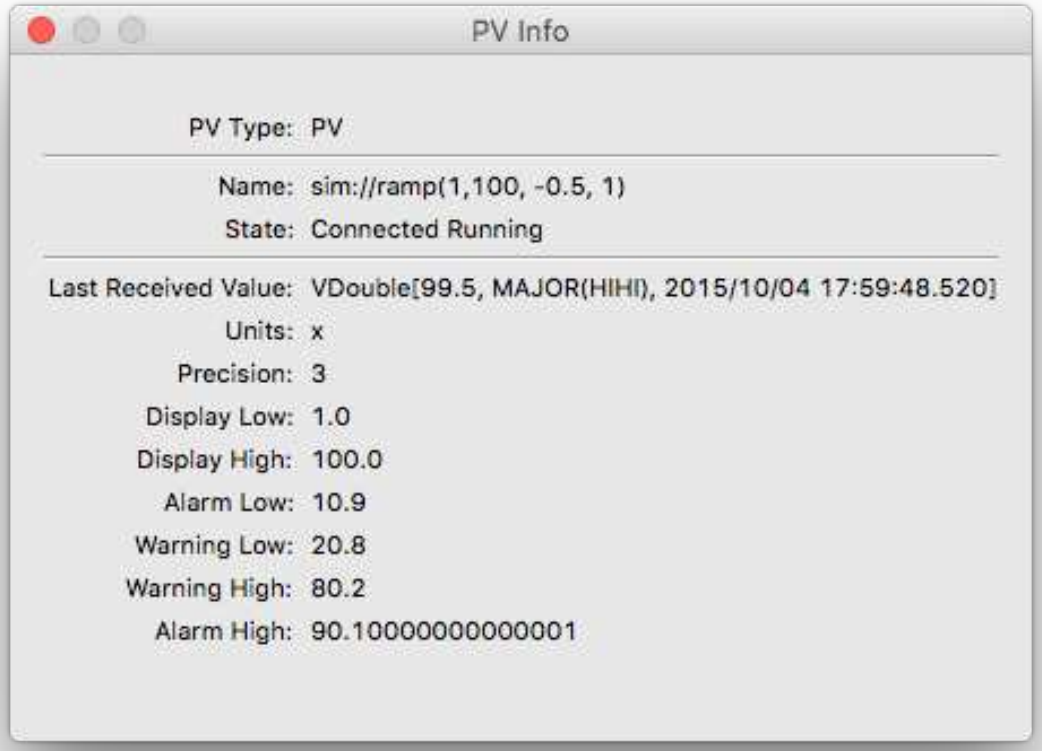


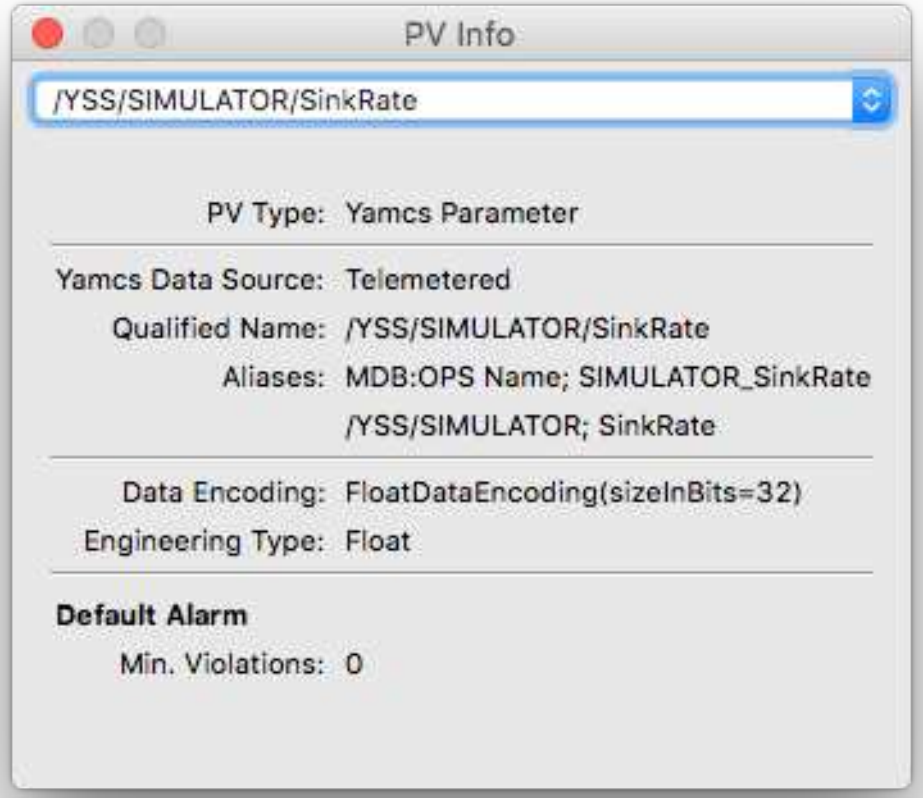
5.2. PV Info

Right-click on a widget backed by a PV, and select **PV Info**. This opens a window where you get extra information on the PVs in that widget. If there are multiple PVs for that widget, select the PV of your interest using the top dropdown selector. For Yamcs parameters, you will see various properties that were defined in the Mission Database.

Currently the displayed information is mostly static. Features which we plan to add include:

- showing continuously updated information on the latest value, and its alarm info.
- showing which context is applicable based on the latest value, for context-dependent Mission Database definitions (e.g. *alarm rule x only applies in contingency mode*).

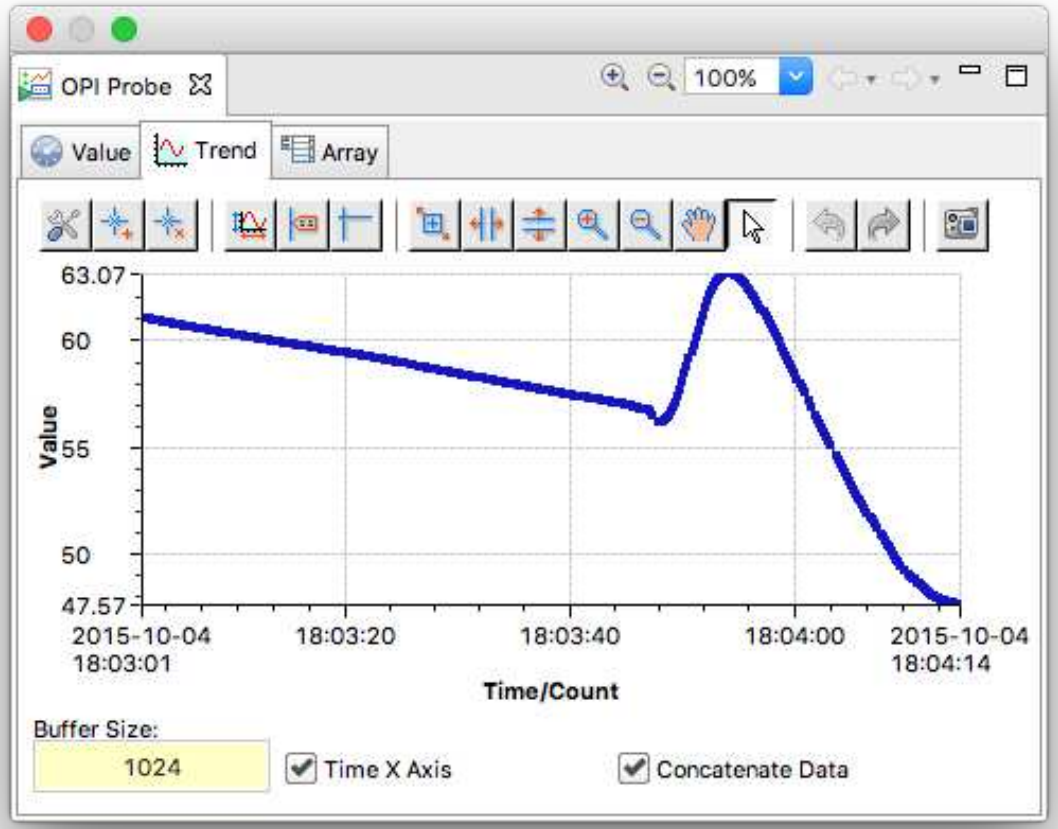



PV Info
/YSS/SIMULATOR/SinkRate
PV Type: Yamcs Parameter
Yamcs Data Source: Telemetered
Qualified Name: /YSS/SIMULATOR/SinkRate
Aliases: MDB:OPS Name; SIMULATOR_SinkRate
/YSS/SIMULATOR; SinkRate
Data Encoding: FloatDataEncoding(sizeInBits=32)
Engineering Type: Float
Default Alarm
Min. Violations: 0

5.3. OPI Probe

Right-click on a widget backed by a PV, and select **Process Variable > OPI Probe**. This opens the OPI Probe view with:

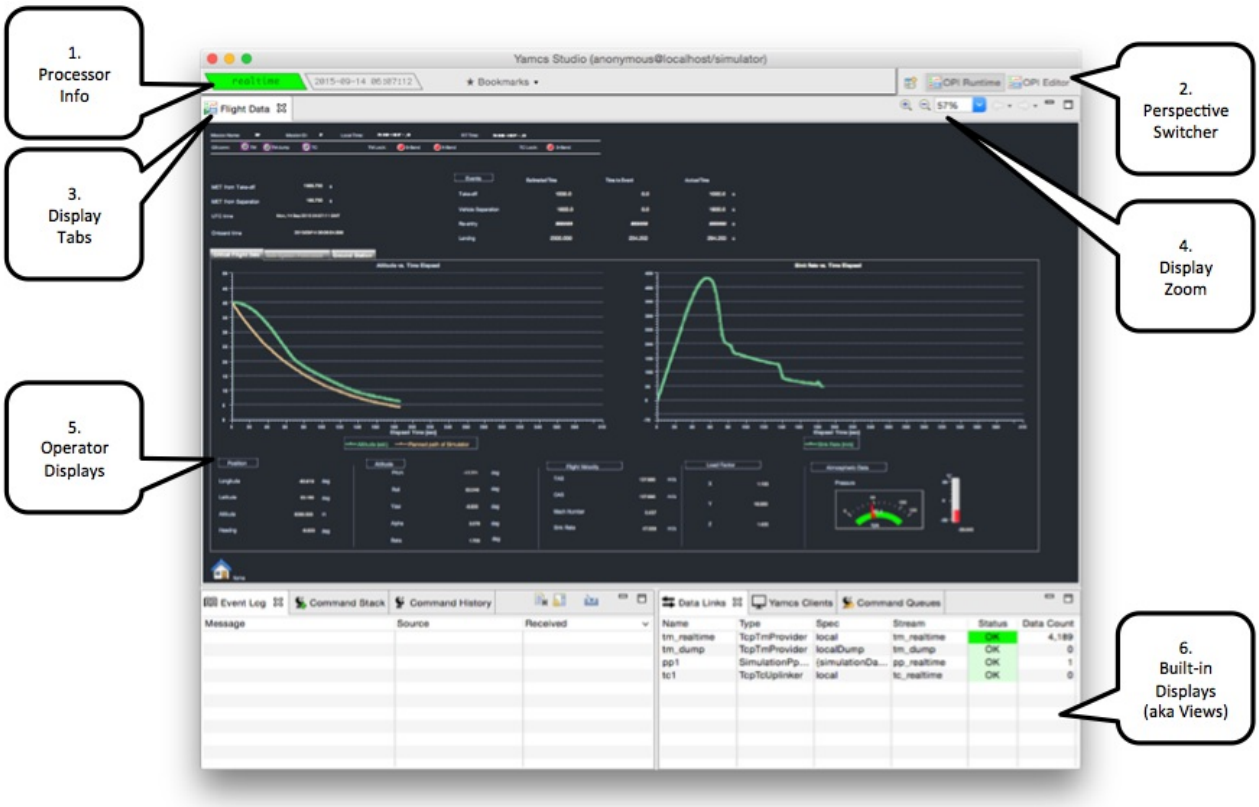
- In the **Value** tab, a meter indicating the validity range. This is however not fully implemented yet, and therefore often shows a very large range of values.
- In the tab **Trend**, a graphical evolution of this PVs value. There is currently no way to navigate to archived data.



Chapter 6. Editing Displays

6.1. Display Runner

The Display Runner window is useful for realtime operations, or for testing out displays as they are being built. The default layout looks like this:



1. Processor Info This zone holds two status indicators. The first indicator light shows the processor that Yamcs Studio is currently listening to. Yamcs supports many concurrent processors (realtime, replay channels). By default Yamcs Studio will always connect to realtime.

   Next to that we see a second indicator which currently shows the processor time as provided by Yamcs. The simulator outputs generation times equal to the local system clock.

2. Display Tabs Displays open in different tabs. By clicking and dragging these tabs we can easily create split screens, or different tab stacks. We can also drag a tab out of its parent window into a new window. In fact, Yamcs Studio is optimised for multi-monitor systems. Window layouts are restored through restarts of Yamcs Studio.

3. Display Zoom The display shown in the picture was configured in such a way that it automatically stretches (while preserving aspect ratio) to fit the available screen space. This behaviour can be turned on or off by the display author. Regardless of its setting, as a display user we can always zoom in or out of the display using these controls.

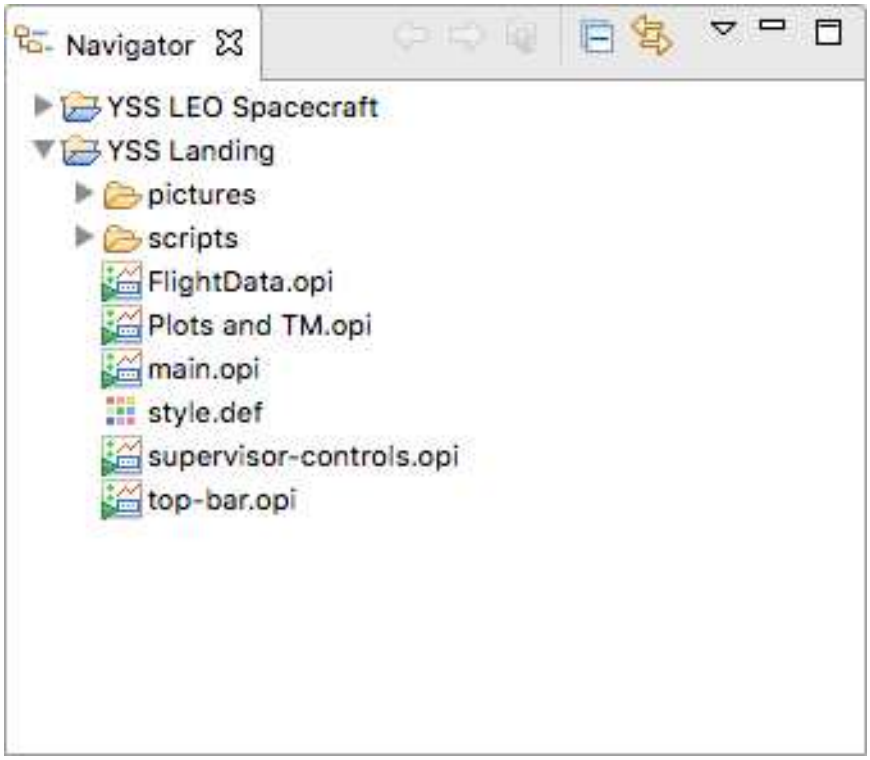
4. Operator Displays This area contains displays that were authored in the Display Builder window. Displays contain any number of widgets. Most widgets can be connected to TM, which will also

make them alarm-sensitive. In practice this means that they will be highlighted using different decorations depending on the alarm level. There are also things like button widgets which can for example open other displays, or launch a telecommand, or open dialog boxes, etc. All widgets are highly customisable using scripts and/or rules.

5. Built-In Displays Yamcs Studio comes with an array of built-in displays that offer more dynamic views on different aspects of Yamcs. These built-in displays (or Views, as Yamcs Studio calls them) cover concepts like commanding, event logging, alarm overviews (upcoming) and archive insight.

6.2. Explorer

The Explorer tab shows a folder-like structure of the resources contained within your current workspace.



Projects

Recall that within Yamcs Studio, you are always working in one workspace only. Within that workspace you create or import *projects*. It is the projects that contain the actual resources (files and/or directories).

Creating a Project

To create a new project, choose **File > New Project**, or right-click in the explorer and choose **New Project** from the pop-up menu.

Importing Existing Projects

To import an existing project, select **File > Import** and choose **Existing Projects into Workspace**. Navigate to the project's folder, and if Yamcs Studio recognizes it as a project you will be able to import it.

Projects are just directories on your disk (usually under version control). Yamcs Studio recognizes existing projects by the metadata which is added under the hidden .metadata folder. This metadata includes project-specific preferences, as well as for example the name of the project.

Resources

Any file can be added to a project or a contained directory. To do so right-click on the desired node to open the popup menu, and choose your desired file type under the **New** item.

To add an existing file to a project (for example a project). Copy it to your clipboard, and paste it onto the node. Alternatively, use the **File > Import > General > File System** option.

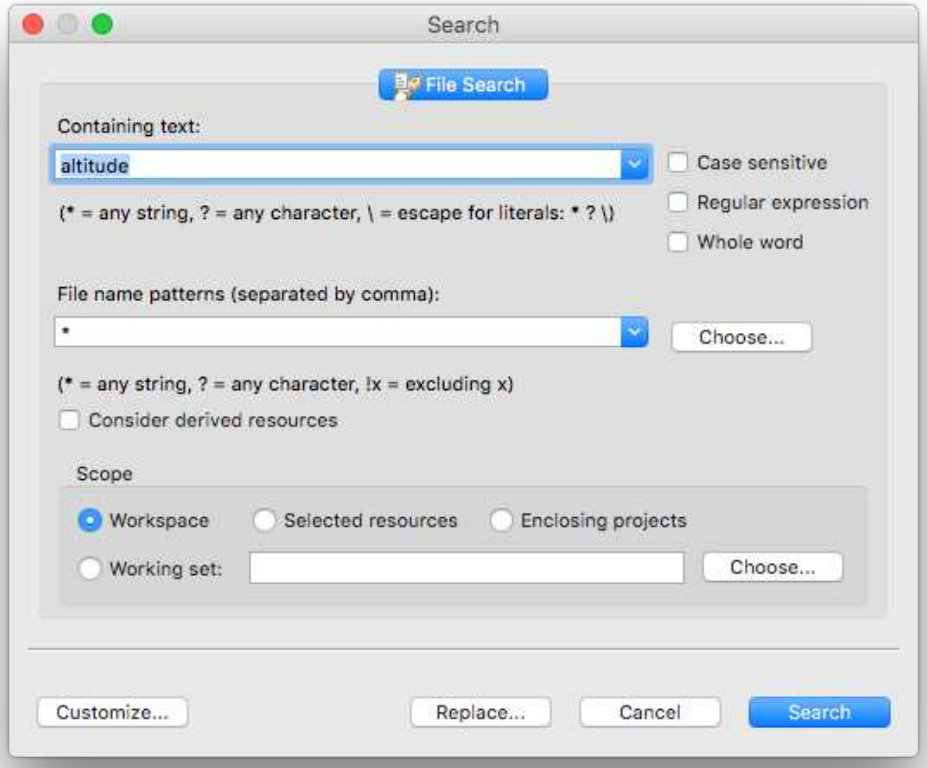
Open a file by double-clicking on it. If you open a file, yet Yamcs Studio does not have a specific handler for the type of file, it will open it with your system default program for that extension.

The default Yamcs Studio distribution handles *.def and *.opi files. It also comes with a built-in text editor for basic editing of many other types of files as well (including *.txt, *.js and *.py).

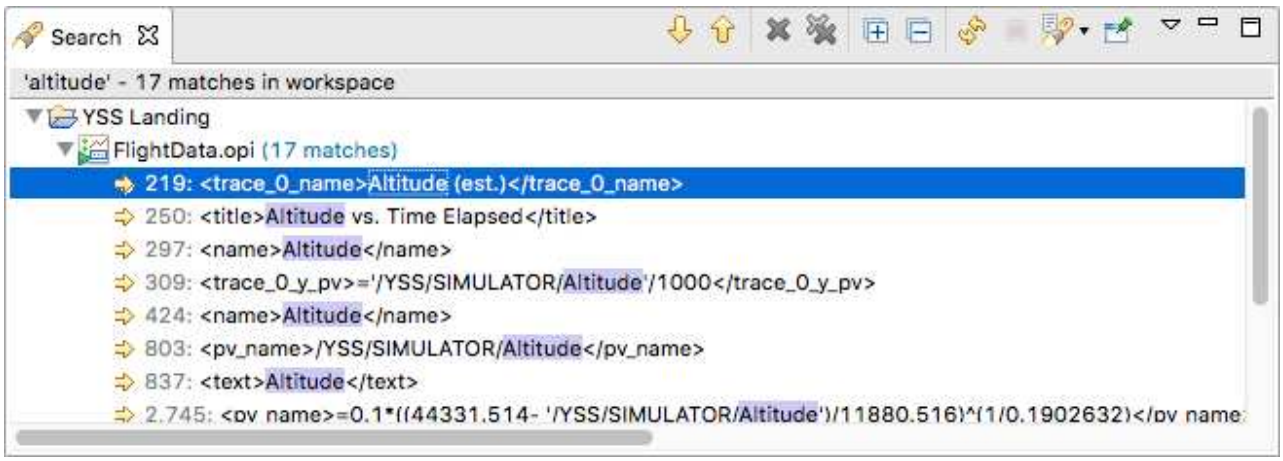
Use right-click **Open With** for more control over how the file is to be opened.

Searching

An advanced search and replace dialog is available from the **Search** menu. If you select a node in the explorer before opening the Search dialog, this dialog will be configured to only search resources under that node.



Your search can include wildcard characters, and can be further specified to only a specific set of resources. The results will be opened in a **Search** view which also allows for replacing occurrences upon right-click.


Search
'altitude' - 17 matches in workspace
YSS Landing
FlightData.opi (17 matches)
219: <trace_0_name>Altitude (est.)</trace_0_name>
250: <title>Altitude vs. Time Elapsed</title>
297: <name>Altitude</name>
309: <trace_0_y_pv>='/YSS/SIMULATOR/Altitude'/1000</trace_0_y_pv>
424: <name>Altitude</name>
803: <pv_name>/YSS/SIMULATOR/Altitude</pv_name>
837: <text>Altitude</text>
2,745: <pv_name>=0.1*((44331.514- '/YSS/SIMULATOR/Altitude')/11880.516)^(1/0.1902632)</pv_name

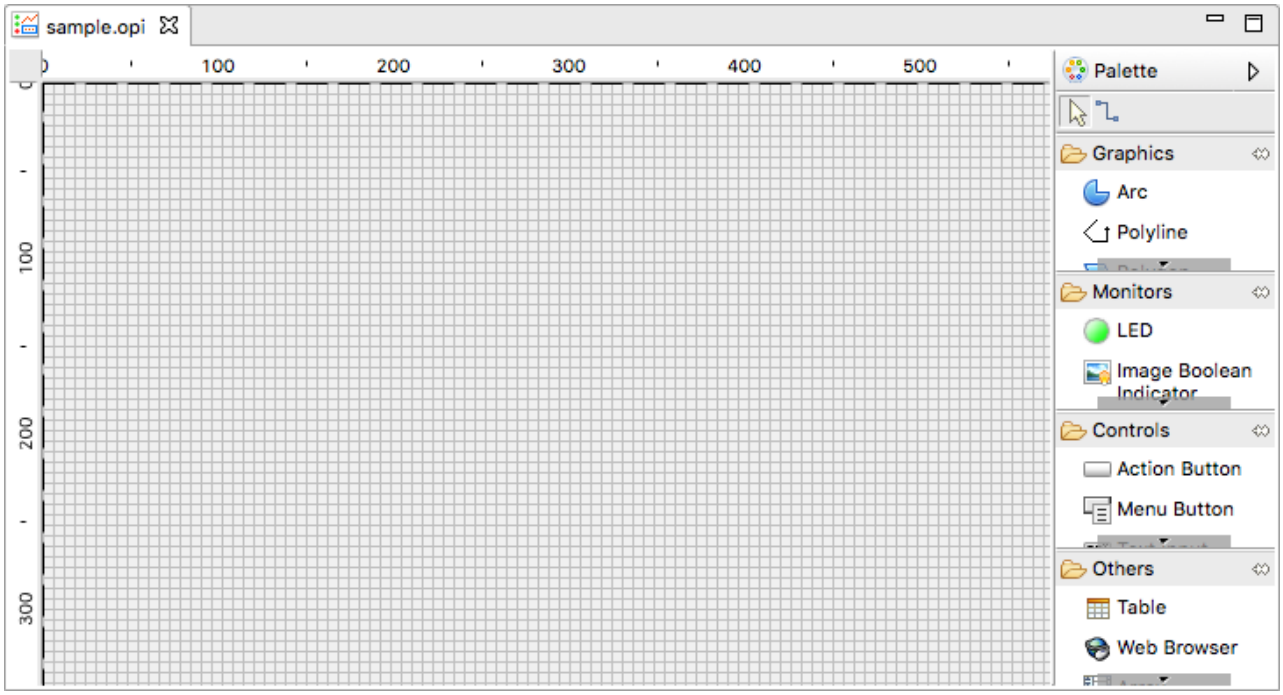
6.3. Editor Area

The Editor Area is a unique zone in Yamcs Studio where files are opened for editing. Typically this would be a display file (*.opi), but it doesn't have to be.

OPI Files

OPI stands for Operator Interface, but is more commonly referred to as a *display*. OPIs are often static, but when needed can be made very dynamic by combining different widgets and PVs together using concepts like Actions, Rules and Scripts.

In the Display Builder window, create a new OPI file, right-click in the Explorer on the desired location, and select **New > OPI File**.



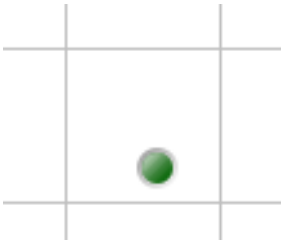
OPI files are created with some default properties, which includes a grid, and a size of 800x600. We will see in the section on Properties that we can edit these properties.

Palette

Notice the Palette attached to the right of the Editor Area. The Palette contains the widgets bundled with your version of Yamcs Studio. Use the palette as your toolbox when you author a display.

To add a widget to your display, click first on its icon in the Palette, then click where you want to put it in the Editor Area.

We select as example an LED.



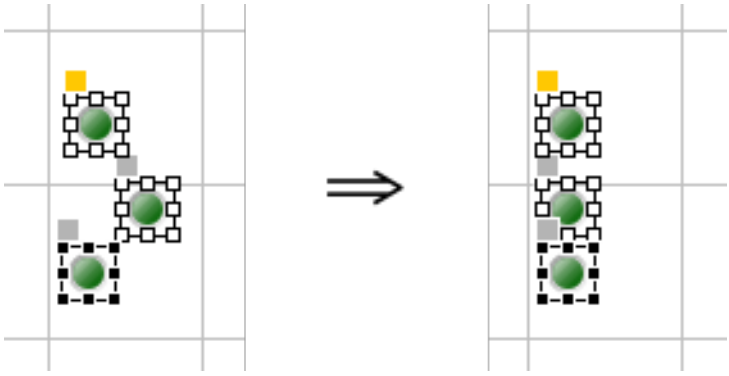
Once the widget has been placed, you can finetune its position and size using the Properties View. Some operations are also readily available in the Editor Area itself using familiar controls. For example, to enlarge a widget, select it, then drag its handles around with the mouse.

You can also move a widget by pressing it, while dragging it to another location.

To select multiple widgets, drag a box around them. To add widgets to an existing selection, hold the Ctrl key (⌘ on Mac) while selecting the widgets one by one. Remove a widget from the selection in similar fashion.

Positioning Widgets

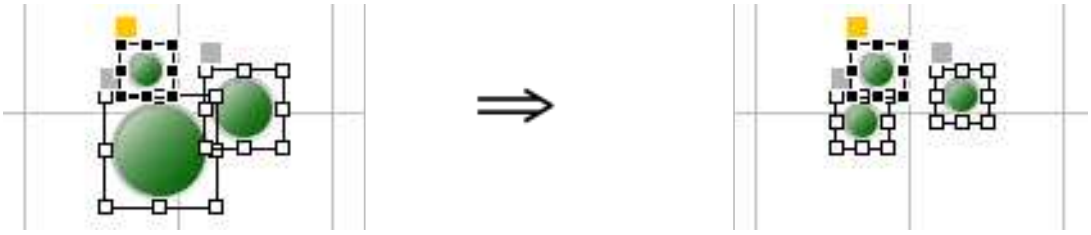
The toolbar of Yamcs Studio contains tools that help us align multiple selected widgets. For example, clicking **Align Left** repositions these three LEDs to the leftmost position.



There are similar tools for vertical alignments, as well as for distributing horizontal or vertical space between selected widgets.

Match Size

We can also standardize the size of selected widgets. For example. By clicking **Match Width** and **Match Height** in sequence, we made these three LEDs the same size.



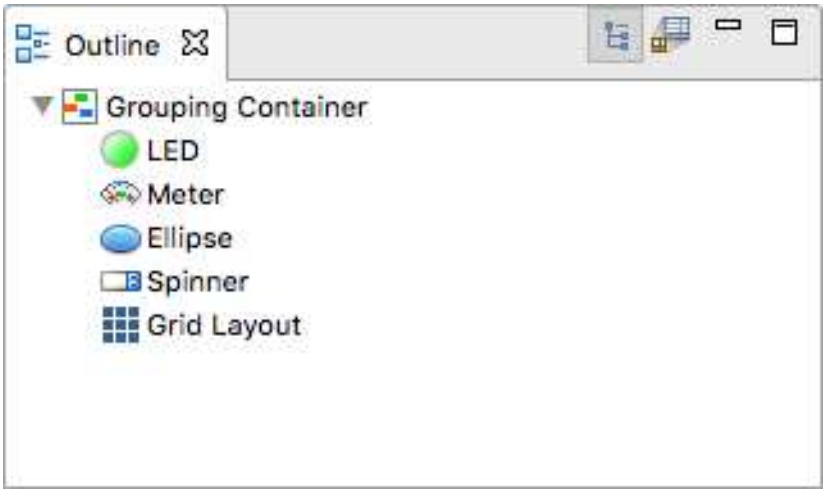
The size of the last selected widget is taken as the reference. This reference widget is highlighted with black instead of white anchor points.

Note that in this particular case of non-square LEDs, clicking only **Match Width** would actually have been sufficient since round LEDs can't take on the shape of an ellipse.

6.4. Outline

The Outline view, available from the Display Builder window, gives a hierarchical breakdown of the widgets contained in the currently active OPI. Some widgets are containers for other widget types, and will be shown as a node in the tree with a sub-node for every child.

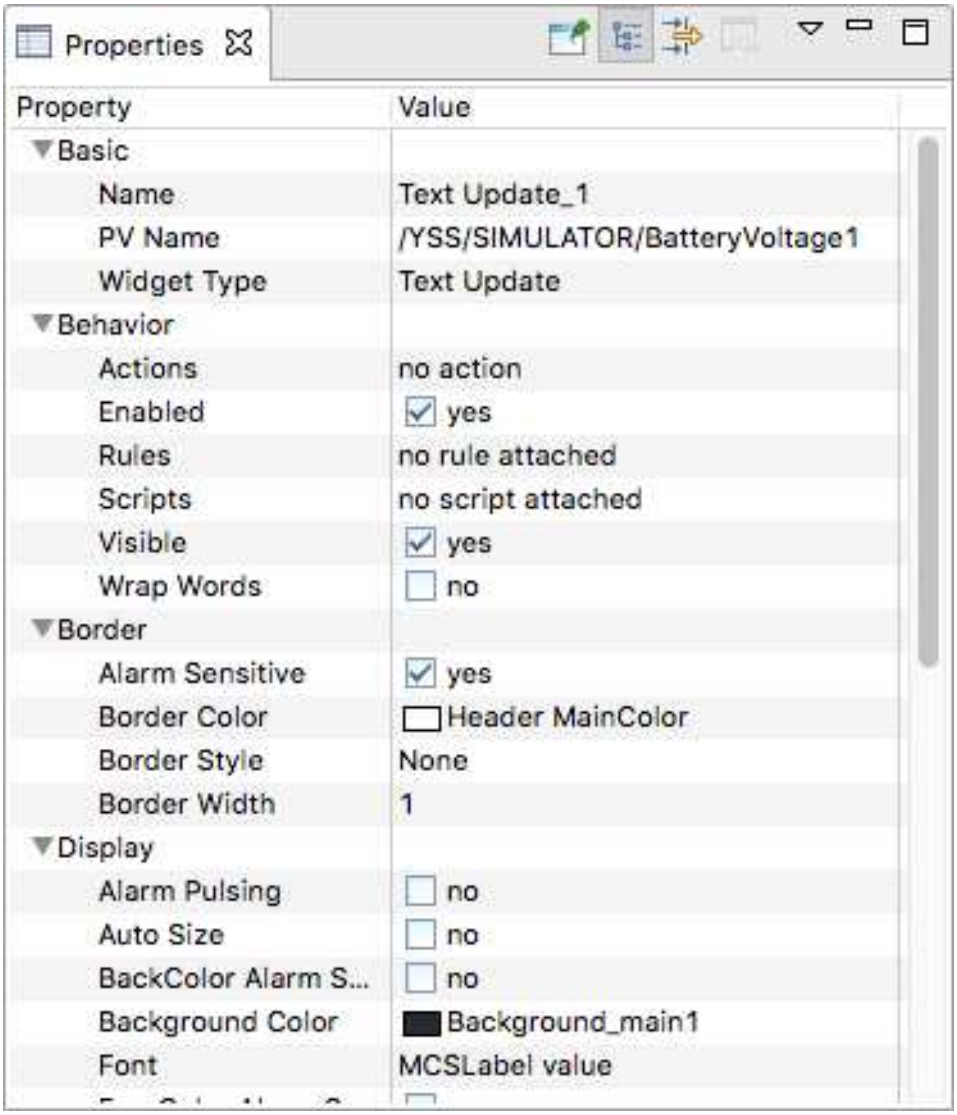
Widgets can be named in the Properties view to make them stand out in the Outline.



6.5. Properties

The Properties view is used in the Display Builder window to edit properties of your display, or to edit properties of a widget.s

Select a widget to see its properties in the Properties view. The contents of this view adapts to your selection. Click in the **Value** column to edit a specific property, depending on the type of property this will trigger different behaviour. For example, if the property is just a numeric value, you can edit it in-place (confirm with **Enter**). If the property represents multiline text or a list of items you will typically have more advanced editing controls in a popup dialog.



Changes are not saved automatically. Remember to select **File > Save All** before you refresh a runtime OPI.

Depending on the types of involved widgets, it may be possible to batch-edit some properties by selecting multiple different widgets.

Widget Properties

Different widgets have different properties, but many of those properties are shared among them. These include:

Name	A name that identifies this widget in the Outline view. There is no constraint on uniqueness, but when not specified by the user, Yamcs Studio will try to determine a unique name by concatenating the widget type with a sequential number.
X **Y** **Width** **Height**	Widgets are contained in a bounding box which is controlled by these properties. X and Y indicate the pixel position of the widget within the display. The origin is located at the top left of the Editor Area. The X and Y position of the widget also indicates the top left of its bounding box. Width and Height indicate the size of the bounding box. Many widgets support automatic scaling within the available bounding box.
PV Name	The unique name of a PV that will be backing this widget. At runtime the value of this PV will be used to control the intrinsic value of the widget, or to decorate it in case of off-nominal state. If the PV concerns a Yamcs parameter, and Yamcs Studio is connected to Yamcs, you will get autocompletion support on parameter names based on the contents of the Mission Database.
Alarm Sensitive	Toggles whether or not the bounding box of this widget will be decorated during runtime based on off-nominal values of its connected PV.
Border Color **Border Style** **Border Width**	Allows drawing the contours of the widget's bounding box using a wide variety of different styles.

OPI Properties

The OPI itself is also a special kind of container widget with editable properties. Click on an empty region of your Editor Area to see these.

Specific properties include:

Show Ruler **Show Grid** **Grid Color** **Grid Space**	Configure the ruler or the grid. Notice that these properties are tied to a specific OPI. The visibility can also be toggled using the toolbar. If the grid is toggled on, the grid lines will work as magnets when positioning widgets.
Snap to Geometry	When enabled, Yamcs Studio snaps your widgets magnetically in place based on the position of neighbouring widgets.
Auto Zoom to Fit All	Controls whether the display as a whole should be zoomed in at runtime such that it fits its available space

6.6. Palette

The Palette is a fold-out zone of the Editor Area that contains the widgets bundled with your version of Yamcs Studio, which may contain less or more widgets than the default widgets documented below.

Widgets come in all different kinds, and are grouped in four different categories.

Graphics

Arc	Rectangle	Label
Polyline	Rounded Rectangle	Image
Polygon	Ellipse	

Monitors

LED	Progress Bar	XY Graph
Image Boolean Indicator	Gauge	Intensity Graph
0.0 Text Update	Thermometer	Byte Monitor
Meter	Tank	

Controls

Action Button	Knob	Image Boolean Button
Menu Button	Scrollbar	Check Box
Text Input	Thumb Wheel	Radio Box
Spinner	Boolean Switch	Choice Button
Scaled Slider	Boolean Button	Combo

Others

Table	Grouping Container	Sash Container
Web Browser	Linking Container	Grid Layout
Array	Tabbed Container	

6.6.1. Graphic Widgets

Arc

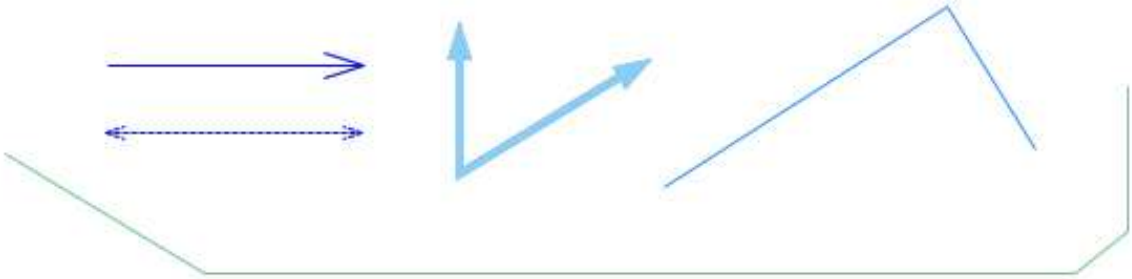
Draws an arc contained by the bounding box. To modify the size of the arc, modify the **X**, **Y**, **Width** and **Height** properties.

By default the arc starts with an angle of 0 degrees (right center). Modify this property using **Start Angle** (in degrees). The arc length is determined using **Total Angle**.

Polyline

Use the polyline tool to draw lines of two or more points. After selecting the widget from the Palette, start by clicking on your desired start location. Every next click will add a new point. Double-click if this is your last point. You can modify your points by moving around the yellow handles. Finetune your points using the **Points** properties.

Polylines have a direction and can be decorated with arrow heads using the various **Arrow** properties.

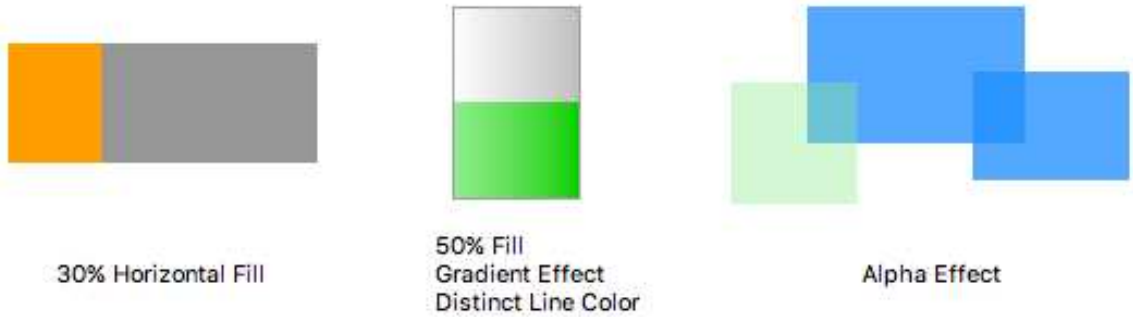


Polygon

Similar in functionality to polylines, but the last point is always connected to the first point to form a closed shape. Unlike polylines, polygons are not directed. The can be filled from left to right or from bottom to top using the **Fill Level**, and **Horizontal Fill** properties. In the section Rules we'll learn that this (or just about any other property) can also be dynamically modified.

Rectangle

Rectangles are constrained polygons. They can be customized in much the same way as polygons. Unlike polygons the background and foreground can also be colored using gradients which makes them look less flat.

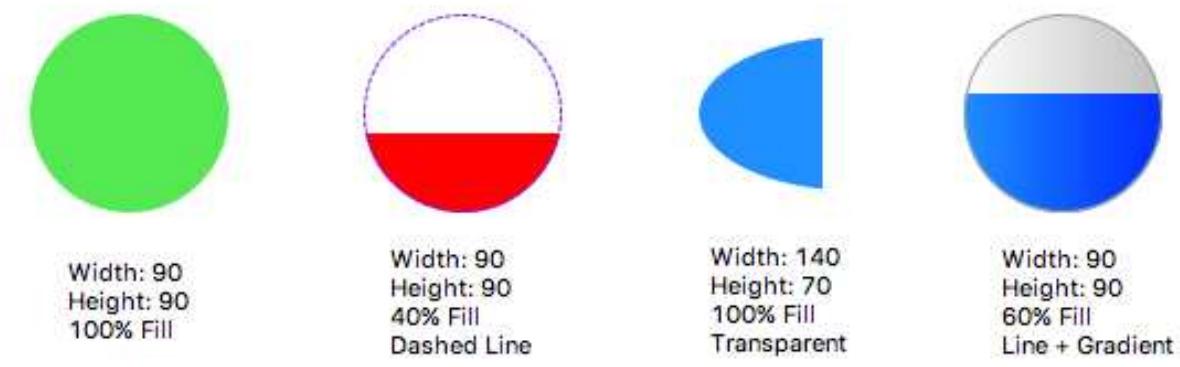


Rounded Rectangle

Rounded rectangles are a specialised form of Rectangles. They support additional properties for controlling **Corner Height** and **Corner Width**.

Ellipse

The Ellipse's shape is determined by its bounding box using the **Width** and **Height** properties. Ellipses support similar properties as rectangles.

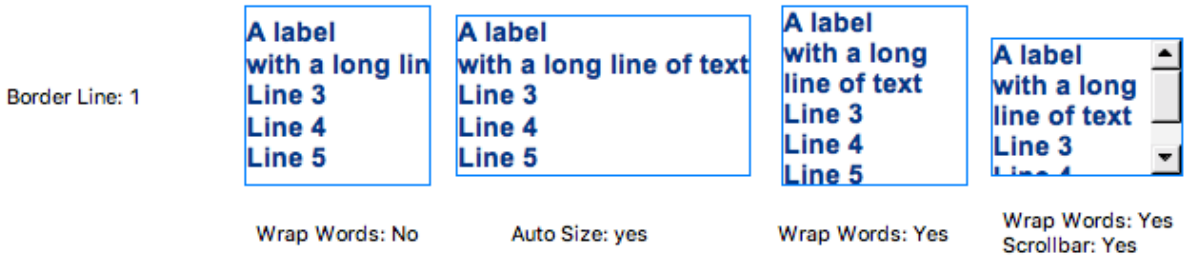


Label

The Label widget allows displaying text within a bounding box indicated by the **X**, **Y**, **Width** and **Height** properties. Adapt your bounding box to the content using **Auto Size** property.

For long texts consider using the **Wrap Words** and **Show Scrollbar** properties.

To edit the text of a label, either select the **Label** property to open an input dialog, or press **F2** to edit the label in-place.



Image

Yamcs Studio supports GIF, JPG, PNG and BMP images. Images must be added to your project before you can select them with the **Image File** property. Transparency is automatically supported as well, but make sure to set the background of the Image widget to the same color as the container.

6.6.2. Monitor Widgets

LED

By default LED widgets are dark green, and light up when the connected PV is non-zero. Using the **State Count** the number of states can be increased to more than two. When doing so, a number of extra properties will be added in the Properties view. You then need to set explicit colors per value.

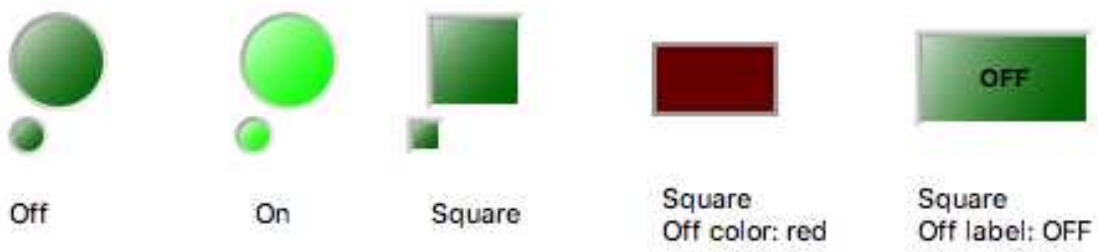


Image Boolean Indicator

With the Image Boolean Indicator you can render images depending on the ON or OFF state of the connected PV. The images need to be part of your workspace.

Among other use cases, this widget provides a practical way to replace the default LED widget with your own custom styling.

Text Update

The Text Update widget shows the textual value of the connected PV. The format can be controlled using properties like **Precision** and **Format Type**.

Both the background and the foreground color can be made alarm-sensitive. This will then apply the alarm colors that are defined in the color scheme.

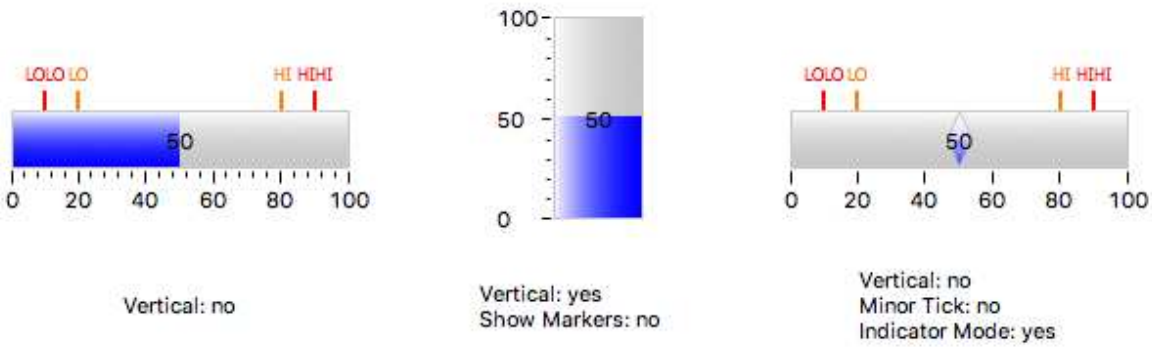
Like Labels, Text Update widgets are drawn in a bounding box which you can control on the canvas by dragging the handles, or by finetuning the properties **Width** and **Height**.

Meter

The Meter widget shows the numeric value of the connected PV on a meter. The exact display of this widget can be controlled using various properties, among which colors for the needle, LOLO, LO, HI and HIHI values, major tick separation, as well as various toggles for the different elements that a meter is composed of.

Progress Bar

A bar graph widget that shows the numeric value of the connected PV as a bar or indicator.

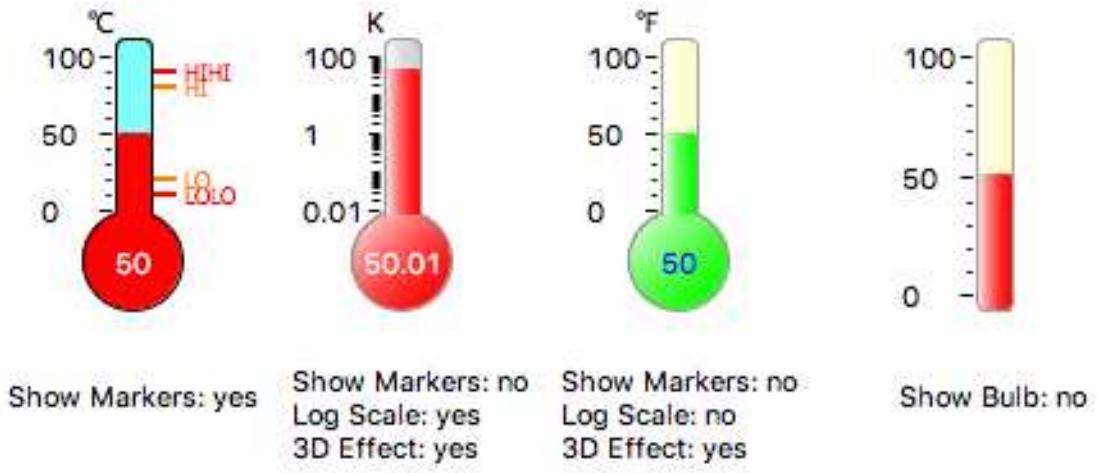


Gauge

The Gauge widget shows the numeric value of the connected PV on a gauge. The exact display of this widget can be controlled using various properties, among which colors for the needle, LOLO, LO, HI and HIHI values, major tick separation, as well as various toggles for the different elements that a gauge is composed of.

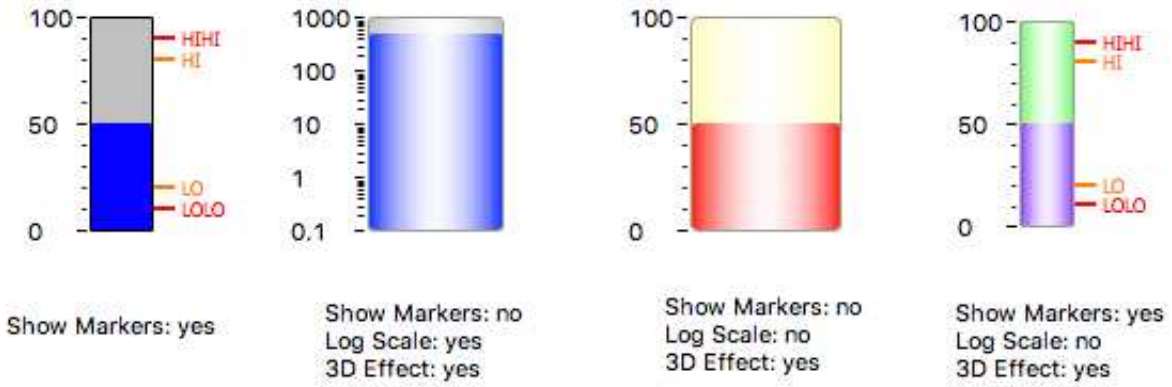
Thermometer

Specialised form of the Progress Bar widget that shows the numeric value of the connected PV as a thermometer.



Tank

Specialised form of the Progress Bar widget that shows the numeric value of the connected PV as a tank.

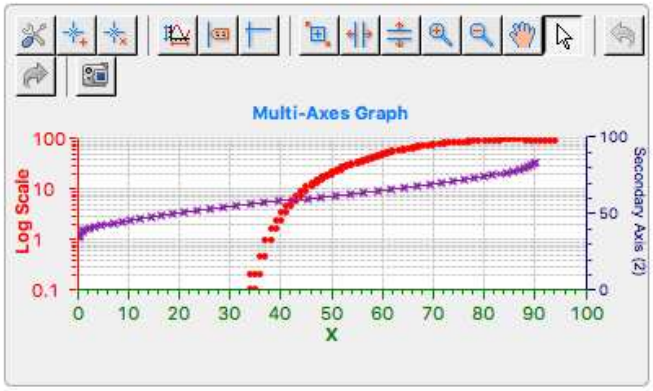


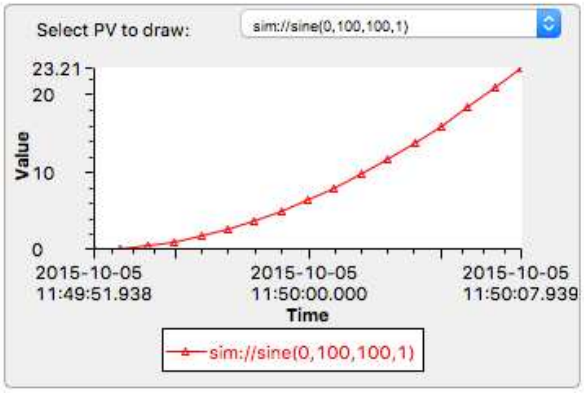
XY Graph

Advanced widget for plotting the numeric value of one or more connected PVs. By combining the different properties it supports line charts, scatter charts, bar charts, step charts and area charts.

To show multiple traces in the same chart, increase the **Trace Count**. This will add a set of properties for every added trace. Every trace can be linked with a separate PV.

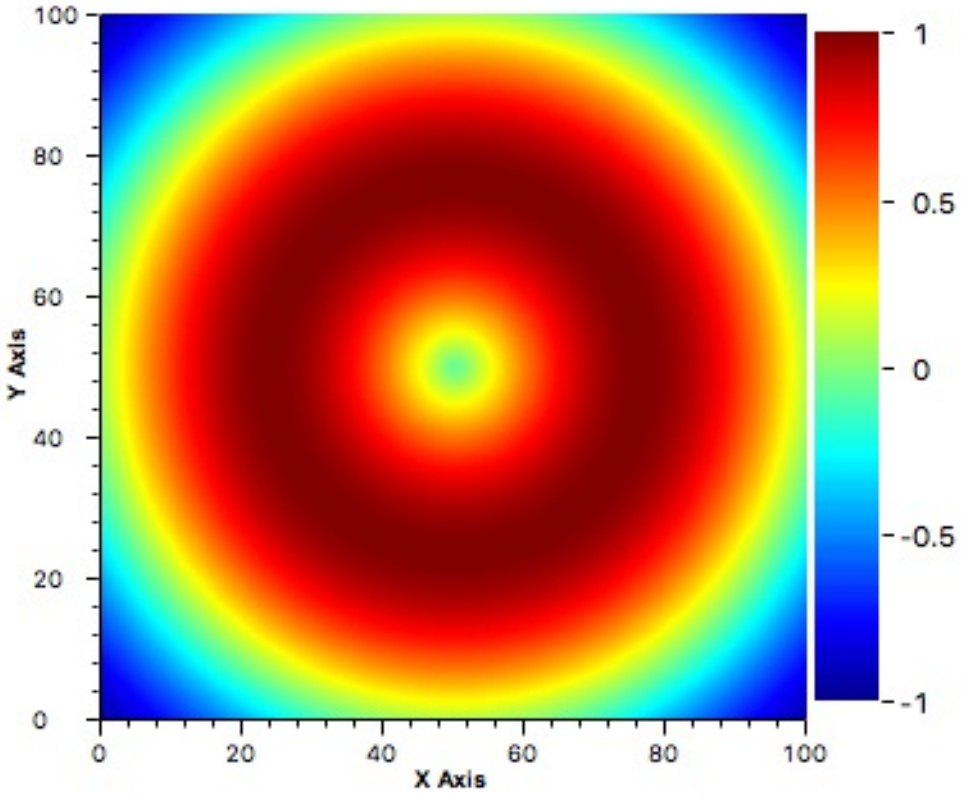
Use the **Show Toolbar** property to toggle visiblity of a toolbar. When visible, this toolbar can be used by the operator to perform zooming operations on the plot.





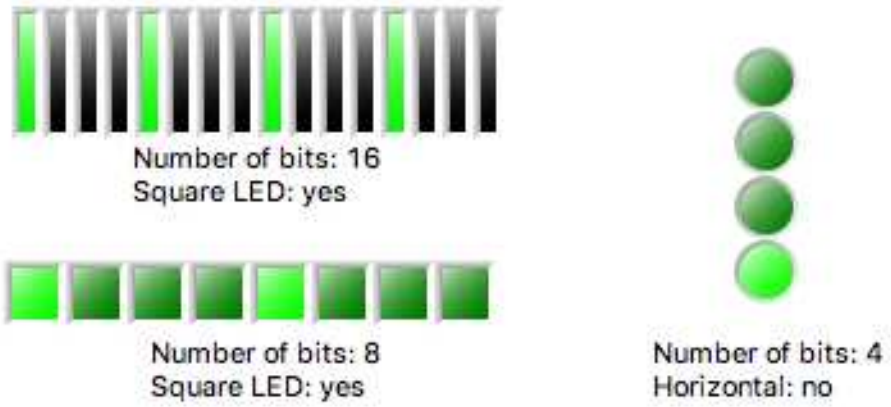
Intensity Graph

Advanced widget for diplaying a 2D array as an image. This widget is meant to be used in combination with scripts. Further documentation pending.



Byte Monitor

Specialised form of the LED widget that shows the numeric long value of the connected PV as a series of LEDs, each LED lighting up if the corresponding bit is true. The rightmost or bottom bit corresponds with the least-significant bit of the PV, but this can be reversed using the **Reverse Bits** property.



6.6.3. Control Widgets

Action Button

A button that can be used to trigger Actions. Use the **Toggle** property to have the option of adding a

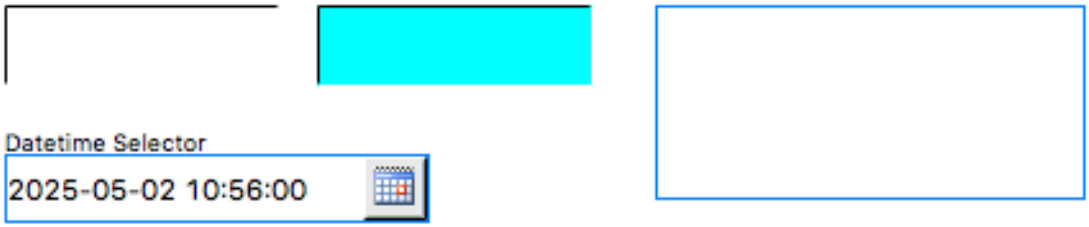
different action on press and release.

Menu Button

A button that will show a menu when it's clicked. The menu will be filled with either the actions from **Actions** property or the string values from the connected enumerated PV.

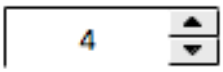
Text Input

A widget that allows the user to write data to the connected (writable) PV. For dates set the **Selector Type** to Datetime.



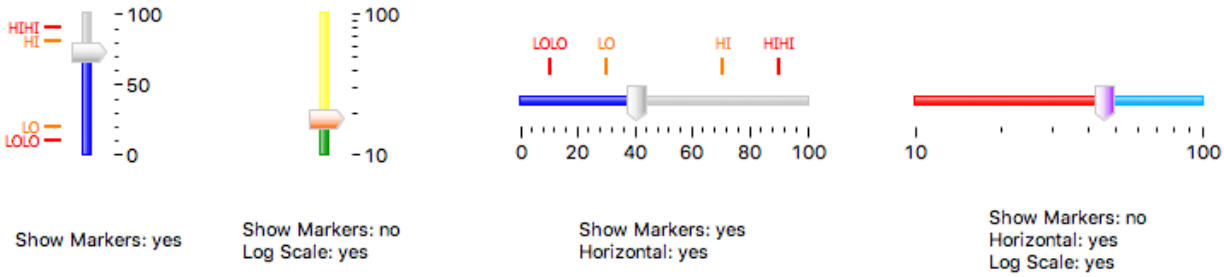
Spinner

The Spinner widget is similar to the Text Input, but allows updating the PV in incremental steps using up and down arrow buttons.



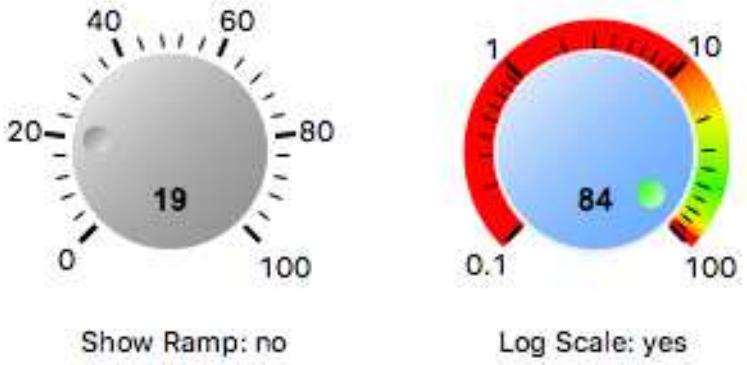
Scaled Slider

This widget allows adjusting the connected PV value using a configurable slider.



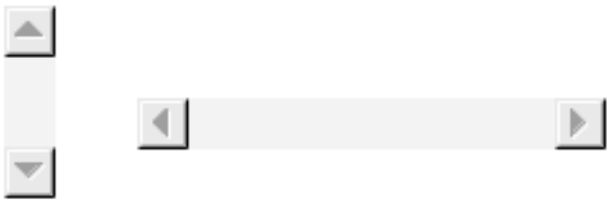
Knob

Knobs allow adjusting the connected PV value by dragging the thumb around.



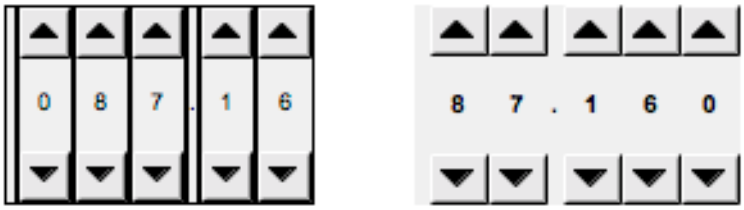
Scrollbar

Scrollbar that allows adjusting the connected PV value by. Useful as a building block for advanced display customizations.



Thumb Wheel

A widget that allows adjusting the decimal value of a PV digit by digit.



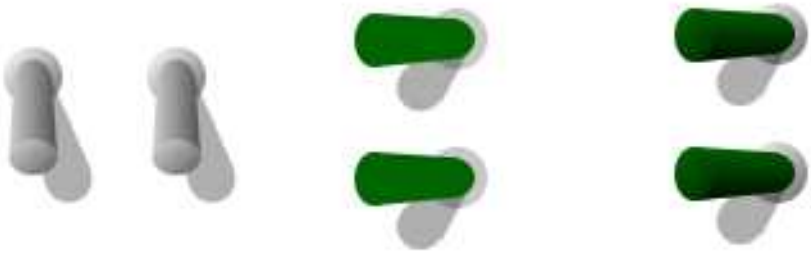
Integer Digits: 3
Decimal Digits: 2

Integer Digits: 2
Decimal Digits: 3

Boolean Switch

A Boolean Switch is able to write 0 or 1 to a single configurable bit of the connected PV or the whole PV if the **Bit** property is set to -1.

If **Width** is greater than **Height**, the switch will render horizontally, otherwise vertically.



Boolean Button

A Boolean Button is able to write 0 or 1 to a single configurable bit of the connected PV or the whole PV if the **Bit** property is set to -1.

Square: No
On Label: ON

Square: Yes
Off Label: OFF

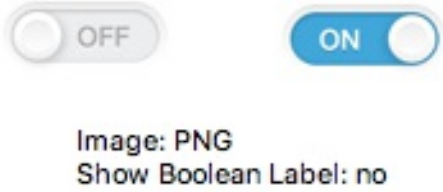
Square: Yes
Off Label: Open
Show LED: no

Square: No
On Label: STOP
Off Label: Run
Show LED: no

Image Boolean Button

Same as the Boolean Button widget, but you can customize the look further by specifying your own

images with the **Off Image** and **On Image** properties. The selected images must be present in the workspace.



Check Box

A Check Box is able to write 0 or 1 to a single configurable bit of the connected PV or the whole PV if the **Bit** property is set to -1. Use the **Label** property to render a value next to the check box.

Radio Box

A Radio Box allows the user to choose between a configurable set of values. Use the **Items** property to specify these values.

Choice Button

Similar to the Radio Box, but with buttons. Toggling a button within the widget, untoggles the previously active button.

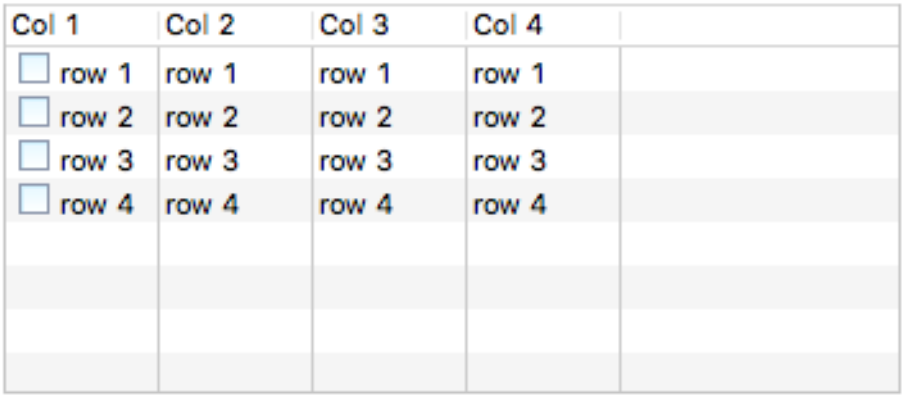
Combo

The Combo widget, as well, makes the user choose between one of its **Items**.

6.6.4. Other Widgets

Table

Advanced widget for structuring tabular data. Does not connect to PVs by itself. Its main use comes when combined with scripts.



Web Browser

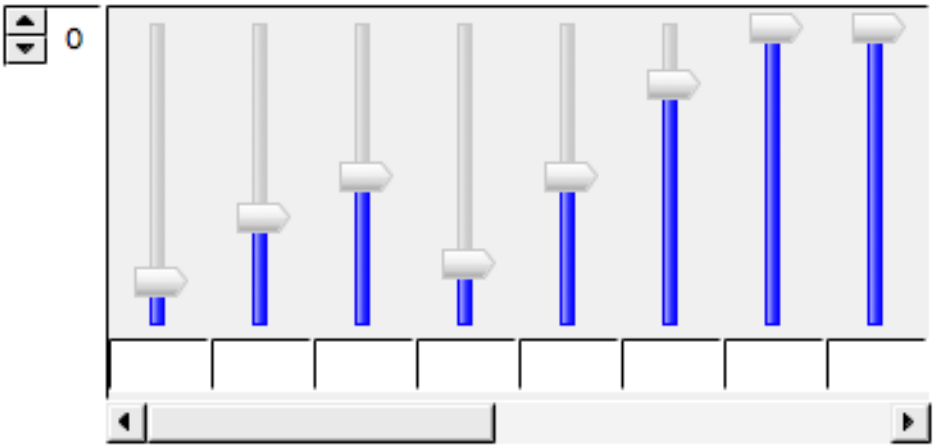
The Web Browser widget allows incorporating a web browser inside a display. This may be useful in managed environments, but in general we would recommend using a standalone web browser. The option is there though. Choose the start page with the **URL** property.

Array

Container widget that repeats another widget. First create an Array instance, than drag another widget on top of it and release. The widget will be repeated for the defined **Array Length**.

Modifying the property of any child modifies it for all the other children as well.

The Array widget must be connected to a an array PV.

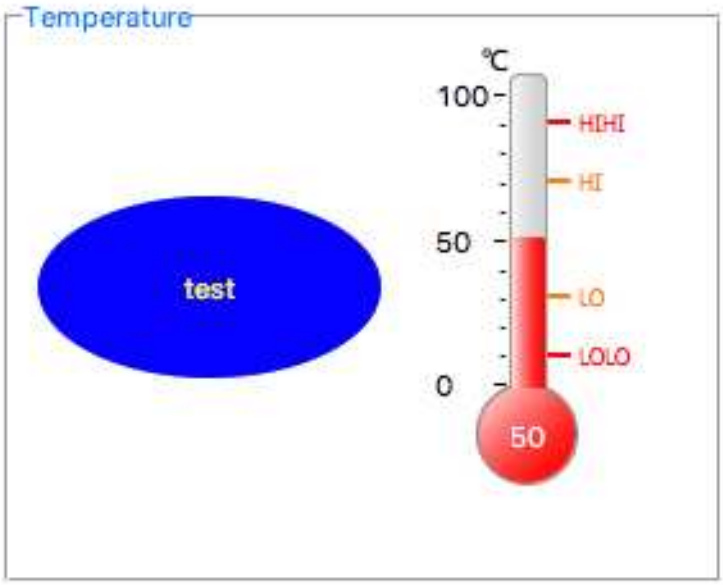


Grouping Container

You can group widgets together using a Grouping Container. Drag any widget on top of it to make it part of the group. Being widgets themselves, grouping containers can have their own backgrounds and borders.

To remove a widget out of a group, select it and drag it out of its container. To remove a group, right-click on it and select **Ungroup**. To add existing widgets to a new group, select them, and then choose **Group** from the right-click menu.

To prevent nested widgets from being individually selectable, set the **Lock Children** property of the group to yes.



Linking Container

Displays can include other displays, or groups of other displays. This is particularly useful to avoid duplication when creating shared components, such as top bars or side bars.

Define the included OPI file with the **OPI File** property, then optionally define the group with the **Group Name** property. If you don't specify a group the entire display is embedded. Tweak properties such as the **Resize Behaviour** to get your desired outcome.

Tabbed Container

Creates a stack of tabs. Define the number of tabs with the **Tab Count** property.

Sash Container

Use the Sash Container to create resizable split panes.

Horizontal: yes

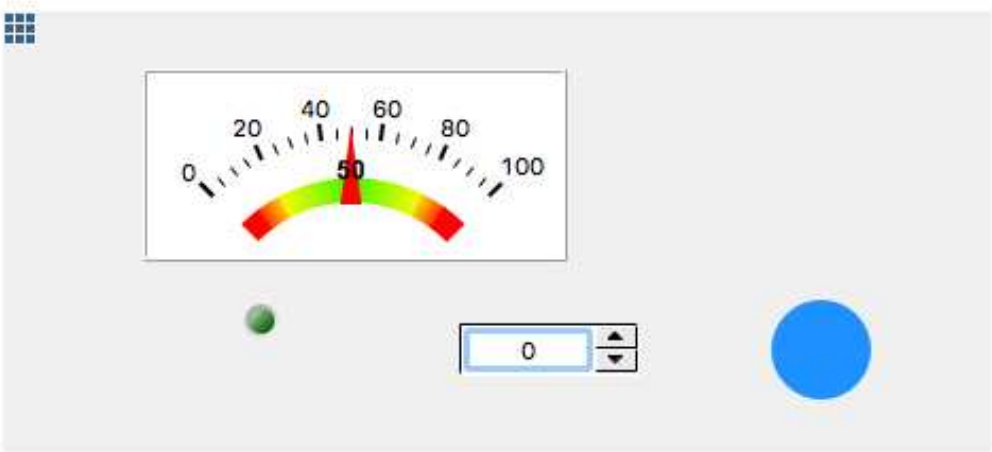
Horizontal: no

Grid Layout

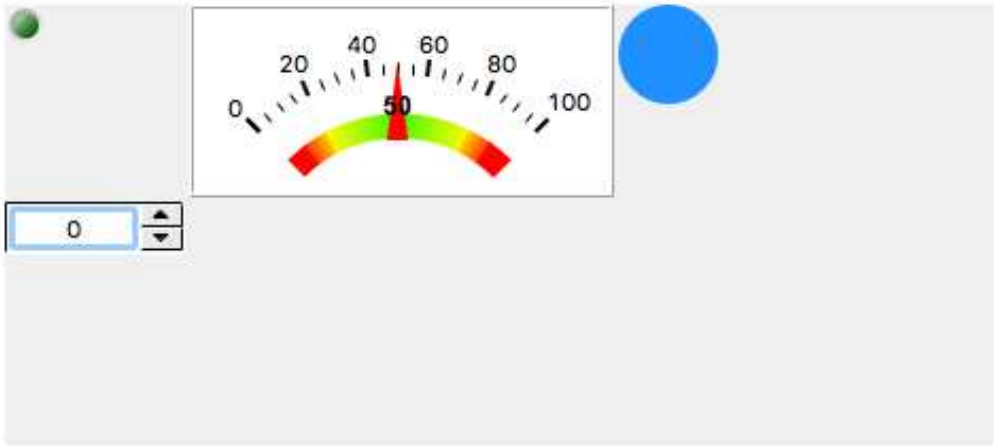
This is a special type of widget that is meant to be attached to a container or to the display itself. At runtime it reorganizes the children of that container in a grid. This usually works best with similarly sized widgets.

Create a Grid Layout by selecting it from the Palette, and clicking on top of the targeted container. You'll notice a grid decoration element in the top left of the container.

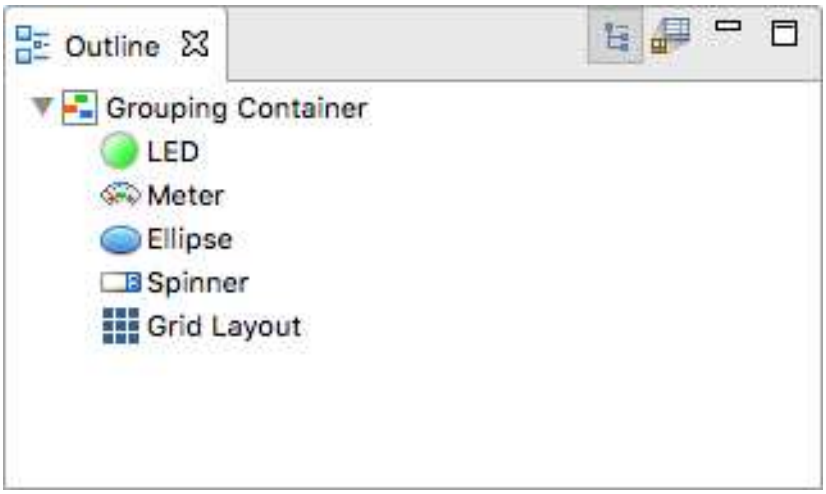
For example, while editing it may look like this:



But at runtime, with 3 columns, it renders like this:

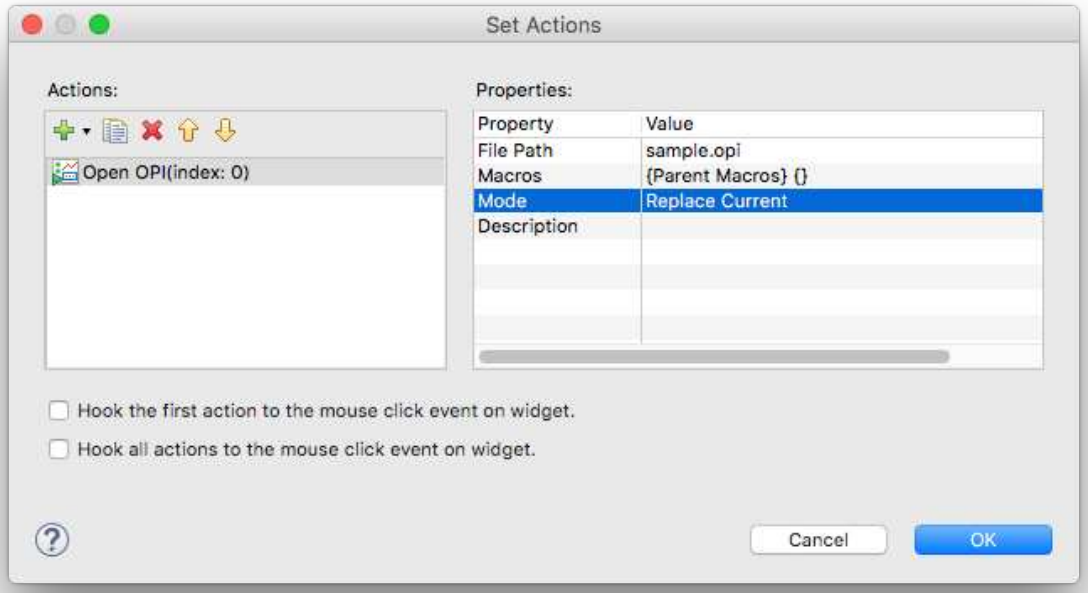


The order of the container's children is important for this layout, and can be modified in the **Outline** view.



6.7. Actions

Widgets have an **Actions** property which is used to trigger actions upon user interaction. The common case is to associate Actions with Control Widgets, but in principle it can be made to work with other widgets as well.



The list of available actions currently includes:

Open OPI	Often used in combination with Action Buttons to organize displays hierarchically. Indicate the workspace path to the OPI with File Path. Use the Mode to select whether the OPI should by default open in the same tab. Note that the runtime user can override this default behaviour by right-clicking the button.
Write PV	Writes the specified value to a PV. The variable $(pv_name) is automatically substituted with the PV attached to the widget.
Execute Command	This executes a command on your operating system. It does not execute a telecommand (although we will probably add such a feature too)
Execute JavaScript	Execute a JavaScript. Link to a script file in your workspace, or alternatively embed it into the Action
Execute Python Script	Execute some Python script. Link to a script file in your workspace, or alternatively embed it into the Action
Play WAV File	Plays the specified sound file
Open File	Opens a workspace file with the default handler
Open Webpage	Open the specified web page with the integrated web browser

6.8. Rules & Scripts

Making OPI displays is flexible using the Palette and the Properties, but the resulting displays are still fairly static. But what if we want to make the position of a widget dynamic based on a PV? Or if we want to dynamically change widget colors?

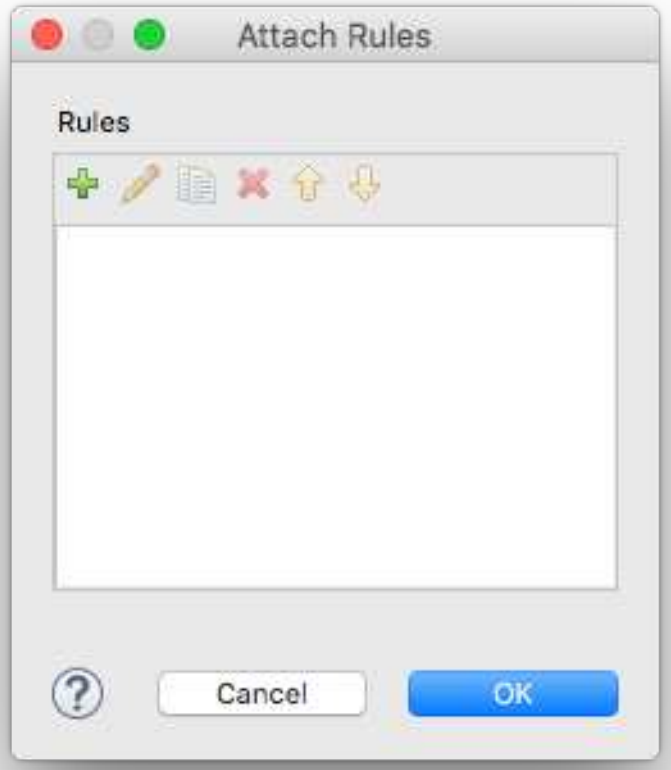
Rules

You may have noticed that every widget has a **Rules** property. Rules are a user-friendly way for adding dynamic behaviour to widgets. They are most often used for changing the widget's properties at runtime.

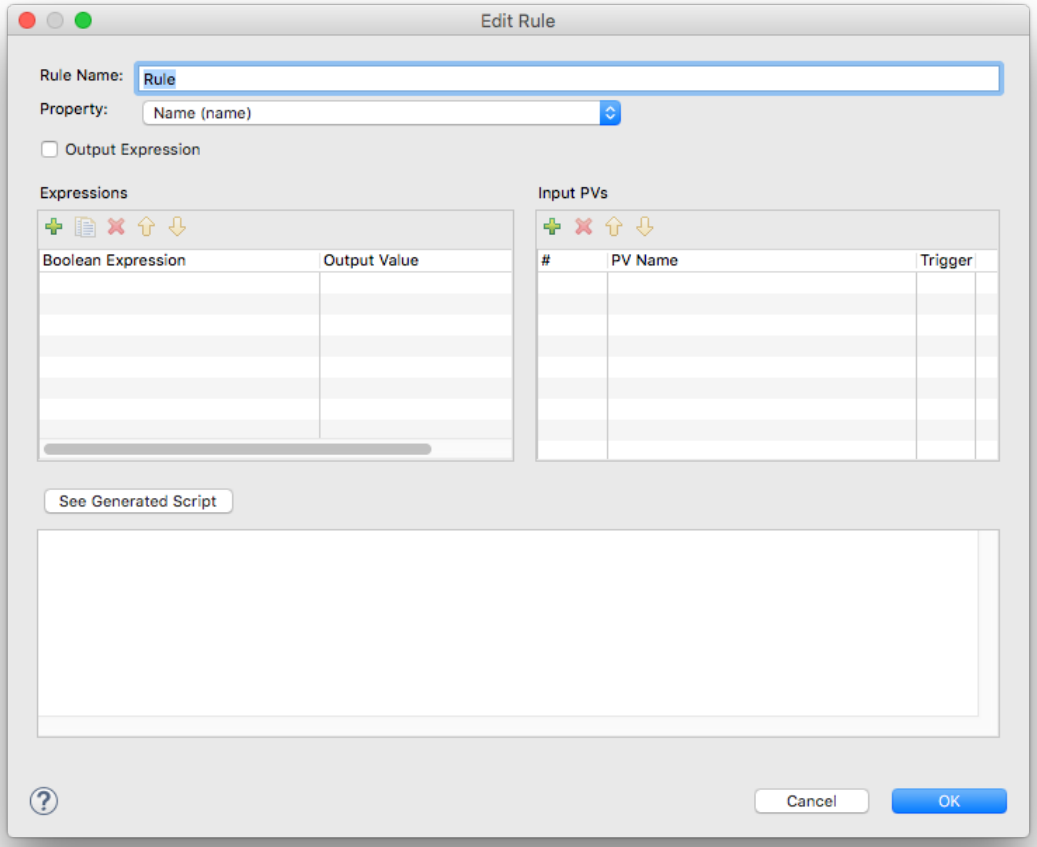
Example

This is best explained with an example. Suppose we have the bright idea to make an LED square when it is off, and round when it is on. The static properties would not allow for such a scenario, we therefore add a rule.

1. Edit the **Rules** property to pop up this dialog.



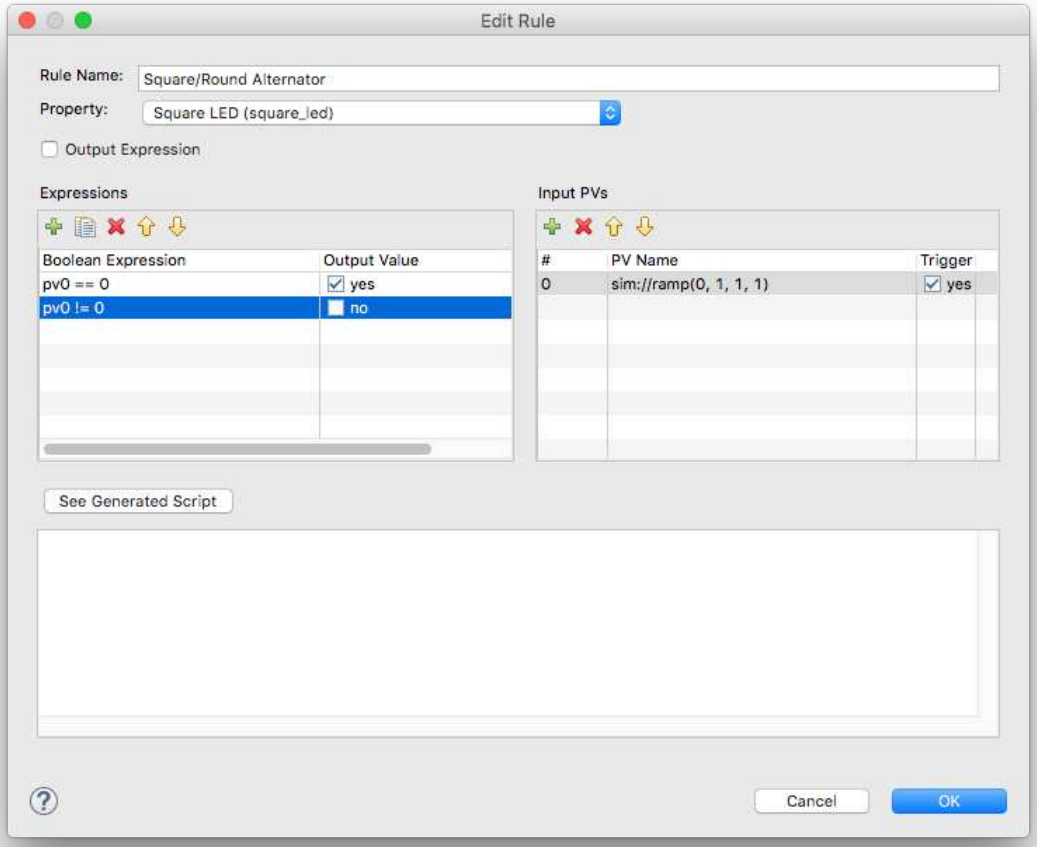
2. Clicking the plus icon gives you this dialog:



3. The first thing to choose is the rule's target **Property**. So select **Square LED**.

4. In the right **Input PVs** table add your input PV. In this example we chose to generate an alternating 0/1 value using a simulated PV. Notice the sequential number in the # column. The first PV is numbered 0. Make sure to check the Trigger checkbox as this will then trigger the execution of the rule whenever the PV's value is updated.

5. Now in the **Expressions** table, fill in your conditions in the **Boolean Expression** column, and add the desired value of the rule's property in the **Output Value** column. The double value of the top-most right PV is available as the variable pv0. The next PV in the list (if applicable) is available as the variable pv1, etc.



6. Confirm your dialogs, save your display and refresh a runtime view of it. You should see the LED's shape now alternating between square and ground.

One can see that this example could be made arbitrarily complex by adding more rules and/or expressions.

Boolean Expression

This input field needs to be expressed in JavaScript. The **Input PVs** are available in different formats:

Type	Example
Double Value	pv0 == 2.2
String Value	pvStr0 != 'abc'
Integer Value	pvInt0 >= 2

In addition, you can access the numeric alarm state of an input PV.

Alarm	Example
Invalid	pvSev0 == -1
No Alarm	pvSev0 == 0
Minor Alarm	pvSev0 == 1
Major Alarm	pvSev0 == 2

If you wish to set a property value that always applies, use true (or 1==1) as the Boolean Expression.

Output Value

The exact form that the **Output Value** column adopts depends on the type of the property. Some properties are colors, so you would see a color picker, other properties expect text, and the above example was a boolean yes/no, so we got a checkbox.

Scripts

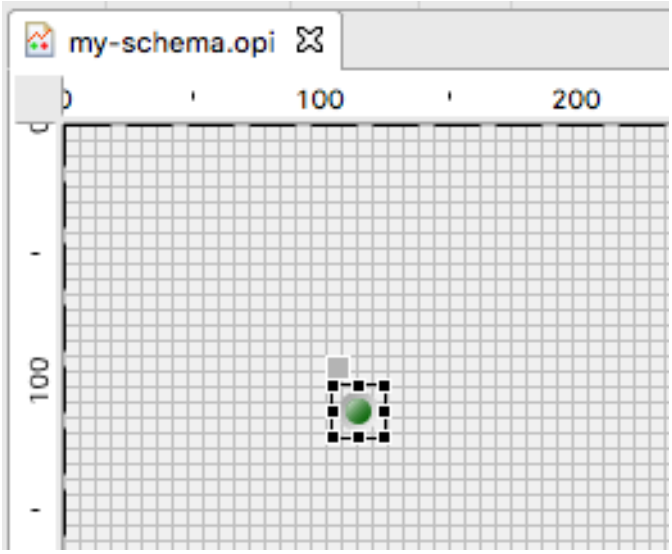
For more advanced dynamic runtime behaviour, we can write scripts (actually Rules are a thin layer on top of scripts). With scripts we can write arbitrary logic that can dynamically manipulate just about any combination of properties for a widget.

Yamcs Studio supports two dynamic languages: JavaScript and Python. Both languages can be used to the same effect, and are available without any external dependencies. As of now, there is no advanced editor support bundled with Yamcs Studio though, so scripts are edited with a plain text editor.

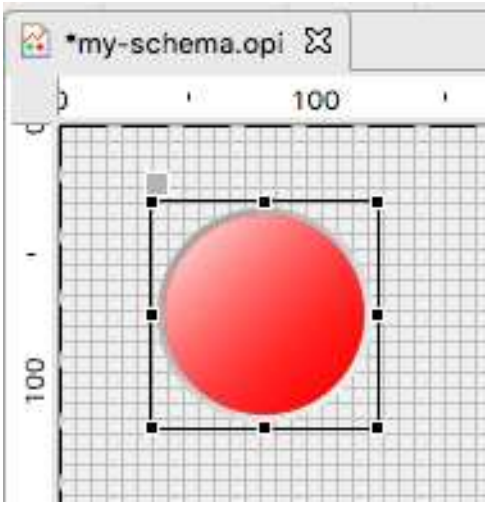
6.9. OPI Schema

The default property values of any widget can be overriden via an **OPI Schema**. A schema is an OPI file like any other, but which contains a set of widgets whose properties are used as defaults.

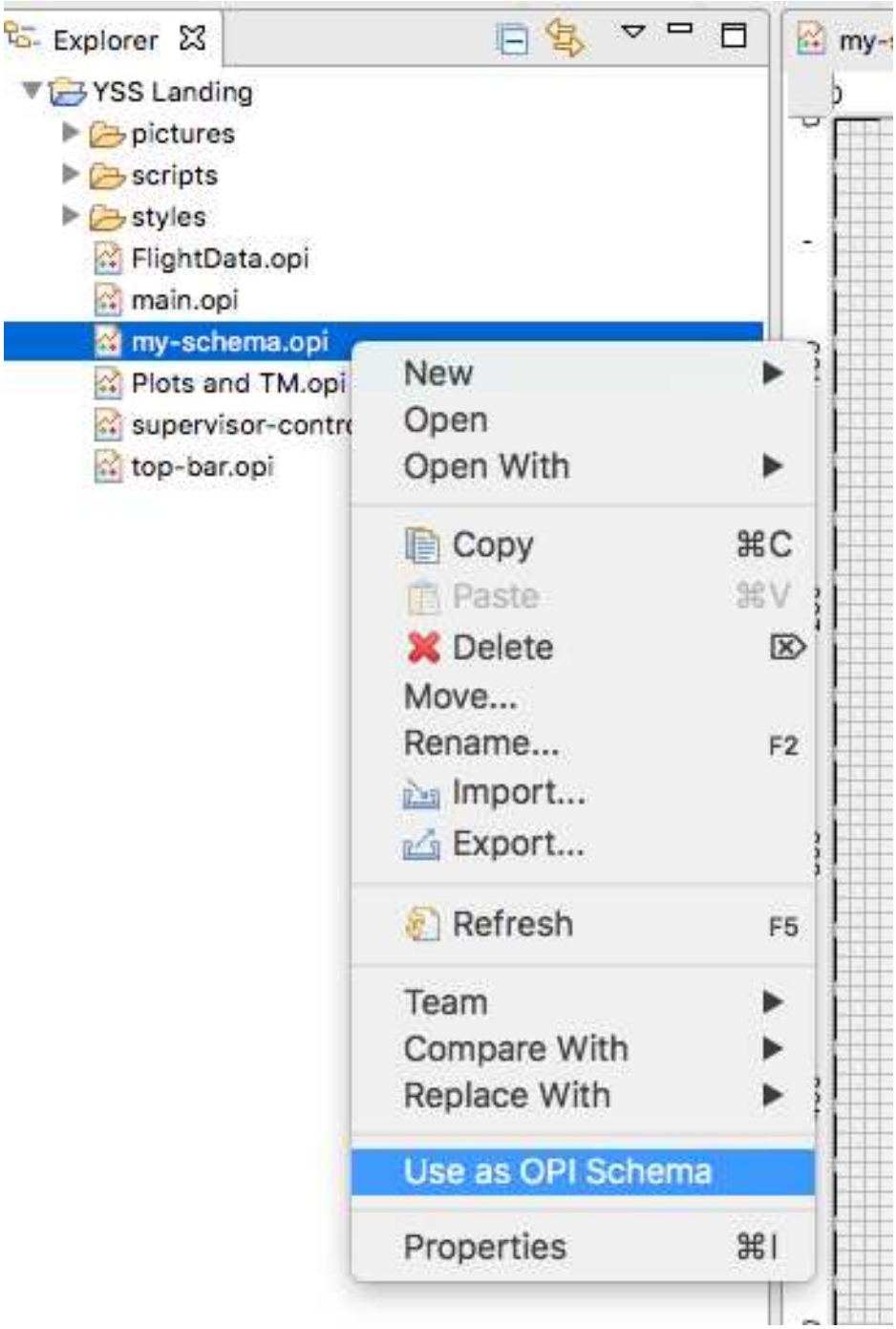
This is best illustrated with an example. Let's say we want to modify the defaults of the LED widget. Create a new OPI file and add a LED to the canvas (anywhere).



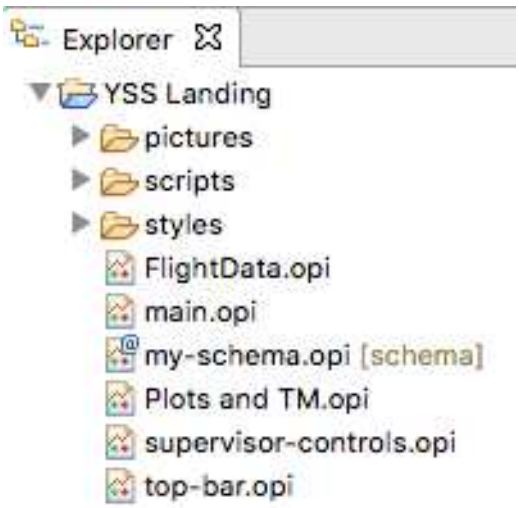
Make the LED bigger and change the colors.



Save your changes. Then right-click your file and select **Use as OPI Schema**.



Notice your schema file now shows a small decorator in the Explorer view:



Whenever you add a new LED to any display, it will now by red and big by default. The same principle can be applied to any other properties of any of the available widgets.

A schema may contain multiple widgets, however if there are multiple widgets of the same type, only the first occurrence is considered. This is the widget that appears the highest in the Outline view .

Only one OPI Schema can be active at a time. To unset the active schema, Right-click it in the Explorer view and untick **Use as OPI Schema**. This will return Display Builder to its default behaviour.

Chapter 7. Views

7.1. Archive

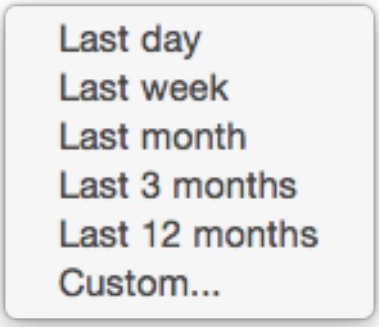
The Archive view represents a visual view of the data stored in the Yamcs archive. Through this view we can also initialize and control replays of archived telemetry.

User Interface

The Archive view always works on a range of indexed data, which it fetches from the server. All further actions like zooming happen client-side on the loaded data range.

Choosing a Data Range

As a first step you should select your data range. Click the pull-down icon to bring up this menu:



You can choose one of the predefined half-open time intervals, or you can select **Custom…** to specify your preferred range. Ranges can be half-open, which means they will always grow to include more bordering data as it becomes available.

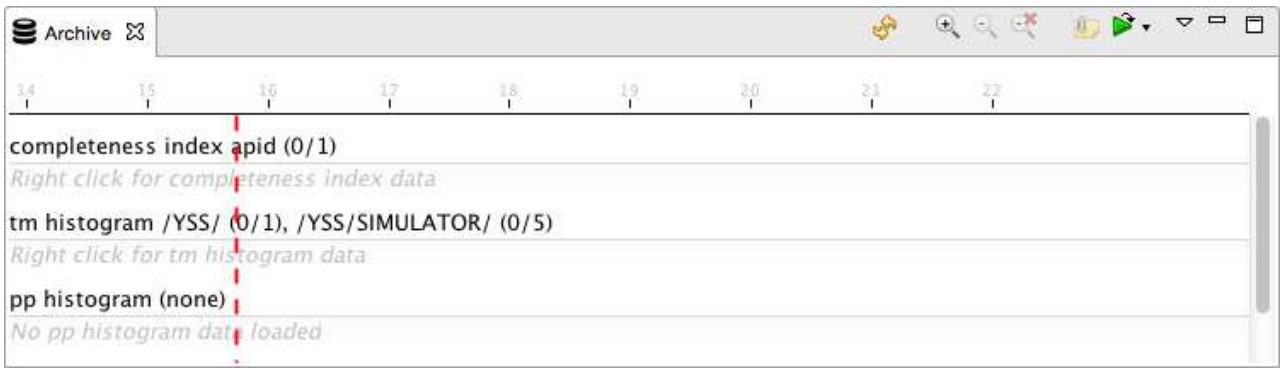
If you choose for example **Last Day**, Yamcs Studio will fetch an index of the archive for that time period, and refresh your view.

Your chosen data range is stored in your user preferences and will be restored the next time you open Yamcs Studio.

Selecting Data

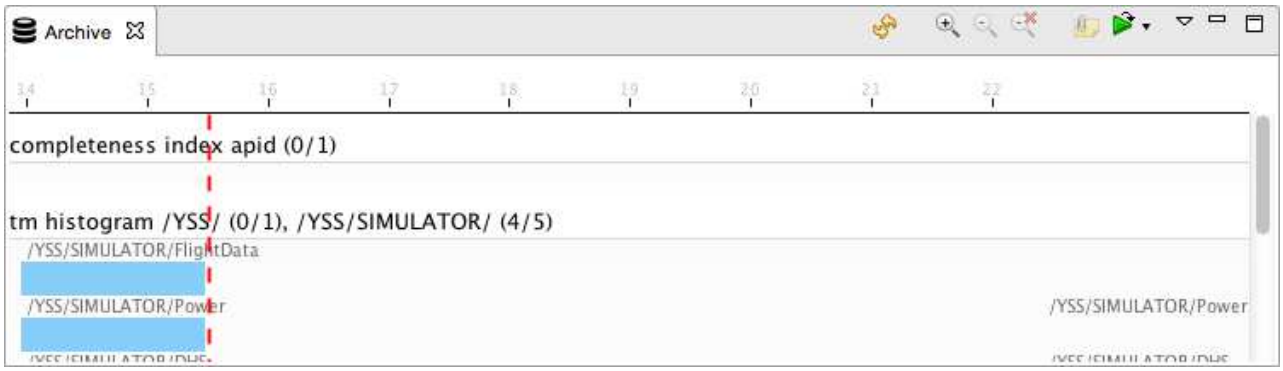
If this is the first time you have opened Yamcs Studio on your workstation, you won't see anything other than some empty zones named:

- Completeness Index APID
- TM Histogram
- PP Histogram
- CMDHIST Histogram



You need to choose which index data you actually want to display in your view. If there is data available for a zone, you can right-click it to bring up a pop-up menu where you select **Add Packets > … > All Packets**. Your view will then update to show the selected packets.

We say *packets* since this is typically what we are interested in when browsing the Archive, but any recorded data can in reality be displayed through the Archive view.



Note that the view does not refresh itself, so hit **Refresh** whenever you want to load the latest data for your selected time range.

Navigating

The vertical red locator shows the current time as provided by Yamcs. When we hover the mouse over the view, a greyed-out locator indicates the current mouse position.

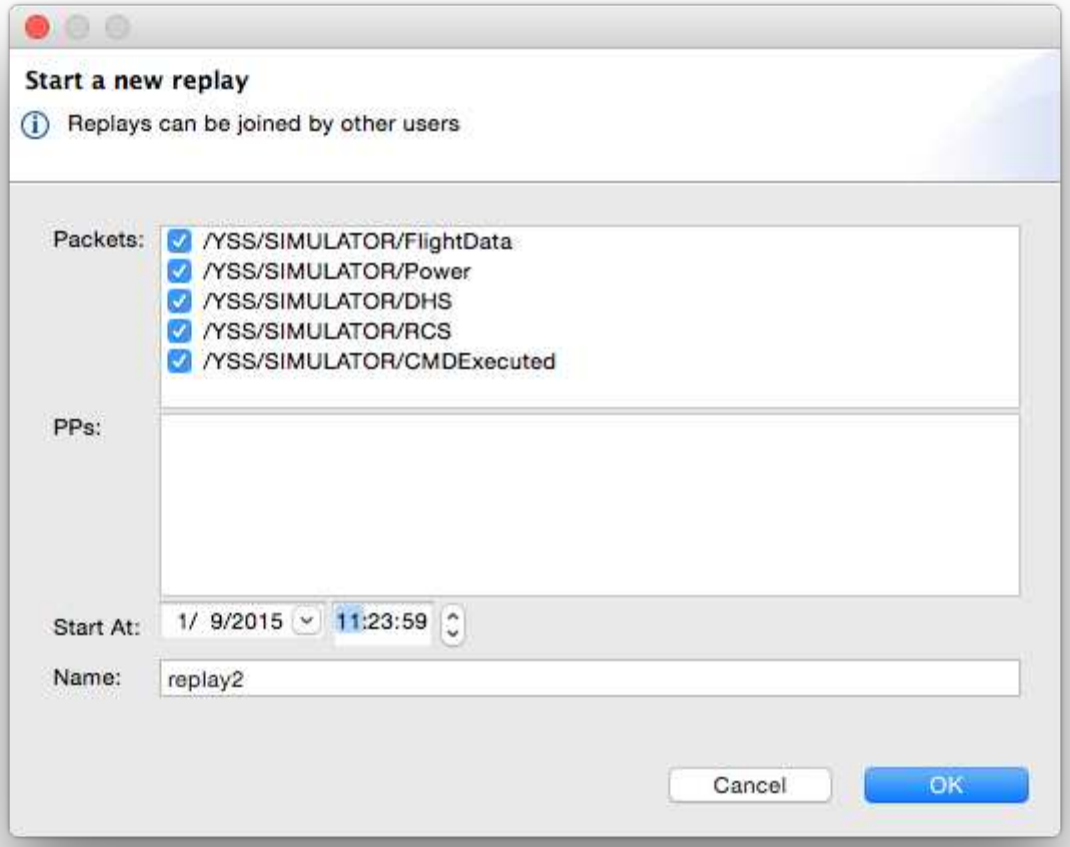
It is also possible to **Zoom In** or **Zoom Out**. If you are interested in a specific range, select a time range by clicking and dragging your mouse over the range before you zoom in using the zoom in button.

Notice that as you are zooming in and out, a horizontal scroll bar appears. This allows you to scroll left and right within the initially load time range.

To clear your zoom stack, select **Clear Zoom**.

Replaying data

We can use this view to replay archived data. Click **Replay**.



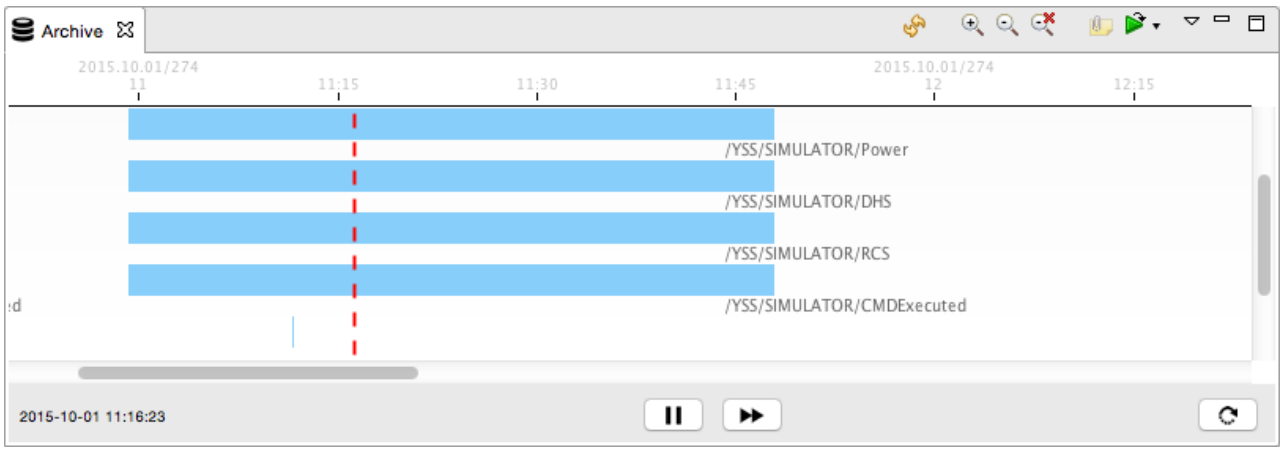
In the dialog box, confirm or filter the suggested selection of data. Currently only telemetry packets and processed parameters that were made visible in the Archive are part of the selectable data.

Modify **Start At** to the time and date you want to start the replay from.

Yamcs will create a processor (next to the built-in *realtime* processor) with the name that you provide in the **Name** field. The exact name that you choose is of no importance (although it needs to be unique), it helps you to identify the processor.

Click OK to start the replay. Yamcs Studio will reset any operator displays you may have opened, and will automatically switch to the newly created replay processor, as visible in the processor indicator in the top-left of your window.

Notice also that the Archive view is now equipped with an additional control bar.



The red locator shows you the current time of the replay processing. Double click anywhere to the left or to the right to make the processing jump to another point in time.

Click **Pause** ❚❚ to pause the processing, and use **Forward** ▶▶ to increase the speed of the processing. This button currently cycles through 5 predefined speed steps.

▶▶	Original Speed
▶▶ 2x	2x Original Speed
▶▶ 4x	4x Original Speed
▶▶ 8x	8x Original Speed
▶▶ 16x	16x Original Speed

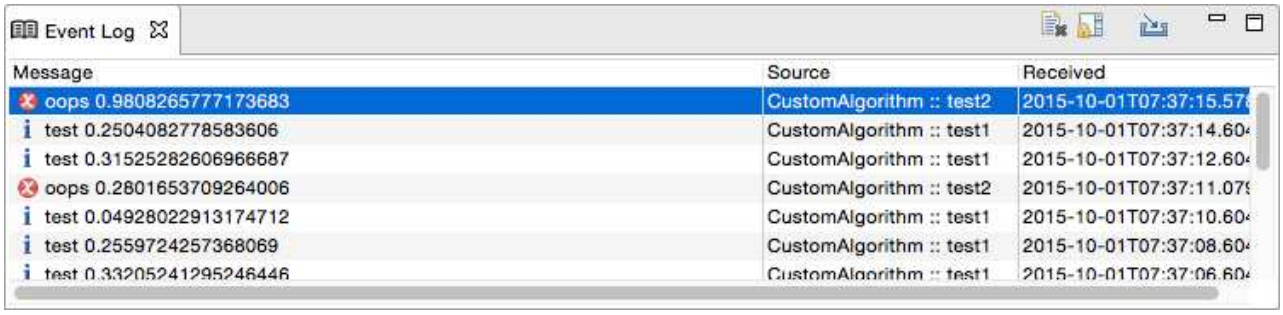
Speeding up will not cause any reset of your displays, as the same data is arriving, just faster.

When you want to leave the replay, there are several possibilities to follow:

- Hit **Return to Realtime** ;
- Open the pull-down menu ▼ next to the **Replay** button to choose a different processor;
- Click on the processor info bar in the top left of the window, to choose a different processor.

7.2. Event Log

The Event Log view displays events from Yamcs Server. This could be on-board events, or events generated by Yamcs itself, whenever something significant occurs.



To load events for an earlier time range, select **Import**.

Clear your view by clicking **Clear**. You can always re-import events again at a later moment.

When Yamcs Studio becomes aware of a new event, it will automatically select and reveal it. You can prevent this default behaviour by toggling the **Scroll Lock**.

7.3. Alarms

This view is under development, and will offer a centralised view of all active alarms, with the opportunity to acknowledge alarms or to mute any sounds. The Alarms view is not yet bundled in current copies of Yamcs Studio, but will be so in the short term.

Currently the only way to be noticed of alarms in Yamcs Studio is by following events (if your Yamcs Server is configured to report alarm state changes as events), or by manually iterating your displays to look for widgets that have red (= major alarm) or orange (= minor alarm) colored borders around them. The upcoming Alarms view aims to improve this process.

7.4. Command Stack

The Command Stack allows operators to prepare stacks of commands for manual command issuing. The process is intentionally click-heavy to make sure that operators are aware of what they are doing.

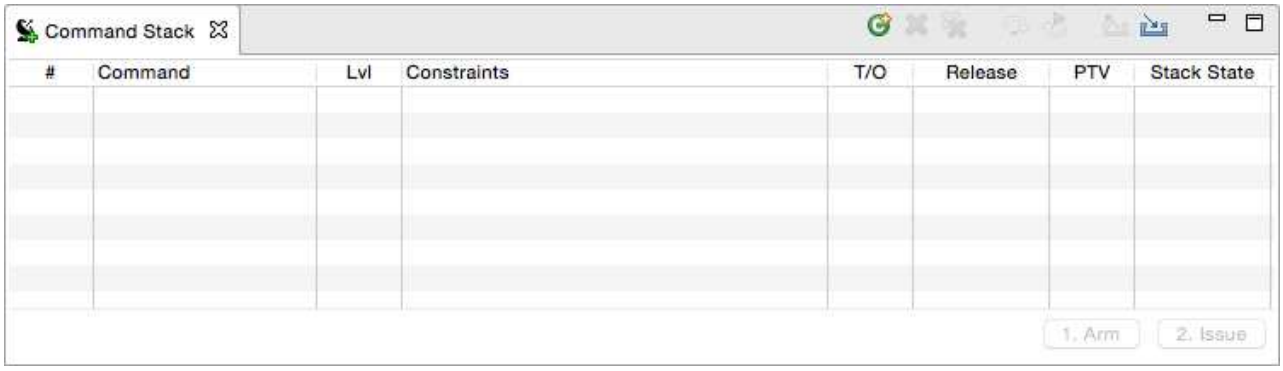
When you issue commands from Yamcs Studio, these are queued to Yamcs Server, which will perform any further steps.

We're keen on bringing many improvements to this view for better editing, but it is usable in its current state.

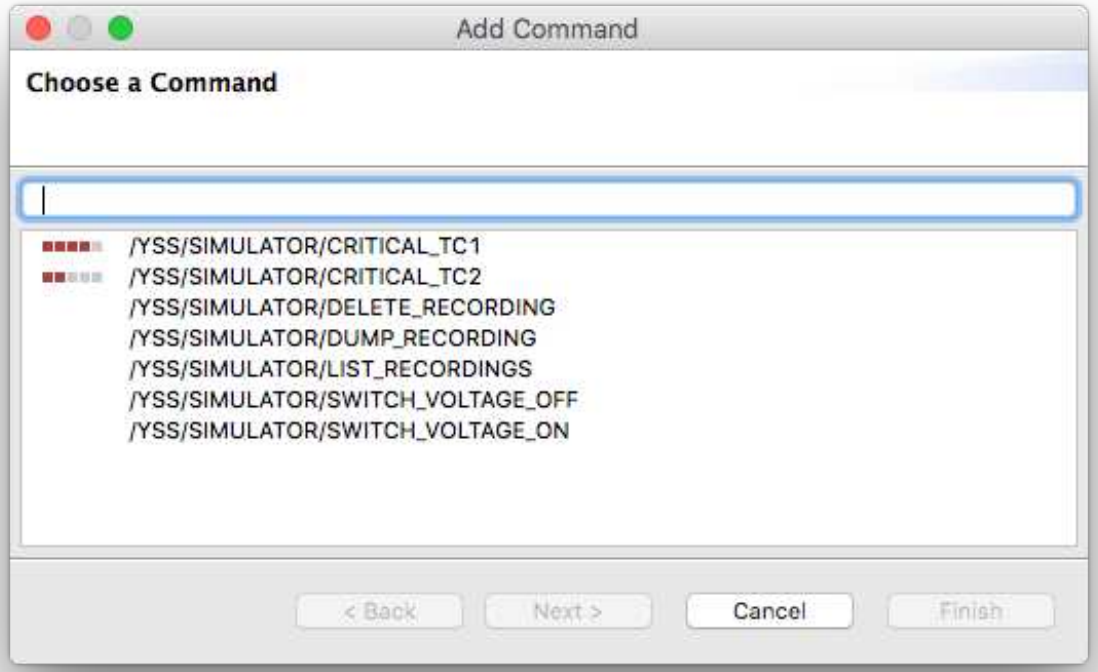
Preparing a Stack

You can prepare a stack of commands only when you are connected to Yamcs. Yamcs Studio uses this connection to retrieve the list of available commands or to perform server-side validations.

When you start Yamcs Studio, the Command Stack view (available from the Display Runner window) is by default shown below the operator displays. If you can't find it back, select **Window > Show View > Command Stack**.



Add a command by clicking the **Add Command** button.



This opens a wizard dialog with the list of available commands. You can filter the list with the search box on top.

Commands are identified by their fully qualified XTCE name. This name matches the hierarchical structure of the commands as defined in the mission database of the connected Yamcs instance.

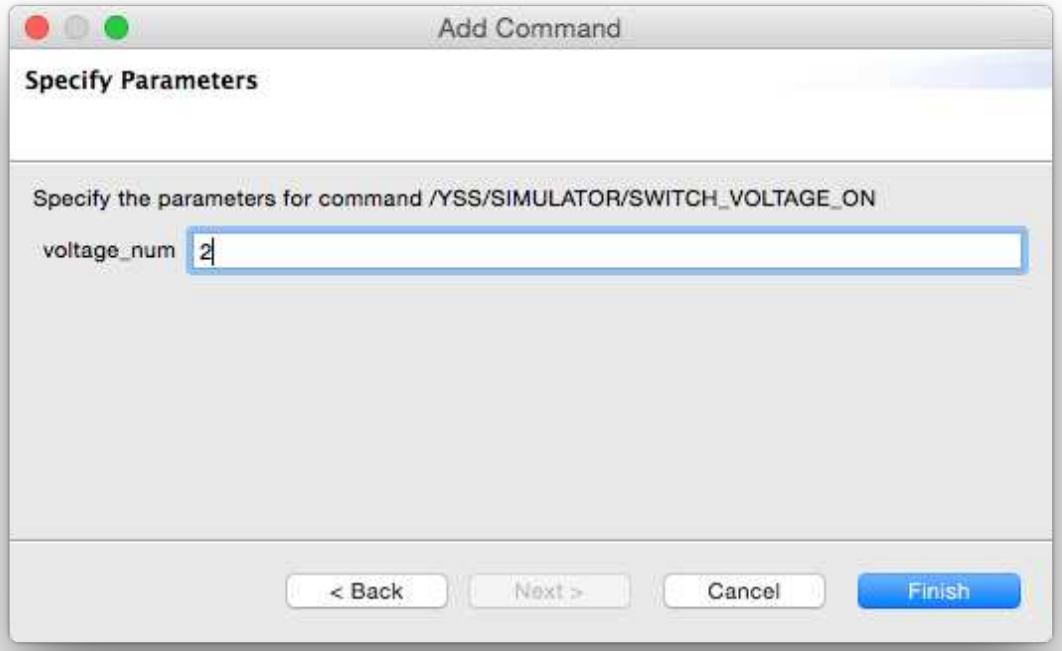
Commands can have varying levels of criticality (called *significance* in XTCE terminology). The icon in the leftmost column indicates the defined criticality for the command.

Icon	Criticality
	Watch
	Warning
	Distress
	Critical
	Severe

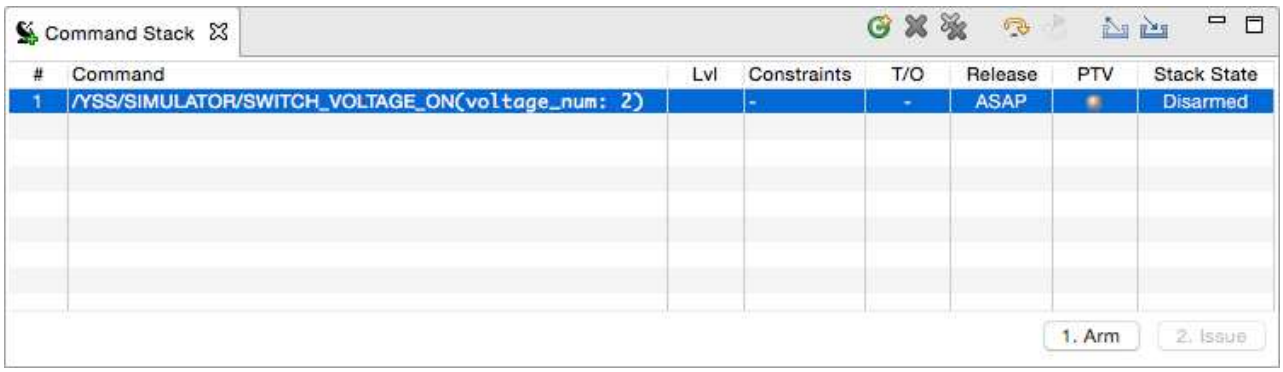
If an explanatory message regarding the criticality was defined in the mission database, this will show in the top title area of the dialog when the command is selected.

Once you've chosen your command, hit **Next** to go the next page in the wizard, where you can specify any arguments that need to be provided for the command. Currently, only numbers or text can be entered.

You can close the wizard from the first page as well by clicking **Finish** instead of **Next**. If the command requires any arguments, you will have a chance to add them afterwards as well by editing your stacked command.

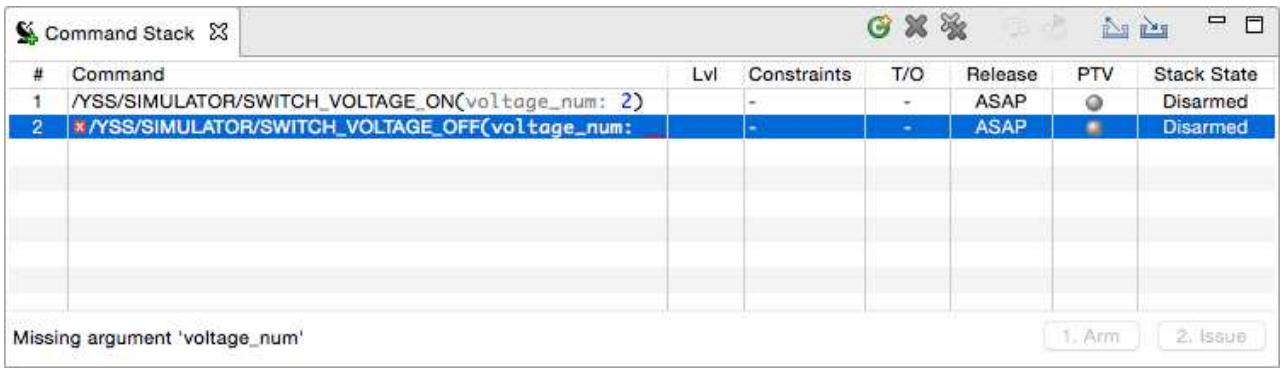


Click **Finish** to append your command to the end of your current stack.



You can review your provided arguments by double clicking the command. To remove the selected command from the stack select **Delete**. Clear the entire stack with **Delete All**.

If a stacked command does not pass static validation checks (sometimes referred to as SPTVs – Static PreTransmission Verification) it will be marked with error indicators. This will prevent the user from attempting further execution of the stack until the error is resolved.



It is not currently possible to move a command up or down the stack.

Executing a Stack

As soon as your manual stack contains at least one command, you can start execution. Execution is top-down, so select the first command of your stack.

There is currently no clear separation between a 'stack editing' mode and a 'stack executing' mode. We are happy with this given the current set of features, but may revise this concept as we add more features (e.g. moving commands up and down the stack).

In the bottom bar click the **1. Arm** button. This will 'arm' your command, and is a first required step before issuing it. There is currently no timeout on how long a command stays armed before being reverted to disarmed. If the command you are arming has a defined criticality (watch, warning, distress, critical or severe), you will receive a confirmation dialog first.

Once your command is armed it will say so in the Stack State column.

Arming a command is a client-side notion. There is no communication with Yamcs during this step. The intention of arming a command is to make the operator aware of his actions, especially when the command is significant. There is currently no support for arming multiple commands together and execution commands in batch.

Now that your command is armed, hit **2. Issue**. There will be no more confirmation dialogs. If the server refuses your request you will receive an error message. This can happen for example when a command is only to be executed under a certain context, and within a certain time frame. These type of settings are currently exclusively defined in the Yamcs Mission Database.

To follow what happens to your command, open up the **Command History** view, explained in the next section.

When the command is issued, the Stack State column is updated, and you can arm the next command in the stack to repeat the process.

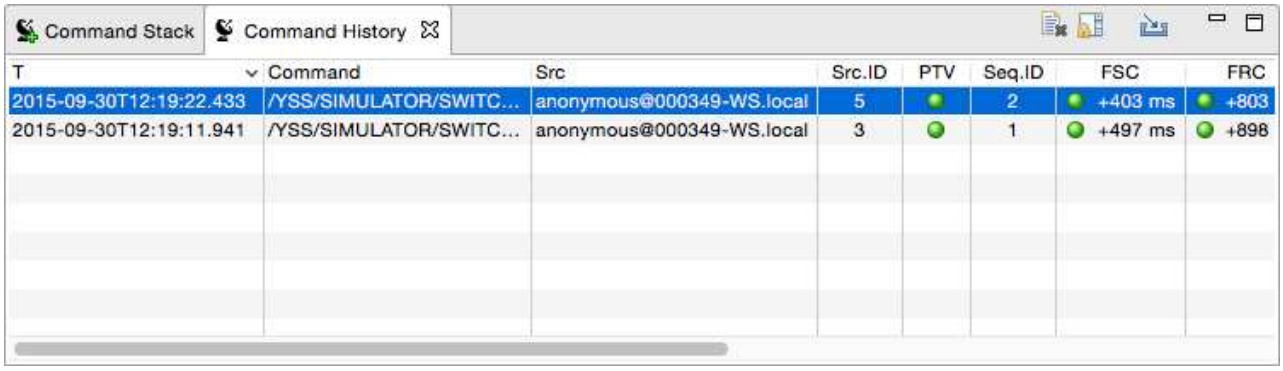
Skip the selected command by selecting **Skip**. Reset the execution state by selecting **Restart**. This will reset all Stack State columns to **Disarmed**.

Importing and Exporting a Stack

When you want to save a stack for future use, you can select **Export** to save your stack in XML-format in any of your drives. Likewise, reuse a stack which you set aside by selecting **Import**.

7.5. Command History

The Command History keeps track of all commands that were issued using Yamcs (not just by yourself, but by anyone).



It will by default only show commands that were received on the realtime processor since you started your copy of Yamcs Studio. To load the command history for an earlier time range, select **Import**.

Clear your view by clicking **Clear**. You can always import the cleared commands again at a later time.

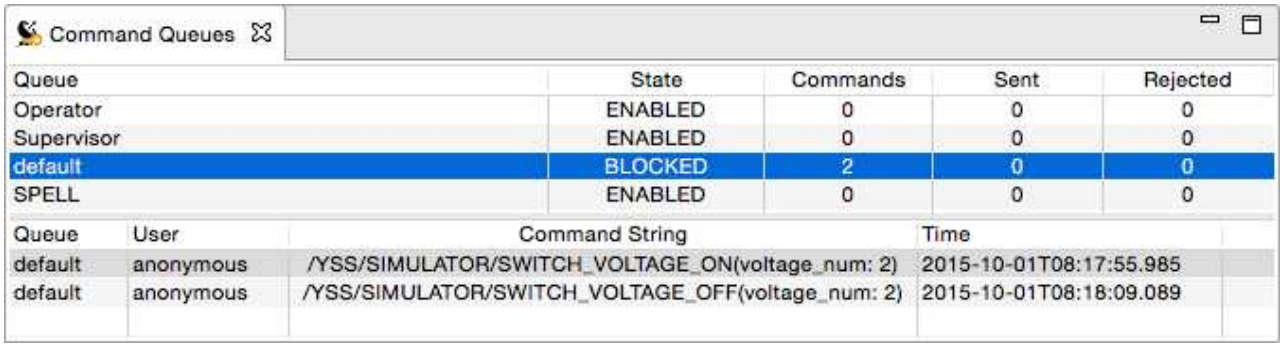
When Yamcs Studio becomes aware of a new command that was issued by Yamcs, it will automatically select and reveal it. You can prevent this default behaviour by toggling the **Scroll Lock**.

The displayed columns are as follows.

T	Time when the command was issued
Command	The command in textual format
Src	The origin of the command. Typically in *user@host* format
Src.ID	The ID that was given to the command by the issuing application. This number is assigned by the source application. In case of Yamcs Studio it is an incremental number that resets to 1 on every restart of Yamcs Studio.
PTV	Result of the Pretransmission Verification as performed by Yamcs. For example, some commands may only be applicable for 10 seconds and needs certain other parameters to be set to specific values. When the PTV bubble colors red, these type of context-dependent checks could not be validated, and therefore the command was not actually issued.
Seq.ID	The id that was determined by Yamcs before further dispatching the command. This is an incremental number that resets on every restart of Yamcs.
Further Columns	Indicate acknowledgments of ground hops as the command is being dispatched. The exact number and name of the columns depends largely on how Yamcs is deployed at your site. Yamcs typically calculates the state of these bubbles based on incoming telemetry. The bubble becomes green or red depending on the verification result. The column value shows the time difference with the issuing time *T*.

7.6. Command Queues

This view allows controlling the Yamcs queues from the point of view of Yamcs Server. With sufficient privileges, queues can be blocked or disabled.



The view is split into two panels:

- The upper panel contains a list with all the defined command queues. The queue name, the current state and the number of commands that are currently in the queue is displayed. Right-clicking on a command queue opens a menu with the possibility to change the state of the queue.

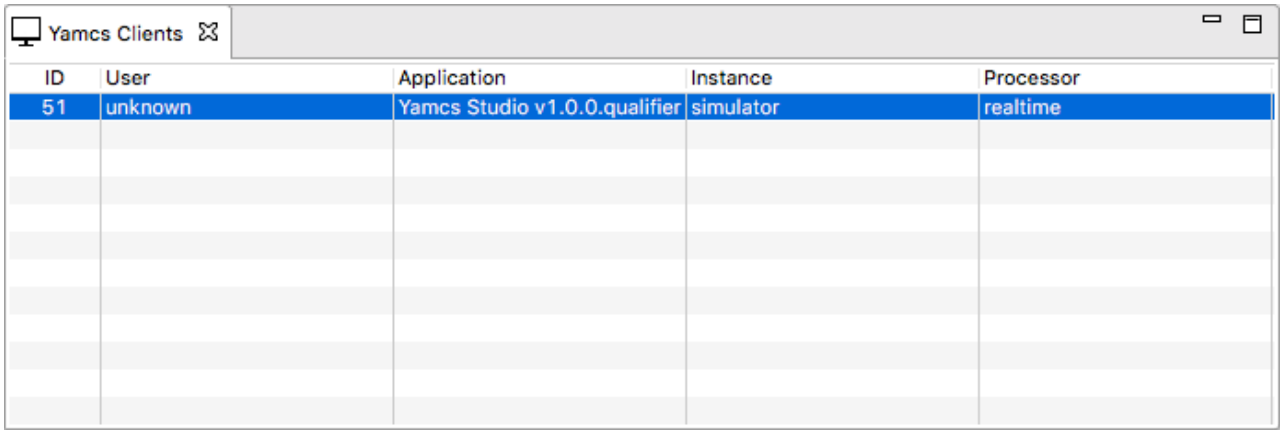
  In addition to setting the new state of the queue, the following additional actions happen when changing the state of a blocked queue:

  - blocked → disabled: all the commands in the queue will be automatically rejected;
  - blocked → enabled: all the commands in the queue will be automatically sent.
- The bottom panel contains the list of commands currently present in the queue that is selected in the upper panel. For each command the queue name, the command id, the user that has sent the command and the command string are displayed. The list is empty if the selected queue is not in the state blocked.

  Right-clicking on a command gives the possibility to **Send** or **Reject** the command.

7.7. Yamcs Clients

The Yamcs Clients view is a read-only view that shows you all the clients that are connected to the same Yamcs Server that you are connected to.



The displayed information includes:

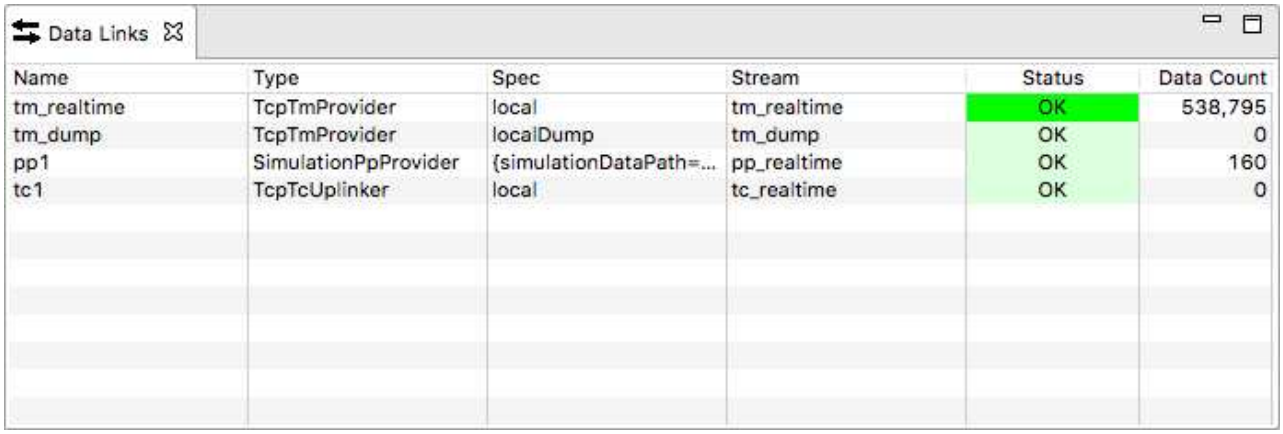
ID	The client ID assigned by Yamcs Server. Useful for debugging.
User	The name of the connected user, or unknown if Yamcs was not secured
Application	The name of the application that the user is using to connect to Yamcs.
Instance	The instance this user is connected to
Processor	The processor this user is connected to

7.8. Data Links

This view provides an overview of the data links of a running Yamcs server.

Data links represent input or output flows to Yamcs. There are three types of Data Links: TM (called TM Providers), TC (called TC Uplinkers) and PP (called PP Providers). TM and PP receive telemetry packets or parameters and inject them into the realtime or dump TM or PP streams. The TC data links subscribe to the realtime TC stream and send data to external systems.

There are different types of providers. For details refer to the Yamcs Server Manual.



Data Links

Name	Type	Spec	Stream	Status	Data Count
tm_realtime	TcpTmProvider	local	tm_realtime	OK	538,795
tm_dump	TcpTmProvider	localDump	tm_dump	OK	0
pp1	SimulationPpProvider	{simulationDataPath=...	pp_realtime	OK	160
tc1	TcpTcUplinker	local	tc_realtime	OK	0

The presented information includes:

Name	Identifier of this link as assigned by Yamcs
Type	The type of this link. For example, TcpTmProvider represents an input of TM over TCP
Spec	Configuration information passed to the provider instance. Significance depends on the type of provider
Stream	The internal stream where the data is either: ● sourced from (in the case of uplinkers), or ● published to (in the case of providers) It is perfectly valid for different providers or uplinkers to use the same stream.
Status	The status of this link. One of: ● OK – if the link is alive ● DISABLED – if the link was disabled ● UNAVAIL – if the link is enabled, but not available The Status background lights bright green if the data count increased within the last 1500 milliseconds.
Data Count	The number of data elements (e.g. packets) that this link published or received from its stream since Yamcs started running.

With sufficient privileges, you can enable or disable a link manually by right-clicking the applicable row and selecting **Enable Link** or **Disable Link**.

Chapter 8. Tuning

8.1. Command Options

Yamcs Studio accepts these command options:

-version
: Print version info and quit

-workspace /some/workspace
: Use the provided workspace. If unspecified the workspace is set to ~/yamcs-studio

-force-workspace-prompt
: Prompt for the workspace

8.2. Capturing Log Output

In case you need to debug an issue with a deployed Yamcs Studio client, it can be useful to capture the logging output. Instructions are specific to the platform.

Linux

Launch the executable from a terminal window while redirecting all output to a file named log.txt

```
./Yamcs\ Studio >log.txt 2>&1
```

Mac OS X

With Terminal navigate into the Yamcs Studio application bundle and launch the executable directly from there while redirecting all output to a file named log.txt. For example:

```
cd Yamcs\ Studio.app/Contents/MacOS
./Yamcs\ Studio >log.txt 2>&1
```

Windows

With Command Prompt navigate into the location where you installed Yamcs Studio and launch the executable while redirecting all output to a file named log.txt. For example:

```
"Yamcs Studio.exe" >log.txt 2>&1
```

8.3. Preference Defaults

Most user preferences are linked to the workspace and saved to a folder.metadata. Whenever a user creates a new workspace, the workspace starts with the default preferences.

These default preferences can be modified by adding or modifiying the Yamcs Studio.ini file in the installation directory of Yamcs Studio. This is often done to ensure that different workstations use similar site-specific configuration.

Some of the more common preferences are documented below.

org.csstudio.opibuilder/colors.list

List of named colors. Entries are separated by semicolons. Each entry is composed as NAME@R,G,B. For example: Major@255,0,0;Minor@255,128,0. You can choose any name, but note that the names Major, Minor, Invalid and Disconnected have a special meaning in Yamcs Studio. They are used for common decorations such as out-of-limit indicators.

org.csstudio.opibuilder/fonts.list

List of named fonts. Entries are separated by semicolons. Each entry is composed as NAME@FONT-STYLE-SIZE. For example: Header 1@Arial-bold-19;Header 2@Arial-bold-15. The font should be availabe on the system. Only 'Liberation Sans' (which is the default) is dynamically loaded when it is missing from the system, this is to ensure that the default font settings produce identical displays on all platforms. Note that font size is expressed in points, not pixels.

org.csstudio.opibuilder/hidden_widgets

Hide the specified widgets from the palette. Widgets are mentioned by their id. and separated by the vertical bar character.

org.csstudio.opibuilder/schema_opi

Workspace reference to the active Schema OPI. For example: /My Project/schema.opi

org.yamcs.studio.core.ui/singleConnectionMode org.yamcs.studio.core.ui/connectionString

When singleConnectionMode is set to true, Yamcs Studio will not open the Connection Manager window, but will only allow connections to a single Yamcs server defined in the connectionString