Yamcs Studio User Guide

Release 1.7.9-SNAPSHOT

Yamcs
MISSION CONTROL

Space Applications Services, NV/SA
Leuvensesteenweg 325
1932 Sint-Stevens-Woluwe
Belgium
spaceapplications.com
yamcs.org

Aerospace Applications North America, Inc.
16850 Saturn Ln, Ste 100
Houston, TX 77058
United States of America
aerospaceapplications-na.com

Copyright © 2025 Space Applications Services NV/SA. All rights reserved.

Contents

About

This is the Yamcs Studio User Guide. You can download Yamcs Studio from the download site. Yamcs Studio is available for Windows, Linux and macOS.

For notes detailing the changes in each release, see the Yamcs Studio Release Notes.

Yamcs Studio is a desktop client for use with Yamcs, the server. In its default configuration, it includes support for authoring and running operator displays, and comes bundled with various built-in displays that highlight different aspects of Yamcs (Alarms, Events, Commanding, Archive playbacks, etc.).

1. Overview

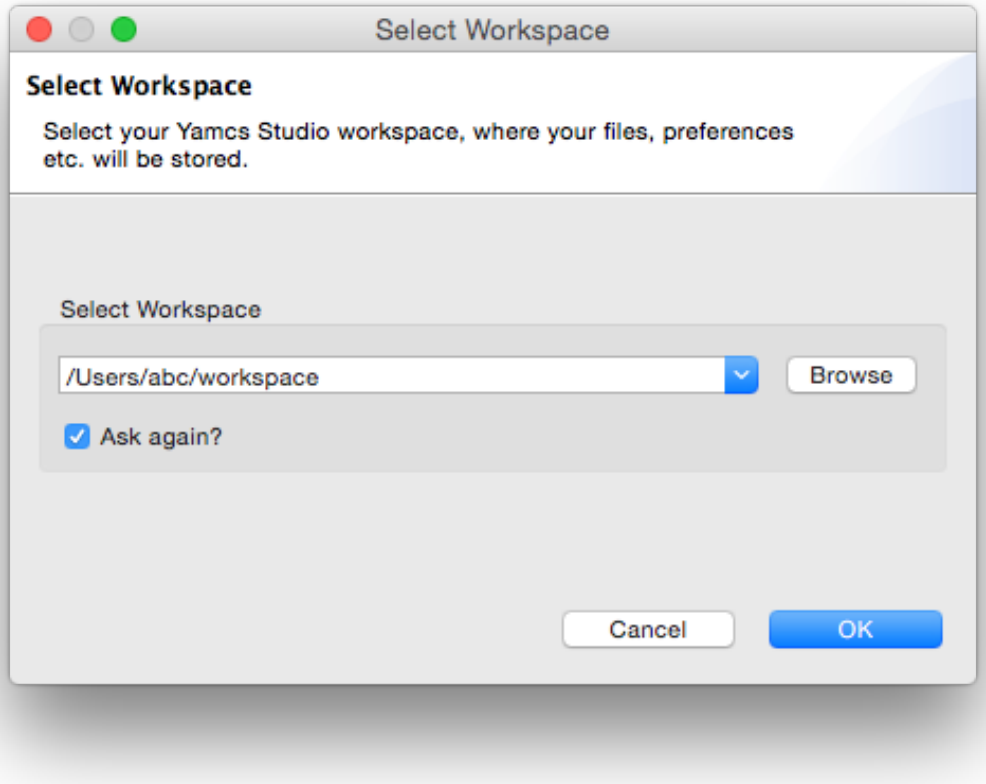
Yamcs Studio is a TM/TC front-end for Yamcs. It contains an editor for building synoptic operator displays and supports basic telecommanding.

1.1 First Steps

1.1.1 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).

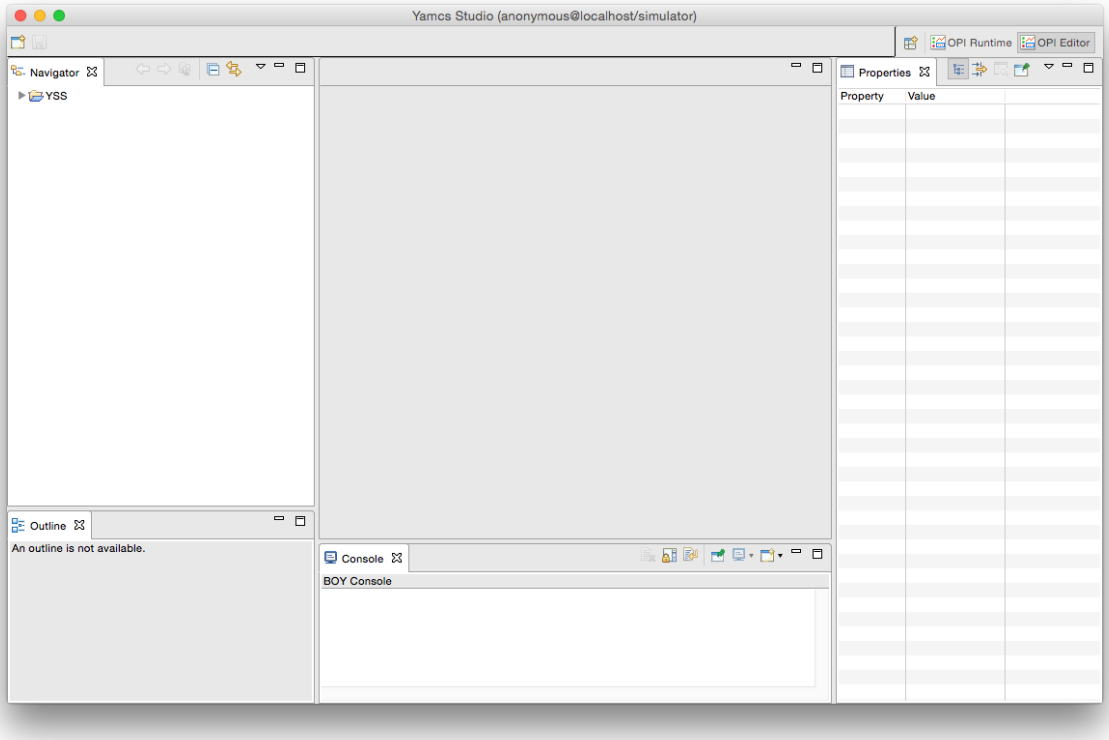
With Yamcs Studio, you are always working on one workspace at a time. Usually workspaces are fairly static, and you can often do with just one of them.



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

1.1.2 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* (page 9) and *Display Runner* (page 21). When Yamcs Studio is launched for the first time on a new workspace the user will be welcomed with the Display Builder window.

1.1.3 Example Project

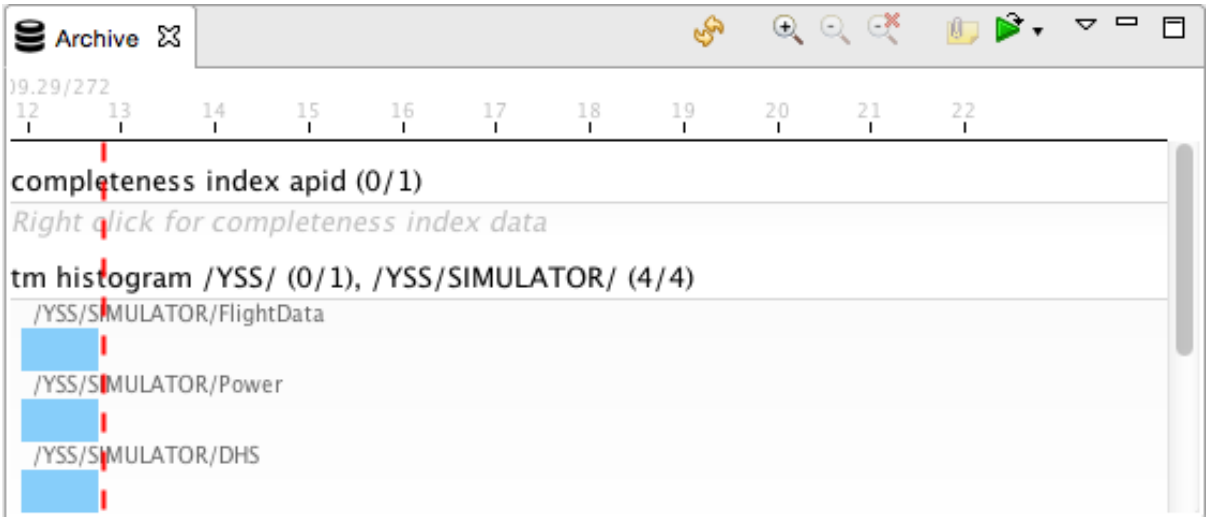
Yamcs Studio comes with a demo project that contains many examples on how to use any of the available widgets. To add this project to your workspace choose **File > New > Example...**, then **OPI Examples**.

1.2 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.

1.2.1 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.



Note: 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.

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

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

Note: 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.

1.2.2 Windows

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

- *Display Builder* (page 9)
- *Display Runner* (page 21)

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

In the top toolbar, you find a play button which opens 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.

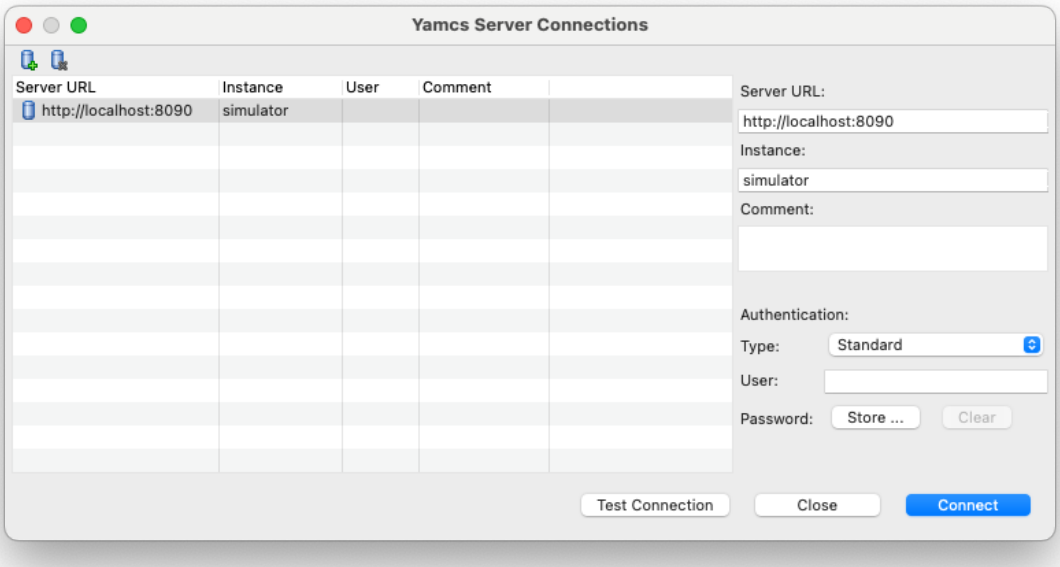
In the top toolbar of the Display Runner there is a pickaxe button which allows you to open or bring to the front the Display Builder window.

1.3 Connect to Yamcs

Yamcs Studio can connect as a client to 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 **Yamcs > Connect...**. This will open the Connections window where you can manage your connections.



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

The right panel contains editable details for the selected server connection:

Server URL (required)
: Specify the base URL for reaching Yamcs. For example `http://localhost:8090`

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.

Type (required)
: Specify the authentication mechanism. One of `Standard` or `Kerberos`.

  If your Yamcs installation does not require authentication `Standard` and do not enter a username.

Standard Authentication

This authentication type applies the standard authentication mechanism of Yamcs where the initial authentication is done by sending a username and password to Yamcs.

User / Password (optional)
: If your Yamcs instance is secured, fill in your user and password here.

  Optionally, the password can be stored to your OS keychain. If you don't do so, you will be prompted each time you attempt to connect to Yamcs.

  If you wish to remove a previously stored password, click **Clear**.

Kerberos Authentication

Using Kerberos, Yamcs Studio assumes the username of the OS user. There is no need for a password, because the protocol allows Yamcs to verify authentication against your Kerberos Distribution Center (KDC).

The use of this mechanism requires extra configuration of your Yamcs Server, to integrate it with your Kerberos realm.

Note: Connection preferences are stored at user level (not workspace), and will continue functioning whenever you upgrade your copy of Yamcs Studio.

The storage mechanism and location is different for each platform:

Linux
: XML file at `~/.java/.userPrefs/org/yamcs/studio/ui/connections/prefs.xml`

MacOS
: Property list file at `~/Library/Preferences/org.yamcs.studio.plist`

Windows
: In the Windows Registry: `HKEY_CURRENT_USER\Software\JavaSoft\Prefs\org\yamcs\studio\ui\connections`

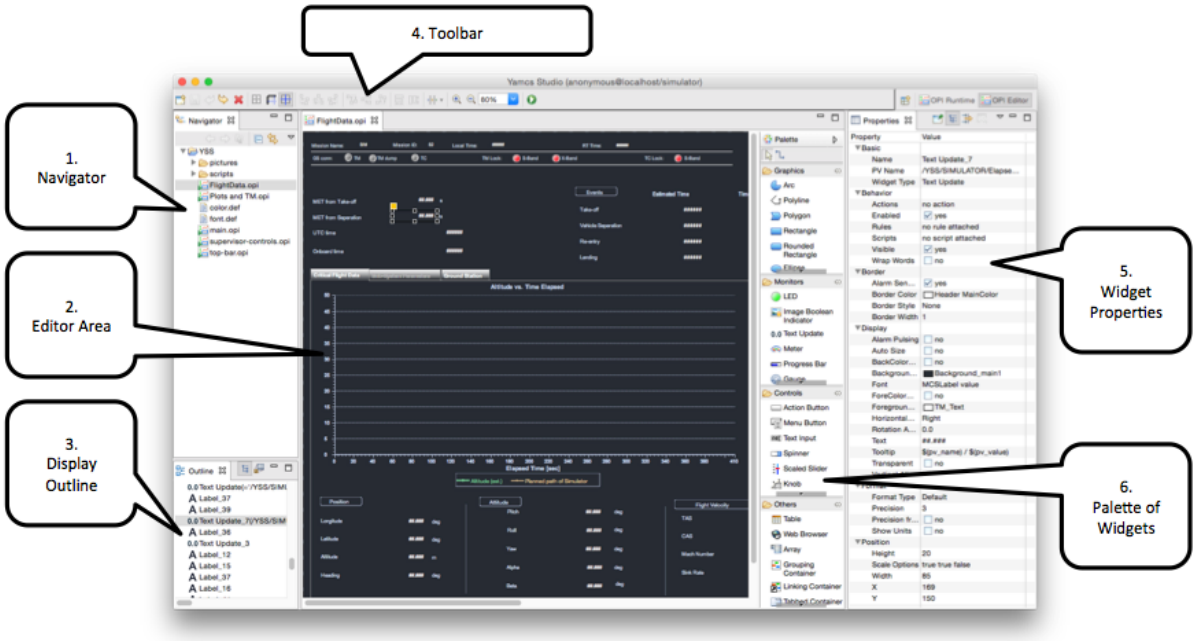
Any saved passwords are stored separately from the connections using Eclipse Secure Storage[1].

---

[1] https://help.eclipse.org/latest/index.jsp?topic=%2Forg.eclipse.platform.doc.user%2Freference%2Fref-securestorage-start.htm

2. Display Builder

The Display Builder window is used to create or edit displays.



1. **Explorer**

   The *Explorer* (page 10) contains all projects within the current workspace. In general a project is at the same level as a mission, but this is not strictly necessary. When we launch Yamcs Studio with a new workspace, it will always automatically create the YSS project. Once you have added your own project, you can remove YSS Landing and it will not be auto created anymore.

   A project contains Operator Displays (`*.opi`), images, custom scripts (`*.js`), etc. By right-clicking an OPI file, displays can be opened in two different modes.

   - In editing mode (default)
   - In runtime mode (via a new view)

2. **Editor Area**

   The *Editor Area* (page 12) contains tabs for every OPI that was opened for editing. This offers familiar editing controls. Widgets can be selected, grouped, dragged and deleted to your personal taste.

3. **Outline**

   The *Outline* (page 14) view presents a hierarchical breakdown of all the widgets within the currently active editor tab. It is useful for finding back widgets. Widgets that were named will be easily identifiable.

4. **Toolbar**

   The toolbar offers context-sensitive controls. This includes general *Save* functionality, as well as handy features like grid toggling or space distribution among different widgets.

5. **Properties**
   The *Properties* (page 15) view shows the properties of widgets (or of the display itself). Notable properties include the **PV Name** which allows you to connect a widget with a specific Yamcs parameter (with autocompletion support). Other properties allow the display author to greatly tweak default widget behaviour. And in cases where the properties are not sufficient, we can always escape to more customization options using rules and scripts (there are properties for adding these as well).

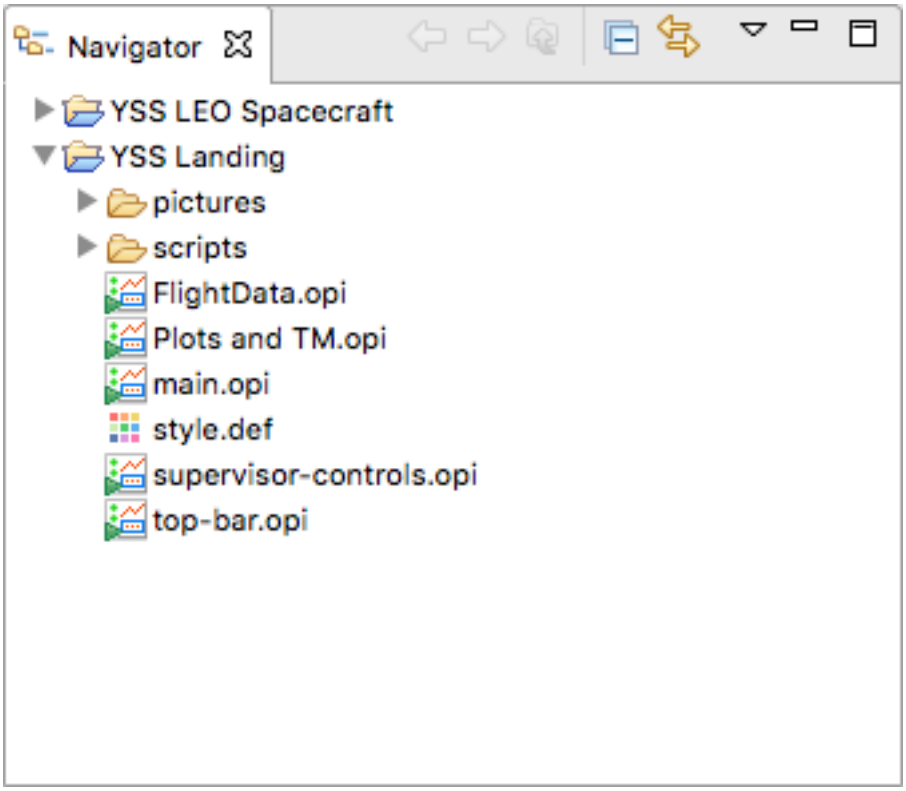
6. **Palette**
   The palette contains the widgets that are available in your copy of Yamcs Studio. Select a widget from the Palette, and then click somewhere in the editor area to place it down.

When you are done doing changes, make sure to save them (**File > Save All**). You can now test out your changes by clicking the launch button from the toolbar.

This will open the *Display Runner* (page 21) window. If you leave this window open, and you save more changes, do a right-click in your display tab and choose **Refresh OPI**. You will do this a lot as you go about editing displays. You can also refresh by hitting **F5**, but make sure that your display actually has focus (for example by clicking somewhere in the editor before hitting **F5**).

2.1 Explorer

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



2.1.1 Projects

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).

2.1.2 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.

Note: 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.

2.1.3 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.

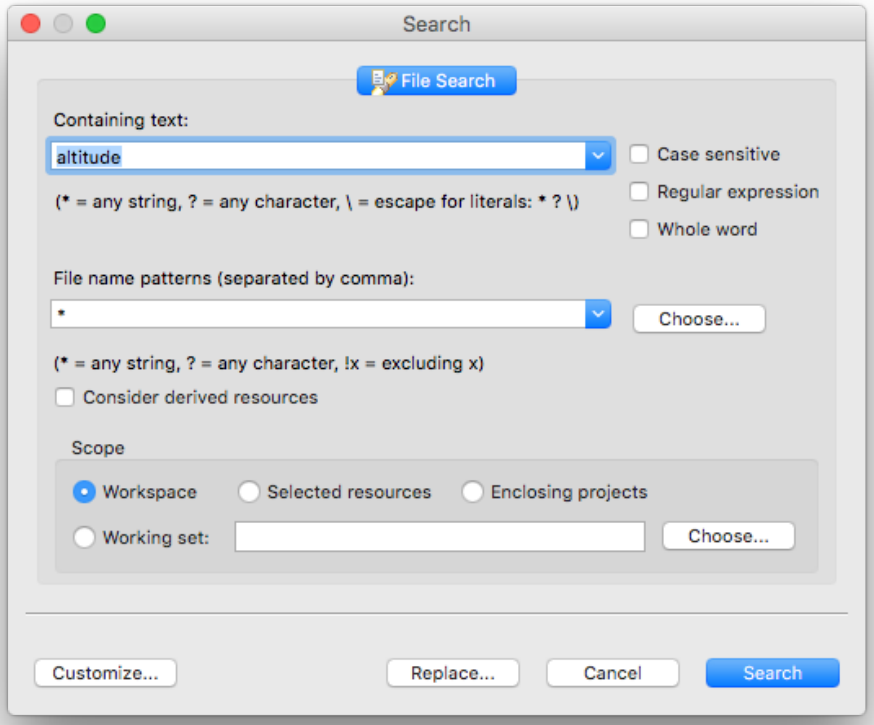
Open a file by double-clicking on it. If you open a file, yet Yamcs Studio does not have a specific handler for the type of file, it will open it with your system default program for that extension.

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`).

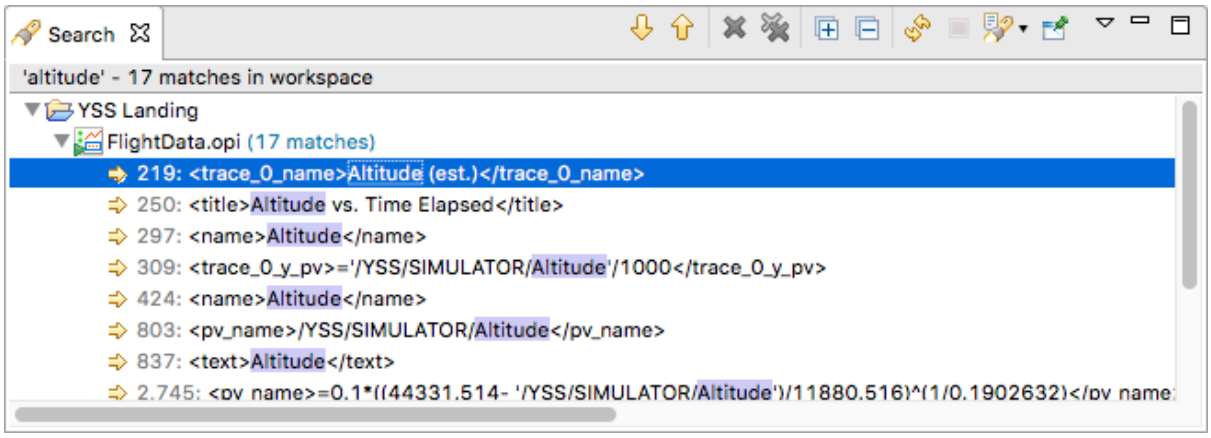
Use right-click **Open With** for more control over how the file is to be opened.

2.1.4 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.



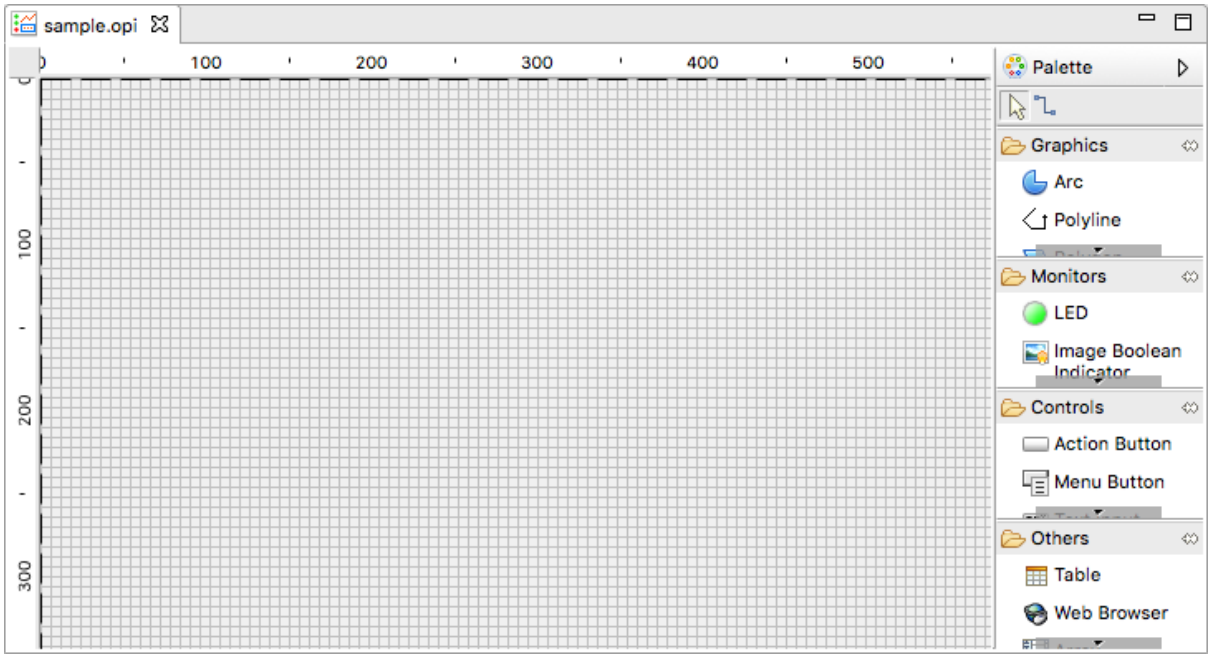
2.2 Editor Area

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.

2.2.1 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* (page 173), *Rules* (page 179) and *Scripts* (page 183).

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



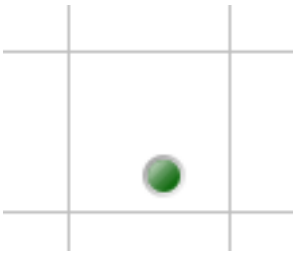
OPI files are created with some default properties, which includes a grid, and a size of 800x600. We will see in the section on *Properties* (page 15) that we can edit these properties.

2.2.2 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.



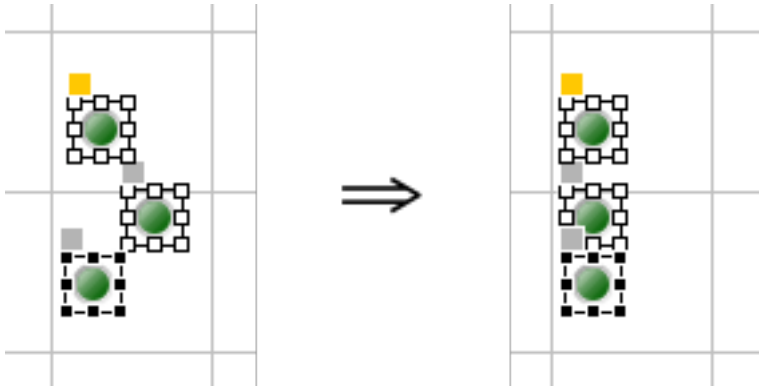
Once the widget has been placed, you can finetune its position and size using the *Properties* (page 15) view. Some operations are also readily available in the Editor Area itself using familiar controls. For example, to enlarge a widget, select it, then drag its handles around with the mouse.

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 (Cmd on Mac) while selecting the widgets one by one. Remove a widget from the selection in similar fashion.

2.2.3 Positioning Widgets

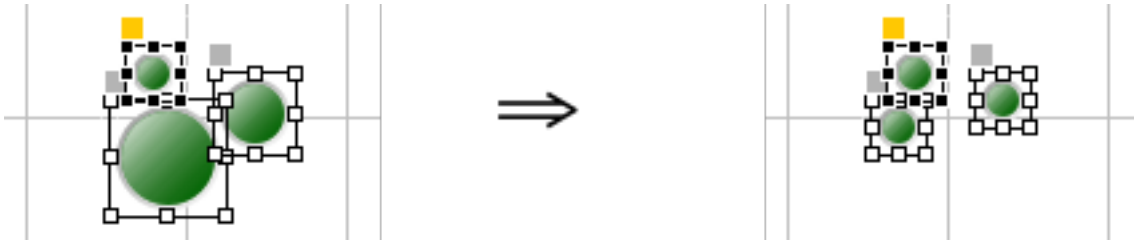
The toolbar of Yamcs Studio contains tools that help us align multiple selected widgets. For example, clicking **Align Left** repositions these three LEDs to the leftmost position.



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

2.2.4 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.



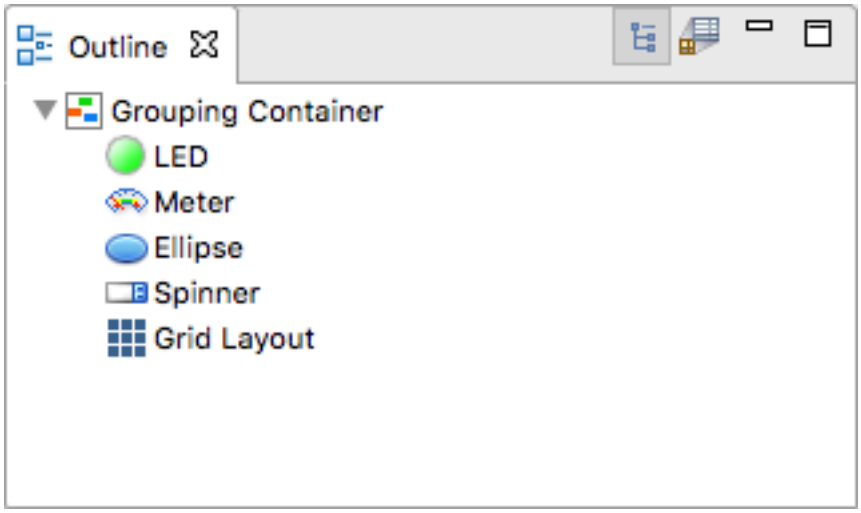
The size of the last selected widget is taken as the reference. This reference widget is highlighted with black instead of white anchor points.

Note: 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.

2.3 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.

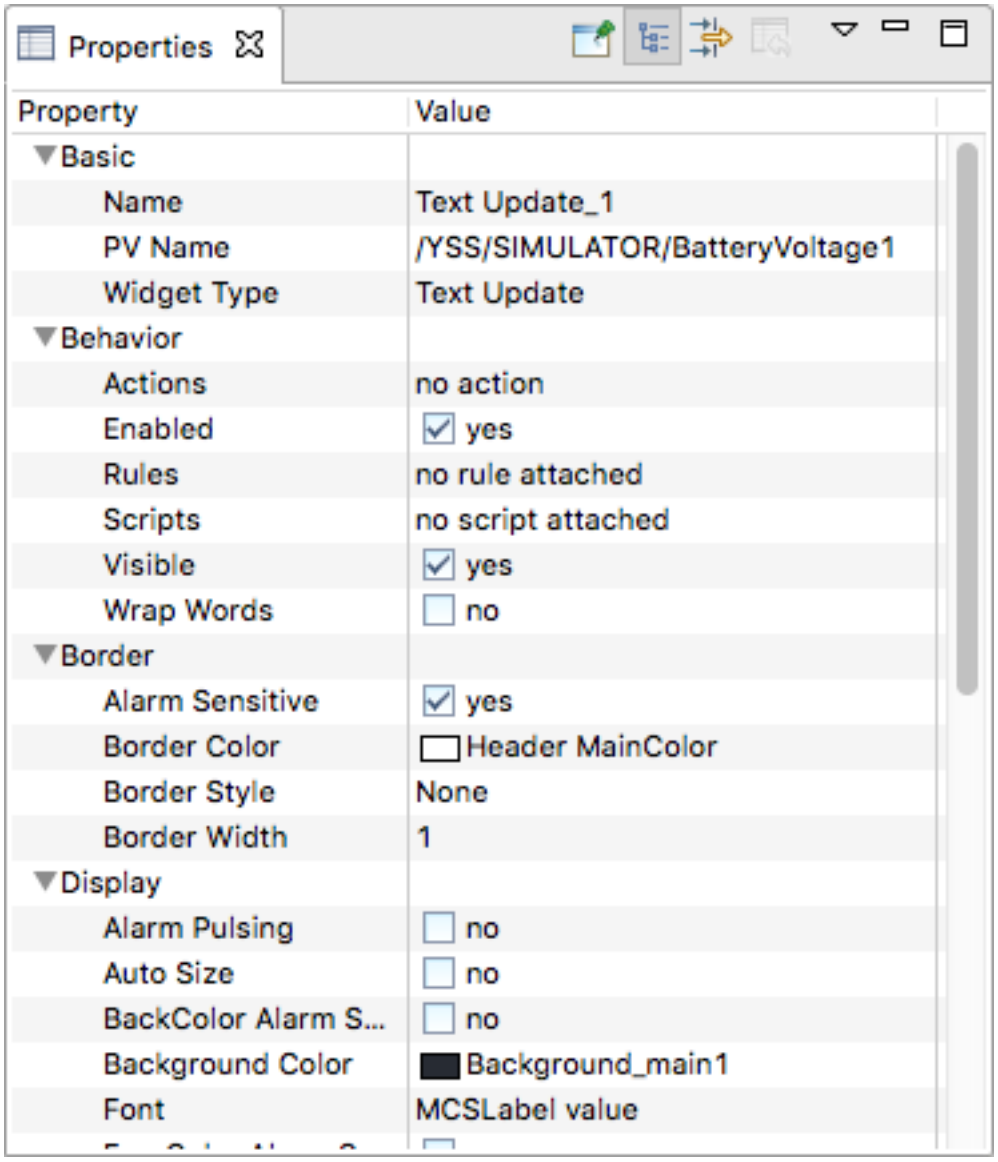
Widgets can be named in the *Properties* (page 15) view to make them stand out in the Outline.



2.4 Properties

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

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.



Note: 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.

2.4.1 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* (page 14) 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* (page 35) 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.

2.4.2 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.

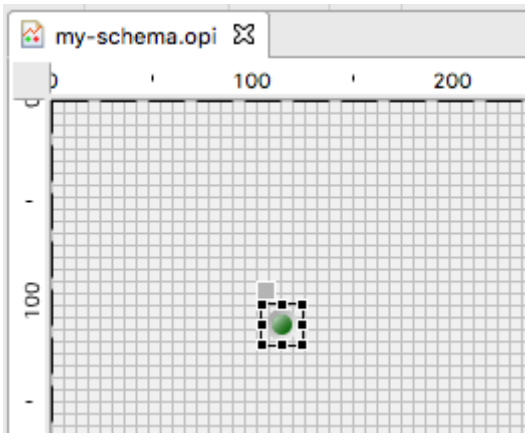
Auto Zoom to Fit All

Controls whether the display as a whole should be zoomed in at runtime such that it fits its available space.

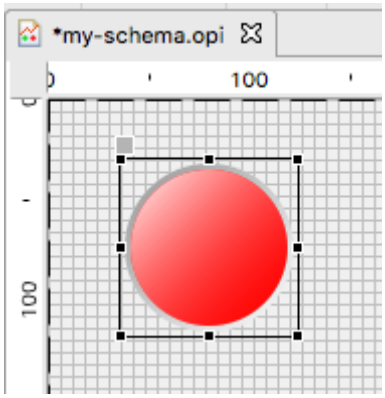
2.5 OPI Schema

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.

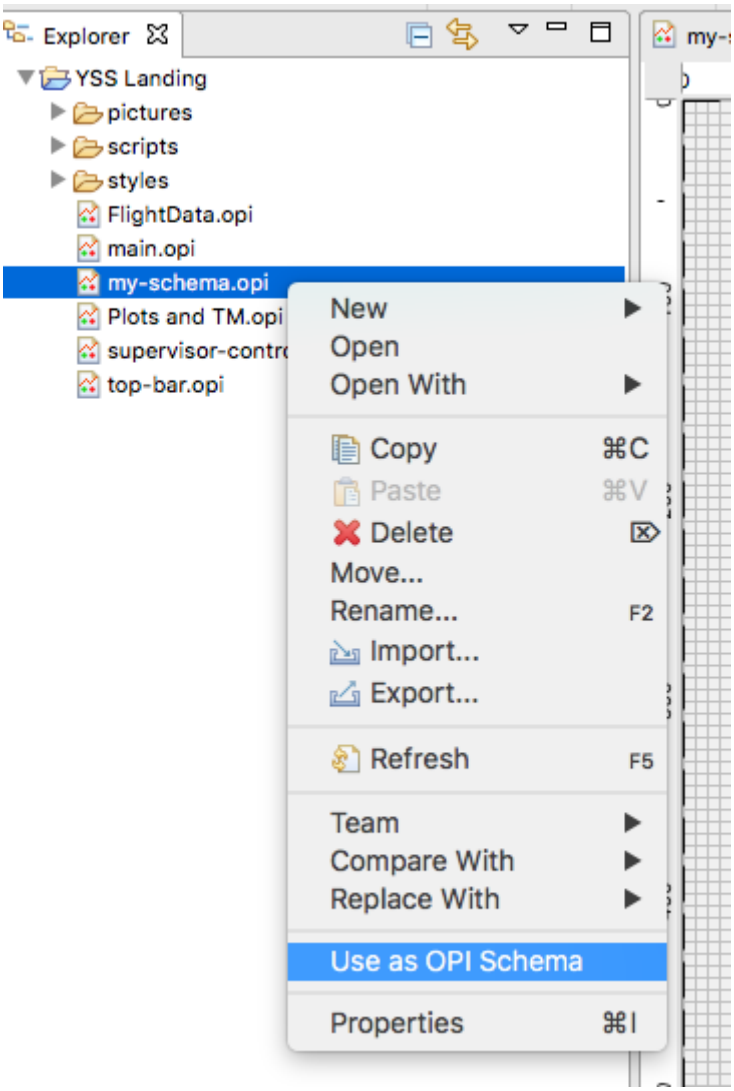
This is best illustrated with an example. Let's say we want to modify the defaults of the LED widget. Create a new OPI file and add a LED to the canvas (anywhere).



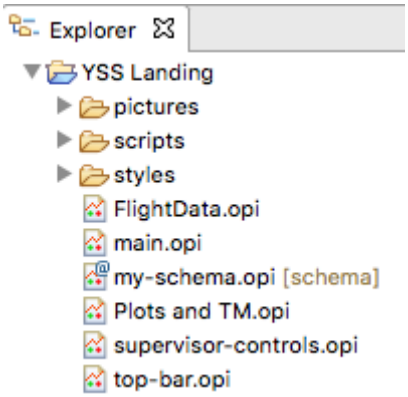
Make the LED bigger and change the colors.



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* (page 10) view:



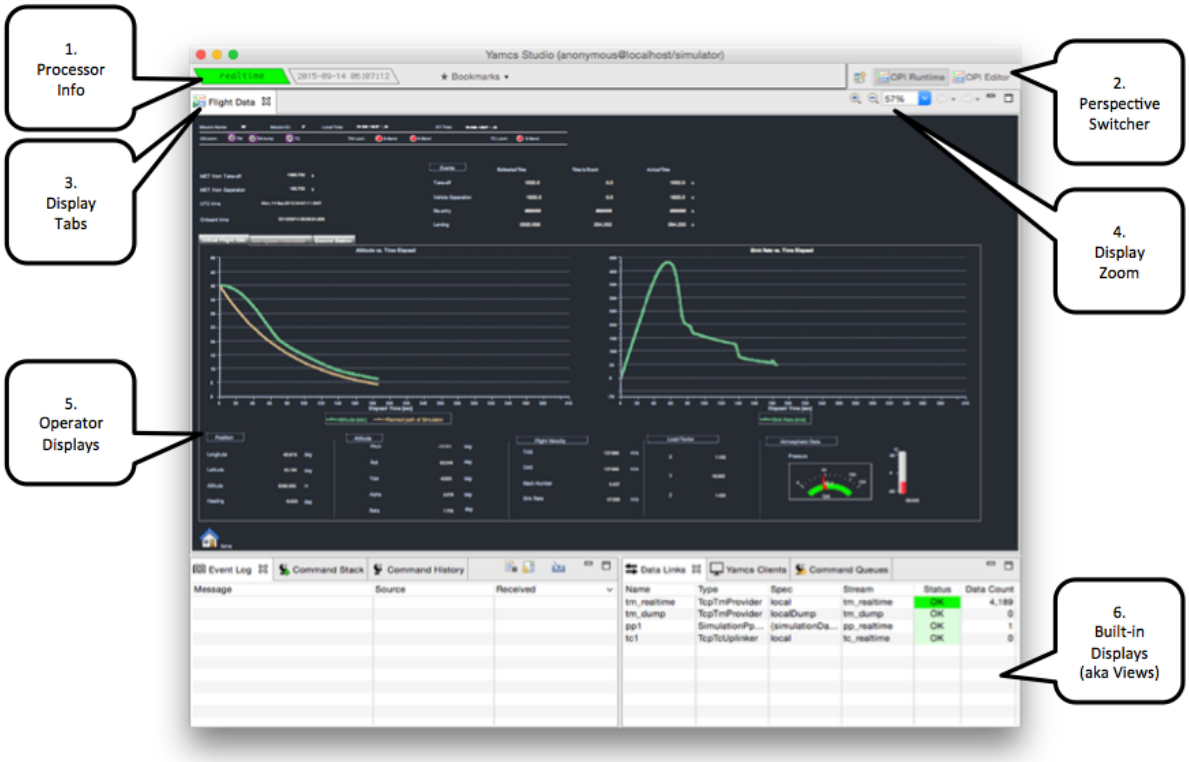
Whenever you add a new LED to any display, it will now by red and big by default. The same principle can be applied to any other properties of any of the available widgets.

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* (page 14) view.

Only one OPI Schema can be active at a time. To unset the active schema, Right-click it in the *Explorer* (page 10) view and untick **Use as OPI Schema**. This will return Display Builder to its default behaviour.

3. 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.

3. **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.

4. **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.

5. **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.
6. **Views**
   Yamcs Studio includes extra views for interacting with other Yamcs functionalities.

3.1 Archive View

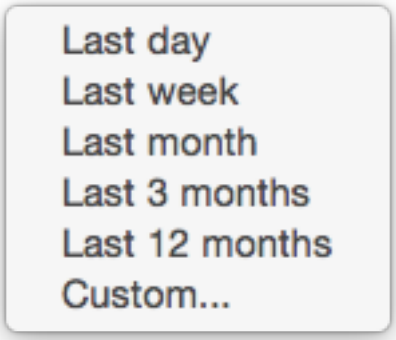
The Archive view represents a visual view of the data stored in the Yamcs archive. Through this view we can also initialize and control replays of archived telemetry.

3.1.1 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.

3.1.1.1 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.

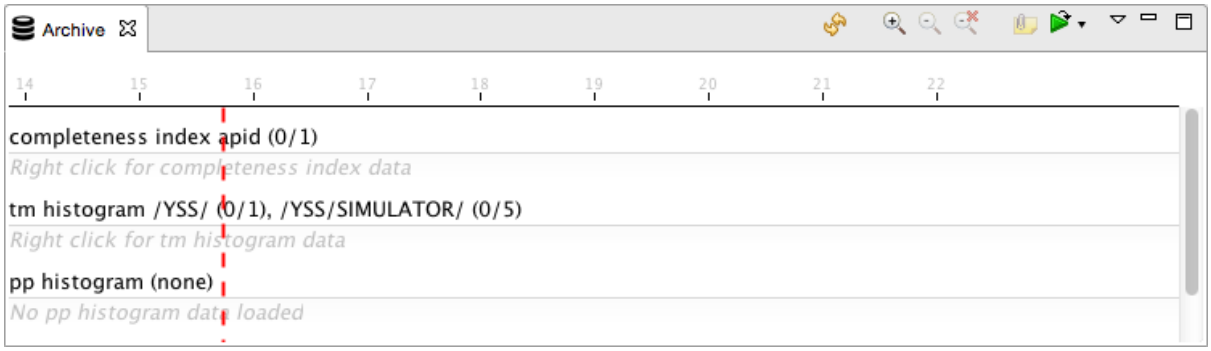
Your chosen data range is stored in your user preferences and will be restored the next time you open Yamcs Studio.

3.1.1.2 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:

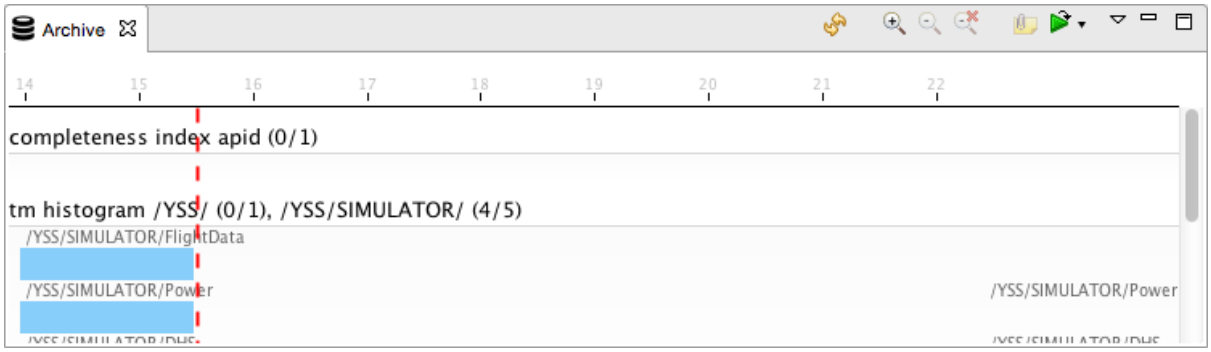
- Completeness Index APID

- 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.

Note: 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.

3.1.1.3 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.

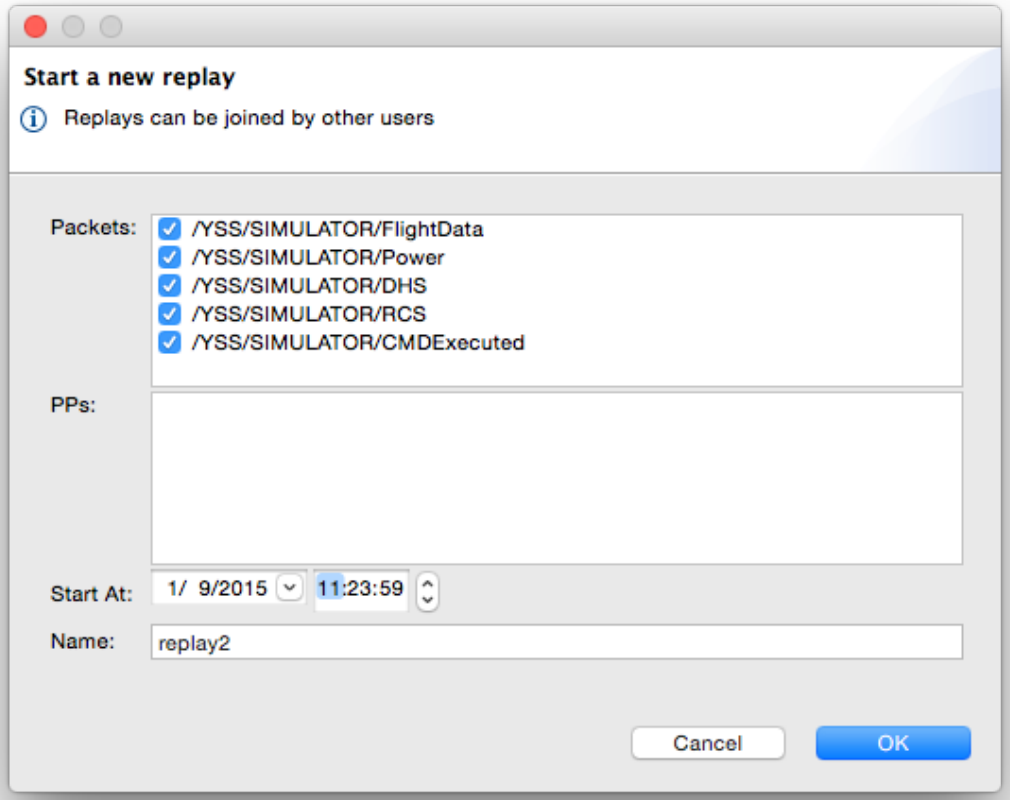
It is also possible to **Zoom In** or **Zoom Out**. If you are interested in a specific range, select a time range by clicking and dragging your mouse over the range before you zoom in using the zoom in button.

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**.

3.1.2 Replaying Data

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



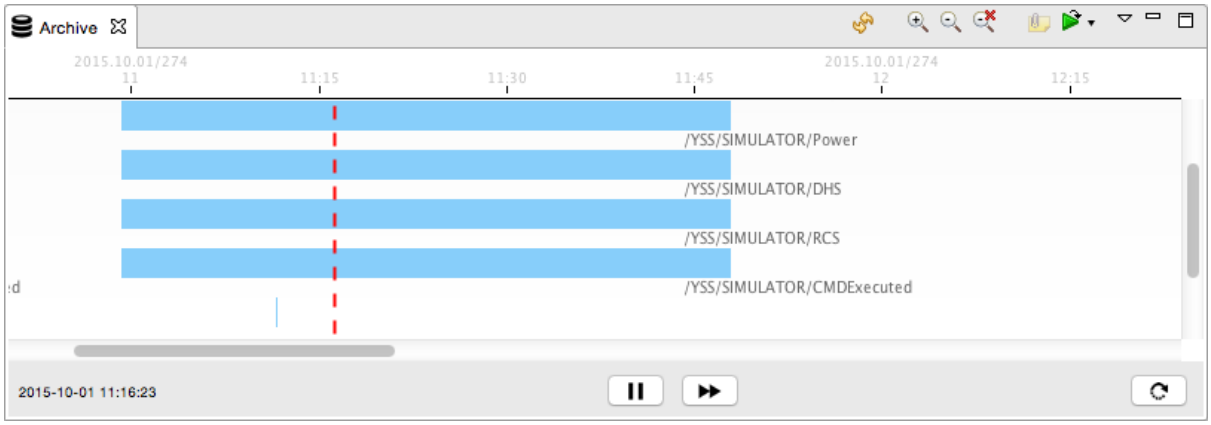
In the dialog box, confirm or filter the suggested selection of data. Currently only telemetry packets and processed parameters that were made visible in the Archive are part of the selectable data.

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

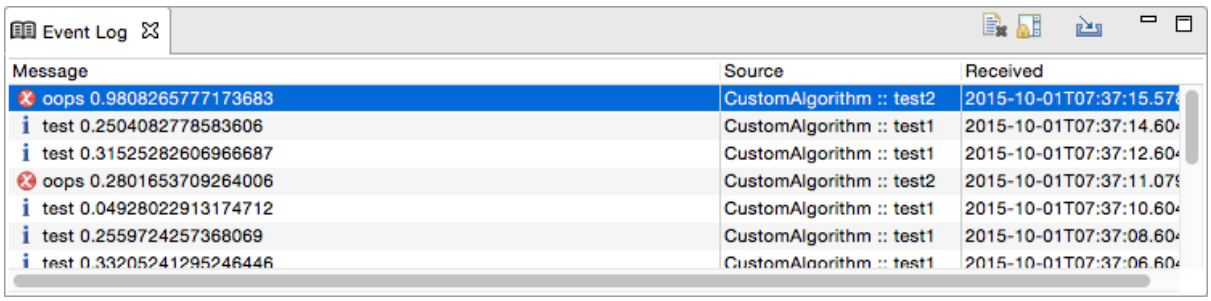
Speeding up will not cause any reset of your displays, as the same data is arriving, just faster.

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

3.2 Event Log View

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**.

3.3 Command Stack View

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.

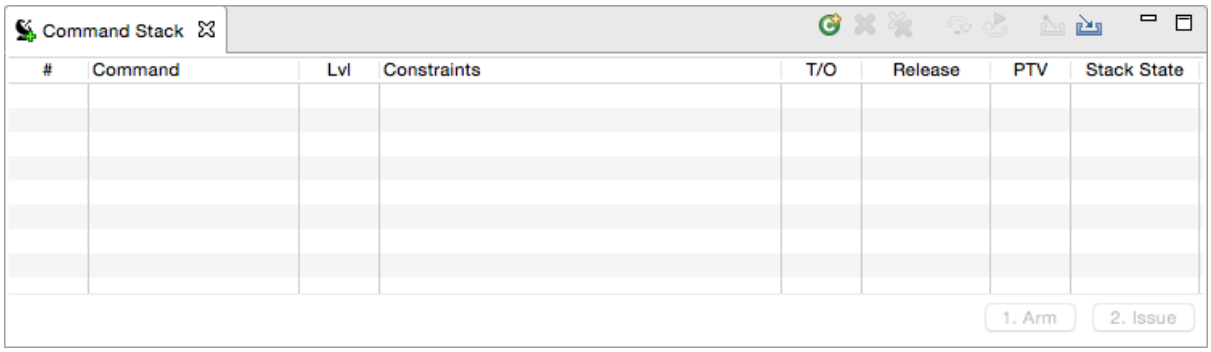
When you issue commands from Yamcs Studio, these are queued to Yamcs Server, which will perform any further steps.

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

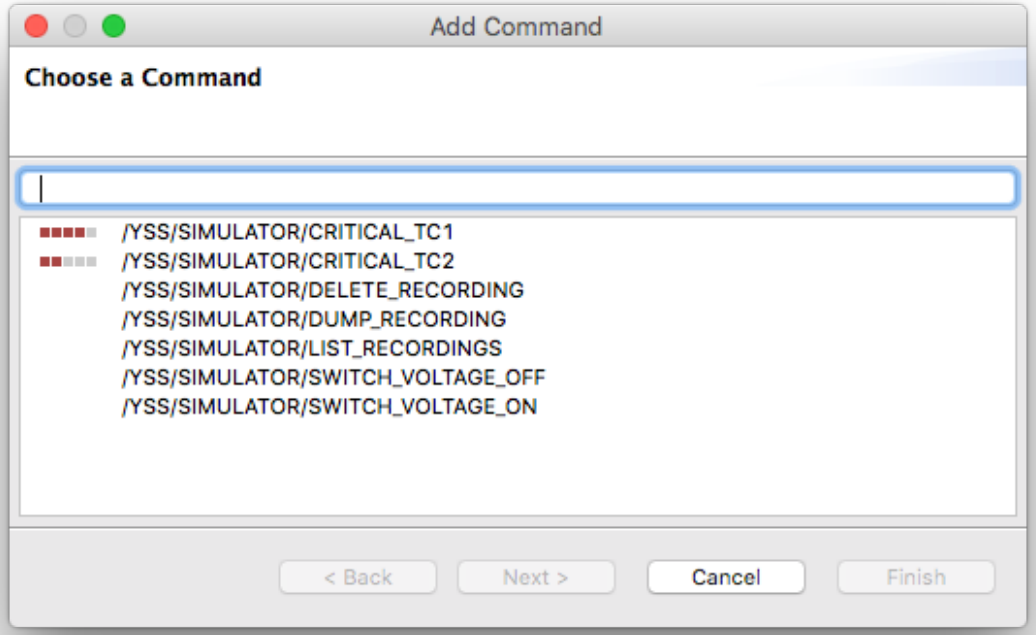
3.3.1 Preparing a Stack

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.

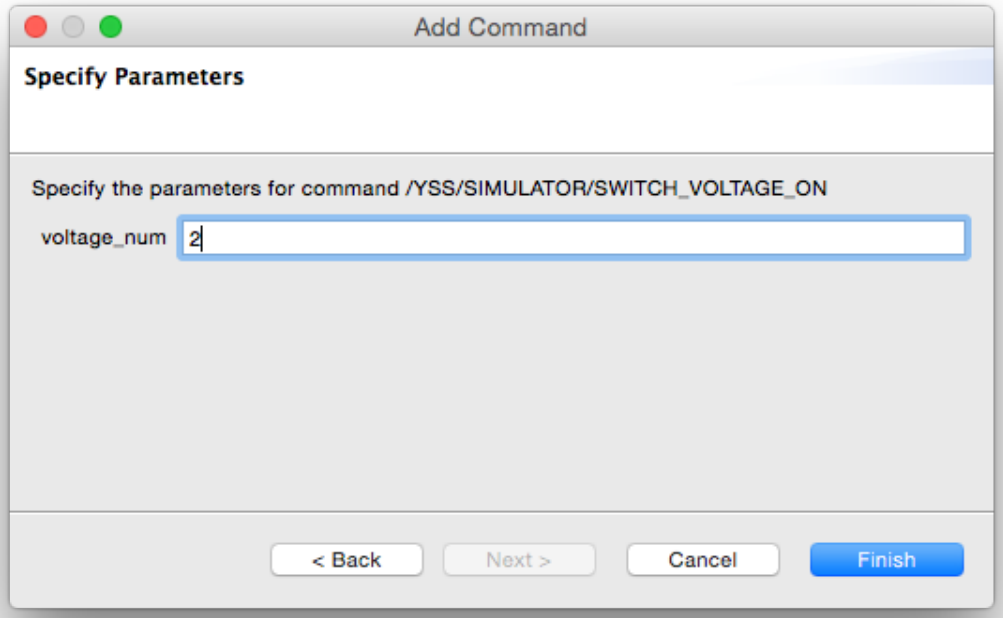
Commands can have varying levels of criticality (called *significance* in XTCE terminology). The icon in the leftmost column indicates the defined criticality for the command.

Icon	Criticality
	Watch
	Warning
	Distress
	Critical
	Severe

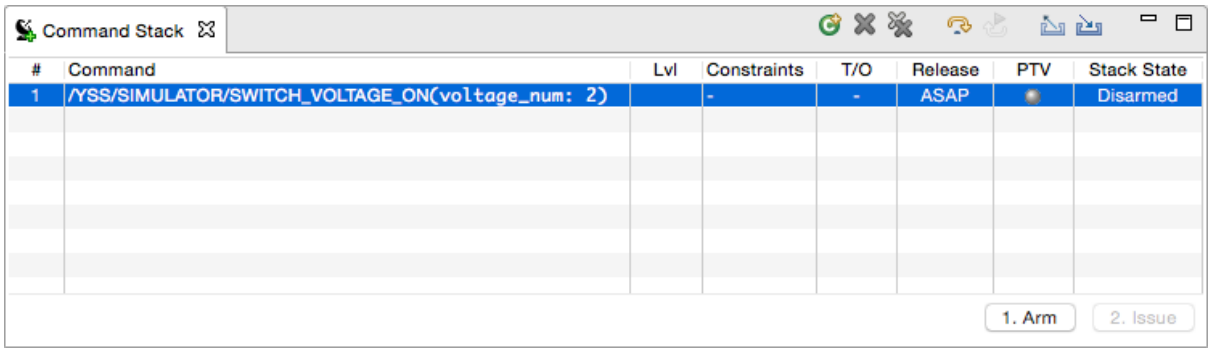
If an explanatory message regarding the criticality was defined in the mission database, this will show in the top title area of the dialog when the command is selected.

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.

Note: 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.

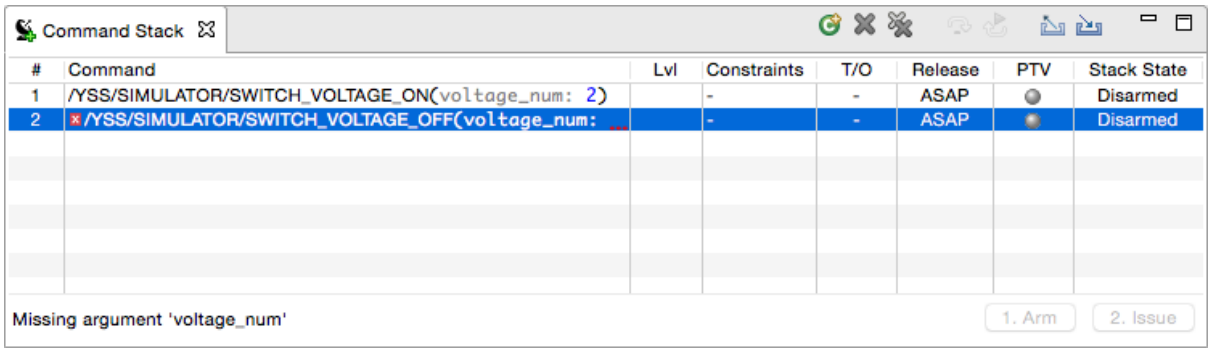


Click **Finish** to append your command to the end of your current stack.



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.



3.3.2 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.

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.

Note: *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.

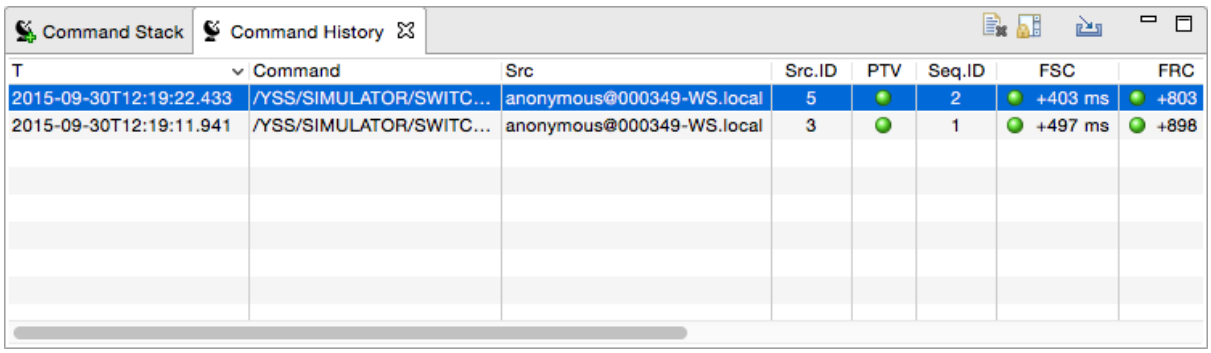
Skip the selected command by selecting **Skip**. Reset the execution state by selecting **Restart**. This will reset all Stack State columns to **Disarmed**.

3.3.3 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**.

3.4 Command History View

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

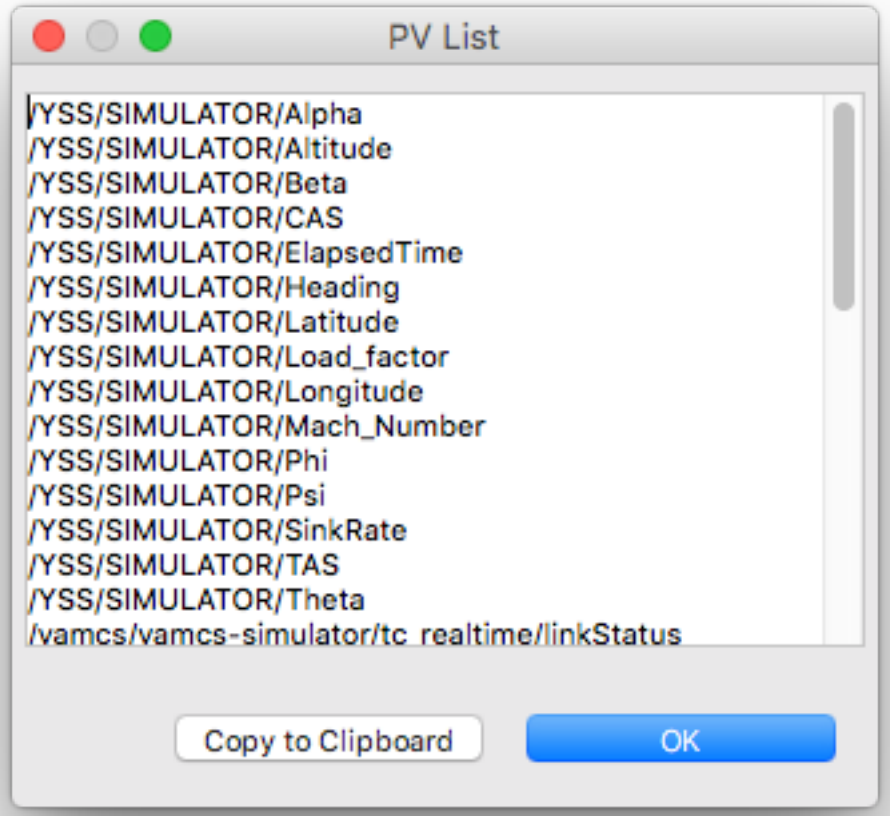
: Indicate acknowledgments of ground hops as the command is being dispatched. The exact number and name of the columns depends largely on how Yamcs is deployed at your site. Yamcs typically calculates the state of these bubbles based on incoming telemetry.

  The bubble becomes green or red depending on the verification result. The column value shows the time difference with the issuing time *T*.

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

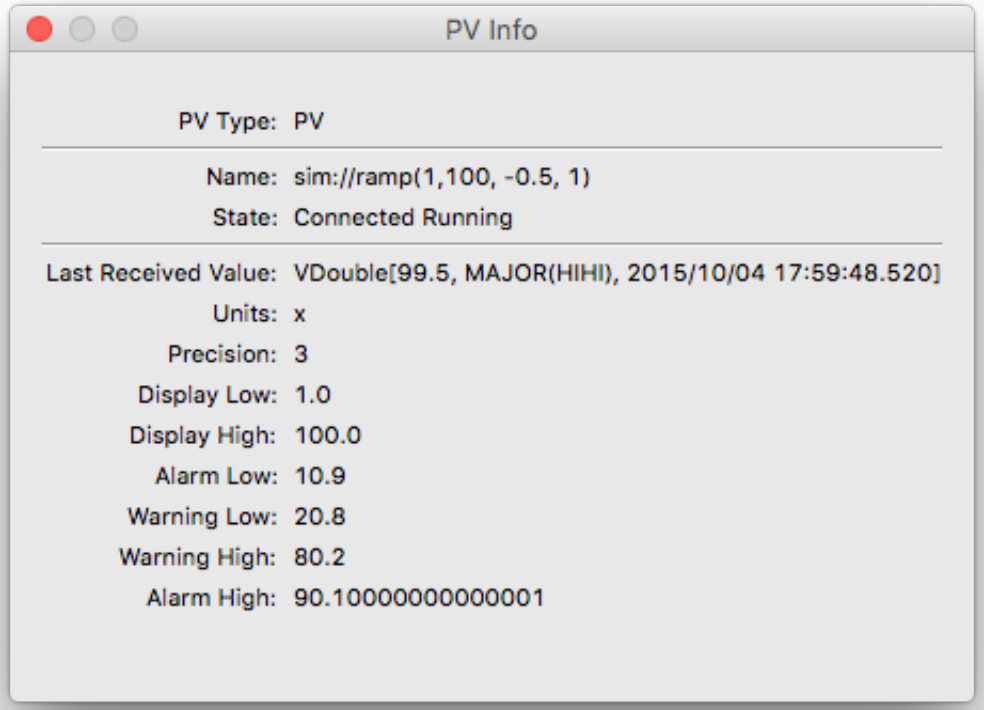
3.5 PV List

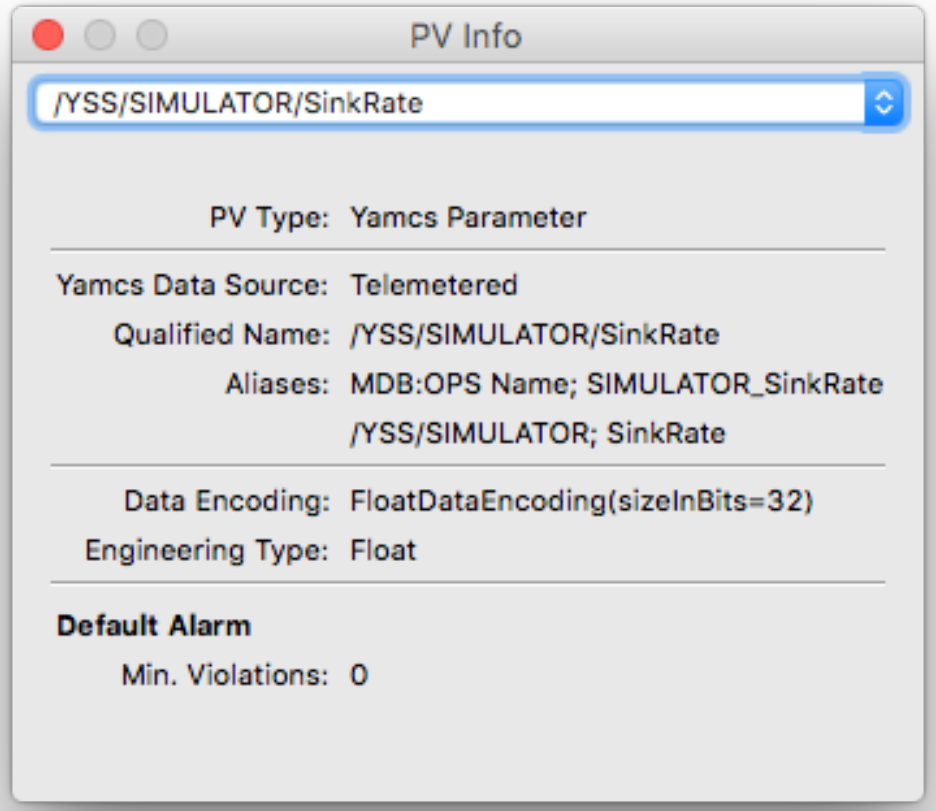
Right-click anywhere in a display, and choose **Dump PV List**, you will see a window listing the unique PVs that are defined inside any widget of that display. This can be useful for quick-fixing runtime issues.



3.6 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.

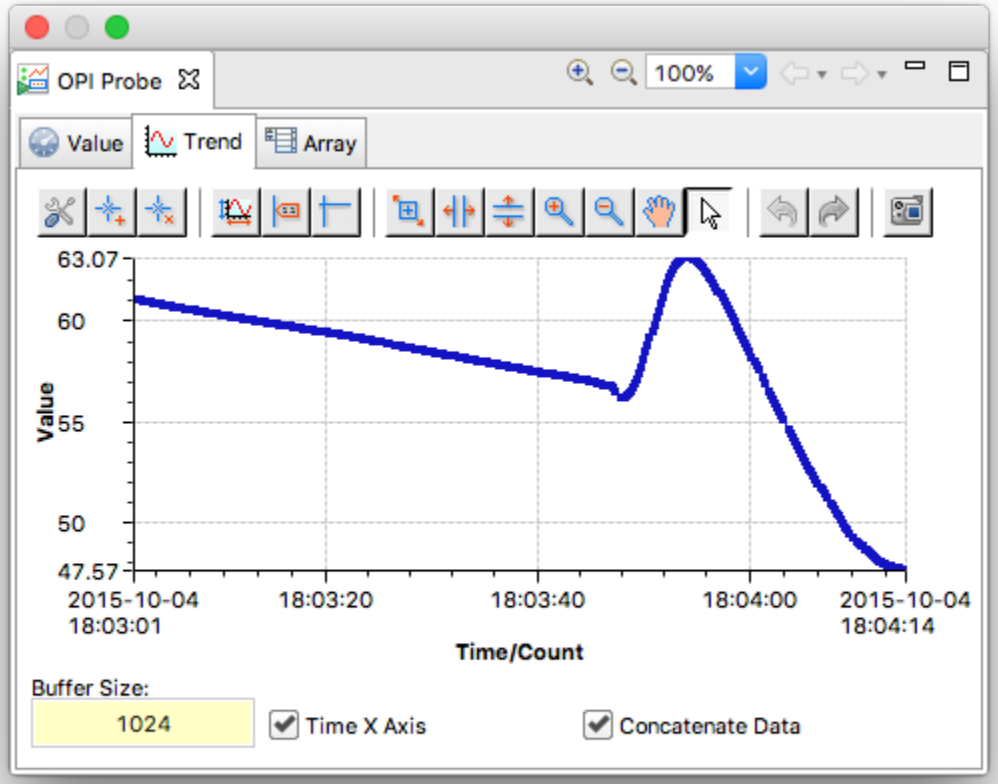

PV Info
PV Type: PV
Name: sim://ramp(1,100, -0.5, 1)
State: Connected Running
Last Received Value: VDouble[99.5, MAJOR(HIHI), 2015/10/04 17:59:48.520]
Units: x
Precision: 3
Display Low: 1.0
Display High: 100.0
Alarm Low: 10.9
Warning Low: 20.8
Warning High: 80.2
Alarm High: 90.10000000000001



3.7 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.
- In the tab **Trend**, a graphical evolution of this PVs value.


OPI Probe
100%
Value
Trend
Array
63.07
60
55
50
47.57
Value
2015-10-04
18:03:01
18:03:20
18:03:40
18:04:00
2015-10-04
18:04:14
Time/Count
Buffer Size:
1024
Time X Axis
Concatenate Data

4. 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*.

4.1 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, or indeed from one display to another. They are very useful when scripting advanced displays.

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 example:

- `loc://foo(1)`
- `loc://bar("abc")`
- `loc://baz(1, 2, 3, 4, 5)`

A local PV can be configured to enforce a specific type by specifying one of `VDouble`, `VString`, `VDoubleArray`, `VStringArray`, `VTable` or `VEnum` in angle brackets. For example, a control widget with `PV Name` set to `loc://foo<VString>` will emit string values, even if the input is numeric.

To combine a type and an initializer use this syntax:

- `loc://foo<VString>("200")`
- `loc://foo<VDouble>(200)`

4.2 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)

4.3 Parameters

Parameter PVs represent a value that is provided by a connected Yamcs server, usually from packet telemetry.

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

Example PV Names:

- `/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`.

LOLO/LO/HI/HIHI Markers

Several widgets support the display of low/high warning and alarm markers. If those widgets have the property **Limits from PV** enabled, the marker positions are mapped from Yamcs severity levels:

- WATCH, WARNING, DISTRESS → LO/HI
- CRITICAL, SEVERE → LOLO/HIHI

Minimum/Maximum

Several widgets display the current PV value on a scale. If those widgets have the property **Limits from PV** enabled:

- The lower bound of the scale matches the most-severe lower bound of the parameter. If Yamcs does not specify any lower bound, the widget reverts to the configured **Minimum** property.
- The upper bound of the scale matches the most-severe upper bound of the parameter. If Yamcs does not specify any upper bound, the widget reverts to the configured **Maximum** property.

Units

Some widgets have the property **Units from PV**. If enabled, Yamcs Studio will compose a unit string based on information provided by Yamcs.

4.4 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`

4.5 State PVs

The namespace `state://` contains a set of PVs that describe state specific to the local Yamcs Studio instantiation.

- `state://yamcs.host`

  Hostname used for connecting to Yamcs

- `state://yamcs.instance`

  Connected Yamcs instance
- `state://yamcs.processor`

  Connected Yamcs processor
- `state://yamcs.serverId`

  Server ID of the connected Yamcs Server
- `state://yamcs.username`

  Username at the connected Yamcs Server
- `state://yamcs.version`

  Version number of the connected Yamcs Server

4.6 System PVs

The namespace `sys://` contains a set of PVs that describe system properties of the local Yamcs Studio instantiation.

- `sys://time`

  Local time
- `sys://user`

  Local OS user (that is: *not* the Yamcs user)
- `sys://host_name`

  Host name of the system that runs Yamcs Studio
- `sys://qualified_host_name`

  Full host name of the system that runs Yamcs Studio
- `sys://system.[PROP]`

  Provides access to system properties available to Yamcs Studio.

5. Widgets

A display is a container for ordered widgets, each occupying an area specified by x, y, width and height coordinates.

The widgets in Yamcs Studio are listed below. Note that a *Display* (page 62) is itself also a kind of widget.

Graphics

Arc (page 42)	*Rectangle* (page 117)	*Label* (page 92)
Polyline (page 108)	*Rounded Rectangle* (page 121)	*Image* (page 74)
Polygon (page 105)	*Ellipse* (page 64)	

Monitors

LED (page 94)	*Progress Bar* (page 111)	*XY Graph* (page 162)
Image Boolean Indicator (page 79)	*Gauge* (page 68)	*Intensity Graph* (page 82)
Text Update (page 153)	*Thermometer* (page 156)	*Byte Monitor* (page 54)
Meter (page 101)	*Tank* (page 144)	

Controls

Action Button (page 40)	*Knob* (page 88)	*Image Boolean Button* (page 76)
Menu Button (page 99)	*Scrollbar* (page 131)	*Check Box* (page 56)
Text Input (page 148)	*Thumb Wheel* (page 160)	*Radio Box* (page 115)
Spinner (page 134)	*Boolean Switch* (page 51)	*Choice Button* (page 59)
Scaled Slider (page 127)	*Boolean Button* (page 48)	*Combo* (page 61)

Others

Table (page 139)	*Grouping Container* (page 72)	*Sash Container* (page 125)
Web Browser (page 169)	*Linking Container* (page 97)	*Grid Layout* (page 71)
Array (page 45)	*Tabbed Container* (page 137)	

5.1 Action Button

Control widget for executing an action when clicked.

For example, a button can be used to open another display.

Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (`pv_name`)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Click Action Index (`push_action_index`)
Index in the list of associated actions of the action that must be executed when the control is clicked.

This property is only visible when **Toggle Button** is disabled.

Push Action Index (`push_action_index`)
Index in the list of associated actions of the action that must be executed when the control is pushed.

This property is only visible when **Toggle Button** is enabled.

Release Action Index (`release_action_index`)
Index in the list of associated actions of the action that must be executed when the control is released.

This property is only visible when **Toggle Button** is enabled.

Enabled (`enabled`)
Unset to make this control widget unusable.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Toggle Button (`toggle_button`)
If yes, this button stays engaged after clicking it. You need to click it a second time to untoggle it.

If no, this button stays engaged only for as long as the mouse stays pressed.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the label.

Icon File (`image`)
An image file to be shown next to the button text.

Text (`text`)
The displayed text.

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

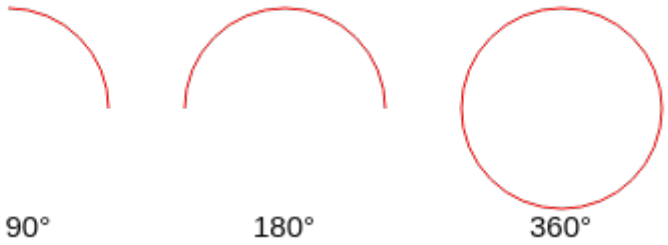
Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

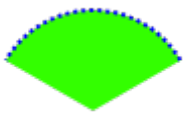
Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

5.2 Arc

Widget that draws an arc shape.

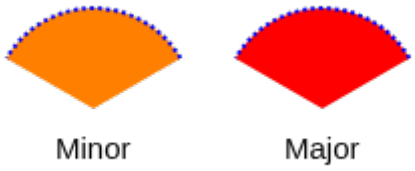


Example of shape fill:



Property	Value
Background Color	rgb(51, 255, 0)
Fill	yes
Foreground Color	rgb(0, 0, 255)
Line Style	Dot
Line Width	2
Start Angle	30
Total Angle	120

The shape background and foreground colors can be made alarm-aware by attaching a PV. Note that the PV value is otherwise ignored.



Property	Value
BackColor Alarm Sensitive	yes

Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (`pv_name`)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

Alpha (`alpha`)
Opacity level of the shape. Should be a value in the range 0 to 255.

Anti Alias (`anti_alias`)
Whether anti-aliasing is enabled for drawing the shape.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the wedge when **Fill** is enabled.

Fill (`fill`)
Whether the shape should be filled.

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the arc stroke.

Line Style (`line_style`)
The type of stroke.

Code	Value	Description
0	Solid	Uninterrupted
1	Dash	Applies the pattern: 6px solid, 2px gap
2	Dot	Applies the pattern: 2px solid, 2px gap
3	DashDot	Applies the pattern: 6px solid, 2px gap, 2px solid, 2px gap
4	DashDotDot	Applies the pattern: 6px solid, 2px gap, 2px solid, 2px gap, 2px solid, 2px gap

Line Width (`line_width`)
Thickness of the shape stroke

Start Angle (`start_angle`)
Start angle in degrees. The coordinate system is anticlockwise, with 0 degrees representing east.

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Total Angle (`total_angle`)
Total angle covered by the arc. The coordinate system is anticlockwise, with 0 degrees representing east.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

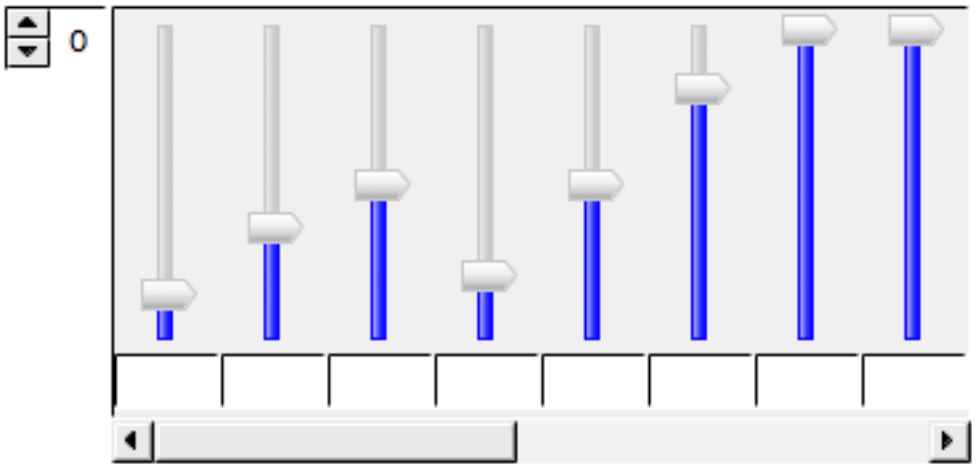
Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

5.3 Array

Widget for reading or writing an array of other widgets of the same type.



When dropping a new widget onto an array widget, that widget will be duplicated by the length of that array. Each singular contained widget is matched to one element of the array.

While editing in Yamcs Studio, changes to any widget element are applied to all other widgets within that array.

If the array widget is backed by an array PV, the **Array Length** as well as **Data Type** are automatically determined from the PV.

An array widget can support large array sizes by showing a scrollbar, and a spinner that allows to jump to a specific array index.

Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (`pv_name`)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Array Length (`array_length`)
Number of array elements.

This property is not available when the array is backed by a PV.

Data Type (`data_type`)
Type of the array. This is the type of the value returned by `widget.getValue()` inside a script.

This property is not available when the array is backed by a PV.

Code	Value
0	double[]
1	String[]
2	int[]
3	byte[]
4	long[]
5	short[]
6	float[]
7	Object[]

Enabled (`enabled`)
Unset to make contained control widgets unusable.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the label.

Horizontal (`horizontal`)
If yes, the array elements are arranged in horizontal direction. Otherwise vertical.

Show Scrollbar (`show_scrollbar`)
Make a scrollbar visible.

Show Spinner (`show_spinner`)
Make a spinner widget for jumping to a specific index within the array.

Spinner Width (`spinner_width`)
The width in pixels of the spinner widget.

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

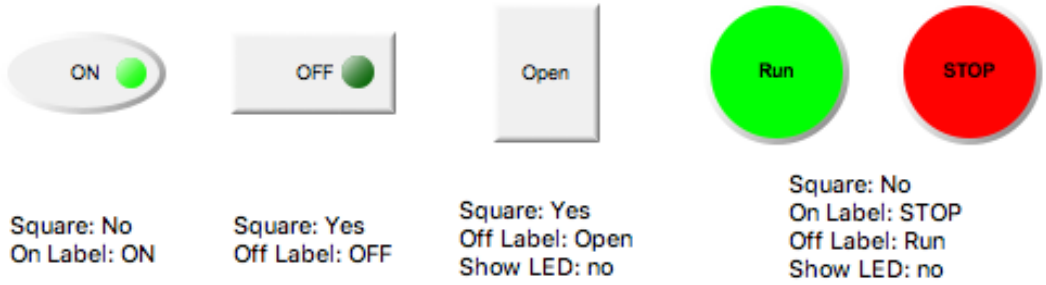
Additional API

Array widgets expose the following additional *Widget* (page 196) API for use in scripting:

getIndex(child)
Get the index of a child widget. If the given widget is not a child this method returns `-1`.

5.4 Boolean Button

Widget for writing 0 or 1 to the attached PV. This widget can also be used to flip a single bit of the attached PV.



Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (`pv_name`)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Bit (`bit`)
Matches the widget's boolean value to a specific bit of the attached PV's value.

If `-1`, any non-zero value is considered true, whereas a zero value is considered false.

This widget writes the numeric values 0 (unselected) or 1 (selected) to the PV.

Confirm Message (`confirm_message`)
The message to be displayed when **Show Confirm Dialog** is set.

Data Type (`data_type`)
Control how the widget boolean value is established.

Code	Value	Description
0	Bit	The widget boolean value matches a specific bit (indicated by the **Bit** property), or the entire value in case the **Bit** property is set to `-1`
1	Enum	The widget boolean value follows the comparison of its value with specific enumeration states (indicated with the **Off State** and **On State** property

Enabled (`enabled`)
Unset to make this control widget unusable.

Off State (`off_state`)
If **Data Type** is set to `Enum`, this indicates the state that matches boolean false.

On State (`on_state`)
If **Data Type** is set to `Enum`, this indicates the state that matches boolean true.

Password (`password`)
If set, this contains the password that needs to be provided by the user when **Show Confirm Dialog** is set, and the user is attempting to change the widget state.

Note: OPIs are rendered on the client. The use of this property provides only a false sense of security.

Push Action Index (`push_action_index`)
Index in the list of associated actions of the action that must be executed when the control is pushed.

Release Action Index (`release_action_index`)
Index in the list of associated actions of the action that must be executed when the control is released.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Show Confirm Dialog (`show_confirm_dialog`)
Show a confirm dialog when an attempt is made to change the widget state.

If the **Password** property is set, the user will be asked to enter that password correctly.

Toggle Button (`toggle_button`)
If yes, this button stays engaged after clicking it. You need to click it a second time to untoggle it.

If no, this button stays engaged only for as long as the mouse stays pressed.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

3D Effect (`effect_3d`)
Whether the rendering includes gradient and shadow effects.

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the label.

Labels from PV (`labels_from_pv`)
If the PV is an enumerated PV, retrieve the labels for the on/off states from the associated PV.

Off Color (`off_color`)
Color of the LED when it is off.

Off Label (`off_label`)
The label text when this widget's boolean state is false.

On Color (`on_color`)
Color of the LED when it is on.

On Label (`on_label`)
The label text when this widget's boolean state is true.

Show Boolean Label (`show_boolean_label`)
Whether the label is visible (controlled with properties **Off Label** and **On Label**).

Show LED (`show_led`)
Whether the LED is visible.

Square Button (`square_button`)
Make the button rectangular.

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

5.5 Boolean Switch

Widget for writing 0 or 1 to the attached PV. This widget can also be used to flip a single bit of the attached PV.

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

Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (`pv_name`)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Bit (`bit`)
Matches the widget's boolean value to a specific bit of the attached PV's value.

If `-1`, any non-zero value is considered true, whereas a zero value is considered false.

This widget writes the numeric values 0 (unselected) or 1 (selected) to the PV.

Confirm Message (`confirm_message`)
The message to be displayed when **Show Confirm Dialog** is set.

Data Type (`data_type`)
Control how the widget boolean value is established.

Code	Value	Description
0	Bit	The widget boolean value matches a specific bit (indicated by the **Bit** property), or the entire value in case the **Bit** property is set to `-1`
1	Enum	The widget boolean value follows the comparison of its value with specific enumeration states (indicated with the **Off State** and **On State** property

Enabled (`enabled`)
Unset to make this control widget unusable.

Off State (`off_state`)
If **Data Type** is set to `Enum`, this indicates the state that matches boolean false.

On State (`on_state`)
If **Data Type** is set to `Enum`, this indicates the state that matches boolean true.

Password (`password`)
If set, this contains the password that needs to be provided by the user when **Show Confirm Dialog** is set, and the user is attempting to change the widget state.

Note: OPIs are rendered on the client. The use of this property provides only a false sense of security.

Push Action Index (`push_action_index`)
Index in the list of associated actions of the action that must be executed when the control is pushed.

Release Action Index (`release_action_index`)
Index in the list of associated actions of the action that must be executed when the control is released.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Show Confirm Dialog (`show_confirm_dialog`)
Show a confirm dialog when an attempt is made to change the widget state.

If the **Password** property is set, the user will be asked to enter that password correctly.

Toggle Button (`toggle_button`)
If yes, this button stays engaged after clicking it. You need to click it a second time to untoggle it.

If no, this button stays engaged only for as long as the mouse stays pressed.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

3D Effect (`effect_3d`)
Whether the rendering includes gradient and shadow effects.

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Boolean Label Position (`boolean_label_position`)
Where to position the label within the widget area (top, bottom, left, right, top-left, top-right, bottom-left, bottom-right).

Code	Value
0	Default
1	Top
2	Left
3	Center
4	Right
5	Bottom
6	Top Left
7	Top Right
8	Bottom Left
9	Bottom Right

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the label.

Off Color (`off_color`)
Color of the switch shaft when it is off.

Off Label (`off_label`)
The label text when this widget's boolean state is false.

On Color (`on_color`)
Color of the switch shaft when it is on.

On Label (`on_label`)
The label text when this widget's boolean state is true.

Show Boolean Label (`show_boolean_label`)
Whether the label is visible (controlled with properties **Off Label** and **On Label**).

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

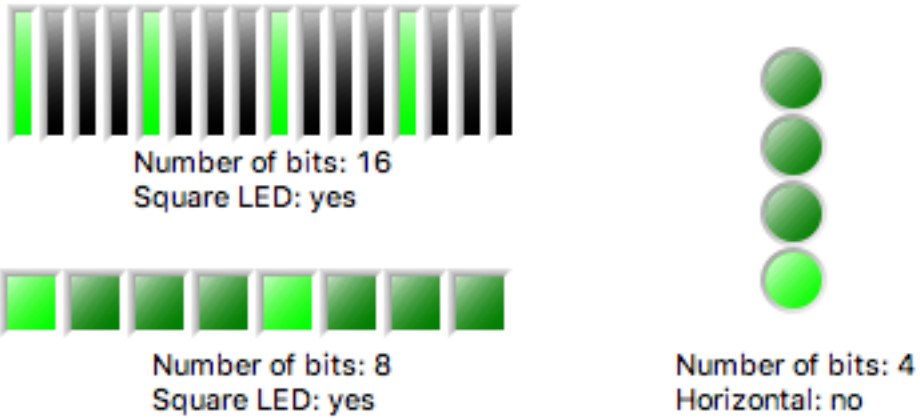
Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

5.6 Byte Monitor

Widget that displays the bits of a numeric value as a series of LEDs.



Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (`pv_name`)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

3D Effect (`effect_3d`)
Whether the rendering includes gradient and shadow effects.

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the label.

Horizontal (`horizontal`)
Direction of the LEDs.

Labels (`label`)
The labels corresponding with each LED. Labels are only visible when **Square LED** is enabled.

LED Border (`led_border`)
The width of the border surrounding each LED bulb.

LED Border Color (`led_border_color`)
The color of the border surrounding each LED bulb.

Number of Bits (`num_bits`)
The number of bits (LEDs) to display.

Off Color (`off_color`)
Color of each LED when it is off.

On Color (`on_color`)
Color of each LED when it is on.

Pack LEDs (`led_packed`)
Collapse borders between successive LEDs. Has best effect when using square LEDs.

Reverse Bits (`bitReverse`)
Left (or top) bit corresponds with the least-significant bit.

Square LED (`square_led`)
Whether the LED shape is round or rectangular.

Start Bit (`startBit`)
Bit position to start displaying from (zero-based).

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

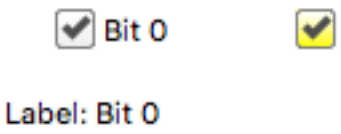
Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

5.7 Check Box

Widget for writing 0 or 1 to the attached PV. This widget can also be used to flip a single bit of the attached PV.



Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (`pv_name`)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Bit (`bit`)
Matches the widget's boolean value to a specific bit of the attached PV's value.

If `-1`, any non-zero value is considered true, whereas a zero value is considered false.

This widget writes the numeric values 0 (unselected) or 1 (selected) to the PV.

Enabled (`enabled`)
Unset to make this control widget unusable.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

Auto Size (`auto_size`)
Adjust the size of the widget to the displayed text.

Note that this can cause unexpected alignment issues when the display is rendered with different fonts.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the label.

Label (`label`)
The displayed text.

Selected Color (`selected_color`)
The color of the check mark.

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

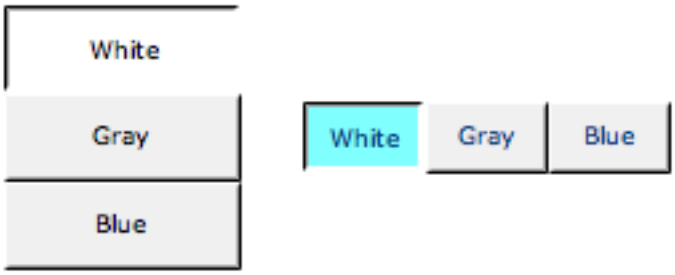
Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

5.8 Choice Button

Widget for writing one of a list of available values to a PV.



Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (`pv_name`)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Enabled (`enabled`)
Unset to make this control widget unusable.

Items (`items`)
The list of available values to choose from.

This property is only available when **Items from PV** is unset.

Items from PV (`items_from_pv`)
If the PV is an enumerated PV, populate the value options based on the list of enumeration states. When selected, each option will write that specific state value to the attached PV.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the label.

Horizontal (`horizontal`)
Direction in which to arrange the items.

Selected Color (`selected_color`)
The color of the dot when selected.

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

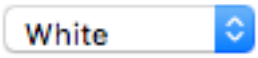
Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

5.9 Combo

Widget for writing one of a list of available values to a PV.



Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (`pv_name`)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Enabled (`enabled`)
Unset to make this control widget unusable.

Items (`items`)
The list of available values to choose from.

This property is only available when **Items from PV** is unset.

Items from PV (`items_from_pv`)
If the PV is an enumerated PV, populate the value options based on the list of enumeration states. When selected, each option will write that specific state value to the attached PV.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the label.

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

5.10 Display

Root container widget for an OPI file.

Basic Properties

Macros (`macros`)
Macros (page 201) available within this container.

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Auto Scale Widgets (`auto_scale_widgets`)
Controls whether autoscaling is enabled.

If so, child widgets are automatically stretched in horizontal and/or vertical direction depending on their individual **Scale Options** property configuration.

Autoscaling can cause shifts in the interdistance between widgets.

Auto Zoom to Fit All (`auto_zoom_to_fit_all`)
If enabled, always zoom the display to fit the available space while preserving ratios.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Display Properties

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Grid Color (`foreground_color`)
Color of grid lines.

Grid Space (`grid_space`)
Space in pixels between grid lines.

Show Close Button (`show_close_button`)
If true, this display's tab does not show the close icon at runtime.

Note that the tab can still be closed through right-click.

Show Edit Range (`show_edit_range`)
If true, two lines matching the display's width and height are visible while editing this OPI display.

Show Grid (`show_grid`)
If true, a grid is visible while editing this OPI display.

Show Ruler (`show_ruler`)
If true, vertical and horizontal rulers are visible while editing this OPI Display.

Snap to Geometry (`snap_to_geometry`)
If true, dragging widgets while editing an OPI display makes the positioning *sticky* with respect to the geometry of other widgets.

Position Properties

Height (`height`)
: Height of the widget area in pixels.

Width (`width`)
: Width of the widget area in pixels

X (`x`)
: X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
: Y-coordinate in pixels of the top-left corner of the widget area.

5.11 Ellipse

Widget that draws an ellipse shape.

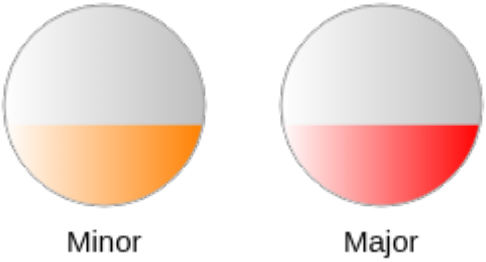
Example of shape fill:

Property	Value
Foreground Color	rgb(255, 0, 0)
Fill Level	40.0
Horizontal Fill	no
Transparent	yes
Line Color	rgb(255, 0, 0)
Line Width	1

Example of gradient effect:

Property	Value
Background Color	rgb(191, 191, 191)
Fill Level	40.0
Foreground Color	rgb(30, 144, 255)
Horizontal Fill	no
Gradient	yes
Line Color	rgb(161, 161, 161)
Line Width	1

The shape background and foreground colors can be made alarm-aware by attaching a PV. Note that the PV value is otherwise ignored. In particular: it does not impact fill level (use a *Tank* (page 144) for this use case).



Property	Value
ForeColor Alarm Sensitive	yes

Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (`pv_name`)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

Alpha (`alpha`)
Opacity level of the shape. Should be a value in the range 0 to 255.

Anti Alias (`anti_alias`)
Whether anti-aliasing is enabled for drawing the shape.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the ellipse shape.

Background Gradient Color (`bg_gradient_color`)
If the **Gradient** property is true, and the **Transparent** property is false, then this indicates the start color of the gradient that is used for the filled part of the shape. The stop color is defined by the property **Background Color**.

Fill Level (`fill_level`)
Percentage of the shape that should be filled with **Foreground Color**. The remaining part of the shape is filled up with **Background Color** (unless the **Transparent** property is enabled).

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the shape fill.

Foreground Gradient Color (`fg_gradient_color`)
If the **Gradient** property is true, then this indicates the start color of the gradient that is used for the filled part of the shape. The stop color is defined by the property **Foreground Color**.

Gradient (`gradient`)
Whether to fill this shape with a gradient.

The gradient for the filled part of this shape are controlled with the properties **Foreground Color** and **Foreground Gradient Color**.

The gradient for the non-filled part of this shape are controlled with the properties **Background Color** and **Background Gradient Color** (unless the **Transparent** property is enabled).

The gradient direction is reverse to the direction set by **Horizontal Fill**.

Horizontal Fill (`horizontal_fill`)
If enabled the fill direction is horizontal (left to right). Otherwise vertical (bottom to top).

Line Color (`line_color`)
Color of the stroke.

Line Style (`line_style`)
The type of stroke.

Code	Value	Description
0	Solid	Uninterrupted
1	Dash	Applies the pattern: 6px solid, 2px gap
2	Dot	Applies the pattern: 2px solid, 2px gap
3	DashDot	Applies the pattern: 6px solid, 2px gap, 2px solid, 2px gap
4	DashDotDot	Applies the pattern: 6px solid, 2px gap, 2px solid, 2px gap, 2px solid, 2px gap

Line Width (`line_width`)
Thickness of the shape stroke

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Transparent (`transparent`)
If true, the unused part of the widget area is left transparent. If false, the unused part uses the **Background Color**.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

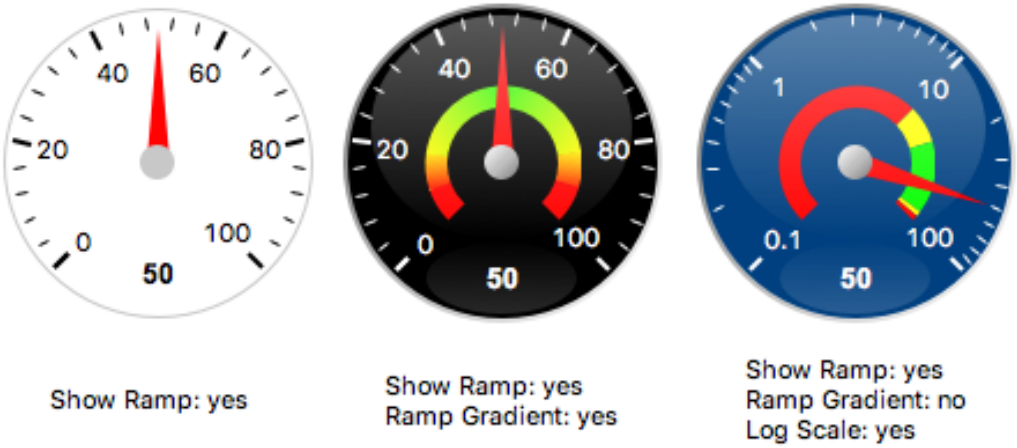
Width (`width`)
Width of the widget area in pixels

X (x)
X-coordinate in pixels of the top-left corner of the widget area.

Y (y)
Y-coordinate in pixels of the top-left corner of the widget area.

5.12 Gauge

Widget that displays a numeric value on a gauge.



Basic Properties

Name (name)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (pv_name)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Widget Type (widget_type)
Readonly property describing the type of this widget.

Behavior Properties

Actions (actions)
Executable *Actions* (page 173) attached to this widget.

Level HI (level_hi)
The high mark.

Level HIHI (level_hihi)
The high high mark.

Level LO (level_lo)
The low mark.

Level LOLO (level_lolo)
The low low mark.

Limits from PV (`limits_from_pv`)
Determine Minimum, LOLO, LO, HI, HIHI, Maximum levels automatically based on the underlying PV.

If the PV is backed by a Yamcs parameter, the mapping is as follows:

Yamcs	Yamcs Studio
WATCH	LO/HI
WARNING	LO/HI
DISTRESS	LO/HI
CRITICAL	LOLO/HIHI
SEVERE	LOLO/HIHI

Maximum (`maximum`)
The upper limit of the widget.

Minimum (`minimum`)
The lower limit of the widget.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

3D Effect (`effect_3d`)
Whether the rendering includes gradient and shadow effects.

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Color HI (`color_hi`)
Color associated with high alarm.

Color HIHI (`color_hihi`)
Color associated with high high alarm.

Color LO (`color_lo`)
Color associated with low alarm.

Color LOLO (`color_lolo`)
Color associated with low low alarm.

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the label.

Log Scale (`log_scale`)
Use a logarithmic scale.

Major Tick Step Hint (`major_tick_step_hint`)
Minimum amount of pixels between major ticks.

Needle Color (`needle_color`)
The color of the needle.

Ramp Gradient (`ramp_gradient`)
Use color transitions on the ramp.

Scale Font (`scale_font`)
The font of the scale.

Scale Format (`scale_format`)
Pattern describing how to format step values shown on the scale.

The pattern follows Java conventions. See https://docs.oracle.com/javase/7/docs/api/java/text/DecimalFormat.html

Some examples:

Value	Format	Printed
1234	#.00	1234.00
12.3456	#.##	12.35
1234	0.###E0	1.234E3

Show HI (`show_hi`)
Whether to show the high mark.

Show HIHI (`show_hihi`)
Whether to show the high high mark.

Show LO (`show_lo`)
Whether to show the low mark.

Show LOLO (`show_lolo`)
Whether to show the low low mark.

Show Minor Ticks (`show_minor_ticks`)
Whether to show minor ticks on the scale.

Show Ramp (`show_markers`)
Whether to show the ramp.

Show Scale (`show_scale`)
Whether to show the ticks and scale labels.

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Transparent Background (`transparent_background`)
If true, the unused part of the widget area is left transparent. If false, the unused part uses the **Background Color**.

Value Label Format (`value_label_format`)
Pattern describing how to format the current value.

The pattern follows Java conventions. See https://docs.oracle.com/javase/7/docs/api/java/text/DecimalFormat.html

Some examples:

Value	Format	Printed
1234	#.00	1234.00
12.3456	#.##	12.35
1234	0.###E0	1.234E3

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

5.13 Grid Layout

Meta widget that can be added to a container for laying out its other contained widgets in a grid.

While editing in Yamcs Studio, a grid icon is visible in the top-left of the container. Right-click this icon and choose **Layout Widgets** to calculate and apply adjusted positions for all widgets belonging to that container.

When running a display, the layout is always calculated prior to rendering.

Grid cell sizes are established using the following algorithm:

1. Group the widgets in rows, respecting the order visible in the Outline view.

2. Per row: apply the largest widget width to all widgets (based on bounding box).
3. Per column: apply the largest widget height (based on bounding box).

Note: Some widgets perform autosizing which may cause the widget area to not use the entire available cell space.

Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Display Properties

Fill Grids (`fill_grids`)
Resize smaller widgets to match the largest widget, thereby filling up the cells.

Grid Gap (`grid_gap`)
Reserve a gap in pixels between grid cells.

Number of Columns (`number_of_columns`)
Number of columns in the grid.

5.14 Grouping Container

Container widget for grouping other widgets. When moving a grouping container, all of its contained widgets are moved with it. X and Y coordinates of contained widgets are relative to the top-left of Grouping Container.

By default a Grouping Container is *Unlocked* as indicated in the top-left corner. Click this indicator to switch the container to *Locked*. Once locked you will not be able to directly select its contained widgets.

Widgets can be added to a group in two ways:

- By dragging and dropping widgets onto the area of an existing Grouping Container.
- By selecting existing widgets and choosing **Create Group** from the right-click context menu.

The bounds of a Grouping Container can be automatically calculated by right-clicking it and choosing **Perform Auto Size**. The new size will account for all the contained widgets to be visible.

Deleting a Grouping Container will also delete all of its children. You can also delete the group without deleting the children, by right-clicking it and choosing **Remove Group**.

Basic Properties

Macros (`macros`)
Macros (page 201) available within this container.

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Enabled (`enabled`)
Unset to make contained control widgets unusable.

Forward Colors (`fc`)
If yes, the **Background Color** and **Foreground Color** set by this container are applied to all contained widgets.

Lock Children (`lock_children`)
If yes, contained widgets are not directly selectable.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Show Scrollbar (`show_scrollbar`)
Show a scrollbar when necessary.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Font (`font`)
The font of the label.

Foreground Color (`foreground_color`)
The color of the label.

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Transparent Background (`transparent`)
Make the container background transparent.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

5.15 Image

Widget that displays an image.

Basic Properties

Image File (`image_file`)
The image file to be shown (GIF, PNG, JPG, BMP).

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Transparent (`transparency`)
Support transparent parts of the drawn image. If false, these parts are filled with the widget **Background Color** instead.

Image Properties

Animation Aligned to Nearest Second (`align_to_nearest_second`)
In case the image is an animated GIF, then start the animation only upon the next second.

Auto Size (`auto_size`)
Adjust the size of the widget to the image size. Has no effect when **Stretch to Fit** is enabled.

Crop Bottom (`crop_bottom`)
Amount of pixels to be cropped from the bottom of the image.

Crop Left (`crop_left`)
Amount of pixels to be cropped from the left of the image.

Crop Right (`crop_right`)
Amount of pixels to be cropped from the right of the image.

Crop Top (`crop_top`)
Amount of pixels to be cropped from the top of the image.

Flip Horizontal (`flip_horizontal`)
Flip the image horizontally.

Flip Vertical (`flip_vertical`)
Flip the image vertically.

No Animation (`no_animation`)
In case the image is an animated GIF, control whether it shows animated or not.

Rotation Angle (`degree`)
Angle in degrees by which to rotate the image clockwise.

Stretch to Fit (`stretch_to_fit`)
Adjust the size of the image to the widget size.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

Width (`width`)
Width of the widget area in pixels

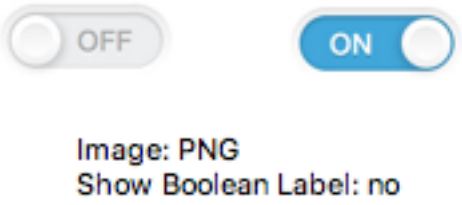
X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

5.16 Image Boolean Button

Widget for writing 0 or 1 to the attached PV. This widget can also be used to flip a single bit of the attached PV.

The widget shows the current state by switching between two configurable images.



Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (`pv_name`)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Bit (`bit`)
Matches the widget's boolean value to a specific bit of the attached PV's value.

If `-1`, any non-zero value is considered true, whereas a zero value is considered false.

This widget writes the numeric values 0 (unselected) or 1 (selected) to the PV.

Confirm Message (`confirm_message`)
The message to be displayed when **Show Confirm Dialog** is set.

Data Type (`data_type`)
Control how the widget boolean value is established.

Code	Value	Description
0	Bit	The widget boolean value matches a specific bit (indicated by the **Bit** property), or the entire value in case the **Bit** property is set to `-1`
1	Enum	The widget boolean value follows the comparison of its value with specific enumeration states (indicated with the **Off State** and **On State** property

Enabled (`enabled`)
Unset to make this control widget unusable.

Off State (`off_state`)
If **Data Type** is set to `Enum`, this indicates the state that matches boolean false.

On State (`on_state`)
If **Data Type** is set to `Enum`, this indicates the state that matches boolean true.

Password (`password`)
If set, this contains the password that needs to be provided by the user when **Show Confirm Dialog** is set, and the user is attempting to change the widget state.

Note: OPIs are rendered on the client. The use of this property provides only a false sense of security.

Push Action Index (`push_action_index`)
Index in the list of associated actions of the action that must be executed when the control is pushed.

Release Action Index (`release_action_index`)
Index in the list of associated actions of the action that must be executed when the control is released.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Show Confirm Dialog (`show_confirm_dialog`)
Show a confirm dialog when an attempt is made to change the widget state.

If the **Password** property is set, the user will be asked to enter that password correctly.

Toggle Button (`toggle_button`)
If yes, this button stays engaged after clicking it. You need to click it a second time to untoggle it.

If no, this button stays engaged only for as long as the mouse stays pressed.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Boolean Label Position (`boolean_label_position`)
Where to position the label within the widget area (top, bottom, left, right, top-left, top-right, bottom-left, bottom-right).

Code	Value
0	Default
1	Top
2	Left
3	Center
4	Right
5	Bottom
6	Top Left
7	Top Right
8	Bottom Left
9	Bottom Right

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the label.

Off Label (`off_label`)
The label text when this widget's boolean state is false.

On Label (`on_label`)
The label text when this widget's boolean state is true.

Show Boolean Label (`show_boolean_label`)
Whether the label is visible (controlled with properties **Off Label** and **On Label**).

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Transparent (`transparency`)
Support transparent parts of the drawn image. If false, these parts are filled with the widget **Background Color** instead.

Animation Aligned to Nearest Second (`align_to_nearest_second`)
In case the image is an animated GIF, then start the animation only upon the next second.

Auto Size (`auto_size`)
Adjust the size of the widget to the image size. Has no effect when **Stretch to Fit** is enabled.

No Animation (`no_animation`)
In case the image is an animated GIF, control whether it shows animated or not.

Off Image (`off_image`)
The image when this widget's boolean state is false.

On Image (`on_image`)
The image when this widget's boolean state is true.

Stretch to Fit (`stretch_to_fit`)
Adjust the size of the image to the widget size.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

5.17 Image Boolean Indicator

Boolean widget that alternates images based on its value.

Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (`pv_name`)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Bit (`bit`)
Matches the widget's boolean value to a specific bit of the attached PV's value.

If `-1`, any non-zero value is considered true, whereas a zero value is considered false.

Data Type (`data_type`)
Control how the widget boolean value is established.

Code	Value	Description
0	Bit	The widget boolean value matches a specific bit (indicated by the **Bit** property), or the entire value in case the **Bit** property is set to `-1`
1	Enum	The widget boolean value follows the comparison of its value with specific enumeration states (indicated with the **Off State** and **On State** property

Off State (`off_state`)
If **Data Type** is set to `Enum`, this indicates the state that matches boolean false.

On State (`on_state`)
If **Data Type** is set to `Enum`, this indicates the state that matches boolean true.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Boolean Label Position (`boolean_label_position`)
Where to position the label within the widget area (top, bottom, left, right, top-left, top-right, bottom-left, bottom-right).

Code	Value
0	Default
1	Top
2	Left
3	Center
4	Right
5	Bottom
6	Top Left
7	Top Right
8	Bottom Left
9	Bottom Right

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the label.

On Label (`on_label`)
The label text when this widget's boolean state is true.

Show Boolean Label (`show_boolean_label`)
Whether the label is visible (controlled with properties **Off Label** and **On Label**).

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Transparent (`transparency`)
Support transparent parts of the drawn image. If false, these parts are filled with the widget **Background Color** instead.

Image Properties

Animation Aligned to Nearest Second (`align_to_nearest_second`)
In case the image is an animated GIF, then start the animation only upon the next second.

Auto Size (`auto_size`)
Adjust the size of the widget to the image size. Has no effect when **Stretch to Fit** is enabled.

No Animation (`no_animation`)
In case the image is an animated GIF, control whether it shows animated or not.

Off Image (`off_image`)
The image when this widget's boolean state is false.

On Image (`on_image`)
The image when this widget's boolean state is true.

Stretch to Fit (`stretch_to_fit`)
Adjust the size of the image to the widget size.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

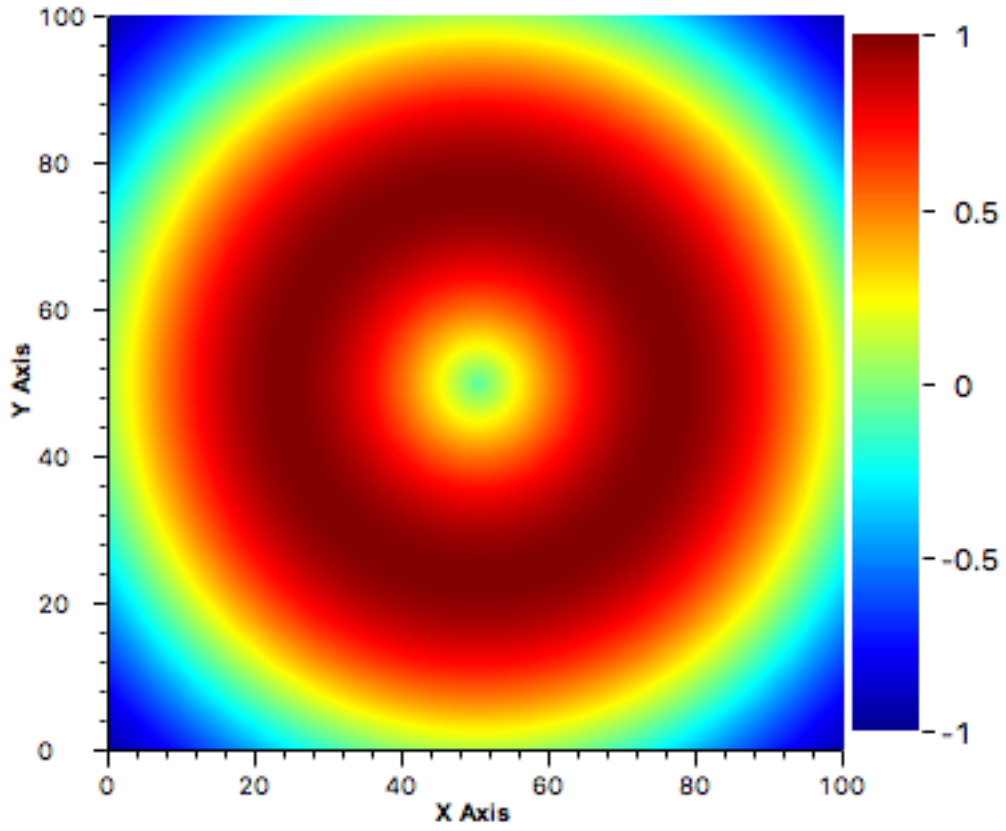
Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

5.18 Intensity Graph

Widget that displays an array of numbers as an image. An Intensity Graph could be used to display a video image, temperature pattern or terrain.



An Intensity Graph can be used in two different modes toggled by the property **RGB Mode**:

1. When not in RGB mode, this widget uses a configurable color map that associates a color with each possible array entry value.

   The input data must then be provided as a single dimensioned array where rows are concatenated to each other. For example, consider the following one-dimensional input data:

   $[p_{1,1}, p_{1,2}, \ldots, p_{1,y}, p_{2,1}, p_{2,2}, \ldots, p_{2,y}, \ldots, p_{x,1}, p_{x,2}, \ldots, p_{x,y}]$

   The widget will interpret this data as a two-dimensional array with `x` rows and `y` columns:

   $[p_{1,1}, p_{1,2}, \ldots, p_{1,y}]$
   $[p_{2,1}, p_{2,2}, \ldots, p_{2,y}]$
   $\ldots$
   $[p_{x,1}, p_{x,2}, \ldots, p_{x,y}]$

2. When in RGB Mode, the input data is again a one-dimensional array of values, but the number of elements (**Data Height** times **Data Width**) must be multiplied by three for describing the red, green and blue values of each pixel. The structure of the data is similar as when using a color map, but each pixel is described by three values: red, green and blue.

   $[p_{1,1,r}, p_{1,1,g}, p_{1,1,b}, \ldots, p_{x,y,r}, p_{x,y,g}, p_{x,y,b}]$

This widget also allows connecting PVs that will receive updates with the profile data matching the input.

Basic Properties

Horizon Profile X PV (`horizon_profile_x_pv_name`)
The output PV which will receive updates with the horizontal profile data on the X axis.

Horizon Profile Y PV (`horizon_profile_y_pv_name`)
The output PV which will receive updates with the horizontal profile data on the Y axis.

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

Pixel Info PV (`pixel_info_pv_name`)
The output PV which will receive updates when the user hovers the graph, or clicks on it. Each such event is sent as as a `VTable` to the PV.

The `VTable` has the following columns:

- `X`: X coordinate of the pixel.
- `Y`: Y coordinate of the pixel.
- `Value`: Value of the image data for this specific pixel.
- `Selected`: True if the user was pressing the mouse button. False otherwise.

PV Name (`pv_name`)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Vertical Profile X PV (`vertical_profile_x_pv_name`)
The output PV which will receives updates with the vertical profile data on the X axis.

Vertical Profile Y PV (`vertical_profile_y_pv_name`)
The output PV which will receives updates with the vertical profile data on the Y axis.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Crop Bottom (`crop_bottom`)
Amount of pixels to be cropped from the bottom of the image.

Crop Left (`crop_left`)
Amount of pixels to be cropped from the left of the image.

Crop Right (`crop_right`)
Amount of pixels to be cropped from the right of the image.

Crop Top (`crop_top`)
Amount of pixels to be cropped from the top of the image.

Data Height (`data_height`)
Number of rows of the input data.

Data Width (`data_width`)
Number of rows of the input data.

Enabled (`enabled`)
Unset to make this control widget unusable.

Maximum (`maximum`)
The upper limit for a singular array element.

Minimum (`minimum`)
The lower limit for a singular array element.

Profile on Single Line (`single_line_profiling`)
If set, profile on a single pixel. A horizontal and vertical guideline is added to the graph. These can be dragged around.

If unset, the profiling is performed on the average of all pixels.

RGB Mode (`rgb_mode`)
Enable RGB mode. In this mode, the color map is not available, and RGB values for each pixel must be provided.

Profiles take the average of r, g and b for each point.

ROI Count (`roi_count`)
The number of regions of interest.

Regions of interest are marked on the graph. For each region of interest a specific set of properties is added as detailed in **ROI Properties**.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Color Map (`color_map`)
Color map for the graph. This maps values to colors. In the dialog you can choose from a set of predefined color maps (defaulting to JET) or make one yourself.

If the option **Interpolate** is selected, the color mapping will be done through linear interpolation of the colors available in the map.

If the option **Auto Scale** is selected, the value range of the color map is remapped to the range indicated by the properties **Minimum** and **Maximum**. If unselected, there is no remapping, and the lookup is direct.

Note: This property has multiple sub-properties. Use in scripts and rules is limited to switching between the predefined maps: GrayScale, JET, ColorSpectrum, Hot, Cool or Shaded.

```
widget.setPropertyValue("color_map", "JET");
```

When doing so the other sub-properties (interpolate, autoscale) default to true.

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the label.

ROI Color (`roi_color`)
The color of the frame for indicating a region of interest.

Show Ramp (`show_ramp`)
Whether to show the color map legend to the right of the graph.

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Position Properties

Graph Area Height (`graph_area_height`)
The height of the actual graph area. The widget size will adjust accordingly.

Graph Area Width (`graph_area_width`)
The width of the actual graph area. The widget size will adjust accordingly.

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

ROI Properties

The property names for the following properties take the form `roi_x_property_name`, where `x` is the zero-based index of that ROI.

Height PV (`roi_x_height_pv`)
PV that provides the height of the ROI.

Title (`roi_x_title`)
The label for this ROI, as shown on the graph.

Visible (`roi_x_visible`)
Whether this ROI is visible on the graph.

Width PV (`roi_x_width_pv`)
PV that provides the width of the ROI.

X PV (`roi_x_x_pv`)
PV that provides the X coordinate of the ROI.

Y PV (`roi_x_y_pv`)
PV that provides the Y coordinate of the ROI.

X Axis Properties

Axis Color (`x_axis_axis_color`)
Color used for rendering X axis ticks and labels.

Axis Title (`x_axis_axis_title`)
Label for the X axis.

Major Tick Step Hint (`x_axis_major_tick_step_hint`)
The minimum space in pixels between major ticks.

Maximum (`x_axis_maximum`)
Upper range of the axis.

Minimum (`x_axis_minimum`)
Lower range of the axis.

Scale Font (`x_axis_scale_font`)
Font used for axis labels.

Show Minor Ticks (`x_axis_show_minor_ticks`)
Whether to show minor ticks on the scale.

Title Font (`x_axis_title_font`)
Font used for the axis title.

Visible (`x_axis_visible`)
Whether to show the X axis.

Y Axis Properties

Axis Color (`y_axis_axis_color`)
Color used for rendering Y axis ticks and labels.

Axis Title (`y_axis_axis_title`)
Label for the Y axis.

Major Tick Step Hint (`y_axis_major_tick_step_hint`)
The minimum space in pixels between major ticks.

Maximum (`y_axis_maximum`)
Upper range of the axis.

Minimum (`y_axis_minimum`)
Lower range of the axis.

Scale Font (`y_axis_scale_font`)
Font used for axis labels.

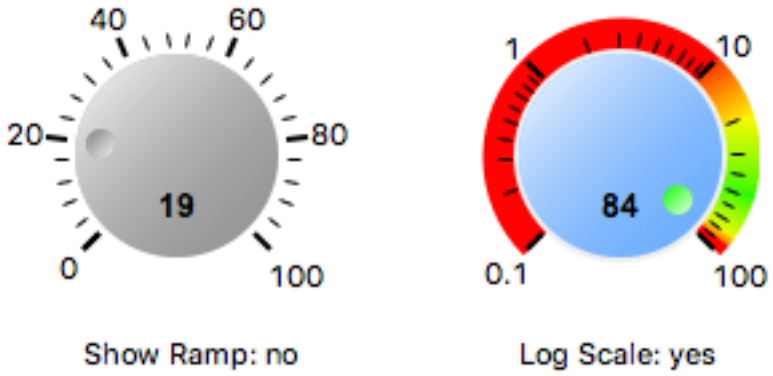
Show Minor Ticks (`y_axis_show_minor_ticks`)
Whether to show minor ticks on the scale.

Title Font (`y_axis_title_font`)
Font used for the axis title.

Visible (`y_axis_visible`)
Whether to show the Y axis.

5.19 Knob

Widget for reading and writing a numeric value.



Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (`pv_name`)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Enabled (`enabled`)
Unset to make this control widget unusable.

Increment (`increment`)
Value added/subtracted when dragging the thumb.

Level HI (`level_hi`)
The high mark.

Level HIHI (`level_hihi`)
The high high mark.

Level LO (`level_lo`)
The low mark.

Level LOLO (`level_lolo`)
The low low mark.

Limits from PV (`limits_from_pv`)
Determine Minimum, LOLO, LO, HI, HIHI, Maximum levels automatically based on the underlying PV.

If the PV is backed by a Yamcs parameter, the mapping is as follows:

Yamcs	Yamcs Studio
WATCH	LO/HI
WARNING	LO/HI
DISTRESS	LO/HI
CRITICAL	LOLO/HIHI
SEVERE	LOLO/HIHI

Maximum (`maximum`)
The upper limit of the widget.

Minimum (`minimum`)
The lower limit of the widget.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

3D Effect (`effect_3d`)
Whether the rendering includes gradient and shadow effects.

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Color HI (`color_hi`)
Color associated with high alarm.

Color HIHI (`color_hihi`)
Color associated with high high alarm.

Color LO (`color_lo`)
Color associated with low alarm.

Color LOLO (`color_lolo`)
Color associated with low low alarm.

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the label.

Knob Color (`knob_color`)
Color of the knob.

Log Scale (`log_scale`)
Use a logarithmic scale.

Major Tick Step Hint (`major_tick_step_hint`)
Minimum amount of pixels between major ticks.

Ramp Gradient (`ramp_gradient`)
Use color transitions on the ramp.

Scale Font (`scale_font`)
The font of the scale.

Scale Format (`scale_format`)
Pattern describing how to format step values shown on the scale.

The pattern follows Java conventions. See https://docs.oracle.com/javase/7/docs/api/java/text/DecimalFormat.html

Some examples:

Value	Format	Printed
1234	#.00	1234.00
12.3456	#.##	12.35
1234	0.###E0	1.234E3

Show HI (`show_hi`)
Whether to show the high mark.

Show HIHI (`show_hihi`)
Whether to show the high high mark.

Show LO (`show_lo`)
Whether to show the low mark.

Show LOLO (`show_lolo`)
Whether to show the low low mark.

Show Minor Ticks (`show_minor_ticks`)
Whether to show minor ticks on the scale.

Show Ramp (`show_markers`)
Whether to show the ramp.

Show Scale (`show_scale`)
Whether to show the ticks and scale labels.

Show Value Label (`show_value_label`)
Whether to show the current value.

Thumb Color (`thumb_color`)
Color of the thumb.

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Transparent Background (`transparent_background`)
If true, the unused part of the widget area is left transparent. If false, the unused part uses the **Background Color**.

Value Label Format (`value_label_format`)
Pattern describing how to format the current value.

The pattern follows Java conventions. See https://docs.oracle.com/javase/7/docs/api/java/text/DecimalFormat.html

Some examples:

Value	Format	Printed
1234	#.00	1234.00
12.3456	#.##	12.35
1234	0.###E0	1.234E3

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

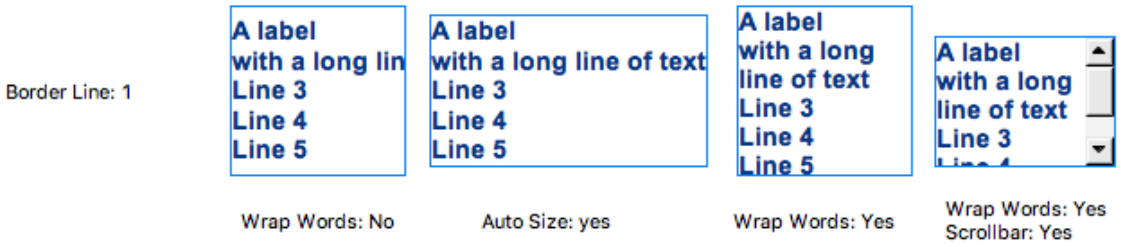
Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

5.20 Label

Widget that draws a text label.



Labels can be edited in-place by pressing `F2` on them.

Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Visible (`visible`)
Manage the visibility of this widget.

Show Scrollbar (`show_scrollbar`)
Show scrollbars when the text overflows the bounding box of this widget.

This property is only visible when **Wrap Words** is set.

Wrap Words (`wrap_words`)
Break lines when they exceed the widget area.

Border Properties

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

Auto Size (`auto_size`)
Adjust the size of the widget to the displayed text.

Note that this can cause unexpected alignment issues when the display is rendered with different fonts.

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Font (`font`)
The font of the label.

Foreground Color (`foreground_color`)
The color of the label.

Horizontal Alignment (`horizontal_alignment`)
Placement of the text within the widget area.

Code	Value
0	Left
1	Center
2	Right

Text (`text`)
The displayed text.

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Transparent (`transparent`)
If true, the unused part of the widget area is left transparent. If false, the unused part uses the **Background Color**.

Vertical Alignment (`vertical_alignment`)
Placement of the text within the widget area.

Code	Value
0	Top
1	Middle
2	Bottom

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

5.21 LED

Boolean widget that displays a value as an ON/OFF LED.

The LED can be made square.

Property	Value
Square LED	yes
Show Boolean Label	yes
Width	40
Height	30

This widget further supports multistate whereby it can assume multiple different color states.

Property	Value
State Count	3
3D Effect	no

Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (`pv_name`)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Bit (`bit`)
Matches the widget's boolean value to a specific bit of the attached PV's value.

If `-1`, any non-zero value is considered true, whereas a zero value is considered false.

Data Type (`data_type`)
Control how the widget boolean value is established.

Code	Value	Description
0	Bit	The widget boolean value matches a specific bit (indicated by the **Bit** property), or the entire value in case the **Bit** property is set to `-1`
1	Enum	The widget boolean value follows the comparison of its value with specific enumeration states (indicated with the **Off State** and **On State** property

Off State (`off_state`)
If **Data Type** is set to `Enum`, this indicates the state that matches boolean false.

On State (`on_state`)
If **Data Type** is set to `Enum`, this indicates the state that matches boolean true.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

State Count (`state_count`)
If set to more than two, this boolean widget becomes a multistate widget, and a number of properties will be added to control each state's specific options: **Color**, **Label** and **Value**.

The `Bit` property then has no more meaning. Instead the widget state will be determined by comparing the widget's numeric value with each state's **Value** property.

Multistate works with numerical values only. It does not work when the **Data Type** is set to `Enum`.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

3D Effect (`effect_3d`)
Whether the rendering includes gradient and shadow effects.

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Bulb Border (`bulb_border`)
The width of the border surrounding the LED bulb.

Bulb Border Color (`bulb_border_color`)
The color of the border surrounding the LED bulb.

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the label.

Off Color (`off_color`)
Color of the LED when it is off.

Off Label (`off_label`)
The label text when this widget's boolean state is false.

On Color (`on_color`)
Color of the LED when it is on.

On Label (`on_label`)
The label text when this widget's boolean state is true.

Show Boolean Label (`show_boolean_label`)
Whether the label is visible (controlled with properties **Off Label** and **On Label**).

Square LED (`square_led`)
Whether the LED shape is round or rectangular.

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

5.22 Linking Container

Container widget that allows to include another OPI file, or a specific group of widgets within another OPI file.

A typical use case would be a common header or footer shown on a series of displays. Or to reuse the same OPI with different *variables*, that is: using macros.

The bounds of a Linking Container can be automatically calculated by right-clicking it and choosing **Perform Auto Size**. The new size will account for all content to be visible.

Basic Properties

Macros (`macros`)
Macros (page 201) available within this container.

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Enabled (`enabled`)
Unset to make contained control widgets unusable.

Group Name (`group_name`)
The name of a specific Grouping Container inside the linked display.

If set, only this particular group will be visible. Otherwise the entire linked display is visible.

OPI File (`opi_file`)
The linked OPI display file.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Font (`font`)
The font of the label.

Foreground Color (`foreground_color`)
The color of the label.

Resize Behaviour (`resize_behaviour`)
How to match the size of the Linking Container with that of the linked display or group.

Code	Value
0	Size *.opi to fit the container
1	Size the container to fit the linked *.opi
2	Don't resize anything, crop if *.opi too large for container
3	Don't resize anything, add scrollbars if *.opi too large for container

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

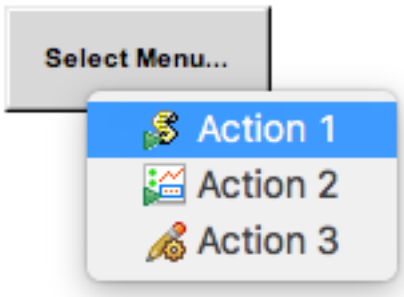
Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

5.23 Menu Button

Control widget for opening a menu when clicked. A menu item is created for each action.



Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (`pv_name`)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Actions from PV (`actions_from_pv`)
If the PV is an enumerated PV, populate actions based on the list of enumeration states. When selected, each action will write that specific state value to the attached PV.

Enabled (`enabled`)
Unset to make this control widget unusable.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the label.

Label (`label`)
The displayed text.

Show Down Arrow (`show_down_arrow`)
Show a down chevron next to the label.

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Transparent (`transparent`)
Do not draw the button background.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

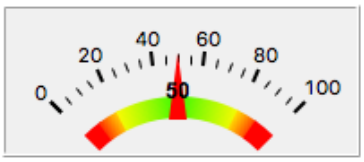
Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

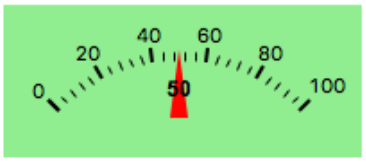
Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

5.24 Meter

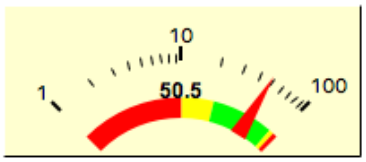
Widget that displays a numeric value on a meter.



Show Ramp: yes
Ramp Gradient: yes



Show Ramp: no



Show Ramp: yes
Ramp Gradient: no
Log Scale: yes

Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (`pv_name`)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Level HI (`level_hi`)
The high mark.

Level HIHI (`level_hihi`)
The high high mark.

Level LO (`level_lo`)
The low mark.

Level LOLO (`level_lolo`)
The low low mark.

Limits from PV (`limits_from_pv`)
Determine Minimum, LOLO, LO, HI, HIHI, Maximum levels automatically based on the underlying PV.

If the PV is backed by a Yamcs parameter, the mapping is as follows:

Yamcs	Yamcs Studio
WATCH	LO/HI
WARNING	LO/HI
DISTRESS	LO/HI
CRITICAL	LOLO/HIHI
SEVERE	LOLO/HIHI

Maximum (`maximum`)
The upper limit of the widget.

Minimum (`minimum`)
The lower limit of the widget.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Color HI (`color_hi`)
Color associated with high alarm.

Color HIHI (`color_hihi`)
Color associated with high high alarm.

Color LO (`color_lo`)
Color associated with low alarm.

Color LOLO (`color_lolo`)
Color associated with low low alarm.

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the label.

Log Scale (`log_scale`)
Use a logarithmic scale.

Major Tick Step Hint (`major_tick_step_hint`)
Minimum amount of pixels between major ticks.

Needle Color (`needle_color`)
The color of the needle.

Ramp Gradient (`ramp_gradient`)
Use color transitions on the ramp.

Scale Font (`scale_font`)
The font of the scale.

Scale Format (`scale_format`)
Pattern describing how to format step values shown on the scale.

The pattern follows Java conventions. See https://docs.oracle.com/javase/7/docs/api/java/text/DecimalFormat.html

Some examples:

Value	Format	Printed
1234	#.00	1234.00
12.3456	#.##	12.35
1234	0.###E0	1.234E3

Show HI (`show_hi`)
Whether to show the high mark.

Show HIHI (`show_hihi`)
Whether to show the high high mark.

Show LO (`show_lo`)
Whether to show the low mark.

Show LOLO (`show_lolo`)
Whether to show the low low mark.

Show Minor Ticks (`show_minor_ticks`)
Whether to show minor ticks on the scale.

Show Ramp (`show_markers`)
Whether to show the ramp.

Show Scale (`show_scale`)
Whether to show the ticks and scale labels.

Show Value Label (`show_value_label`)
Whether to show the current value.

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Value Label Format (`value_label_format`)
Pattern describing how to format the current value.

The pattern follows Java conventions. See https://docs.oracle.com/javase/7/docs/api/java/text/DecimalFormat.html

Some examples:

Value	Format	Printed
1234	#.00	1234.00
12.3456	#.##	12.35
1234	0.###E0	1.234E3

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

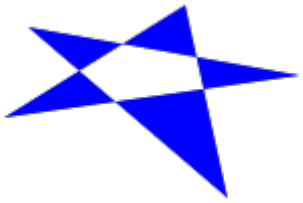
Width (`width`)
Width of the widget area in pixels

X (x)
X-coordinate in pixels of the top-left corner of the widget area.

Y (y)
Y-coordinate in pixels of the top-left corner of the widget area.

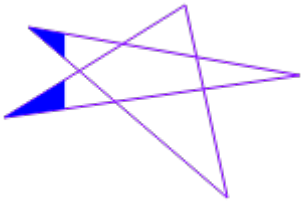
5.25 Polygon

Widget that draws a polygon shape.



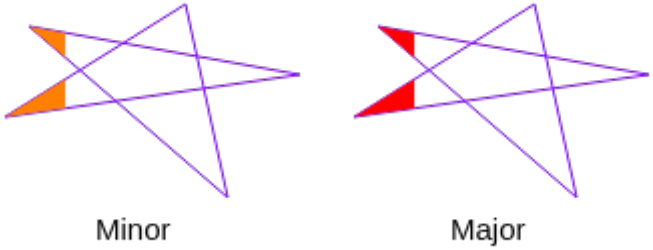
To draw a polygon, choose the tool from the Palette, and click on the start location. Every next click will add a new point. Double-click to indicate this is the last point. The last point is connected to the first point to form a closed shape. Points can be repositioned using the yellow handles.

Example of shape fill:



Property	Value
Fill Level	20.0
Horizontal Fill	yes
Transparent	yes
Line Width	1

The shape background and foreground colors can be made alarm-aware by attaching a PV. Note that the PV value is otherwise ignored.



Property	Value
ForeColor Alarm Sensitive	yes

Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (`pv_name`)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

Alpha (`alpha`)
Opacity level of the shape. Should be a value in the range 0 to 255.

Anti Alias (`anti_alias`)
Whether anti-aliasing is enabled for drawing the shape.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the polygon shape.

Fill Level (`fill_level`)
Percentage of the shape that should be filled with **Foreground Color**. The remaining part of the shape is filled up with **Background Color** (unless the **Transparent** property is enabled).

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the shape fill.

Horizontal Fill (`horizontal_fill`)
If enabled the fill direction is horizontal (left to right). Otherwise vertical (bottom to top).

Line Color (`line_color`)
Color of the stroke.

Line Style (`line_style`)
The type of stroke.

Code	Value	Description
0	Solid	Uninterrupted
1	Dash	Applies the pattern: 6px solid, 2px gap
2	Dot	Applies the pattern: 2px solid, 2px gap
3	DashDot	Applies the pattern: 6px solid, 2px gap, 2px solid, 2px gap
4	DashDotDot	Applies the pattern: 6px solid, 2px gap, 2px solid, 2px gap, 2px solid, 2px gap

Line Width (`line_width`)
Thickness of the shape stroke

Points (`points`)
The point array that describes the shape of this widget.

Rotation Angle (`rotation_angle`)
Angle in degrees by which to rotate the shape clockwise.

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Transparent (`transparent`)
If true, the unused part of the widget area is left transparent. If false, the unused part uses the **Background Color**.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

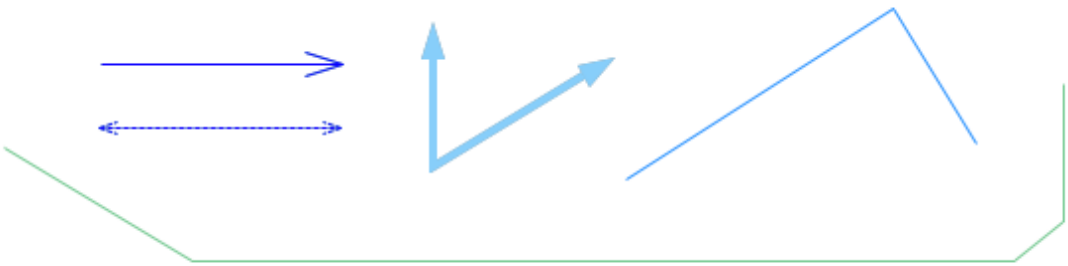
Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

5.26 Polyline

Widget that draws a line or polyline, possibly with arrowheads.



To draw a polyline, choose the tool from the Palette, and click on the start location. Every next click will add a new point. Double-click to indicate this is the last point. Points can be repositioned using the yellow handles.

Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (`pv_name`)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

Alpha (`alpha`)
Opacity level of the shape. Should be a value in the range 0 to 255.

Anti Alias (`anti_alias`)
Whether anti-aliasing is enabled for drawing the shape.

Arrow Length (`arrow_length`)
Length of the arrowheads in pixels.

Arrows (`arrows`)
Which arrowheads to show.

Code	Value
0	None
1	From
2	To
3	Both

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Fill Arrow (`fill_arrow`)
Whether to fill arrowheads.

Fill Level (`fill_level`)
Percentage of the shape that should be filled with **Foreground Color**. The remaining part of the shape is filled up with **Background Color** (unless the **Transparent** property is enabled).

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the label.

Horizontal Fill (`horizontal_fill`)
If enabled the fill direction is horizontal (left to right). Otherwise vertical (bottom to top).

Line Style (`line_style`)
The type of stroke.

Code	Value	Description
0	Solid	Uninterrupted
1	Dash	Applies the pattern: 6px solid, 2px gap
2	Dot	Applies the pattern: 2px solid, 2px gap
3	DashDot	Applies the pattern: 6px solid, 2px gap, 2px solid, 2px gap
4	DashDotDot	Applies the pattern: 6px solid, 2px gap, 2px solid, 2px gap, 2px solid, 2px gap

Line Width (`line_width`)
Thickness of the shape stroke

Points (`points`)
The point array that describes the shape of this widget.

Rotation Angle (`rotation_angle`)
Angle in degrees by which to rotate the shape clockwise.

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Transparent (`transparent`)
If true, the unused part of the widget area is left transparent. If false, the unused part uses the **Background Color**.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

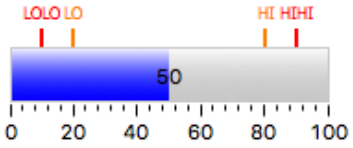
Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

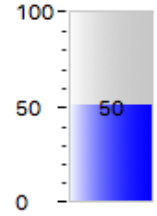
Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

5.27 Progress Bar

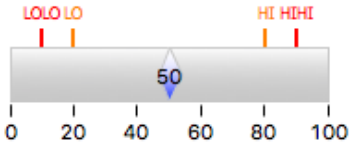
Widget that displays a numeric value on a bar graph.



Vertical: no



Vertical: yes
Show Markers: no



Vertical: no
Minor Tick: no
Indicator Mode: yes

Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (`pv_name`)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Level HI (`level_hi`)
The high mark.

Level HIHI (`level_hihi`)
The high high mark.

Level LO (`level_lo`)
The low mark.

Level LOLO (`level_lolo`)
The low low mark.

Limits from PV (`limits_from_pv`)
Determine Minimum, LOLO, LO, HI, HIHI, Maximum levels automatically based on the underlying PV.

If the PV is backed by a Yamcs parameter, the mapping is as follows:

Yamcs	Yamcs Studio
WATCH	LO/HI
WARNING	LO/HI
DISTRESS	LO/HI
CRITICAL	LOLO/HIHI
SEVERE	LOLO/HIHI

Maximum (`maximum`)
The upper limit of the widget.

Minimum (`minimum`)
The lower limit of the widget.

Origin (`origin`)
Customize the start point of the bar fill (value within the range **Minimum**, **Maximum**).

Ignored if the property **Origin Ignored** is set.

Origin Ignored (`origin_ignored`)
If set, consider the property **Origin**.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

3D Effect (`effect_3d`)
Whether the rendering includes gradient and shadow effects.

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Color Fillbackground (`color_fillbackground`)
Color of the bar background.

Color HI (`color_hi`)
Color associated with high alarm.

Color HIHI (`color_hihi`)
Color associated with high high alarm.

Color LO (`color_lo`)
Color associated with low alarm.

Color LOLO (`color_lolo`)
Color associated with low low alarm.

Fill Color (`fill_color`)
Color of the bar fill (or indicator).

FillColor Alarm Sensitive (`fillcolor_alarm_sensitive`)
If the PV is in alarm state, then **Fill Color** matches the corresponding alarm color.

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the label.

Horizontal (`horizontal`)
Direction of the bar.

Indicator Mode (`indicator_mode`)
If true, the current value is indicated on the bar, but the bar is not filled.

Log Scale (`log_scale`)
Use a logarithmic scale.

Major Tick Step Hint (`major_tick_step_hint`)
Minimum amount of pixels between major ticks.

Scale Font (`scale_font`)
The font of the scale.

Scale Format (`scale_format`)
Pattern describing how to format step values shown on the scale.

The pattern follows Java conventions. See https://docs.oracle.com/javase/7/docs/api/java/text/DecimalFormat.html

Some examples:

Value	Format	Printed
1234	#.00	1234.00
12.3456	#.##	12.35
1234	0.###E0	1.234E3

Show HI (`show_hi`)
Whether to show the high mark.

Show HIHI (`show_hihi`)
Whether to show the high high mark.

Show Label (`show_label`)
Whether to show the current value.

Show LO (`show_lo`)
Whether to show the low mark.

Show LOLO (`show_lolo`)
Whether to show the low low mark.

Show Markers (`show_markers`)
Whether to show LOLO, LO, HI and HIHI markers.

Show Minor Ticks (`show_minor_ticks`)
Whether to show minor ticks on the scale.

Show Scale (`show_scale`)
Whether to show the ticks and scale labels.

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Transparent Background (`transparent_background`)
If true, the unused part of the widget area is left transparent. If false, the unused part uses the **Background Color**.

Value Label Format (`value_label_format`)
Pattern describing how to format the current value.

The pattern follows Java conventions. See https://docs.oracle.com/javase/7/docs/api/java/text/DecimalFormat.html

Some examples:

Value	Format	Printed
1234	#.00	1234.00
12.3456	#.##	12.35
1234	0.###E0	1.234E3

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

5.28 Radio Box

Widget for writing one of a list of available values to a PV.

Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (`pv_name`)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Enabled (`enabled`)
Unset to make this control widget unusable.

Items (`items`)
The list of available values to choose from.

This property is only available when **Items from PV** is unset.

Items from PV (`items_from_pv`)
If the PV is an enumerated PV, populate the value options based on the list of enumeration states. When selected, each option will write that specific state value to the attached PV.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the label.

Horizontal (`horizontal`)
Direction in which to arrange the items.

Selected Color (`selected_color`)
The background color of the selected item.

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

Width (`width`)
Width of the widget area in pixels

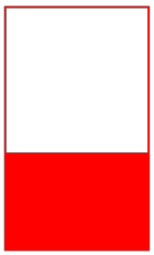
X (x)
X-coordinate in pixels of the top-left corner of the widget area.

Y (y)
Y-coordinate in pixels of the top-left corner of the widget area.

5.29 Rectangle

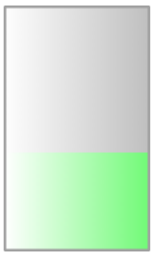
Widget that draws a rectangle shape.

Example of shape fill:



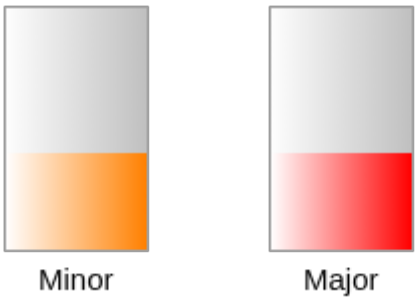
Property	Value
Foreground Color	rgb(255, 0, 0)
Fill Level	40.0
Horizontal Fill	no
Transparent	yes
Line Color	rgb(255, 0, 0)
Line Width	1

Example of gradient effect:



Property	Value
Background Color	rgb(191, 191, 191)
Fill Level	40.0
Foreground Color	rgb(114, 250, 120)
Horizontal Fill	no
Gradient	yes
Line Color	rgb(161, 161, 161)
Line Width	1

The shape background and foreground colors can be made alarm-aware by attaching a PV. Note that the PV value is otherwise ignored. In particular: it does not impact fill level (use a *Tank* (page 144) for this use case).



Property	Value
ForeColor Alarm Sensitive	yes

Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (`pv_name`)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

Alpha (`alpha`)
Opacity level of the shape. Should be a value in the range 0 to 255.

Anti Alias (`anti_alias`)
Whether anti-aliasing is enabled for drawing the shape.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the rectangle shape.

Background Gradient Color (`bg_gradient_color`)
If the **Gradient** property is true, and the **Transparent** property is false, then this indicates the start color of the gradient that is used for the filled part of the shape. The stop color is defined by the property **Background Color**.

Fill Level (`fill_level`)
Percentage of the shape that should be filled with **Foreground Color**. The remaining part of the shape is filled up with **Background Color** (unless the **Transparent** property is enabled).

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the shape fill.

Foreground Gradient Color (`fg_gradient_color`)
If the **Gradient** property is true, then this indicates the start color of the gradient that is used for the filled part of the shape. The stop color is defined by the property **Foreground Color**.

Gradient (`gradient`)
Whether to fill this shape with a gradient.

The gradient for the filled part of this shape are controlled with the properties **Foreground Color** and **Foreground Gradient Color**.

The gradient for the non-filled part of this shape are controlled with the properties **Background Color** and **Background Gradient Color** (unless the **Transparent** property is enabled).

The gradient direction is reverse to the direction set by **Horizontal Fill**.

Horizontal Fill (`horizontal_fill`)
If enabled the fill direction is horizontal (left to right). Otherwise vertical (bottom to top).

Line Color (`line_color`)
Color of the stroke.

Line Style (`line_style`)
The type of stroke.

Code	Value	Description
0	Solid	Uninterrupted
1	Dash	Applies the pattern: 6px solid, 2px gap
2	Dot	Applies the pattern: 2px solid, 2px gap
3	DashDot	Applies the pattern: 6px solid, 2px gap, 2px solid, 2px gap
4	DashDotDot	Applies the pattern: 6px solid, 2px gap, 2px solid, 2px gap, 2px solid, 2px gap

Line Width (`line_width`)
Thickness of the shape stroke

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Transparent (`transparent`)
If true, the unused part of the widget area is left transparent. If false, the unused part uses the **Background Color**.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

Width (`width`)
Width of the widget area in pixels

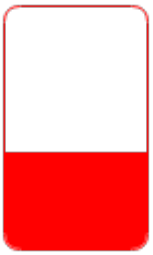
X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

5.30 Rounded Rectangle

Widget that draws a rectangle shape with rounded corners.

Example of shape fill:



Property	Value
Foreground Color	rgb(255, 0, 0)
Fill Level	40.0
Horizontal Fill	no
Transparent	yes
Line Color	rgb(255, 0, 0)
Line Width	1

Example of gradient effect:

Property	Value
Background Color	rgb(191, 191, 191)
Fill Level	40.0
Foreground Color	rgb(114, 250, 120)
Horizontal Fill	no
Gradient	yes
Line Color	rgb(161, 161, 161)
Line Width	1

The shape background and foreground colors can be made alarm-aware by attaching a PV. Note that the PV value is otherwise ignored. In particular: it does not impact fill level (use a *Tank* (page 144) for this use case).

Property	Value
ForeColor Alarm Sensitive	yes

Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (`pv_name`)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

Alpha (`alpha`)
Opacity level of the shape. Should be a value in the range 0 to 255.

Anti Alias (`anti_alias`)
Whether anti-aliasing is enabled for drawing the shape.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the rectangle shape.

Background Gradient Color (`bg_gradient_color`)
If the **Gradient** property is true, and the **Transparent** property is false, then this indicates the start color of the gradient that is used for the filled part of the shape. The stop color is defined by the property **Background Color**.

Corner Height (`corner_height`)
Corner height in pixels.

Corner Width (`corner_width`)
Corner width in pixels.

Fill Level (`fill_level`)
Percentage of the shape that should be filled with **Foreground Color**. The remaining part of the shape is filled up with **Background Color** (unless the **Transparent** property is enabled).

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the shape fill.

Foreground Gradient Color (`fg_gradient_color`)
If the **Gradient** property is true, then this indicates the start color of the gradient that is used for the filled part of the shape. The stop color is defined by the property **Foreground Color**.

Gradient (`gradient`)
Whether to fill this shape with a gradient.

The gradient for the filled part of this shape are controlled with the properties **Foreground Color** and **Foreground Gradient Color**.

The gradient for the non-filled part of this shape are controlled with the properties **Background Color** and **Background Gradient Color** (unless the **Transparent** property is enabled).

The gradient direction is reverse to the direction set by **Horizontal Fill**.

Horizontal Fill (`horizontal_fill`)
If enabled the fill direction is horizontal (left to right). Otherwise vertical (bottom to top).

Line Color (`line_color`)
Color of the stroke.

Line Style (`line_style`)
The type of stroke.

Code	Value	Description
0	Solid	Uninterrupted
1	Dash	Applies the pattern: 6px solid, 2px gap
2	Dot	Applies the pattern: 2px solid, 2px gap
3	DashDot	Applies the pattern: 6px solid, 2px gap, 2px solid, 2px gap
4	DashDotDot	Applies the pattern: 6px solid, 2px gap, 2px solid, 2px gap, 2px solid, 2px gap

Line Width (`line_width`)
Thickness of the shape stroke

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Transparent (`transparent`)
If true, the unused part of the widget area is left transparent. If false, the unused part uses the **Background Color**.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

5.31 Sash Container

Container widget for grouping widgets in two panels. X and Y coordinates of contained widgets are relative to the top-left of the specific sash panel they belong to.

The panels can be oriented left/right or top/bottom. The available container space can be redistributed at runtime by dragging the sash.

Horizontal: yes

Horizontal: no

Basic Properties

Macros (`macros`)
Macros (page 201) available within this container.

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Enabled (`enabled`)
Unset to make contained control widgets unusable.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Font (`font`)
The font of the label.

Foreground Color (`foreground_color`)
The color of the label.

Horizontal (`horizontal`)
If yes, the sash uses a left and a right panel. Otherwise the sash uses a top panel and a bottom panel.

Sash Position (`sash_position`)
Initial position of the sash (value between 0 and 1).

Sash Style (`sash_style`)
Select the style of the sash:

Code	Value
0	None
1	Rounded
2	Ridged
3	Etched
4	Line
5	Double Lines

Sash Width (`sash_width`)
Width in pixels of the sash. A minimum width of 1 is enforced.

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Transparent (`transparent`)
Make the container background transparent.

Panel 1 (Left/Up) Properties

Auto Scale Children (`panel1_auto_scale_children`)
If yes, the contained widgets are auto scaled while the sash divider is being moved. This auto scaling respects the **Scale Options** set by each widget.

If no, contained widgets keep their size, and a scrollbar is added to the sash panel when necessary.

Panel 2 (Right/Down) Properties

Auto Scale Children (`panel2_auto_scale_children`)
If yes, the contained widgets are auto scaled while the sash divider is being moved. This auto scaling respects the **Scale Options** set by each widget.

If no, contained widgets keep their size, and a scrollbar is added to the sash panel when necessary.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

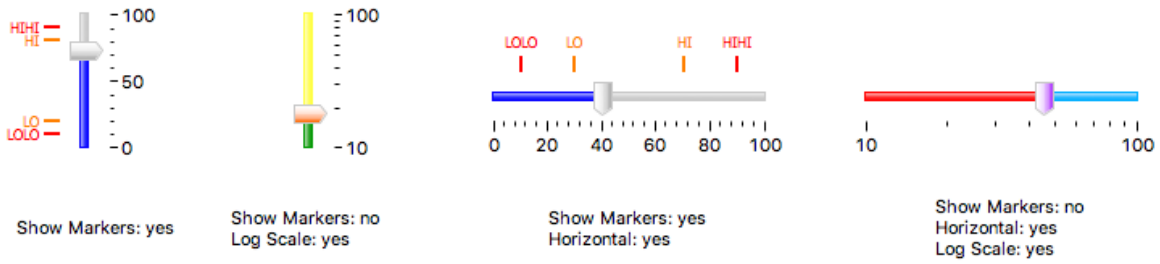
Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

5.32 Scaled Slider

Widget for reading and writing a numeric value in increments.



Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (`pv_name`)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Enabled (`enabled`)
Unset to make this control widget unusable.

Level HI (`level_hi`)
The high mark.

Level HIHI (`level_hihi`)
The high high mark.

Level LO (`level_lo`)
The low mark.

Level LOLO (`level_lolo`)
The low low mark.

Limits from PV (`limits_from_pv`)
Determine Minimum, LOLO, LO, HI, HIHI, Maximum levels automatically based on the underlying PV.

If the PV is backed by a Yamcs parameter, the mapping is as follows:

Yamcs	Yamcs Studio
WATCH	LO/HI
WARNING	LO/HI
DISTRESS	LO/HI
CRITICAL	LOLO/HIHI
SEVERE	LOLO/HIHI

Maximum (`maximum`)
The upper limit of the widget.

Minimum (`minimum`)
The lower limit of the widget.

Page Increment (`page_increment`)
Increment added/subtracted when:

- Clicking the track next to the thumb.
- Pressing `PgUp` or `PgDn`.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Step Increment (`step_increment`)
Increment added/subtracted when:

- Dragging the slider thumb.
- Pressing `Left` or `Right` (in case of a horizontal slider).
- Pressing `Up` or `Down` (in case of a vertical slider).

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

3D Effect (`effect_3d`)
Whether the rendering includes gradient and shadow effects.

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Color Fillbackground (`color_fillbackground`)
Color of the bar background.

Color HI (`color_hi`)
Color associated with high alarm.

Color HIHI (`color_hihi`)
Color associated with high high alarm.

Color LO (`color_lo`)
Color associated with low alarm.

Color LOLO (`color_lolo`)
Color associated with low low alarm.

Fill Color (`fill_color`)
Color of the bar up and until the current value.

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the label.

Horizontal (`horizontal`)
Direction of the bar.

Log Scale (`log_scale`)
Use a logarithmic scale.

Major Tick Step Hint (`major_tick_step_hint`)
Minimum amount of pixels between major ticks.

Scale Font (`scale_font`)
The font of the scale.

Scale Format (`scale_format`)
Pattern describing how to format step values shown on the scale.

The pattern follows Java conventions. See https://docs.oracle.com/javase/7/docs/api/java/text/DecimalFormat.html

Some examples:

Value	Format	Printed
1234	#.00	1234.00
12.3456	#.##	12.35
1234	0.###E0	1.234E3

Show HI (`show_hi`)
Whether to show the high mark.

Show HIHI (`show_hihi`)
Whether to show the high high mark.

Show LO (`show_lo`)
Whether to show the low mark.

Show LOLO (`show_lolo`)
Whether to show the low low mark.

Show Markers (`show_markers`)
Whether to show LOLO, LO, HI and HIHI markers.

Show Minor Ticks (`show_minor_ticks`)
Whether to show minor ticks on the scale.

Show Scale (`show_scale`)
Whether to show the ticks and scale labels.

Thumb Color (`thumb_color`)
Color of the thumb.

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Transparent Background (`transparent_background`)
If true, the unused part of the widget area is left transparent. If false, the unused part uses the **Background Color**.

Value Label Format (`value_label_format`)
Pattern describing how to format the current value.

The pattern follows Java conventions. See https://docs.oracle.com/javase/7/docs/api/java/text/DecimalFormat.html

Some examples:

Value	Format	Printed
1234	#.00	1234.00
12.3456	#.##	12.35
1234	0.###E0	1.234E3

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

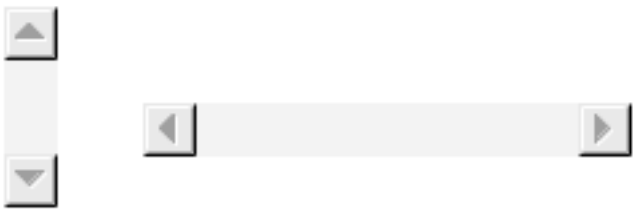
Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

5.33 Scrollbar

Widget for writing to a numeric PV.



Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (`pv_name`)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Bar Length (`bar_length`)
Thumb size, relative to the value range.

Enabled (`enabled`)
Unset to make this control widget unusable.

Limits from PV (`limits_from_pv`)
Determine Minimum and Maximum automatically based on the underlying PV.

If the PV is backed by a Yamcs parameter, the mapping is as follows:

Yamcs	Yamcs Studio
WATCH	LO/HI
WARNING	LO/HI
DISTRESS	LO/HI
CRITICAL	LOLO/HIHI
SEVERE	LOLO/HIHI

Maximum (`maximum`)
The upper limit of the widget.

Minimum (`minimum`)
The lower limit of the widget.

Page Increment (`page_increment`)
Increment added/subtracted when:

- Clicking the scroll track next to the thumb.
- Pressing `PgUp` or `PgDn`.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Step Increment (`step_increment`)
Increment added/subtracted when:

- Clicking the arrow buttons.
- Pressing `Left` or `Right` (in case of a horizontal bar).
- Pressing `Up` or `Down` (in case of a vertical bar).

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the label.

Horizontal (`horizontal`)
Yes for a horizontal bar, otherwise vertical.

Show Value Tip (`show_value_tip`)
Display the current value while it is changing.

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

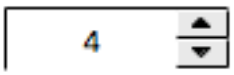
Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

5.34 Spinner

Widget for writing to a numeric PV in increments.



Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (`pv_name`)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Enabled (`enabled`)
Unset to make this control widget unusable.

Limits from PV (`limits_from_pv`)
Determine Minimum and Maximum automatically based on the underlying PV.

If the PV is backed by a Yamcs parameter, the mapping is as follows:

Yamcs	Yamcs Studio
WATCH	LO/HI
WARNING	LO/HI
DISTRESS	LO/HI
CRITICAL	LOLO/HIHI
SEVERE	LOLO/HIHI

Maximum (`maximum`)
The upper limit of the widget.

Minimum (`minimum`)
The lower limit of the widget.

Page Increment (`page_increment`)
Increment added/subtracted when pressing `PgUp` or `PgDn`.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Step Increment (`step_increment`)
Increment added/subtracted when clicking the arrow buttons, or when pressing `Up` or `Down`.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Button on Left (`buttons_on_left`)
Place the arrow buttons left of the input field.

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the label.

Format (`format`)
How to format the PV value.

Code	Value
0	Decimal
1	Exponential
2	Hex

Horizontal Alignment (`horizontal_alignment`)
Placement of the text within the widget area.

Code	Value
0	Left
1	Center
2	Right

Horizontal Buttons Layout (`horizontal_buttons_layout`)
Place the arrow buttons next to each other.

Precision (`precision`)
Precision in case of Decimal format.

Ignored when ** Precision from PV** is enabled.

Precision from PV (`precision_from_pv`)
Get precision information based on the associated PV.

> **Note:** Yamcs does not currently support providing precision information

Show Text (`show_text`)
Display current value. If unset only the arrow buttons are visible.

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Transparent (`transparent`)
Do not draw the input background.

Vertical Alignment (`vertical_alignment`)
Placement of the text within the widget area.

Code	Value
0	Top
1	Middle
2	Bottom

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
: If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

Width (`width`)
: Width of the widget area in pixels

X (`x`)
: X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
: Y-coordinate in pixels of the top-left corner of the widget area.

5.35 Tabbed Container

Container widget for grouping widgets in a number of tabs. X and Y coordinates of contained widgets are relative to the top-left of the container.

While editing in Yamcs Studio, tabs can be organized via the right-click context menu.

Basic Properties

Macros (`macros`)
: *Macros* (page 201) available within this container.

Name (`name`)
: Human-readable name of this widget. Shown in the **Outline** view.

Widget Type (`widget_type`)
: Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
: Executable *Actions* (page 173) attached to this widget.

Enabled (`enabled`)
: Unset to make contained control widgets unusable.

Rules (`rules`)
: *Rules* (page 179) attached to this widget.

Scripts (`scripts`)
: *Scripts* (page 183) attached to this widget.

Tab Count (`tab_count`)
The number of tabs.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

Active Tab (`active_tab`)
The tab that is visible.

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Foreground Color (`foreground_color`)
The color of the label.

Horizontal Tabs (`horizontal_tabs`)
If yes, tabs are organised horizontally (at the top of the widget area).

If no, tabs are organised vertically (at the left of the widget area).

Minimum Tab Height (`minimum_tab_height`)
The minimum height in pixels of tabs.

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Transparent (`transparent`)
Make the container background transparent.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (y)
Y-coordinate in pixels of the top-left corner of the widget area.

Tab-specific Properties

The property names for the following properties take the form `tab_x_property_name`, where `x` is the zero-based index of that tab.

Background Color (`tab_x_background_color`)
The color of the background of the tab's bounding box.

Enabled (`tab_x_enabled`)
Unset to make a tab unselectable, and make its contained control widgets unusable.

Font (`tab_x_font`)
Font used for the tab title.

Foreground Color (`tab_x_foreground_color`)
The text color used for the tab title.

Icon Path (`tab_x_icon_path`)
Path to an image that is used as an icon next to the tab title.

Title (`tab_x_title`)
The title of the tab.

Additional API

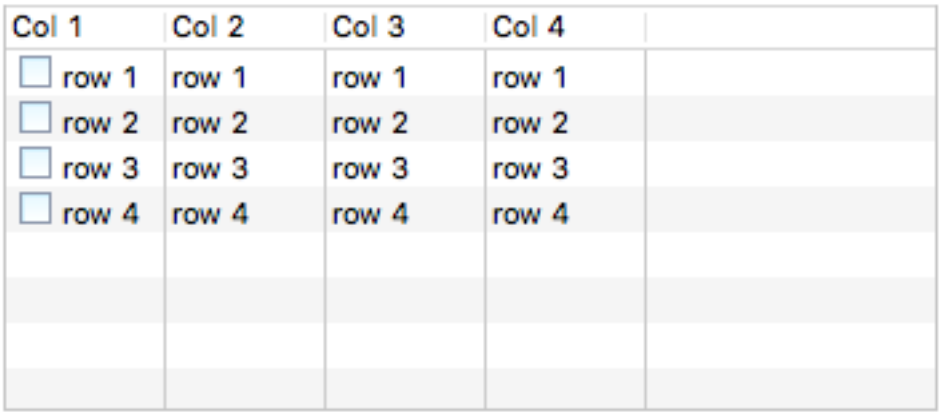
Tabbed Container widgets expose the following additional *Widget* (page 196) API for use in scripting:

getActiveTabIndex()
Get the index of the active tab.

setActiveTabIndex(index)
Activate a specific tab.

5.36 Table

Widget that works like a spreadsheet.



Not all functionalities are exposed as properties. The main use case for this widget is to be populated dynamically from within scripts.

For example, the following JavaScript would append and reveal a row. To make the script run upon display initialization, attach it to the Table widget with a trigger PV set to the formula =1.

```
var table = widget.getTable();

var rowIndex = table.appendRow();
table.setCellText(rowIndex, 0, Math.random());
table.revealRow(rowIndex);
```

Any row column can be modified. The table is automatically extended as needed:

```
var table = widget.getTable();
table.setCellText(4, 5, Math.random());
```

Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Editable (`editable`)
Same as **Enabled**, but if unset, cells can still be selected.

Enabled (`enabled`)
If set, every cell can be modified at runtime by clicking on it. Rows and columns can be managed by using the right-click context menu available from table content.

If unset, cells can't be modified, nor can anything be selected.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Column Headers (`column_headers`)
Configure the following properties for each column: **Column Title**, **Column Width**, **Editable** and **CellEditor**.

CellEditor is one of `TEXT`, `DROPDOWN`, `CHECKBOX` or `CUSTOMIZED`.

In case of `DROPDOWN`, the available items should be set through scripting. Here for column 2:

```
var table = widget.getTable();
var options = Java.to(["Abc", "Def", "Ghi"], "java.lang.String[]");
table.setColumnCellEditorData(2, options);
```

In case of `CHECKBOX`, the boolean labels default to "Yes" and "No". These values can be customized through scripting. Here for column 2:

```
var table = widget.getTable();
var options = Java.to(["ON", "OFF"], "java.lang.String[]");
table.setColumnCellEditorData(2, options);
```

In case of `CUSTOMIZED` you must provide a custom `CellEditor`. This is an advanced use case and not further detailed.

Column Headers Visible (`column_headers_visible`)
Unset to hide column headers.

Columns Count (`columns_count`)
Number of columns.

Default Content (`default_content`)
Optional initial table data.

Foreground Color (`foreground_color`)
The color of the label.

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

Additional API

Table widgets expose the following additional *Widget* (page 196) API for use in scripting:

getTable()
Returns a SpreadSheetTable object for modifying the spreadsheet underlying this table.

setAllowedHeaders(headers)
Restrict the column names to the given string array. If this is set, adding a column at runtime will require to select from one of the available headers.

Class: SpreadSheetTable

addCellEditingListener(listener)
Adds a listener that gets notified whenever a cell is edited.

Usage with JavaScript is as follows:

```
var SpreadSheetTable = Java.type(
    "org.csstudio.swt.widgets.natives.SpreadSheetTable");

var table = widget.getTable();
table.addCellEditingListener(
    new SpreadSheetTable.ITableCellEditingListener({
        cellValueChanged: function(row, col, oldValue, newValue) {
            ConsoleUtil.writeInfo("A cell was modified");
        }
    })
);
```

addModifiedListener(listener)
Adds a listener that gets notified whenever content is modified.

Usage with JavaScript is as follows:

```
var SpreadSheetTable = Java.type(
    "org.csstudio.swt.widgets.natives.SpreadSheetTable");

var table = widget.getTable();
table.addModifiedListener(
    new SpreadSheetTable.ITableModifiedListener({
        modified: function(content) {
            ConsoleUtil.writeInfo("Content was modified");
        }
    })
);
```

addSelectionChangedListener(listener)
Adds a listener that gets notified whenever the selection of the table changes.

Usage with JavaScript is as follows:

```
var SpreadSheetTable = Java.type(
    "org.csstudio.swt.widgets.natives.SpreadSheetTable");

var table = widget.getTable();
table.addSelectionChangedListener(
    new SpreadSheetTable.ITableSelectionChangedListener({
        selectionChanged: function(selection) {
            ConsoleUtil.writeInfo("Selection has changed");
        }
    })
);
```

appendRow()
Adds a row at the bottom of to the spreadsheet, returning the row index.

autoSizeColumns()
Calculate and apply automatic widths for all columns.

deleteColumn(index)
Deletes a column.

deleteRow(index)
Deletes a row.

getCellText(row, col)
Returns the text of a specific cell.

getColumnCount()
Returns the number of columns.

isColumnEditable(index)
Returns `true` if a column is editable.

getColumnHeaders()
Returns an array with the column headers.

getContent()
Returns all spreadsheet data as a two-dimensional array.

getRowCount()
Returns the number of rows.

getSelection()
Returns the current selected data as a two-dimensional array.

insertColumn(index)
Insert a new column, where each value is initialized to an empty string.

insertRow(index)
Insert a new row, where each value is initialized to an empty string.

isEditable()
Returns `true` if the spreadsheet is editable.

isEmpty()
Returns `true` if the spreadsheet is empty.

refresh()
Refresh the table to reflect its content.

revealRow(index)
Scroll a specific row into view.

setCellBackground(row, col, color)
Set the background color of a cell.

Colors can be obtained from *ColorFontUtil* (page 185).

setCellForeground(row, col, color)
Set the foreground color of a cell.

Colors can be obtained from *ColorFontUtil* (page 185).

setCellText(row, col, text)
Set the text of a cell. If the row index is beyond the current row count, the spreadsheet is extended as necessary.

setColumnCellEditorData(col, data)
Set data required for a specific Cell Editor.

In the case of a Cell Editor of type `DROPDOWN`, data should be a Java array of strings. In JavaScript this can be created like this:

```
var data = Java.to(["Abc", "Def", "Ghi"], "java.lang.String[]");
```

In the case of a Cell Editor of type `CHECKBOX`, data should be an array of two strings. One representing the on label, and one representing the off label. In JavaScript this can be created like this:

```
var data = Java.to(["ON", "OFF"], "java.lang.String[]");
```

setColumnCellEditorType(col, type)
Set the editor for cells of a specific column. Type must be one of `TEXT`, `DROPDOWN`, `CHECKBOX` or `CUSTOMIZED`.

setColumnEditable(col, editable)
Set whether the given column is editable or not.

setColumnHeader(col, header)
Set the header for a specific column.

setColumnHeaders(headers)
Set multiple column headers at once. If the given array is larger than the current column count, new columns will be appended to the right.

setColumnHeaderVisible(show)
Set whether to show headers or not.

setColumnsCount(count)
Set the number of columns. If the number is less than the current number of columns, columns will be deleted from the right. If the number is greater than the current number of columns, new columns will be appended to the right.

setColumnWidth(col, width)
Set the pixel width of a specific column.

setColumnWidths(widths)
Set multiple column widths at once. If the given array is larger than the current column count, new columns will be appended to the right.

setContent(content)
Replace the current contents (a two-dimensional array) of this spreadsheet with the given content.

setEditable(editable)
Set whether this spreadsheet is editable or not.

setFont(font)
Set the font used in this spreadsheet.

Fonts can be obtained from *ColorFontUtil* (page 185).

setRowBackground(row, color)
Set the background color of a row.

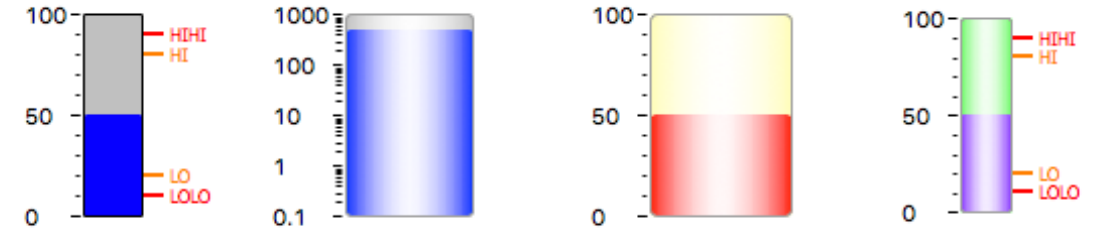
Colors can be obtained from *ColorFontUtil* (page 185).

setRowForeground(row, color)
Set the foreground color of a column.

Colors can be obtained from *ColorFontUtil* (page 185).

5.37 Tank

Widget that displays a numeric value as a fillable tank.



Show Markers: yes

Show Markers: no
Log Scale: yes
3D Effect: yes

Show Markers: no
Log Scale: no
3D Effect: yes

Show Markers: yes
Log Scale: no
3D Effect: yes

Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (`pv_name`)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Level HI (`level_hi`)
The high mark.

Level HIHI (`level_hihi`)
The high high mark.

Level LO (`level_lo`)
The low mark.

Level LOLO (`level_lolo`)
The low low mark.

Limits from PV (`limits_from_pv`)
Determine Minimum, LOLO, LO, HI, HIHI, Maximum levels automatically based on the underlying PV.

If the PV is backed by a Yamcs parameter, the mapping is as follows:

Yamcs	Yamcs Studio
WATCH	LO/HI
WARNING	LO/HI
DISTRESS	LO/HI
CRITICAL	LOLO/HIHI
SEVERE	LOLO/HIHI

Maximum (`maximum`)
The upper limit of the widget.

Minimum (`minimum`)
The lower limit of the widget.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

3D Effect (`effect_3d`)
Whether the rendering includes gradient and shadow effects.

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Color Fillbackground (`color_fillbackground`)
Color of the bar background.

Color HI (`color_hi`)
Color associated with high alarm.

Color HIHI (`color_hihi`)
Color associated with high high alarm.

Color LO (`color_lo`)
Color associated with low alarm.

Color LOLO (`color_lolo`)
Color associated with low low alarm.

Fill Color (`fill_color`)
Color of the bar fill (or indicator).

FillColor Alarm Sensitive (`fillcolor_alarm_sensitive`)
If the PV is in alarm state, then **Fill Color** matches the corresponding alarm color.

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the label.

Log Scale (`log_scale`)
Use a logarithmic scale.

Major Tick Step Hint (`major_tick_step_hint`)
Minimum amount of pixels between major ticks.

Scale Font (`scale_font`)
The font of the scale.

Scale Format (`scale_format`)
Pattern describing how to format step values shown on the scale.

The pattern follows Java conventions. See https://docs.oracle.com/javase/7/docs/api/java/text/DecimalFormat.html

Some examples:

Value	Format	Printed
1234	#.00	1234.00
12.3456	#.##	12.35
1234	0.###E0	1.234E3

Show HI (`show_hi`)
Whether to show the high mark.

Show HIHI (`show_hihi`)
Whether to show the high high mark.

Show LO (`show_lo`)
Whether to show the low mark.

Show LOLO (`show_lolo`)
Whether to show the low low mark.

Show Markers (`show_markers`)
Whether to show LOLO, LO, HI and HIHI markers.

Show Minor Ticks (`show_minor_ticks`)
Whether to show minor ticks on the scale.

Show Scale (`show_scale`)
Whether to show the ticks and scale labels.

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Transparent Background (`transparent_background`)
If true, the unused part of the widget area is left transparent. If false, the unused part uses the **Background Color**.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

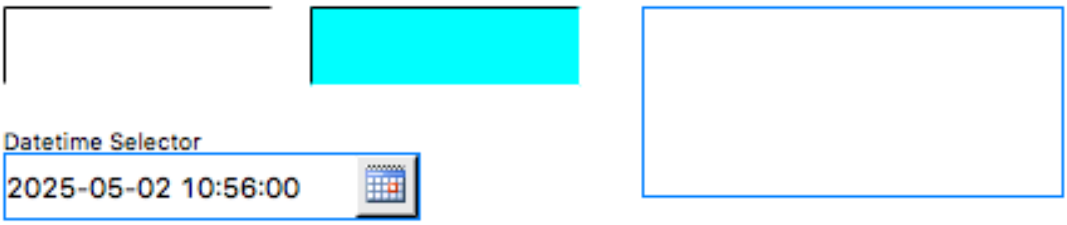
Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

5.38 Text Input

Widget for reading and writing to a PV.



Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (`pv_name`)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Style (`style`)
Type of widget.

Prefer the use of *Classic*. It leads to better compatibility between different OS platforms.

Code	Value
0	Classic
1	Native

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Confirm Message (`confirm_message`)
The message to be displayed when **Show Confirm Dialog** is set.

Enabled (`enabled`)
Unset to make this control widget unusable.

Limits from PV (`limits_from_pv`)
Determine Minimum and Maximum automatically based on the underlying PV.

If the PV is backed by a Yamcs parameter, the mapping is as follows:

Yamcs	Yamcs Studio
WATCH	LO/HI
WARNING	LO/HI
DISTRESS	LO/HI
CRITICAL	LOLO/HIHI
SEVERE	LOLO/HIHI

Maximum (`maximum`)
The upper limit of the widget.

Minimum (`minimum`)
The lower limit of the widget.

Multi-line input (`multiline_input`)
Allow for multiline input.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

Auto Size (`auto_size`)
Adjust the size of the widget to the displayed text.

Note that this can cause unexpected alignment issues when the display is rendered with different fonts.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Datetime Format (`datetime_format`)
The format of a selected datetime. The format may include the following patterns: `yyyy` (year), `MM` (month), `dd` (day), `HH` (hour), `mm` (minute) and `ss` (seconds).

Examples:

- `yyyy-MM-dd HH:mm:ss`
- `MM/dd/yyyy HH:mm:ss`
- `MM/dd/yyyy`

This property is only visible when **Selector Type** is set to `Datetime`.

File Return Part (`file_return_part`)
Which part of the selected path should be returned.

This property is only visible when **Selector Type** is set to `File`.

Code	Value
0	Full Path
1	Name & Extension
2	Name Only
3	Directory

File Source (`file_source`)
What files the selector can choose from.

This property is only visible when **Selector Type** is set to `File`.

Code	Value
0	Workspace
1	Local File System

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the label.

Horizontal Alignment (`horizontal_alignment`)
Placement of the text within the widget area.

Code	Value
0	Left
1	Center
2	Right

Rotation Angle (`rotation_angle`)
Angle in degrees by which to rotate the shape clockwise.

Selector Type (`selector_type`)
Use a special type of selector.

Code	Value
0	None
1	File
2	Datetime

Text (`text`)
Default text (before a PV value arrives).

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Transparent (`transparent`)
If true, the unused part of the widget area is left transparent. If false, the unused part uses the **Background Color**.

Format Properties

Format Type (`format_type`)
How to format the PV value.

Code	Value	Description
0	Default	Use a default format type according to the value type.
1	Decimal	
2	Exponential	Example: `2.023E10`
3	Hex 32	Example: `0xFDC205`
4	String	Print a string representation
5	Hex 64	Same as "Hex 32", but supporting long numbers too. Example: `0xF0DEADBEEF`
6	Compact	If the value is numeric, use either a Decimal or Exponential format, whichever is the shortest.
7	Engineering	Use engineering notation: exponent of ten is power of a thousand. Example: `20.23E9`
8	Sexagesimal	Format as degrees (or hours), minutes, and seconds with colons in-between. Example: `12:45:10.2`
9	Sexagesimal HMS	Same as sexagesimal, but the value is assumed to be in radians, and expressed as hours, minutes and seconds.
10	Sexagesimal DMS	Same as sexagesimal, but the value is assumed to be in radians, and expressed as degrees, minutes and seconds.
11	Time String (Unix Millis)	Print Unix Milliseconds as a time string. The format is specified in **Preferences > Date Format** (default: `yyyy-MM-dd HH:mm:ss.SSS`).

Precision (`precision`)
Precision in case of Default or Decimal format.

Ignored when **Precision from PV** is enabled.

Precision from PV (`precision_from_pv`)
Get precision information based on the associated PV.

Note: Yamcs does not currently support providing precision information

Show Low/High (`show_lohi`)
Show ↓ symbol when the PV exceeds a lower warning or alarm limit, and ↑ symbol when the PV exceeds an upper warning or alarm limit.

Show Units (`show_units`)
If the PV is backed by a parameter with unit information, show these units.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

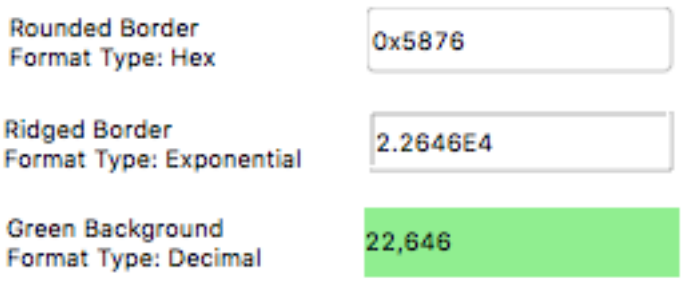
Width (`width`)
Width of the widget area in pixels

X (x)
X-coordinate in pixels of the top-left corner of the widget area.

Y (y)
Y-coordinate in pixels of the top-left corner of the widget area.

5.39 Text Update

Widget that displays a PV value.



Basic Properties

Name (name)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (pv_name)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Widget Type (widget_type)
Readonly property describing the type of this widget.

Behavior Properties

Actions (actions)
Executable *Actions* (page 173) attached to this widget.

Rules (rules)
Rules (page 179) attached to this widget.

Scripts (scripts)
Scripts (page 183) attached to this widget.

Visible (visible)
Manage the visibility of this widget.

Wrap Words (wrap_words)
Break lines when they exceed the widget area.

Border Properties

Alarm Sensitive (border_alarm_sensitive)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

Auto Size (`auto_size`)
Adjust the size of the widget to the displayed text.

Note that this can cause unexpected alignment issues when the display is rendered with different fonts.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the label.

Horizontal Alignment (`horizontal_alignment`)
Placement of the text within the widget area.

Code	Value
0	Left
1	Center
2	Right

Rotation Angle (`rotation_angle`)
Angle in degrees by which to rotate the shape clockwise.

Text (`text`)
Default text (before a PV value arrives).

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Transparent (`transparent`)
If true, the unused part of the widget area is left transparent. If false, the unused part uses the **Background Color**.

Vertical Alignment (`vertical_alignment`)
Placement of the text within the widget area.

Code	Value
0	Top
1	Middle
2	Bottom

Format Properties

Format Type (`format_type`)
How to format the PV value.

Code	Value	Description
0	Default	Use a default format type according to the value type.
1	Decimal	
2	Exponential	Example: `2.023E10`
3	Hex 32	Example: `0xFDC205`
4	String	Print a string representation
5	Hex 64	Same as "Hex 32", but supporting long numbers too. Example: `0xF0DEADBEEF`
6	Compact	If the value is numeric, use either a Decimal or Exponential format, whichever is the shortest.
7	Engineering	Use engineering notation: exponent of ten is power of a thousand. Example: `20.23E9`
8	Sexagesimal	Format as degrees (or hours), minutes, and seconds with colons in-between. Example: `12:45:10.2`
9	Sexagesimal HMS	Same as sexagesimal, but the value is assumed to be in radians, and expressed as hours, minutes and seconds.
10	Sexagesimal DMS	Same as sexagesimal, but the value is assumed to be in radians, and expressed as degrees, minutes and seconds.
11	Time String (Unix Millis)	Print Unix Milliseconds as a time string. The format is specified in **Preferences > Date Format** (default: `yyyy-MM-dd HH:mm:ss.SSS`).

Precision (`precision`)
Precision in case of Default or Decimal format.

Ignored when **Precision from PV** is enabled.

Precision from PV (`precision_from_pv`)
Get precision information based on the associated PV.

Note: Yamcs does not currently support providing precision information

Show Low/High (`show_lohi`)
Show ↓ symbol when the PV exceeds a lower warning or alarm limit, and ↑ symbol when the PV

exceeds an upper warning or alarm limit.

Show Units (`show_units`)
If the PV is backed by a parameter with unit information, show these units.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

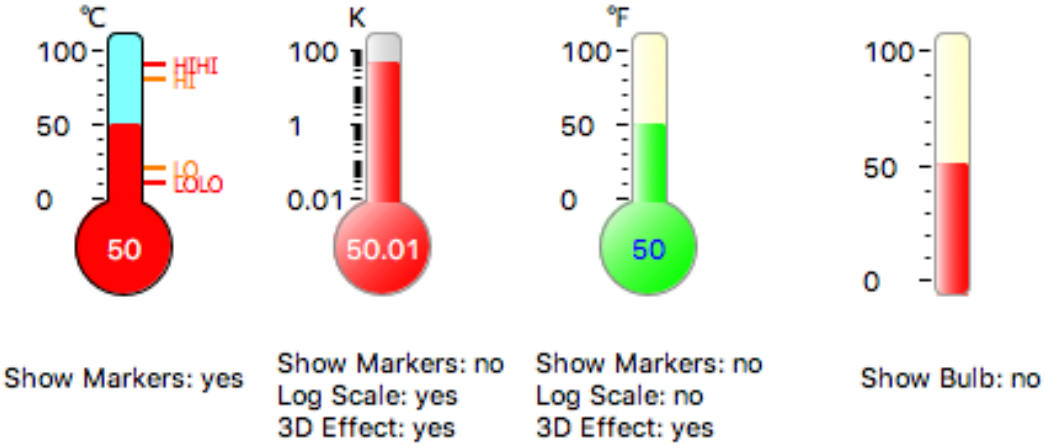
Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

5.40 Thermometer

Widget that displays a numeric value as a thermometer.



Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (`pv_name`)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Level HI (`level_hi`)
The high mark.

Level HIHI (`level_hihi`)
The high high mark.

Level LO (`level_lo`)
The low mark.

Level LOLO (`level_lolo`)
The low low mark.

Limits from PV (`limits_from_pv`)
Determine Minimum, LOLO, LO, HI, HIHI, Maximum levels automatically based on the underlying PV.

If the PV is backed by a Yamcs parameter, the mapping is as follows:

Yamcs	Yamcs Studio
WATCH	LO/HI
WARNING	LO/HI
DISTRESS	LO/HI
CRITICAL	LOLO/HIHI
SEVERE	LOLO/HIHI

Maximum (`maximum`)
The upper limit of the widget.

Minimum (`minimum`)
The lower limit of the widget.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

3D Effect (`effect_3d`)
Whether the rendering includes gradient and shadow effects.

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Color Fillbackground (`color_fillbackground`)
Color of the bar background.

Color HI (`color_hi`)
Color associated with high alarm.

Color HIHI (`color_hihi`)
Color associated with high high alarm.

Color LO (`color_lo`)
Color associated with low alarm.

Color LOLO (`color_lolo`)
Color associated with low low alarm.

Fill Color (`fill_color`)
Color of the bar fill (or indicator).

FillColor Alarm Sensitive (`fillcolor_alarm_sensitive`)
If the PV is in alarm state, then **Fill Color** matches the corresponding alarm color.

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the label.

Log Scale (`log_scale`)
Use a logarithmic scale.

Major Tick Step Hint (`major_tick_step_hint`)
Minimum amount of pixels between major ticks.

Scale Font (`scale_font`)
The font of the scale.

Scale Format (`scale_format`)
Pattern describing how to format step values shown on the scale.

The pattern follows Java conventions. See https://docs.oracle.com/javase/7/docs/api/java/text/DecimalFormat.html

Some examples:

Value	Format	Printed
1234	#.00	1234.00
12.3456	#.##	12.35
1234	0.###E0	1.234E3

Show Bulb (`show_bulb`)
Whether to show the bulb.

Show HI (`show_hi`)
Whether to show the high mark.

Show HIHI (`show_hihi`)
Whether to show the high high mark.

Show LO (`show_lo`)
Whether to show the low mark.

Show LOLO (`show_lolo`)
Whether to show the low low mark.

Show Markers (`show_markers`)
Whether to show LOLO, LO, HI and HIHI markers.

Show Minor Ticks (`show_minor_ticks`)
Whether to show minor ticks on the scale.

Show Scale (`show_scale`)
Whether to show the ticks and scale labels.

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Transparent Background (`transparent_background`)
If true, the unused part of the widget area is left transparent. If false, the unused part uses the **Background Color**.

Unit (`unit`)
The thermometer unit to be displayed.

Code	Value
0	Celcius
1	Fahrenheit
2	Kelvin
3	None

Value Label Format (`value_label_format`)
Pattern describing how to format the current value.

The pattern follows Java conventions. See https://docs.oracle.com/javase/7/docs/api/java/text/DecimalFormat.html

Some examples:

Value	Format	Printed
1234	#.00	1234.00
12.3456	#.##	12.35
1234	0.###E0	1.234E3

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

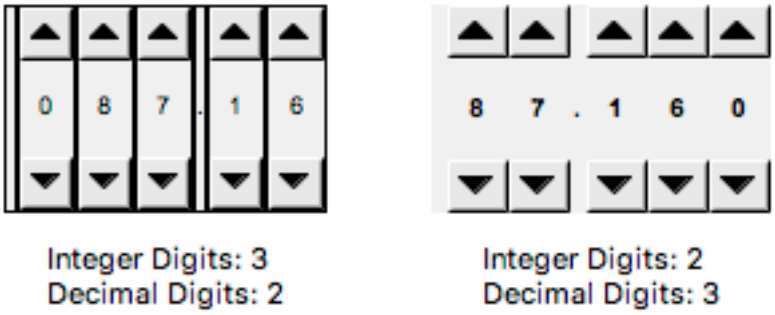
Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

5.41 Thumb Wheel

Widget for reading or writing a numeric PV by digit.



Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (`pv_name`)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Decimal Digits (`decimalDigits`)
Number of decimal digits.

Enabled (`enabled`)
Unset to make this control widget unusable.

Integer Digits (`integerDigits`)
Number of integer digits.

Maximum (`maximum`)
The upper limit of the widget.

Minimum (`minimum`)
The lower limit of the widget.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

Focused Frame Color (`focusedFrameColor`)
Color of the border of the focused digit.

Font (`font`)
The font of the label.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the label.

Internal Frame Color (`internalFrameColor`)
Color of the border surrounding each digit.

Internal Frame Thickness (`internalFrameSize`)
Thickness in pixels of the border surrounding each digit.

Show Buttons (`show_buttons`)
Show the arrow buttons for manipulating the digits.

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

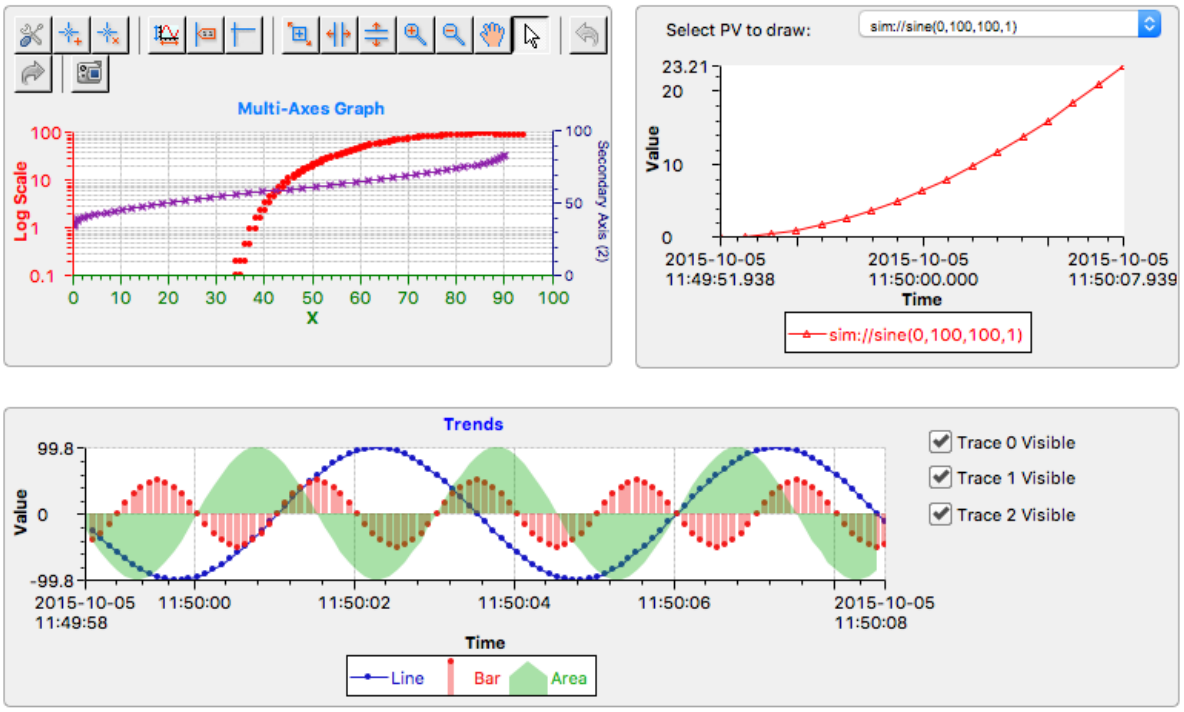
Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

5.42 XY Graph

Widget for plotting one or two-dimensional data.



Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

PV Name (`pv_name`)
The name of the main PV for this widget. If set, the widget's value follows value updates of the corresponding PV.

It is not a requirement to use a PV. You may also control the widget value directly through scripting.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Axis Count (`axis_count`)
The number of axis. Must be between 2 and 4. For each axis, a property group is added. The first axis is always considered the primary X axis. The second axis is always considered the primary Y axis. Up to two other axes may be added, whether they are X or Y is controlled with the **Y Axis** property of each secondary axis.

Enabled (`enabled`)
Unset to make this control widget unusable.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Trace Count (`trace_count`)
The number of traces. For each trace, a property group is added.

Trigger PV (`trigger_pv`)
PV that serves as the trigger for traces that have the property **Update Mode** set to `Trigger`.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Alarm Sensitive (`border_alarm_sensitive`)
If the PV is in alarm state, the widget border and style change to alarm mode.

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

Alarm Pulsing (`alarm_pulsing`)
If enabled, the PV is in alarm state, and the properties **BackColor Alarm Sensitive** and/or **ForeColor Alarm Sensitive** are used, then the corresponding colors will fade in and out to draw operator's attention.

BackColor Alarm Sensitive (`backcolor_alarm_sensitive`)
If the PV is in alarm state, then **Background Color** matches the corresponding alarm color.

Background Color (`background_color`)
The color of the background of the widget's bounding box. Only visible when the widget uses a border style that fills up the widget area.

ForeColor Alarm Sensitive (`forecolor_alarm_sensitive`)
If the PV is in alarm state, then **Foreground Color** matches the corresponding alarm color.

Foreground Color (`foreground_color`)
The color of the title.

Plot Area Background Color (`plot_area_background_color`)
Background color used to colorize the plot area.

Show Legend (`show_legend`)
Whether to show the trace legend.

Show Plot Area Border (`show_plot_area_border`)
Whether to show a border around the plot area.

Show Toolbar (`show_toolbar`)
Whether to show a toolbar with advanced controls.

Title (`title`)
The title of the graph.

Title Font (`title_font`)
The font used to render the title.

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Transparent (`transparent`)
If true, the unused part of the widget area is left transparent. If false, the unused part uses the **Background Color**.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

Axis Properties

The property names for the following properties take the form `axis_x_property_name`, where `x` is the zero-based index of that axis.

The first two axes are always the *primary* X and Y axis. Other axes are considered *secondary* and have additional properties **Left/Bottom Side** and **Y Axis**.

Auto Scale (`axis_x_auto_scale`)
Whether to automatically adjust the scale of this axis.

Auto Scale Threshold (`axis_x_auto_scale_threshold`)
Value in the range `[0-1]` representing a portion of the plot area. If **Auto Scale** is enabled, it will only trigger if current spare space exceeds this threshold.

Axis Color (`axis_x_axis_color`)
The color of this axis. Used for colorizing ticks, title and labels.

Axis Title (`axis_x_axis_title`)
The title of this axis.

Dash Grid Line (`axis_x_dash_grid_line`)
Whether to use a dashed line for the grid matching this axis. Otherwise solid.

Grid Color (`axis_x_grid_color`)
The color of the grid matching this axis.

Left/Bottom Side (`axis_x_left_bottom_side`)
If true this axis is shown on the left (Y axis) or bottom (X axis) side. If false this axis is shown on the right (Y axis) or to (X axis) side.

This property is only available for *secondary* axes (index > 1).

Log Scale (`axis_x_log_scale`)
Use a logarithmic scale.

Maximum (`axis_x_maximum`)
The upper bound the axis range.

Minimum (`axis_x_minimum`)
The lower bound of the axis range.

Scale Font (`axis_x_scale_font`)
The font used to render the scale labels.

Scale Format (`axis_x_scale_format`)
The format used to render scale labels.

The pattern follows Java conventions. See https://docs.oracle.com/javase/7/docs/api/java/text/DecimalFormat.html

Some examples:

Value	Format	Printed
1234	#.00	1234.00
12.3456	#.##	12.35
1234	0.###E0	1.234E3

Show Grid (`axis_x_show_grid`)
Whether to show a grid matching this axis.

Time Format (`axis_x_time_format`)
The format used in case this axis should be used for showing time.

Code	Value	Description
0	None	This axis is not time-based (number formatting defined by **Scale Format**).
1	yyyy-MM-dd HH:mm:ss	
2	yyyy-MM-dd HH:mm:ss.SSS	
3	HH:mm:ss	
4	HH:mm:ss.SSS	
5	HH:mm	
6	yyyy-MM-dd	
7	MMMMM d	
8	Auto	The format of the time label is automatically determined.

Title Font (`axis_x_title_font`)
Font used for rendering the axis title.

Visible (`axis_x_visible`)
Whether this axis is visible.

Y Axis (`axis_x_y_axis`)
Whether this is an Y axis.

This property is only available for *secondary* axes (index > 1).

Trace Properties

The property names for the following properties take the form `trace_x_property_name`, where `x` is the zero-based index of that trace.

Anti Alias (`trace_x_anti_alias`)
Smoothen this trace.

Buffer Size (`trace_x_buffer_size`)
Size of the FIFO buffer underlying this trace. When the buffer is full older items get deleted.

Concatenate Data (`trace_x_concatenate`)
This property is only useful when using with array PVs. Leave it enabled for other PV types.

If yes, whenever the array PV is updated, all of the new array entries are appended to the existing trace data.

If no, whenever the array PV is updated, all of the new array entries replace all the existing trace data.

Line Width (`trace_x_line_width`)
Thickness of the trace. If the **Trace Type** is set to `Bar`, this signifies the bar width.

Name (`trace_x_name`)
Name of the trace, as visible in the legend.

Plot Mode (`trace_x_plot_mode`)
Specifies what to do when the underlying buffer fills up.

Code	Value	Description
0	Plot last n pts.	Show the last updates, removing older points when buffer is full.
1	Plot n pts & stop.	Stop updating the plot when the buffer is full. No data gets removed from the buffer.

Point Size (`trace_x_point_size`)
Size in pixels of points if **Point Style** is not set to `None`.

Point Style (`trace_x_point_style`)
How to stylize data points.

Code	Value
0	None
1	Point
2	Circle
3	Filled Circle
4	Triangle
5	Filled Triangle
6	Square
7	Filled Square
8	Diamond
9	Filled Diamond
10	X Cross
11	Cross
12	Bar

Trace Color (`trace_x_trace_color`)
Color of this trace.

Trace Type (`trace_x_trace_type`)
Type of trace visualization.

Code	Value
0	Solid Line
1	Dash Line
2	Point
3	Bar
4	Area
5	Line Area
6	Step Vertically
7	Step Horizontally
8	Dash Dot Line
9	Dash Dot Dot Line
10	Dot Line

Update Delay (`trace_x_update_delay`)
Throttle plot updates by the given amount of milliseconds. In case of multiple traces, the shortest update time takes precedence.

Update Mode (`trace_x_update_mode`)
Specify when PV updates should be added to the FIFO buffer underlying this trace.

Code	Value	Description
0	X or Y	Update the buffer whenever the X PV or the Y PV has changed. No received data will be missed with this mode.
1	X and Y	Update the buffer only as soon as both the X PV and the Y PV have received an update. Only the last value for each is added to the buffer, so it is possible to miss some values with this mode.
2	X	Update the buffer whenever the X PV has changed. Data coming from the Y PV may be missed with this mode (for example because Y PV updates faster than X PV).
3	Y	Update the buffer whenever the Y PV has changed. Data coming from the X PV may be missed with this mode (for example because X PV updates faster than Y PV).
4	Trigger	Update the buffer only whenever the **Trigger PV** has changed. This is one of the graph properties shared between all traces. Data coming from both the X PV and the Y PV maybe be missed with this mode (for example because they update faster than the trigger PV).

Visible (`trace_x_visible`)
Whether this trace should be visible.

X Axis Index (`trace_x_x_axis_index`)
Index of the axis that is X axis of this trace.

X PV (`trace_x_x_pv`)
The PV providing x values. If empty, this trace is assumed to be chronological.

Y Axis Index (`trace_x_y_axis_index`)
Index of the axis that is Y axis of this trace.

Y PV (`trace_x_y_pv`)
The PV providing y values. By default this is set to the macro `$(pv_name)` so that a user can more simply populate only the **PV Name** field, as the common use case is to render plots with only one X and Y axis.

Additional API

XY Graph widgets expose the following additional *Widget* (page 196) API for use in scripting:

clearGraph()
Clear the graph (deletes trace buffers).

getXBuffer(trace)
Returns the current X axis values for the given trace index as an array of doubles.

getYBuffer(trace)
Returns the current Y axis values for the given trace index as an array of doubles.

5.43 Web Browser

Widget that embeds the native browser engine.

Basic Properties

Name (`name`)
Human-readable name of this widget. Shown in the **Outline** view.

URL (`url`)
The website address.

Widget Type (`widget_type`)
Readonly property describing the type of this widget.

Behavior Properties

Actions (`actions`)
Executable *Actions* (page 173) attached to this widget.

Enabled (`enabled`)
Unset to make this control widget unusable.

Rules (`rules`)
Rules (page 179) attached to this widget.

Scripts (`scripts`)
Scripts (page 183) attached to this widget.

Visible (`visible`)
Manage the visibility of this widget.

Border Properties

Border Color (`border_color`)
The color of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Border Style (`border_style`)
The type of *border* (page 175). Some border styles also colorize the background of the widget's bounding box.

Border Width (`border_width`)
The thickness of the widget border.

Has no meaning with certain types of border styles (for example, raised borders have a fixed style).

Display Properties

Show Toolbar (`show_toolbar`)
Show the browser's toolbar to allow for navigation.

Tooltip (`tooltip`)
Tooltip when mouse hovers this widget.

Position Properties

Height (`height`)
Height of the widget area in pixels.

Scale Options (`scale_options`)
If autoscaling is enabled on the *Display* (page 62), then this property allows controlling whether and how this widget participates.

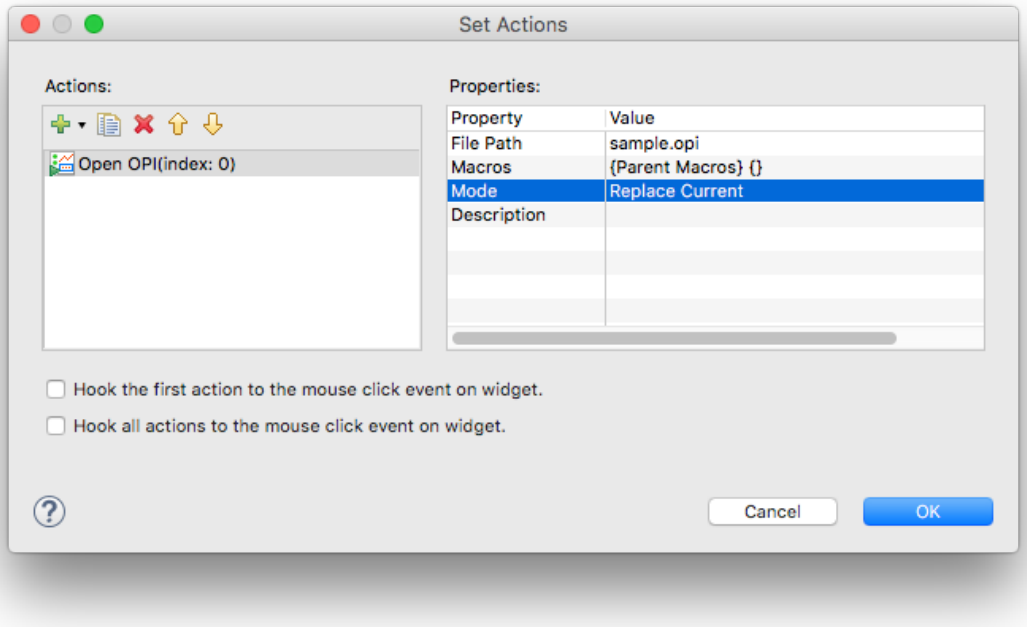
Width (`width`)
Width of the widget area in pixels

X (`x`)
X-coordinate in pixels of the top-left corner of the widget area.

Y (`y`)
Y-coordinate in pixels of the top-left corner of the widget area.

6. Actions

Widgets have an **Actions** property which is used to trigger actions upon user interaction. The common case is to trigger Actions with an *Action Button* (page 40).



Actions can be linked to a push event of a button, or the release event of a toggle button. All actions of a widget are also available via the right-click context menu of that widget.

When using a widget scripts, an action can also be triggered programmatically using *executeAction(index)* (page 196).

The available actions are:

Open OPI

Opens another OPI.

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 *macro* (page 201) `$(pv_name)` is automatically substituted with the PV attached to the widget.

Execute System Command

This executes a command on your operating system.

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.

Run Command
: This runs a telecommand on the currently connected Yamcs processor.

Run Command Stack
: This runs all commands in a Yamcs Command Stack file (extension `*.ycs`).

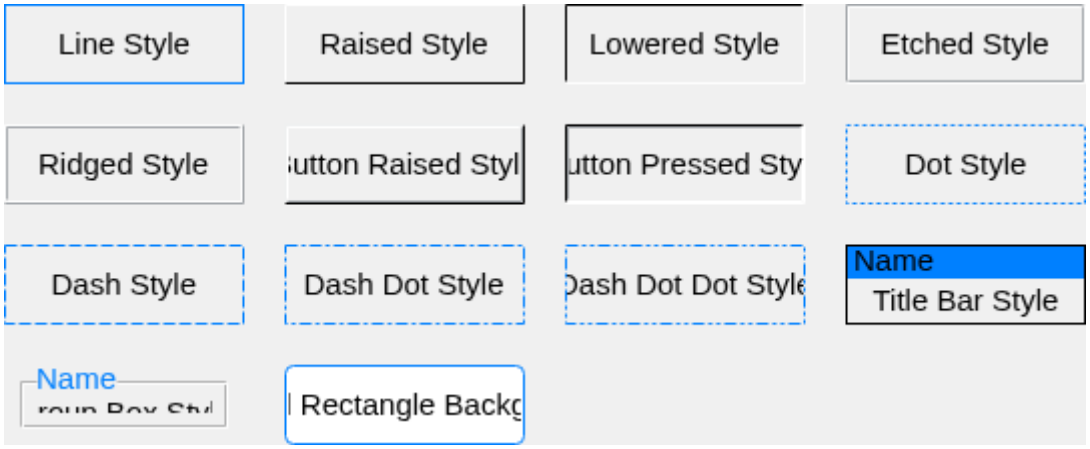
Play WAV File
: Plays the specified sound file.

Open File
: Opens a workspace file with the default handler.

Open Webpage
: Open the specified web page with the integrated web browser.

7. Borders

All widgets can show a border surrounding their bounding box. The border look is controlled by the properties **Border Color**, **Border Width** and **Border Style**.



The codes for these borders are:

Code	Value	Comment
0	None	
1	Line Style	
2	Raised Style	
3	Lowered Style	
4	Etched Style	
5	Ridged Style	
6	Button Raised Style	
7	Button Pressed Style	
8	Dot Style	
9	Dash Style	
10	Dash Dot Style	
11	Dash Dot Dot Style	
12	Title Bar Style	Title is the value of the **Name** property. Title background uses **Background Color**.
13	Group Box Style	Title is the value of the **Name** property. Border and title background use **Background Color**. Title foreground uses **Border Color**.
14	Round Rectangle Background	Applies **Background Color** of the widget.
15	Empty Background	

PV-sensitivity

When a widget is backed by a PV, the border transforms depending on the PV state.

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
Solid pink border around the widget

Alarm-sensitivity

If the widget has the border property **Alarm Sensitive** set, the border transforms to a 2 pixel wide alarm border when the PV is in alarm state.

Minor Alarm
: Solid orange border around the widget

Major Alarm
: Solid red border around the widget

If the PV is connected to a Yamcs parameter, the mapping is done as follows:

- WATCH, WARNING, DISTRESS → MINOR
- CRITICAL, SEVERE → MAJOR

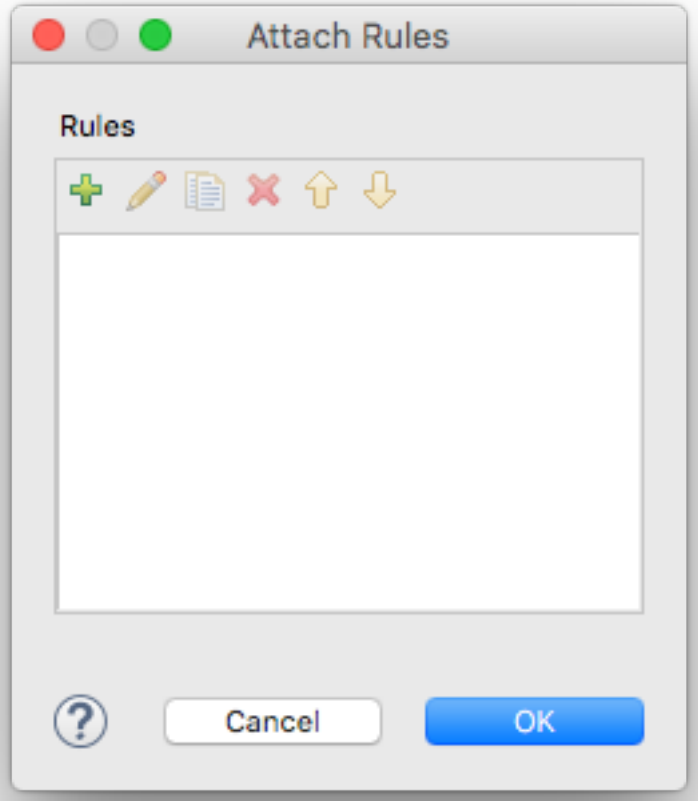
8. Rules

Widgets have many static properties. With *rules* we can modify these properties dynamically, for example based on the incoming value updates of a Parameter PV.

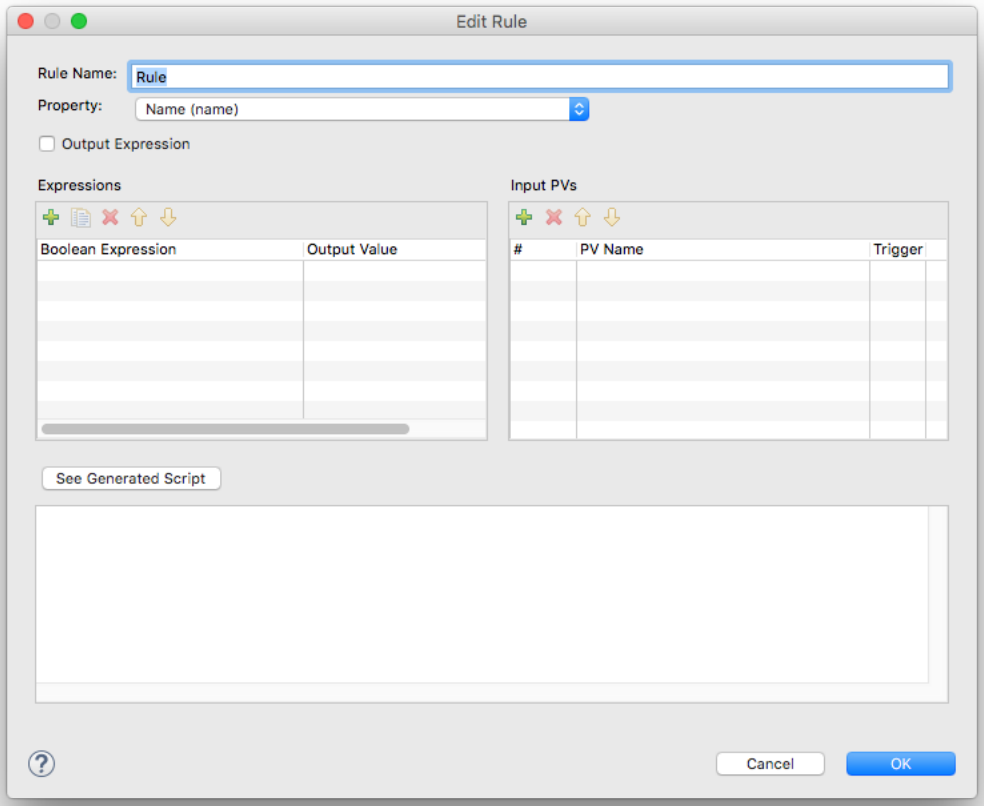
Rules are simpler to create than *Scripts* (page 183), but offer only a subset of dynamic functionality. In fact, under the hood rules are automatically converted to a script.

Assume that want to make an LED square when it is off, and round when it is on. The static properties would not allow for such behaviour, so we add a rule.

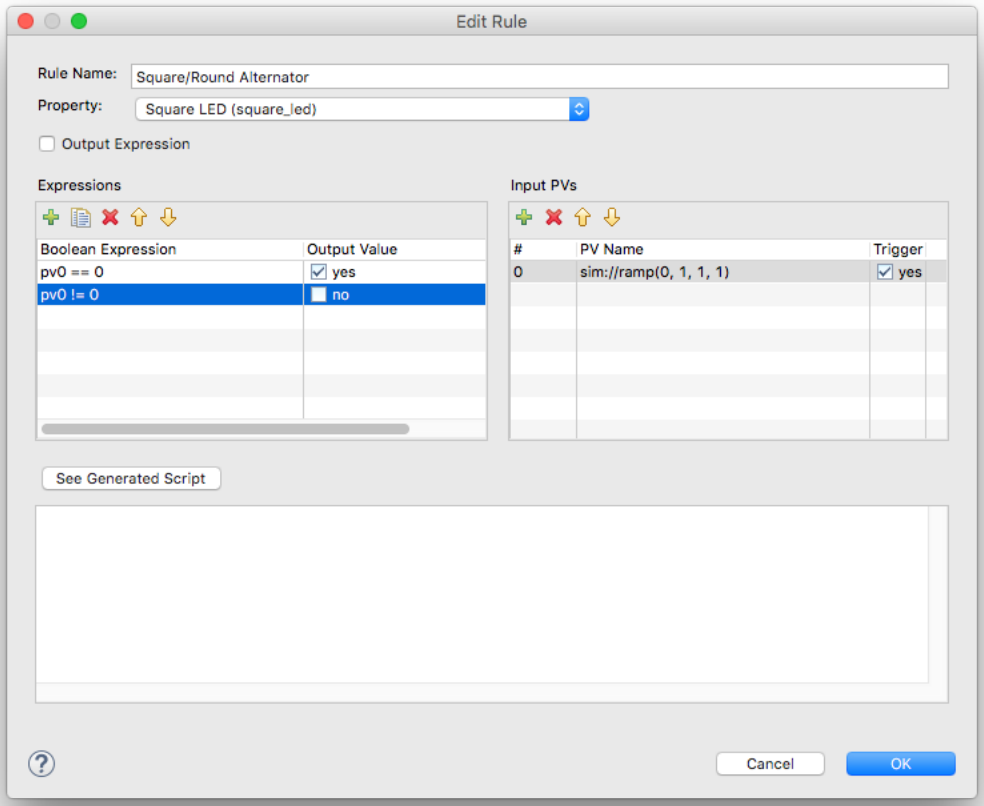
1. Click the **Rules** property fo the LED to pop up this dialog.



2. Click the plus icon, which opens this dialog:



3. First choose the rule's target **Property**. Select **Square LED**.
4. In the right **Input PVs** table add your input PV. In this example we choose to generate an alternating 0/1 value using a *simulated PV* (page 37). 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. In the **Expressions** table, fill in the conditions in the **Boolean Expression** column, and add the matching 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 all dialogs, save your display and refresh a runtime view of it. You should see the LED's shape now alternating between square and round.

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`

The numeric alarm state of an input PV is is accessible as follows:

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

Note: 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.

9. Scripts

Widgets have many properties and methods that can be modified dynamically. *Rules* (page 179) are one way to do that. For more advanced manipulations and control logic, scripts are the right tool.

Scripts can be attached to any widget (or the display itself) and execute only when one of their declared *trigger PVs* gets updated.

Yamcs Studio supports scripts in two different languages:

JavaScript
: Supports ECMAScript 5.1, and a limited set of ECMAScript 6 features.

  The Java implementation that executes JavaScript is called Nashorn. For more on Nashorn, refer to https://github.com/openjdk/nashorn.

Python
: Supports Python 2.7.

  The Java implementation that executes Python scripts is called Jython. For more on Jython, refer to https://www.jython.org.

  Note that Yamcs Studio does *not* make use of the Python distribution available to your system.

Within both scripting languages you have bridged access to the same Java API that allows interacting with its widget, other widgets, PVs or Yamcs Studio itself. Only the syntax looks different.

JavaScript is generally preferred, because it increases the interoperability of your OPI files with the web implementation available on the Yamcs web interface. Python is not at all supported by Yamcs web.

Isolation

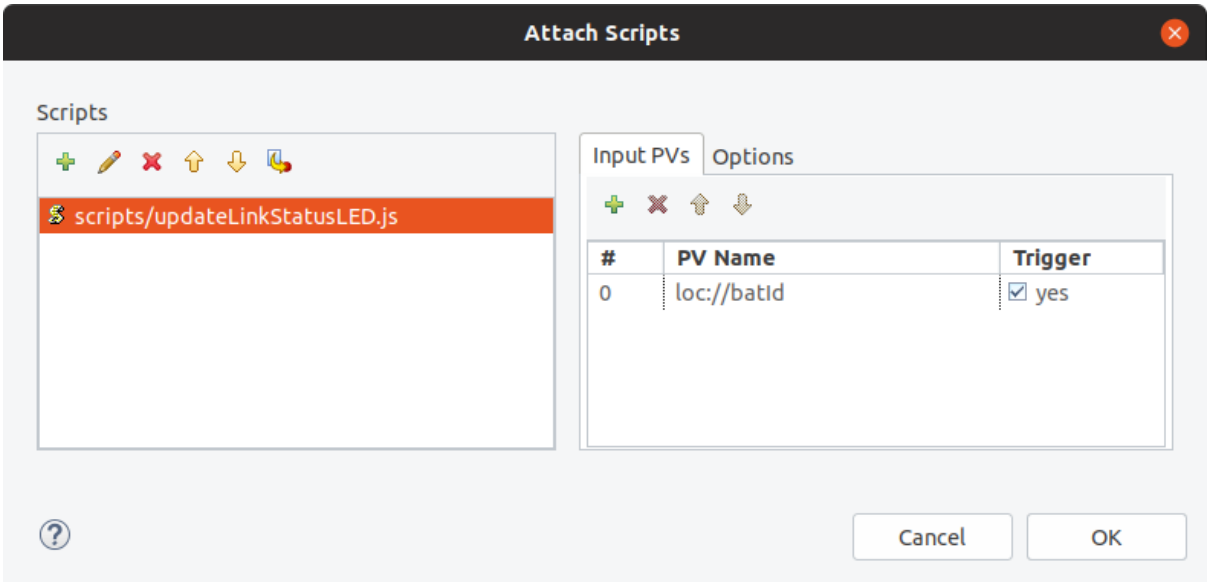
Each script runs with separate scope. It is therefore not possible to access variables from within another script. If you want to share data between scripts, it is possible to do so by creating a local PV and using that as a means for communication.

On the other hand, consecutive executions *do* make use of the same scope, so it is possible to memorize state by using global variables.

9.1 Attach a Script

Scripts can be attached to any widget, or the display itself.

Select the **Scripts** property in the **Properties** view, to open the **Attach Scripts** dialog.



Use this dialog to add or remove scripts. A script can point to a Script file available in the workspace, or alternatively it may be embedded directly within the display.

Input PVs

The panel to the right lists the PVs that must be accessible during script execution. They are available from the global array `pvs` in the same order as they appear in the dialog.

Each script must have at least one Input PV that has the **Trigger** checkbox enabled, otherwise it would never execute.

If you require to execute a script that only runs when the display initializes, use a formula `=1` as your trigger PV.

By default a script executes only when all of its inputs are connected and have a value, and one of its trigger PVs is updated.

Options

The default conditions for execution can be customized in the **Options** tab:

Execute anyway even if some PVs are disconnected.
: Set this flag to run a script even if one of the input PVs is not yet available.

Do not execute the script if error was detected.
: Set this flag to abort further executions of this script as soon as a failure is detected.

9.2 Script API

Global Variables

The following global variables are available in all scripts:

`triggerPV`
: *PV* (page 195) object for the PV that triggered this particular script execution. Sample usage:

```
if (triggerPV === pvs[1]) {
    ConsoleUtil.writeInfo("I was triggered by the second input PV.");
}
```

`pvs`
: Array of *PV* (page 195) objects. One for each Input PV. The order in the array matches with the order in which they appear in the *Attach Scripts* (page 183) dialog.

`widget`
: *Widget* (page 196) object for the widget that this script is attached to.

`display`
: *Widget* (page 196) object for the display that this script's widget belongs to.

Utilities

9.2.1 ColorFontUtil

Utility class for creating font and color objects, required to interact with some widget properties.

The following colors are directly variable:

`BLACK`, `BLUE`, `CYAN`, `DARK_GRAY`, `GRAY`, `GREEN`, `LIGHT_BLUE`, `ORANGE`, `PINK`, `PURPLE`, `RED`, `WHITE`, `YELLOW`.

Other colors can be created using these methods:

getColorFromRGB(red, green, blue)
: Returns a color object for the specified red, green and blue components.

getColorFromHSB(hue, saturation, brightness)
: Returns a color object based on an HSB representation.

- **hue**: value between 0 and 360
- **saturation**: value between 0 and 1
- **brightness**: value between 0 and 1

To create a font object use this method:

getFont(name, height, style)
: Returns a font object.

- **name**: the name of the font
- **height**: font height in points
- **style**: One of:
  - `0` for normal
  - `1` for bold
  - `2` for italic
  - `3` for bold and italic

Example

The following scripts do the same but in different ways:

```
var red = ColorFontUtil.RED;
widget.setPropertyValue("background_color", red);
```

```
var red = ColorFontUtil.getColorFromRGB(255, 0, 0);
widget.setPropertyValue("background_color", red);
```

```
var red = ColorFontUtil.getColorFromHSB(0, 1, 1);
widget.setPropertyValue("background_color", red);
```

9.2.2 ConsoleUtil

Utility class for writing text messages to the **Console** view.

The following methods are available.

writeInfo(message)
: Displays a general message in the Console view.

writeWarning(message)
: Displays a warning message in the Console view.

writeError(message)
: Displays an error message in the Console view.

Example

```
ConsoleUtil.writeError("Something went wrong");
```

9.2.3 DataUtil

Helper methods for creating Java-compatible arguments.

createDoubleArray(size)
: Returns a new Java double array of the given size.

createIntArray(size)
: Returns a new Java int array of the given size.

createMacrosInput(include_parent_macros)
: Create a new `MacrosInput` object, useful when creating a container widget through scripting.

  include_parent_macros is a boolean value that indicates if the MacrosInput should inherit or start blank.

toJavaDoubleArray(array)
: Returns a Java array that matches the provided script array.

toJavaIntArray(array)
: Returns a Java array that matches the provided script array.

Example

The following scripts do the same but in different ways:

```
// Create a Java array, then add to it
var arr = DataUtil.createDoubleArray(100);
for (var i = 0; i < 100; i++) {
    arr[i] = i;
}
pvs[0].setValue(arr);
```

```
// Create a JavaScript array, then convert it to Java
var arr = [];
for (var i = 0; i < 100; i++) {
    arr[i] = i;
}
pvs[0].setValue(DataUtil.toJavaDoubleArray(arr));
```

Note that the same can actually also be achieved by not using `DataUtil`, but instead the `Java` class available to Nashorn scripts:

```
var arr = [];
for (var i = 0; i < 100; i++) {
    arr[i] = i;
}
pvs[0].setValue(Java.to(arr, "double[]"));
```

9.2.4 FileUtil

Helper methods for working with files.

Absolute workspace paths take the form /MyProject/some/file where MyProject is the name of one of the projects in your workspace.

When providing a relative workspace path, you must pass the current widget too (available in all scripts using the global variable `widget`).

readTextFile(path [, widget])
: Returns the content of a file as a string.

  The path can be an absolute path on the local file system, or a relative path inside the workspace. In case of a relative path, the widget argument must be provided as the reference for resolving the path.

writeTextFile(path, inWorkspace [, widget], text, append)
: Writes a string to a file.

  The path can be an absolute path on the local file system, or a relative path inside the workspace. In case of a relative path, the widget argument must be provided as the reference for resolving the path.

  `inWorkspace` must be true if the path should be interpreted as a workspace path. False for a local file system file. This allows to distinguish a workspace path like /MyProject/somefile.txt from an equally valid local file system path.

  Set `append` to true, if the file must be appended to if it already exists.

getInputStreamFromFile(path [, widget])
: Return a `java.io.InputStream` for a file.

  The path can be an absolute path on the local file system, or a relative path inside the workspace. In case of a relative path, the widget argument must be provided as the reference for resolving the path.

  This is an advanced method which provides access to a pure Java object. Use with caution, and close the stream when you're done.

loadXMLFile(path [, widget])
: Return an `org.jdom2.Element` for an XML file.

  The path can be an absolute path on the local file system, or a relative path inside the workspace. In case of a relative path, the widget argument must be provided as the reference for resolving the path.

  This is an advanced method which provides access to a pure Java object. Use with caution.

openFile(path [, widget])
: Open a file with the default editor.

  The path can be an absolute path on the local file system, or a relative path inside the workspace. In case of a relative path, the widget argument must be provided as the reference for resolving the path.

openWebPage(url)
: Open the given URL with a web browser.

playWavFile(path [, widget])
: Play a WAV file.

  The path can be an absolute path on the local file system, or a relative path inside the workspace. In case of a relative path, the widget argument must be provided as the reference for resolving the path.

openFileDialog(inWorkspace)
Open a file selector.

If `inWorkspace` is true, the selector will only allow workspace files to be chosen. If false, the selector allows to choose from the local file system.

This method returns the selected path, or null if the user cancelled.

openFileDialog(startingFolder)
Open a local file system selector starting at the specified folder.

This method returns the selected path, or null if the user cancelled.

saveFileDialog(inWorkspace)
Open a save-file selector.

If `inWorkspace` is true, the selector will only allow workspace files to be chosen. If false, the selector allows to choose from the local file system.

This method returns the selected save path, or null if the user cancelled.

saveFileDialog(startingFolder)
Open a local file system save-file selector.

This method returns the selected save path, or null if the user cancelled.

openDirectoryDialog()
Open a directory selector on the local file system.

This method returns the selected path, or null if the user cancelled.

openDirectoryDialog(startingFolder)
Open a directory selector on the local file system starting at the specified folder.

This method returns the selected path, or null if the user cancelled.

workspacePathToSysPath(workspacePath)
Returns the local file system path for the given workspace path.

Example

```
var targetFile = FileUtil.saveFileDialog(false);
if (targetFile) {
    var text = "file content";
    FileUtil.writeTextFile(targetFile, false, text, false);
}
```

9.2.5 GUIUtil

Helper methods for window-level operations on Yamcs Studio.

fullScreen()
Enter or exit full-screen.

openConfirmDialog(message)
Open a confirmation dialog with the given message.

Returns `true` if the user confirmed.

openInformationDialog(message)
Open an information dialog with the given message.

openWarningDialog(message)
Open a warning dialog with the given message.

openErrorDialog(message)
Open an error dialog with the given message.

openPasswordDialog(message, password)
Open a password input dialog.

Returns `true` if the user entered the expected password.

> **Note:** OPIs are rendered on the client. The use of this method provides only a false sense of security.

Example

```
if (GUIUtil.openConfirmDialog("Are you sure?")) {
    // Do something clever.
}
```

9.2.6 ParameterInfo

ParameterInfo objects are returned from the call `Yamcs.getParameterInfo(pv)`.

New in version 1.7.5.

name
Short name of this parameter (relative to its parent system).

qualifiedName
Fully qualified name of this parameter.

shortDescription
Short description for this parameter.

longDescription
Long description for this parameter.

dataSource
Source of values for this parameter.

units
Units for this parameter.

type
Engineering type of values, as specified by the MDB.

rawType
Raw type of values, as specified by the MDB.

getAlias(namespace)
Returns the alias for this parameter, for a given namespace.

9.2.7 PVUtil

Helper methods for working with PV objects.

createPV(pvName, widget)
Create, start and return a *PV* (page 195) attached to the given widget.

If the widget is destroyed (for example, because the display is closed), the PV goes down with it automatically.

The pvName argument can be any name that can also be used as a PV Name property. For example: `loc://foo`, `/myproject/Voltage`, `=2 + 3`, `sim://noise`, ...

getDouble(pv)
Return a double representation for the current value of the given PV.

getDoubleArray(pv)
Return an array of doubles for the current value of the provided PV.

The array length is 1 if the value is not an array.

getLong(pv)
Return a long representation for the current value of the given PV.

getLongArray(pv)
Return an array of longs for the current value of the provided PV.

The array length is 1 if the value is not an array.

getString(pv)
Return a string representation for the current value of the given PV.

getStringArray(pv)
If pv is an array PV, return its current value as an array of strings, where each entry is a string representation of the actual value.

The array length is 1 if the value is not an array.

getTimeInMilliseconds(pv)
Returns the time for the PV's current value as the number of milliseconds since the UNIX epoch. If the PV is backed by a Yamcs parameter, this would match that parameter's generation time.

getTimeString(pv [, pattern])
Return a formatted string for the timestamp of the PV's current value. If the PV is backed by a Yamcs parameter, this would match that parameter's generation time.

If the **pattern** argument is provided, it must be a string that follows Java conventions. See https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html

getSeverity(pv)
Returns the current severity of the given PV as a number with the following mapping:

- `-1`: UNDEFINED or INVALID
- `0`: NONE
- `1`: MAJOR
- `2`: MINOR

getSeverityString(pv)
Returns the string value of the current severity of the given PV. One of `UNDEFINED`, `INVALID`, `NONE`, `MINOR` or `MAJOR`.

getLabels(pv)
Returns an array with all possible enum values for an enumerated PV.

getStatus(pv)
Returns the status text that might describe the severity specific to the type of PV.

For a Yamcs parameter this can be the value `LOW`, `HIGH`, `LOLO`, `HIHI` or `NONE`.

getSize(pv)
Return the array length of the current PV value. If the current value is not an array, returns `1`.

getUnits(pv)
Returns the units text for a PV value.

writePV(pvName, value [, timeout])
Write a value to a specific PV Name. In the background this will create a temporary PV object, write to it, and finally stop that PV.

A custom timeout in seconds may be provided. The default is 10 seconds.

This method has no return value. It returns immediately and does not wait for the write (or timeout) to occur.

Note: If you already have a connected *PV* (page 195) instance `x`, it is simpler to call `x.setValue(value)` on it.

Example

Create a PV, listen to it, then write to it.

```
var IPVListener = Java.type("org.yamcs.studio.data.IPVListener");

var myPV = PVUtil.createPV("loc://test", widget);
myPV.addListener(new IPVListener({
    valueChanged: function(pv) {
        ConsoleUtil.writeInfo("Write successful: " + PVUtil.getString(pv));
    }
}));

ConsoleUtil.writeInfo("Writing...");
myPV.setValue(123);
```

Set a widget property based on a PV's severity.

```
var severity = PVUtil.getSeverityString(pvs[0]);
var color;
switch (severity) {
    case "NONE":
        color = ColorFontUtil.GREEN;
        break;
    case "MAJOR":
        color = ColorFontUtil.RED;
        break;
    case "MINOR":
        color = ColorFontUtil.ORANGE;
        break;
    default:
        color = ColorFontUtil.PINK;
}
widget.setPropertyValue("foreground_color", color);
```

9.2.8 ScriptUtil

The following methods are available.

openOPI(widget, path, target [, macros])

Open an OPI specified by an absolute or relative workspace path.

Relative paths are resolved in relation to the display file of the provided widget object.

`target` can take the following values:

- `0`: New tab
- `1`: Replace current OPI.
- `2`: New window
- `7`: Detached view
- `8`: New shell

Custom `macros` can be provided to the new OPI. Use *DataUtil.createMacrosInput(false)* (page 186)

closeCurrentOPI()
Close the currently active OPI display.

closeAssociatedOPI(widget)
Close the OPI display that hosts the given widget.

executeSystemCommand(command, timeout)
Run a local system command.

The timeout argument indicates the maximum number of seconds that the command is allowed to execute.

This method returns immediately. Any stdout or stderr is sent to the **Console** view.

execInUI(runnable, widget)
Run some logic on the UI thread.

`runnable` must be a `java.lang.Runnable` implementation.

This is an advanced method. Use with caution.

executeEclipseCommand(commandId [, parameters])
Execute an eclipse command by specifying its identifier, and optionally a String array with parameters.

This is an advanced method. Available command IDs are not documented, and need to be reverse engineered from source code.

Example

```
ScriptUtil.openOPI(widget, "foo.opi", 0, null);
```

```
var macros = DataUtil.createMacrosInput(false);
macros.put("foo", "abc");
macros.put("bar", "def");
ScriptUtil.openOPI(widget, "foo.opi", 0, macros);
```

9.2.9 WidgetUtil

Helper methods for creating widgets at runtime.

createWidgetModel(type)
Returns the model for a new widget.

Types

The following is a list of types for all available widgets:

Widget	Type
Action Button	org.csstudio.opibuilder.widgets.ActionButton
Arc	org.csstudio.opibuilder.widgets.arc
Array	org.csstudio.opibuilder.widgets.array
Boolean Button	org.csstudio.opibuilder.widgets.BoolButton
Boolean Switch	org.csstudio.opibuilder.widgets.BoolSwitch
Byte Monitor	org.csstudio.opibuilder.widgets.bytemonitor

continues on next page

continued from previous page

Widget	Type
Check Box	org.csstudio.opibuilder.widgets.checkbox
Choice Button	org.csstudio.opibuilder.widgets.ChoiceButton
Combo	org.csstudio.opibuilder.widgets.combo
Ellipse	org.csstudio.opibuilder.widgets.Ellipse
Gauge	org.csstudio.opibuilder.widgets.gauge
Grid Layout	org.csstudio.opibuilder.widgets.gridLayout
Grouping Container	org.csstudio.opibuilder.widgets.groupingContainer
Image	org.csstudio.opibuilder.widgets.Image
Image Boolean Button	org.csstudio.opibuilder.widgets.ImageBoolButton
Image Boolean Indicator	org.csstudio.opibuilder.widgets.ImageBoolIndicator
Intensity Graph	org.csstudio.opibuilder.widgets.intensityGraph
Knob	org.csstudio.opibuilder.widgets.knob
Label	org.csstudio.opibuilder.widgets.Label
LED	org.csstudio.opibuilder.widgets.LED
Linking Container	org.csstudio.opibuilder.widgets.linkingContainer
Menu Button	org.csstudio.opibuilder.widgets.MenuButton
Meter	org.csstudio.opibuilder.widgets.meter
Polygon	org.csstudio.opibuilder.widgets.polygon
Polyline	org.csstudio.opibuilder.widgets.polyline
Progress Bar	org.csstudio.opibuilder.widgets.progressbar
Radio Box	org.csstudio.opibuilder.widgets.radioBox
Rectangle	org.csstudio.opibuilder.widgets.Rectangle
Rounded Rectangle	org.csstudio.opibuilder.widgets.RoundedRectangle
Sash Container	org.csstudio.opibuilder.widgets.sashContainer
Scaled Slider	org.csstudio.opibuilder.widgets.scaledslider
Scrollbar	org.csstudio.opibuilder.widgets.scrollbar
Spinner	org.csstudio.opibuilder.widgets.spinner
Tabbed Container	org.csstudio.opibuilder.widgets.tab
Table	org.csstudio.opibuilder.widgets.table
Tank	org.csstudio.opibuilder.widgets.tank
Text Input	org.csstudio.opibuilder.widgets.TextInput
Text Update	org.csstudio.opibuilder.widgets.TextUpdate
Thermometer	org.csstudio.opibuilder.widgets.thermometer
Thumb Wheel	org.csstudio.opibuilder.widgets.ThumbWheel
Web Browser	org.csstudio.opibuilder.widgets.webbrowser

continues on next page

continued from previous page

Widget	Type
XY Graph	org.csstudio.opibuilder.widgets.xyGraph

Example

```
var opiFile = triggerPV.getValue();

var linkingContainer = WidgetUtil.createWidgetModel(
    "org.csstudio.opibuilder.widgets.linkingContainer");
linkingContainer.setPropertyValue("opi_file", opiFile);

// 1 = Size the container to fit the linked OPI
linkingContainer.setPropertyValue("resize_behaviour", 1);
// 1 = Line Style
linkingContainer.setPropertyValue("border_style", 1);

widget.removeAllChildren();
widget.addChild(linkingContainer);
widget.performAutosize();
```

9.2.10 Yamcs

The following methods are available.

issueCommand(commandName, args)
Issue a telecommand on the currently connected Yamcs processor.

runCommandStack(path [, widget])
Runs all commands in a Yamcs Command Stack (`*.ycs`).

The path can be an absolute path on the local file system, or a relative path inside the workspace. In case of a relative path, the widget argument must be provided as the reference for resolving the path.

getMonitoringResult(pv)
Returns the *monitoring result* of a Yamcs Parameter PV. One of `IN_LIMITS`, `WATCH`, `WARNING`, `DISTRESS`, `CRITICAL` or `SEVERE`.

Returns null when no monitoring check was performed.

getAcquisitionStatus(pv)
Returns the *acquisition status* of a Yamcs Parameter PV. One of `ACQUIRED`, `NOT_RECEIVED`, `INVALID` or `EXPIRED`.

New in version 1.7.5.

getGenerationTime(pv)
Returns the *generation time* of a Yamcs Parameter PV in ISO-8601 format.

New in version 1.7.5.

getReceptionTime(pv)
Returns the *reception time* of a Yamcs Parameter PV in ISO-8601 format.

New in version 1.7.5.

getParameterInfo(pv)
Returns a *ParameterInfo* (page 189) object containing the MDB info for a Yamcs Parameter PV.

New in version 1.7.5.

Example

```
Yamcs.issueCommand('/YSS/SIMULATOR/SWITCH_VOLTAGE_ON', {
    voltage_num: 1
});
```

```
Yamcs.runCommandStack('/My Project/stacks/example.ycs');
```

Support Classes

9.2.11 PV

PV instances belong to a single widget. The PV Name identifies what data it represents, and may come from various different data sources.

getName()
: Returns the PV Name.

getValue()
: Returns the current value of this PV.

setValue(value)
: Write a value to this PV. This method returns immediately.

setValue(value, timeout)
: Write a value to this PV and block until either the write was successful, or the timeout (in milliseconds) was reached.

  Returns true if the write was successful.

start()
: Connect and start listening to updates. It is not allowed to start an already started PV.

stop()
: Disconnect a PV. Its listeners are preserved for a potential future restart.

addListener(listener)
: Add an IPVListener implementation for receiving updates. Example:

```
var IPVListener = Java.type("org.yamcs.studio.data.IPVListener");
myPV.addListener(new IPVListener({
    valueChanged: function(pv) {
        ConsoleUtil.writeInfo("An update: " + PVUtil.getString(pv));
    }
}));
```

  Note: In most circumstances it would be preferrable to add the PV of interest as a trigger to the Input PVs of a particular script, so that value updates can more simply be accessed from the global `pvs` array.

removeListener(listener)
: Stop an IPVListener implementation from receiving updates.

isConnected()
: Returns true if this PV is currently *connected*. What this means depends on the underlying datasource. For example, if it concerns a Yamcs parameter, *connected* means that the WebSocket subscription is up and running.

isWriteAllowed()
: Returns true if this PV may be used in control widgets.

  PVs for Yamcs parameters accept all writes, and leave the finally decision to Yamcs itself.

9.2.12 Widget

When a script is executed it has access to a `widget` variable which allows for interaction with the widget that the script is attached to.

Other widgets can also be retrieved using the `display` variable:

```
otherWidget = display.getWidget("someOtherWidgetName");
```

All Widgets

The following methods are common to all widgets:

executeAction(index)
Run one of the widget's actions by specifying its index (zero-based).

getPropertyValue(name)
Retrieve the value for one of the widget's properties.

setPropertyValue(name, value [, force])
Update a widget property to a new value.

If force is true, the update occurs even if the new value equals the current value.

getPVByName(pvName)
Retrieve one of the widget's PVs by name. This includes PVs used for both rules and scripts.

Returns a *PV* (page 195) object, or null if no PV with this name is known to the widget.

setVar(name, value)
Save any custom data for later retrieval by name.

getVar(name)
Retrieve custom data by name.

getMacroValue(name)
Retrieve a macro string value by name.

getName()
Returns the widget's name.

Shortcut for `getPropertyValue("name")`.

setX(x)
Update the X coordinate of the widget area.

Shortcut for `setPropertyValue("x", x)`.

setY(y)
Update the Y coordinate of the widget area.

Shortcut for `setPropertyValue("y", y)`.

setWidth(width)
Update the width of the widget area.

Shortcut for `setPropertyValue("width", width)`.

setHeight(height)
Update the height of the widget area.

Shortcut for `setPropertyValue("height", height)`.

setEnabled(enabled)
Control whether the widget is enabled.

setVisible(visible)
Update the visiblity of this widget.

Shortcut for `setPropertyValue("visible", visible)`.

getValue()
Return the value of the *widget*.

A widget can have a value without having an attached PV. If it does have an attached PV, the value of the widget is almost always identical to that of the PV.

setValue(value)
Manually set the value of the *widget*.

This does *not* update any PV. It updates only the displayed value. If the widget is also following updates from a PV, the value may well be overwritten whenever that PV updates.

This method should be called on the UI thread. If unsure, use the otherwise identical method `setValueInUIThread`.

setValueInUIThread(value)
Same as `setValue`, but forces a switch to the UI thread.

PV Widgets

Widgets that have the **PV Name** property can read or write a PV. They have the following additional methods:

getPV([propertyName])
Return the *PV* (page 195) object for a specific widget property.

If `propertyName` is not specified, it defaults to `pv_name`, which is the name of the main PV property **PV Name**.

getPVName()
Returns a string with the main **PV Name** for this widget.

getAllPVNames()
Return a string array with all PV Names that are used in any of the widget's PV-like properties.

getPVValue(propertyName)
Return the current value for a specific PV property.

setPVValue(propertyName, value)
Write a new value to the PV used by the given property.

Container Widgets

Container widgets are those that contain other widgets:

- *Array* (page 45)
- *Display* (page 62)
- *Grouping Container* (page 72)
- *Linking Container* (page 97)
- *Sash Container* (page 125)
- *Tabbed Container* (page 137)

They have the following additional methods:

getWidget(name)
Get a descendant widget of this container by name.

getChild(name)
Get a direct child widget of this container by name.

getChildren()
Returns all direct child widgets, in order.

addChild(widgetModel)
Add a child widget to this container.

`widgetModel` is an object that can be obtained using *WidgetUtil.createWidgetModel* (page 192)

addChildToRight(widgetModel)
Add a child widget to this container, while adjusting its X coordinate such that it is added to the right of other child widgets.

`widgetModel` is an object that can be obtained using *WidgetUtil.createWidgetModel* (page 192)

addChildToBottom(widgetModel)
Add a child widget to this container, while adjusting its Y coordinate such that it is added below other child widgets.

`widgetModel` is an object that can be obtained using *WidgetUtil.createWidgetModel* (page 192)

removeChild(widget)
Remove the given child widget from this container.

removeChildByName(name)
Remove the direct child widget of this container by name.

removeAllChildren()
Remove all child widgets.

performAutosize()
Adjust the container size to fit its child widgets.

getValue()
Returns an array with all of its children's values (unless `setValue` was used to set another type of value).

setValue(value)
If the given value is an array of length equal to the number of child widgets, the values are written respectively to those widgets.

Otherwise, the value as a whole is written to each child.

Examples

Update a widget's value *without* using the **PV Name** property:

```
var v = PVUtil.getString(pvs[0]);
if (v == "DISABLED") {
    widget.setValue(1.0);
} else if (v == "OK") {
    widget.setValue(2.0);
} else {
    widget.setValue(3.0);
}

widget.setPropertyValue("tooltip", v);
```

Print the name and type of a container's children:

```
var children = widget.getChildren();
for (var i = 0; i < children.length; i++) {
    var child = children[i];
    var name = child.getPropertyValue("name");
    var type = child.getPropertyValue("widget_type");
```

(continues on next page)

(continued from previous page)

```
    ConsoleUtil.writeInfo("Widget: " + name + ", type: " + type);
}
```

9.3 Accessing Java

In addition to the common *Script API* (page 184) utilities, Yamcs Studio allows bridged access within scripts to entire Java packages.

Yamcs Studio uses the Eclipse RCP framework, so for example, here we make use of JFace to open a message dialog:

```
var MessageDialog = Java.type("org.eclipse.jface.dialogs.MessageDialog");
MessageDialog.openInformation(
    null, "Attention", "I was triggered by a script");
```

If you go in this direction, at some point it may make more sense to extend or fork Yamcs Studio at a Java level, rather than using display scripting.

Note further that such scripts are not compatible when rendering your OPI displays via the Yamcs web interface.

Full documentation of Java and Eclipse APIs is well outside of scope of this document, so we leave this topic with a complex example that uses Eclipse SWT and Java I/O. The script logic is as follows:

1. Download a random image from Internet.
2. Save this image to a file within the workspace.
3. Create or reuse a standalone window to display the image.

```
var BufferedInputStream = Java.type("java.io.BufferedInputStream");
var Display = Java.type("org.eclipse.swt.widgets.Display");
var FileOutputStream = Java.type("java.io.FileOutputStream");
var File = Java.type("java.io.File");
var Image = Java.type("org.eclipse.swt.graphics.Image");
var ImageData = Java.type("org.eclipse.swt.graphics.ImageData");
var Label = Java.type("org.eclipse.swt.widgets.Label");
var Shell = Java.type("org.eclipse.swt.widgets.Shell");
var SWT = Java.type("org.eclipse.swt.SWT");
var URL = Java.type("java.net.URL");

function downloadNewImage() {
  var cacheDir = FileUtil.workspacePathToSysPath("/") + "/images_cache/";
  new File(cacheDir).mkdir();
  var filename = cacheDir + "cache" + Date.now() + ".jpg";

  var urlIn = new BufferedInputStream(
    new URL("https://source.unsplash.com/random").openStream()
  );
  var fileOut = new FileOutputStream(filename);
  var ByteArray = Java.type("byte[]");
  var buf = new ByteArray(1024);
  var n;
  while ((n = urlIn.read(buf, 0, 1024)) != -1) {
    fileOut.write(buf, 0, n);
  }
  urlIn.close();
  fileOut.close();

  return filename;
}

function getShell(display, origin) {
  var shells = display.getShells();
  for (var i = 0; i < shells.length; i++) {
    if (shells[i].getData("origin") == origin) {
```

(continues on next page)

(continued from previous page)

```
      return shells[i];
    }
  }
  var newShell = new Shell(display);
  newShell.setData("origin", origin);
  newShell.setData("label", new Label(newShell, SWT.BORDER));
  return newShell;
}

var filename = downloadNewImage();
var display = Display.getCurrent();
var imgData = new ImageData(filename);
var image = new Image(display, imgData);
var shell = getShell(display, "test-script-shell");
var clientArea = shell.getClientArea();
var label = shell.getData("label");
label.setLocation(clientArea.x, clientArea.y);
label.setImage(image);
label.pack();
shell.pack();
shell.open();
```

10. Macros

Macros are placeholders that can be used in string-based property values. They are substituted at runtime, and before the property gets interpreted.

Macros use the syntax `${mymacro}` or `$(mymacro)`.

Predefined Macros

The following macros are predefined with special meaning:

`$(DID)`
: Unique identifier for a display instance.

  For example, if a display uses a *local PV* (page 35) called `loc://$(DID)_foo`, each instance of that display will use a different local variable, without interfering with potential other instances of that display.

`$(DNAME)`
: The name of the display.

`$(LCID)`
: Unique identifier for a *Linking Container* (page 97) instance. If a display contains multiple Linking Container instances referring to the same display, the `$(LCID)` macro will evaluate to a different unique value in each.

Property Macros

In the context of a widget, all of its properties can be accessed in other properties using property identifiers as the macro name. For example: `$(pv_name)`, `$(border_width)`, ...

Custom Macros

You can define custom macros at different locations:

1. In user preferences under `OPI Runtime`.
2. In the configuration of an **Open OPI** widget action (or the equivalent **ScriptUtil.openOPI** script utility).
3. In the **Macros** property of a display.
4. In the **Macros** property of a container widget.

Macros are evaluated for each widget. If a macro with the same name is defined at multiple levels, they overwrite each other in the above order. For example: (4) wins from (1).

If the option **Include macros from parent** is unticked, macros from higher levels are not inherited.

Example

The **Tooltip** property of all widgets has the following default value:

```
$(pv_name)
$(pv_value)
```

When the display is running, the tooltip of widgets will show the value of the **PV Name** property, as well as its current value.

11. Tuning

11.1 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

11.2 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
```

11.3 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`