
Testmanager 1.2.6

User Manual

Florian Pose, fp@igh.de
Ingenieurgemeinschaft IgH

Essen, 19th October 2026
Revision fdaaa972

Contents

1. Introduction	1
1.1. Main Window	2
2. Installation	5
2.1. Linux	5
2.2. Windows	6
2.3. Building from source	7
2.4. Command line	7
3. Data Sources	9
3.1. Adding a data source	9
3.2. Observing the connection	11
3.3. Authentication	12
3.4. Browsing a data source	12
4. Layouts and Edit Mode	15
4.1. Layout files	15
4.2. Edit mode	16
4.3. Property editor	18
4.4. Tabs	18
5. Widgets	21
5.1. Dial	22
5.2. Table view	23
5.3. Text	23
6. Connecting Signals and Parameters	25
6.1. Drag and drop	25
6.2. Process data connections	25
6.3. Selecting vector elements	27
6.4. Data slots dock	28
7. Legend	29

8. Stylesheets	31
8.1. Levels	31
8.2. Stylesheet editor	31
8.3. Selectors	33
8.4. Resources	33
8.5. Dynamic properties	33
9. Python Scripts	35
9.1. Script editor	35
9.2. Combining scripts and stylesheets	36
9.3. Error messages	37
9.4. Python shell	37
10. Example Layout	39
10.1. Running the example	39
10.2. What the example shows	39
11. Parameter Sets	43
11.1. Creating parameter sets	43
11.2. Exporting	43
11.3. Importing	45
11.4. Parameter set widget	45
12. Broadcasts	47
13. Messages	49
13.1. Configuration	49
13.2. Displaying messages	49
14. Replacing Data Sources	51
A. Keyboard Shortcuts	53
B. Python API	55
B.1. Script variables	55
B.2. The <code>widget</code> object	55
B.3. The <code>tm</code> module	55

List of Figures

1.1. Testmanager main window, showing the example layout connected to the PdServ example process	1
1.2. Main window in edit mode	2
1.3. Status bar with network throughput	3
2.1. Download button on GitLab	6
2.2. Extracted ZIP archive with testmanager.exe	6
2.3. About dialog	7
3.1. Connect dialog	10
3.2. Connect dialog with TLS options	10
3.3. Data source tooltip	11
3.4. Context menu of a data source	11
3.5. Login dialog	12
3.6. Data source tree	13
3.7. Filtered data source tree	13
3.8. Context menu of a variable	14
4.1. Save changes?	15
4.2. Layout file format update	15
4.3. Tab in edit mode	16
4.4. Context menu of the canvas	17
4.5. Context menu of a widget	17
4.6. Marked unconnected widgets	18
4.7. Property editor	19
4.8. Context menu of a tab header	19
4.9. Edit tab dialog	19
4.10. Detached tab	20
5.1. All widgets at a glance	21
5.2. Dial configuration dialog	22
5.3. Table view configuration dialog	23
5.4. Text configuration dialog	23
6.1. Connecting a variable via drag and drop	25

6.2. Tooltip with connected variables	26
6.3. Process data connections dialog	26
6.4. Applying a value to all connections	27
6.5. Variable selector	28
6.6. Data slots dock	28
7.1. Graph with legend	29
7.2. Context menu of the legend	29
8.1. The same widget without and with stylesheet	31
8.2. Stylesheet editor	32
8.3. Stylesheet templates	32
8.4. Inserting a color	32
8.5. Inserting a resource	33
9.1. Python script editor	35
9.2. Script templates	36
9.3. Highlighting via script and stylesheet	36
9.4. Script error	37
9.5. Python shell	37
10.1. The example layout, connected to pdserv-example-st	40
11.1. Parameter sets dock	43
11.2. Adding a folder via drag and drop	44
11.3. Context menu of a parameter set	44
11.4. Saving a parameter file	44
11.5. Loading a parameter file	45
11.6. Showing differences only	46
11.7. Multi-dimensional parameter	46
11.8. Parameter set widget	46
12.1. SendBroadcast widget	47
12.2. Received broadcasts	47
13.1. Configure messages	49
13.2. Messages of a data source	50
13.3. Messages dock	50
14.1. Replace data source	51

1. Introduction

Testmanager is a graphical application for visualising and changing the signals and parameters of [EtherLab](#) control processes. It connects to one or more processes via the MSR protocol, using the [PdCom](#) library on the client side and [PdServ](#) on the process side.

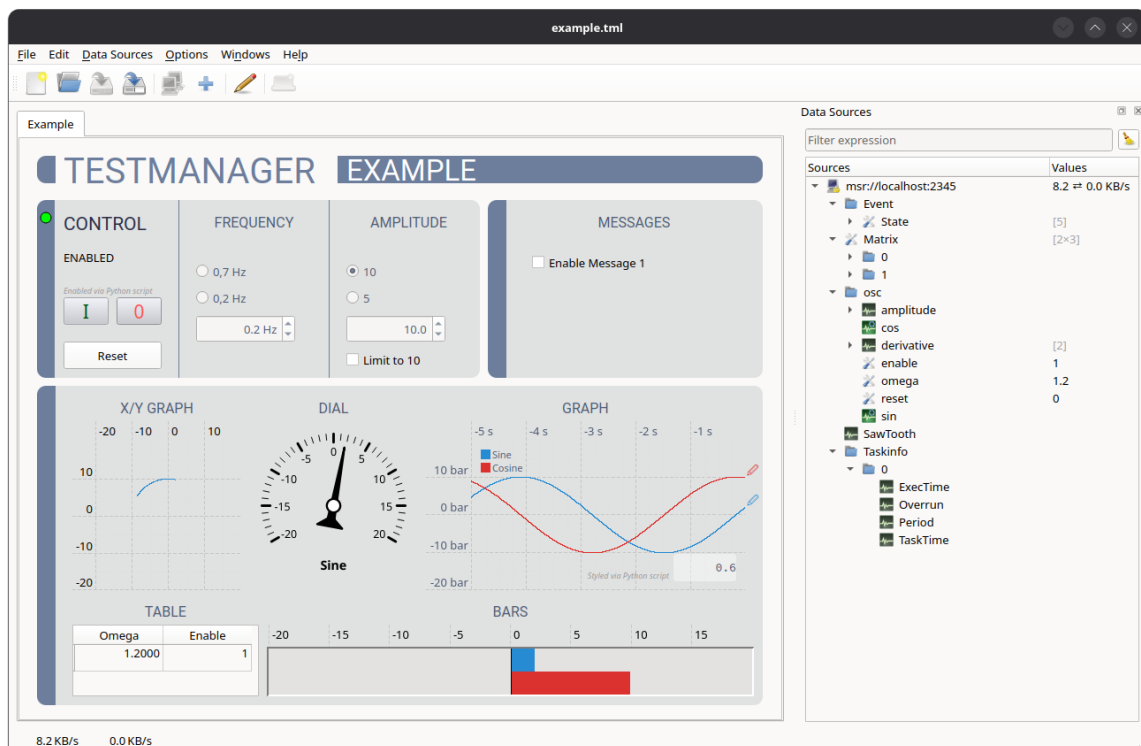


Figure 1.1.: Testmanager main window, showing the example layout connected to the PdServ example process

The signals and parameters of the connected processes are listed in a tree. They can be linked to operating and visualisation widgets (bar graphs, dials, LEDs, time graphs, buttons, input fields, ...) simply by dragging them onto the widgets. In this way, an operator interface is created without programming. The arrangement of the widgets is stored in a *layout file*.

1. Introduction

Further features include:

- Arbitrary number of tabs, which can be detached as separate windows.
- Stylesheets on global, tab and widget level.
- Python scripts that react on process values.
- Parameter sets, which can be saved, compared and restored.
- Display of process messages and broadcasts.
- Encrypted (TLS) and authenticated connections.

💡 Tip

Many topics of this manual are also covered by short videos.

1.1. Main Window

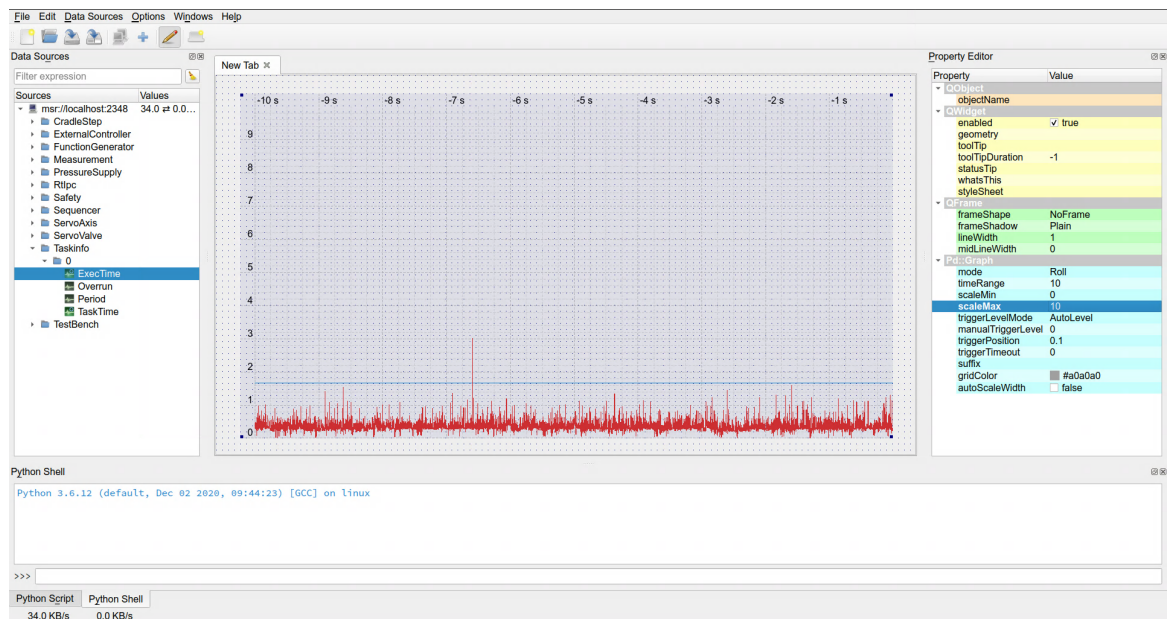


Figure 1.2.: Main window in edit mode

The main window consists of the following areas:

- **Menu and tool bar** at the top, with actions for layout files, adding tabs and toggling the edit mode.
- **Tabs** in the centre. Each tab is a canvas that holds the widgets.
- **Docks** around the tabs, which can be moved, stacked on top of each other (as in the bottom area of the figure) or detached:

- *Data Sources*: tree of the connected processes and their variables,
 - *Property Editor*: properties of the selected widget,
 - *Data Slots*: process data connections of the selected widgets,
 - *Python Script* and *Python Shell*,
 - *Parameter Sets*,
 - *Broadcasts* and *Messages*.
- **Status bar** at the bottom.

The **Windows** menu shows or hides the individual docks and the tool bar.

The status bar shows the total network throughput of all data sources, received data on the left and sent data on the right. The throughput of a single data source is displayed in the *Values* column of the data source tree.

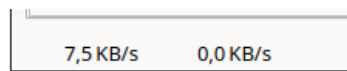


Figure 1.3.: Status bar with network throughput

2. Installation

2.1. Linux

Packages for several distributions are provided via the [openSUSE Build Service](#). They pull in the required libraries PdCom, QtPdCom and QtPdWidgets automatically.

The package also registers the MIME type for layout files (*.tml) and a desktop entry, so layout files can be opened by double-clicking them in a file manager.

2.1.1. openSUSE (RPM-based)

Add the EtherLab RPM repository:

```
sudo zypper ar \
  https://download.opensuse.org/repositories/science:/EtherLab/15.6 \
  etherlab
```

You are now asked, if you trust the repository key. Then install Testmanager and the PdServ example (later used for demonstration):

```
sudo zypper in pdserv-example testmanager-ng
```

2.1.2. Debian (deb-based)

Add the EtherLab repository to sources.list:

```
export KEYRING=/usr/share/keyrings/etherlab.gpg
export REPO=https://download.opensuse.org/repositories/science:/EtherLab/Debian_13
curl -fsSL "$REPO/Release.key" | gpg --dearmor | sudo tee "$KEYRING" >/dev/null
echo "deb [signed-by=$KEYRING] $REPO/ ./" \
  | sudo tee /etc/apt/sources.list.d/etherlab.list > /dev/null
sudo apt-get update
```

Then install Testmanager and the PdServ example (later used for demonstration):

```
sudo apt-get install libpdserv3-example testmanager-ng
```

2.2. Windows

For Windows, Testmanager is provided as a ZIP archive, which can be downloaded from GitLab.

Therefore go to [GitLab](#) and click on the blue "Download for Windows" button:

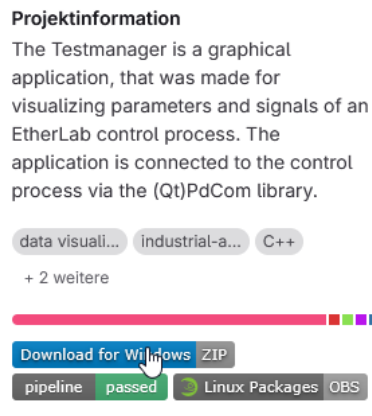


Figure 2.1.: Download button on GitLab

There is no installer. Instead you can just extract the downloaded archive and start the binary from the extracted folder:

1. Download the [ZIP archive](#).
2. Extract the complete archive to a local directory.
3. Start `testmanager.exe` from the extracted directory.

! Important

Do not start `testmanager.exe` directly from within the ZIP archive. Testmanager needs the other files of the archive (libraries, bundled Python), which are only available after extracting.

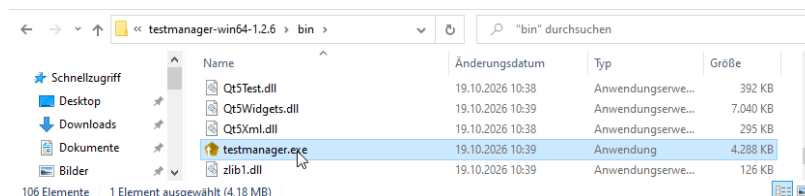


Figure 2.2.: Extracted ZIP archive with testmanager.exe

You can the for example create a shortcut by dragging the binary (`testmanager.exe`) to your desktop with the right mouse button.

Note

If Testmanager fails to start, an error message is displayed. Please include its text, together with the version from **Help → About Testmanager**, in any support request.

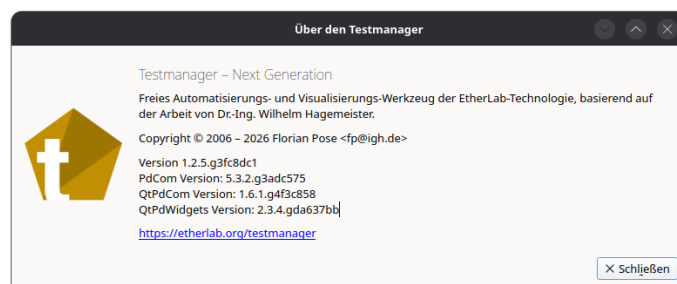


Figure 2.3.: About dialog

2.3. Building from source

See the [README.md](#) file in the source tree for build instructions.

2.4. Command line

`testmanager` [OPTIONS] [LAYOUT]

Option	Description
LAYOUT	Layout file (*.tml) to load on startup.
-c, --connect	Connect immediately and try to stay connected.
-n, --new	Start with an empty layout.
-h, --help	Print usage information.

3. Data Sources

A *data source* is a control process that provides signals and parameters via the network. In order to connect to it, it needs to offer a protocol that is supported by the PdCom library (for example the MSR protocol). A typical data source is a real-time application built with PdServ, for example a Simulink model generated with EtherLab or a C/C++ application using the PdServ library.

Tip

If PdServ is installed, the command `pdserv-example-st` starts a small example process that can be used to try out all features described in this manual. See the [example chapter](#) for a complete walk-through.

3.1. Adding a data source

Select **Data Sources** → **Add data source...** or right-click into the data source tree and select **Add datasource...**. Enter the URL of the process:

URL scheme	Default port	Description
<code>msr://</code>	2345	Unencrypted connection
<code>msrs://</code>	4523	Connection encrypted via TLS

For example, `msr://localhost` connects to a process on the local machine. Process variables are referenced by URLs of the form `msr://host:port/path/to/variable`.

For encrypted connections, the server certificate can either be ignored or verified against a custom CA certificate. These settings are stored per host and port.

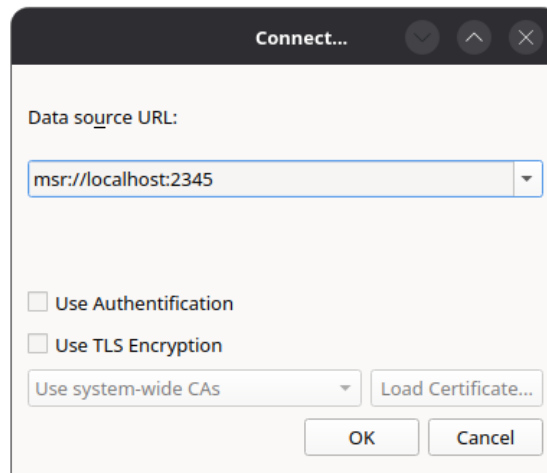


Figure 3.1.: Connect dialog

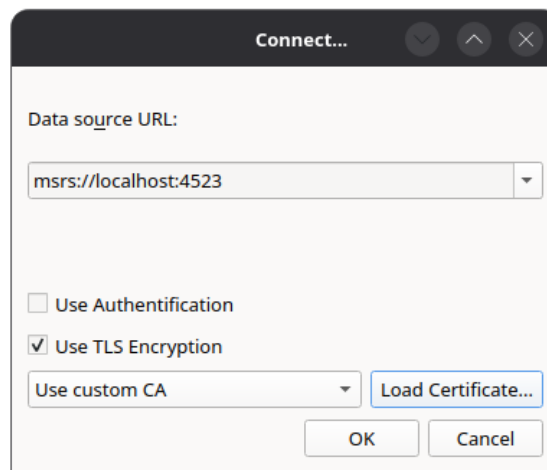


Figure 3.2.: Connect dialog with TLS options

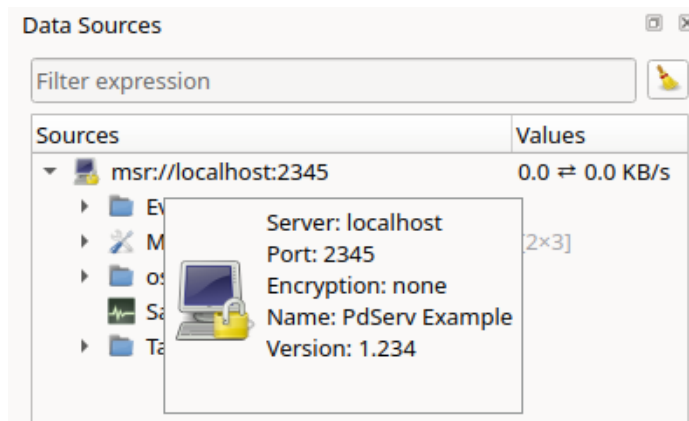


Figure 3.3.: Data source tooltip

3.2. Observing the connection

The context menu of a data source offers the following actions: **Connect**, **Disconnect**, **Login**, **Logout**, **Remove**, **Replace...**, **Show messages...** and **Configure messages...**

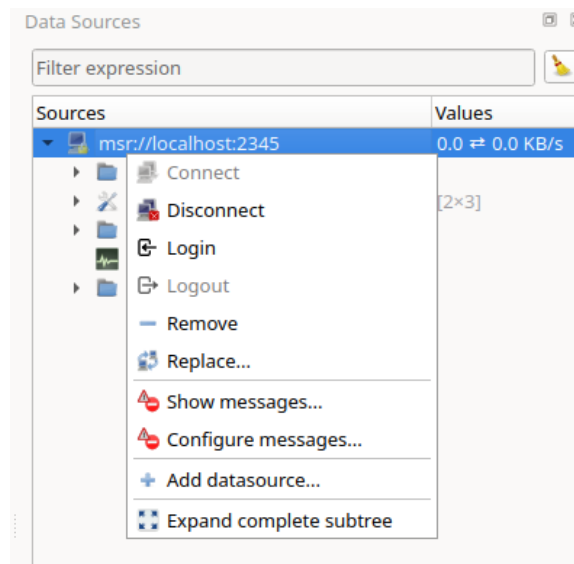


Figure 3.4.: Context menu of a data source

Data Sources → **Connect to data sources** connects all data sources referenced in the current layout.

3.3. Authentication

If the process requires authentication, select **Login** from the context menu of the data source and enter user name and password.

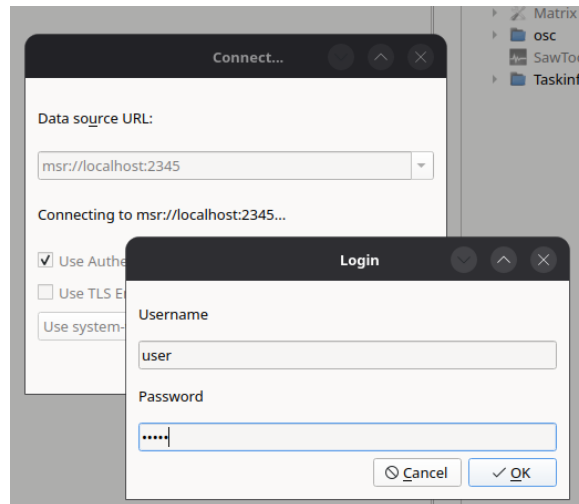


Figure 3.5.: Login dialog

3.4. Browsing a data source

After connecting, the *Data Sources* dock shows a tree of all signals and parameters of the process.

- *Signals* are values calculated by the process. They can only be read.
- *Parameters* can be read and written.

The icon in front of a variable shows its kind:

Icon	Meaning
	Signal, currently not subscribed
	Signal, subscribed periodically
	Signal, subscribed event-based
	Parameter

Each variable has a *sample time*. The value 0 means that the variable is not sampled

Sources	Values
msr://localhost:2345	0.0 ⇌ 0.0 KB/s
Event	
Matrix	[2×3]
0	
0	0
1	1
2	2
1	
osc	
amplitude	
Limit	20
Modulation	
Setpoint	10
cos	
derivative	[2]
0	

Figure 3.6.: Data source tree

periodically, but transmitted *event-based*, i.e. only when its value changes. Parameters are always event-based.

! Important

Within one connection to a process, a signal can be transmitted either event-based (period 0) *or* periodically (period > 0). Therefore use either event-based or periodic transmission consistently for a signal.

A filter expression above the tree limits the displayed variables.

Data Sources	Values
Lim	
msr://localhost:2345	0.0 ⇌ 0.0 KB/s
osc	
amplitude	
Limit	20

Figure 3.7.: Filtered data source tree

The context menu of a variable offers:

- **Copy variable URL:** Copies the URL to the clipboard.
- **Select subscribers:** Selects all widgets that are connected to the variable.
- **Expand complete subtree.**

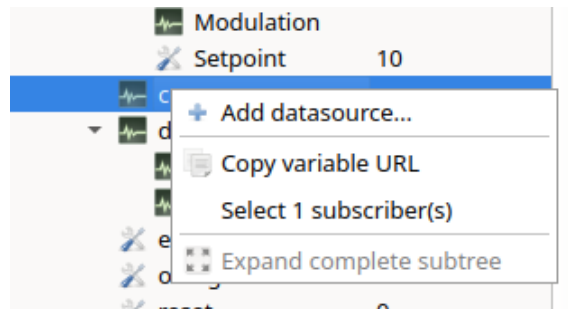


Figure 3.8.: Context menu of a variable

4. Layouts and Edit Mode

4.1. Layout files

The arrangement of tabs and widgets, their connections to process variables, stylesheets, scripts and parameter sets are stored in a *layout file* with the extension `.tml`. Use **File** → **New**, **Load...**, **Save** and **Save As...** to manage layouts. Recently used layouts are listed under **File** → **Recent files**.

When closing Testmanager or loading another layout, Testmanager asks whether unsaved changes should be stored.

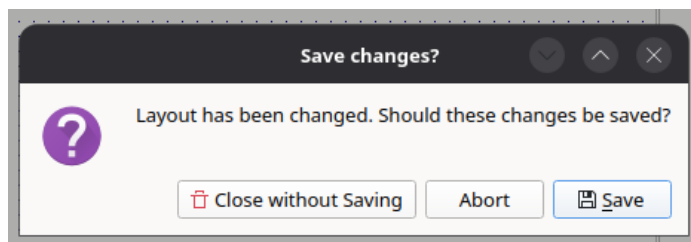


Figure 4.1.: Save changes?

Layout files of older versions are converted automatically when they are loaded. Testmanager informs about this, because older versions will not be able to read the file after saving.

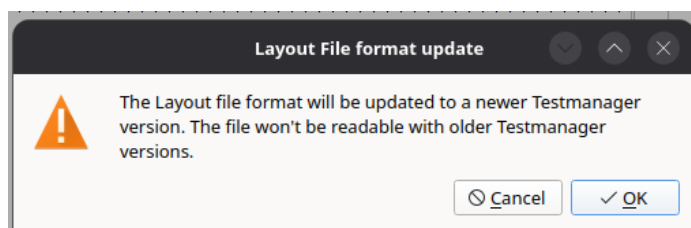


Figure 4.2.: Layout file format update

💡 Tip

Resources like images and message files are referenced relative to the directory of the layout file. Keep them together with the layout, for example in a version-controlled directory.

4.2. Edit mode

Testmanager has two modes: In *operation mode*, the widgets can be used to observe and change process values. In *edit mode*, widgets can be added, moved, resized and configured. Toggle the edit mode with **Edit Mode** in the tool bar or the **Options** menu.

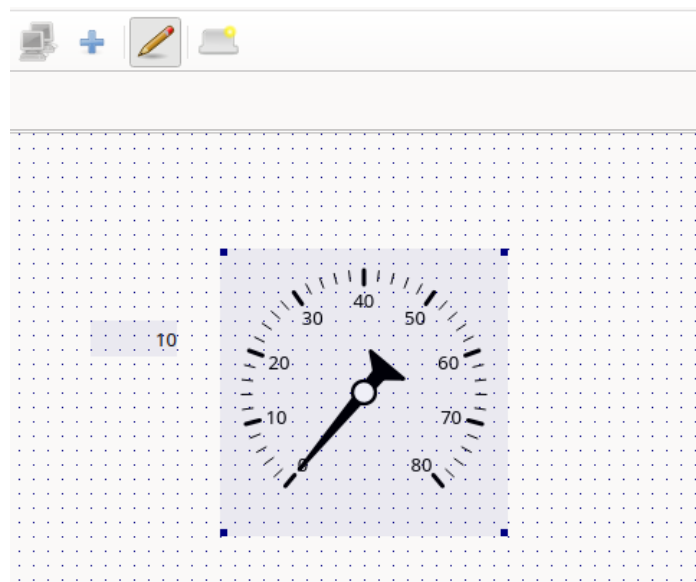


Figure 4.3.: Tab in edit mode

Right-click on an empty area of the canvas to add widgets, paste widgets from the clipboard, edit the stylesheet of the tab or mark all widgets that are not yet connected to process variables.

Right-click on a widget to copy or remove it, edit its process data connections, jump to its variables, align it, change its stacking order, edit its stylesheet or script, or show its legend.

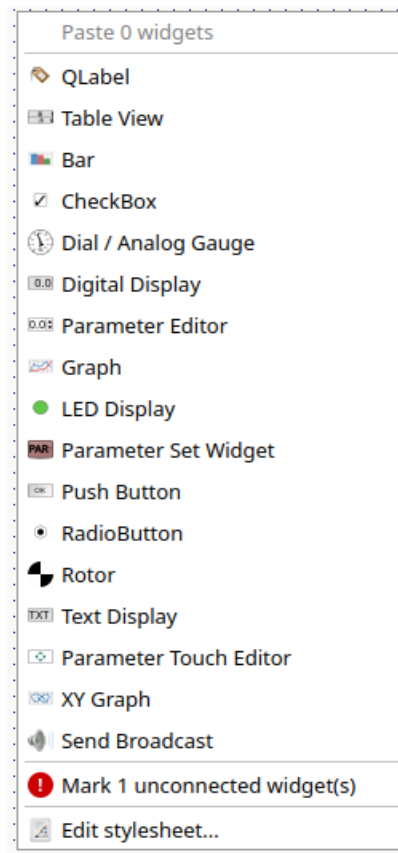


Figure 4.4.: Context menu of the canvas

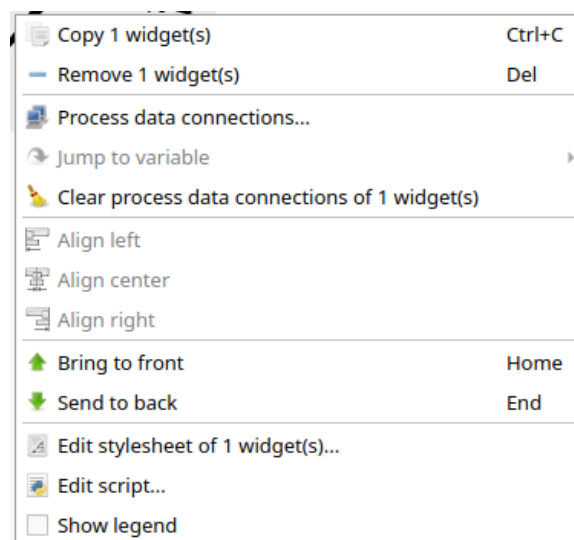


Figure 4.5.: Context menu of a widget

4.2.1. Unconnected widgets

In edit mode, widgets that are not yet connected to any process variable are marked (see the figure below). The status bar reports the number of unconnected widgets and of widgets whose variables could not be found in the connected data sources.

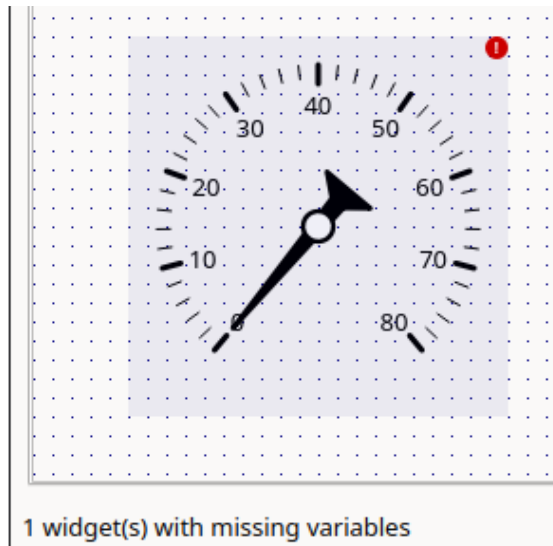


Figure 4.6.: Marked unconnected widgets

4.3. Property editor

The *Property Editor* dock shows the properties of the selected widget. Changes are applied immediately. Right-click a property and select **Reset** to restore its default value.

4.4. Tabs

A layout consists of one or more tabs. Use **Add Tab** to create a new tab. The context menu of the tab header allows to rename, detach or remove a tab. A detached tab is shown in a separate window, which is useful for setups with multiple screens.

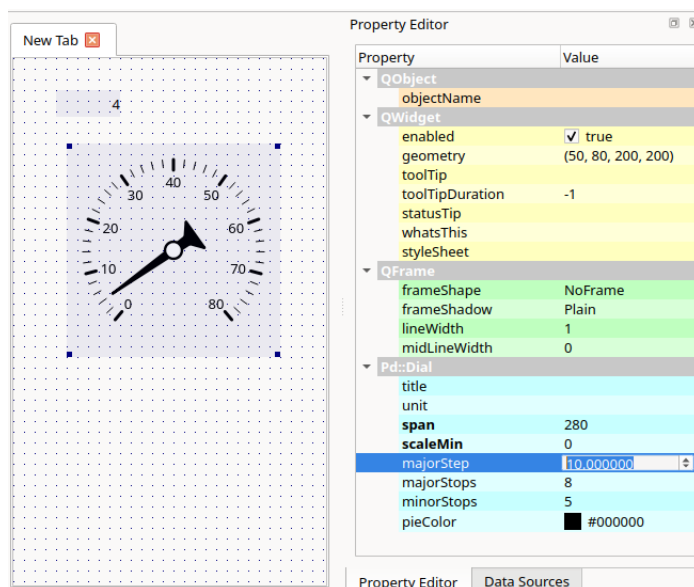


Figure 4.7.: Property editor

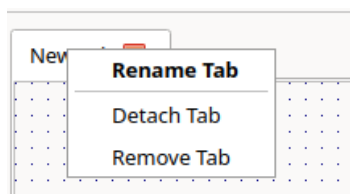


Figure 4.8.: Context menu of a tab header

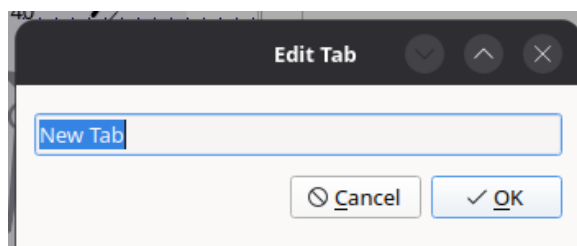


Figure 4.9.: Edit tab dialog

4. Layouts and Edit Mode

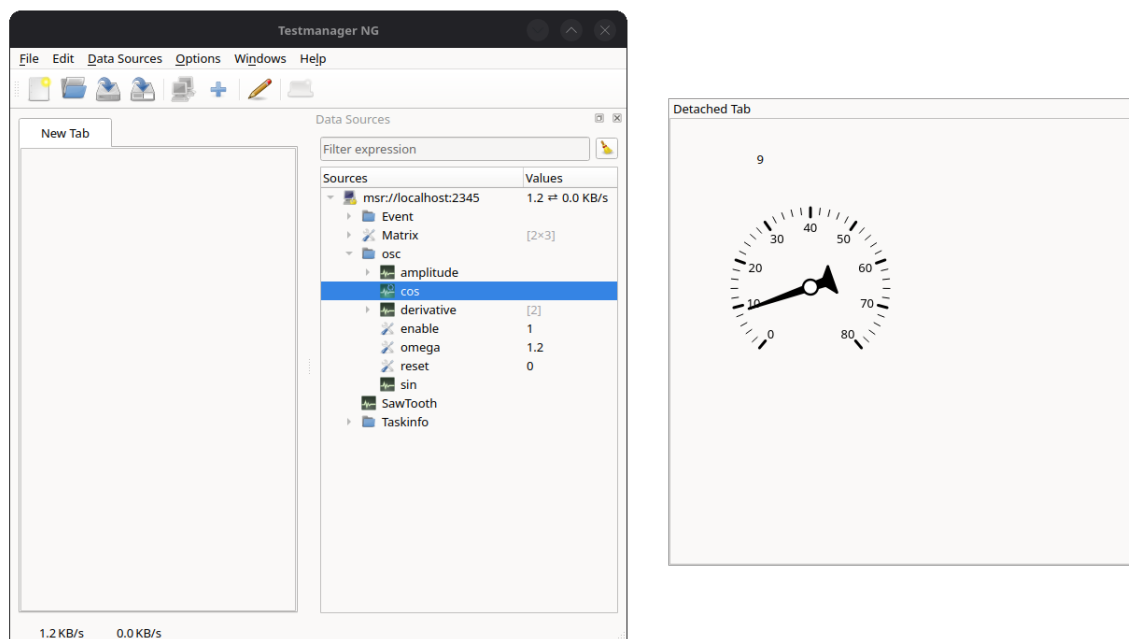


Figure 4.10.: Detached tab

5. Widgets

Testmanager uses the widgets of the [QtPdWidgets](#) library.

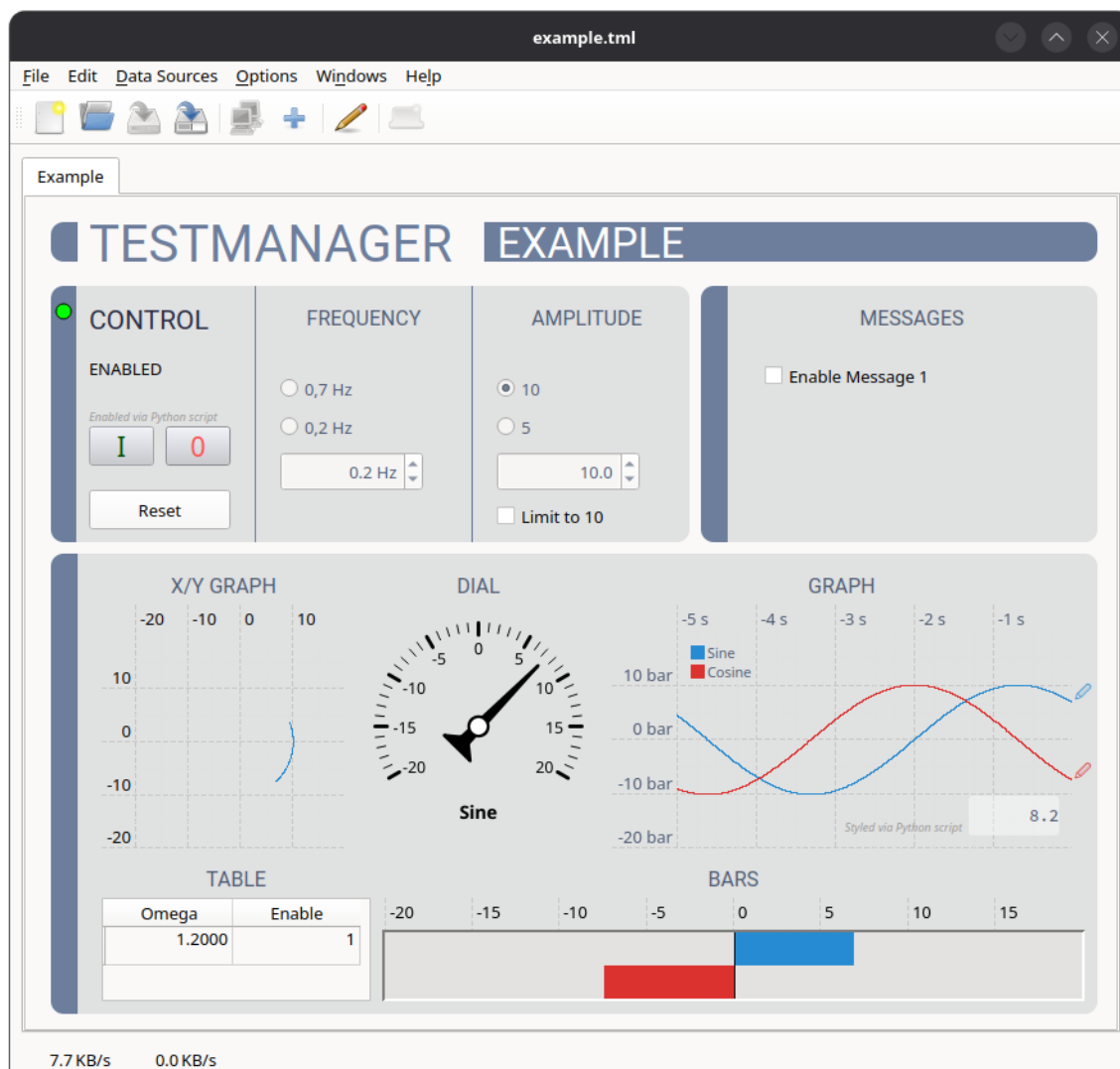


Figure 5.1.: All widgets at a glance

Widget	Purpose	Variables
Bar	Bar graph	Signals, vectors
CheckBox	Switch a boolean parameter	Parameter
Dial	Analog gauge	Signal, parameter
Digital	Numeric display	Signal
DoubleSpinBox	Numeric input	Parameter
Graph	Time graph	Signals
XYGraph	X/Y graph	Two signals
Led	Boolean indicator	Signal
ParameterSetWidget	Apply parameter sets	—
PushButton	Set a parameter on click	Parameter
QLabel	Static text or image	—
RadioButton	Select a value of a parameter	Parameter
Rotor	Rotating display	Signals
SendBroadcast	Send a broadcast message	—
TableView	Table of vector values	Signals, parameters
Text	Text depending on a value	Signal
TouchEdit	Numeric input for touch screens	Parameter

5.1. Dial

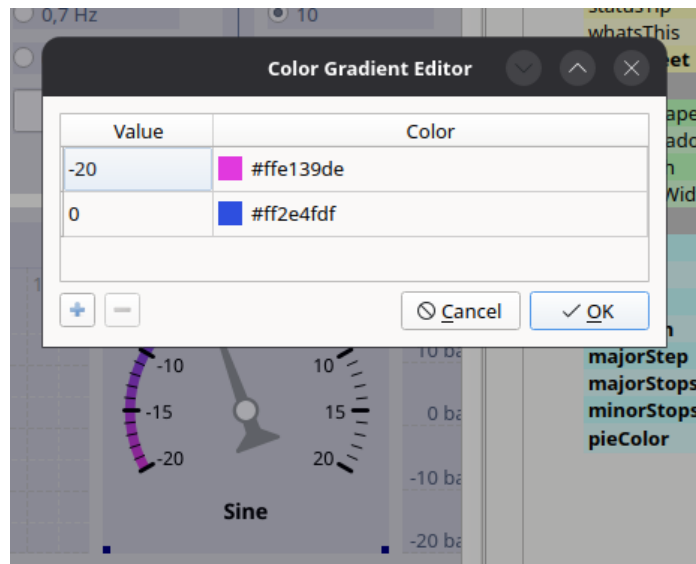


Figure 5.2.: Dial configuration dialog

5.2. Table view

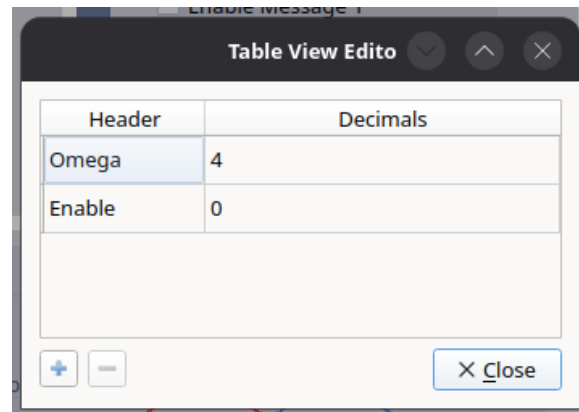


Figure 5.3.: Table view configuration dialog

5.3. Text

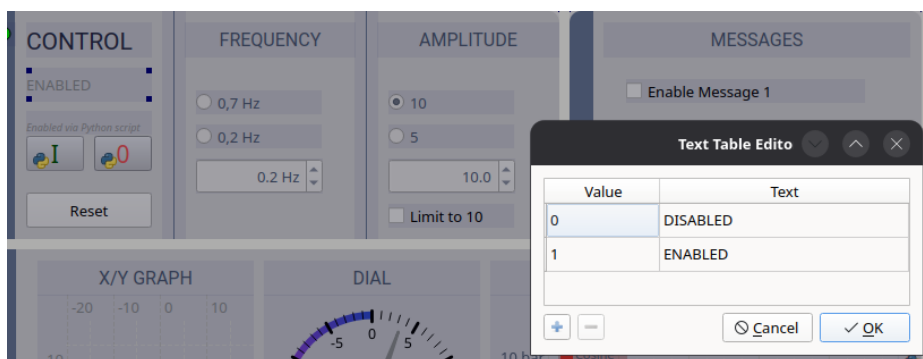


Figure 5.4.: Text configuration dialog

6. Connecting Signals and Parameters

6.1. Drag and drop

To connect a process variable to a widget, drag it from the data source tree onto the widget in edit mode. A tooltip indicates, whether the variables will be connected or replace the existing connections.

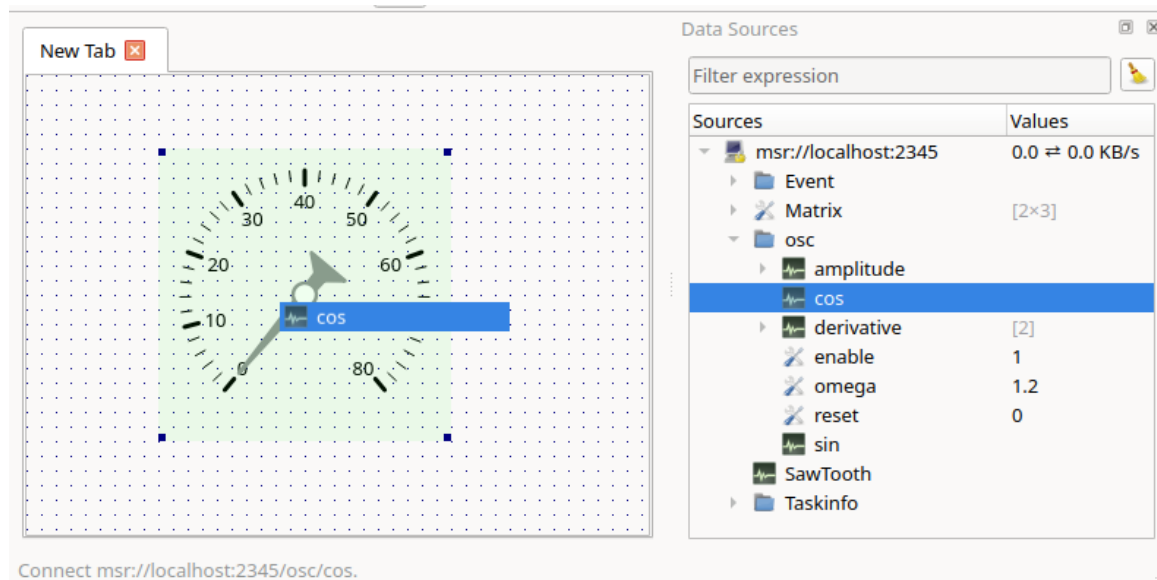


Figure 6.1.: Connecting a variable via drag and drop

When hovering over a widget, a tooltip lists all connected variables.

6.2. Process data connections

Select **Process data connections...** from the context menu of a widget to fine-tune its connections.

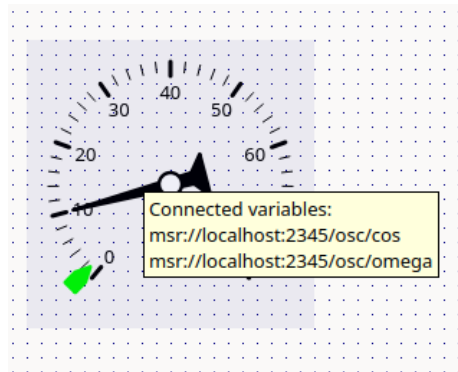


Figure 6.2.: Tooltip with connected variables

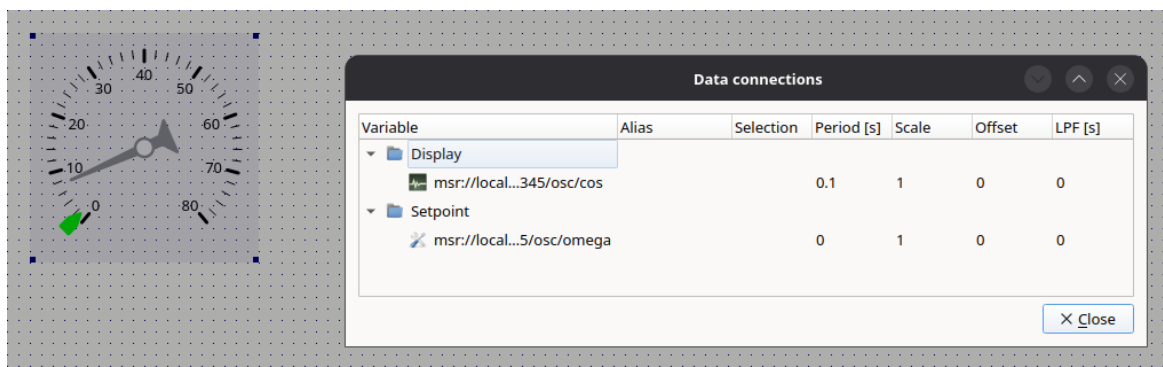


Figure 6.3.: Process data connections dialog

Column	Description
Alias	Name displayed instead of the variable path (e.g. in legends)
Selection	Element of a vector or matrix variable
Period [s]	Sample time; 0 means event-based transmission
Scale	Factor applied to the value
Offset	Offset added to the scaled value
LPF [s]	Time constant of a first-order low-pass filter
Color	Color of the variable in graphs

! Important

Within one connection to a process, a signal can be transmitted either event-based (period 0) *or* periodically (period > 0). Therefore use either event-based or periodic transmission consistently for a signal.

Right-click a column to apply a value to all connections or to the selected connections and their children.

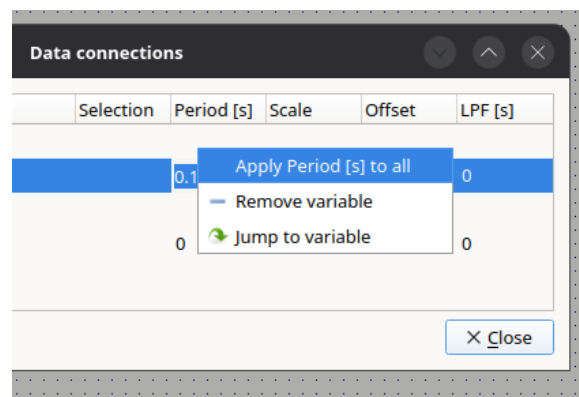


Figure 6.4.: Applying a value to all connections

6.3. Selecting vector elements

If a vector or matrix variable is connected to a widget that expects a scalar, the element can be chosen in the *Variable Selector* dialog.

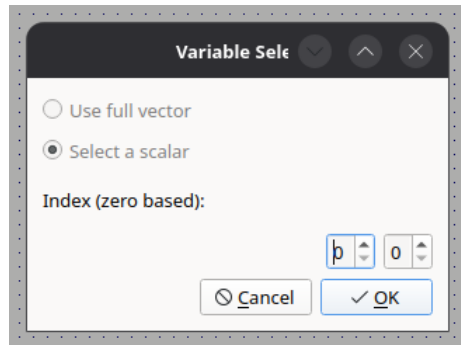


Figure 6.5.: Variable selector

6.4. Data slots dock

The *Data Slots* dock shows the connections of the selected widgets. Use **Jump to variable** to locate a connected variable in the data source tree.

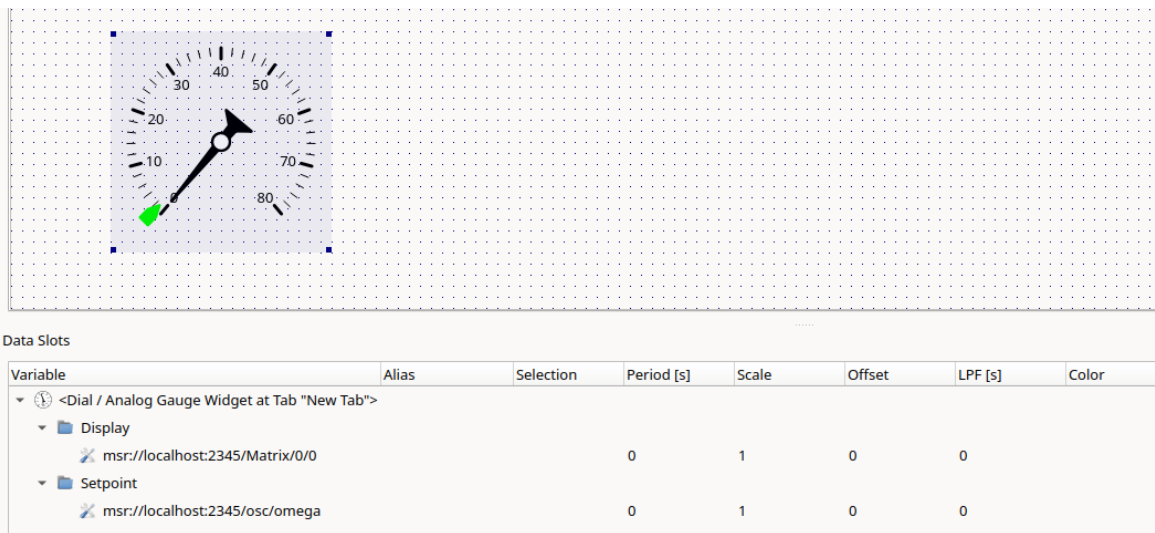


Figure 6.6.: Data slots dock

7. Legend

Each widget can display a *legend*, showing the names (or aliases) and colors of the connected variables. Enable it with **Show legend** from the context menu of the widget.

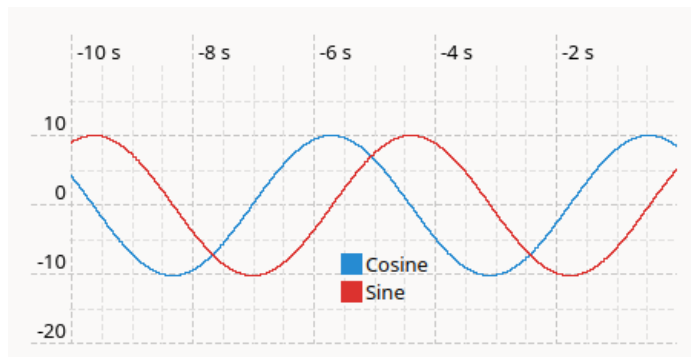


Figure 7.1.: Graph with legend

The context menu of the legend allows to hide it, to change the text alignment and to edit its stylesheet.

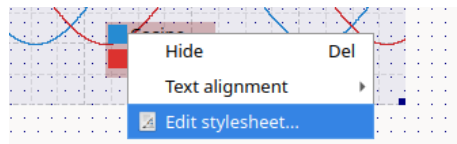


Figure 7.2.: Context menu of the legend

8. Stylesheets

The appearance of Testmanager layouts can be customised with [Qt stylesheets](#), whose syntax is similar to CSS.

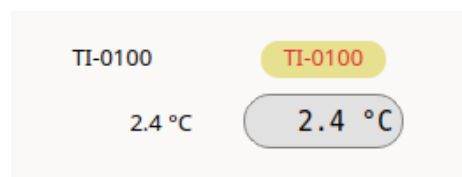


Figure 8.1.: The same widget without and with stylesheet

8.1. Levels

Stylesheets can be defined on three levels:

Level	Where	Applies to
Global	Options → Global stylesheet...	All tabs of the layout
Tab	Context menu of the canvas	All widgets of the tab
Widget	Context menu of one or more widgets	The selected widgets

More specific stylesheets override general ones.

8.2. Stylesheet editor

The tool bar of the editor helps inserting:

- **Resources** (images) for `background-image`, `border-image` and `image`,
- **Colors** for color properties (with alpha channel),
- **Fonts**,
- **Templates** with frequently used rules.

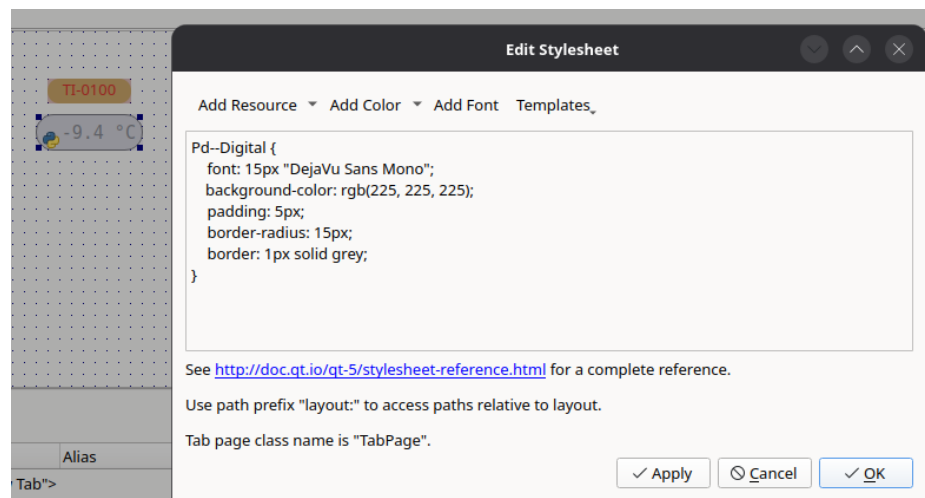


Figure 8.2.: Stylesheet editor

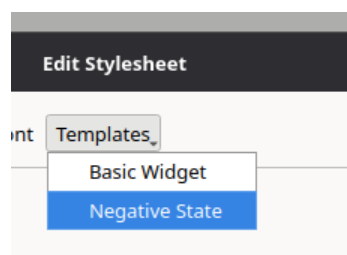


Figure 8.3.: Stylesheet templates

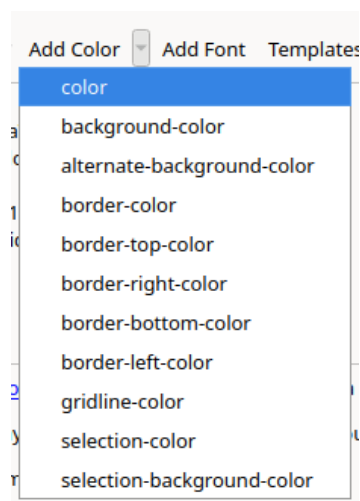


Figure 8.4.: Inserting a color

8.3. Selectors

Widgets are selected by their class name. As the class names of the QtPdWidgets contain the namespace `Pd::`, which is not valid in stylesheets, the double colon has to be replaced by two dashes:

```
Pd--Bar {
    background-color: white;
}
```

Testmanager provides the following special classes:

Class	Description
TabPage	The canvas of a tab
Legend	The legend of a widget

8.4. Resources

Images inserted via the editor are referenced relative to the layout file using the `layout:` prefix, so the layout directory can be moved as a whole:

```
QLabel#logo {
    image: url(layout:images/logo.svg);
}
```

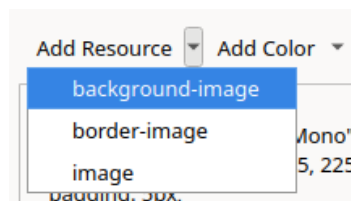


Figure 8.5.: Inserting a resource

8.5. Dynamic properties

Rules can depend on dynamic properties, which can be set from Python scripts (see [Python Scripts](#)):

8. Stylesheets

```
QWidget[negative=true] {  
    color: red;  
}
```

9. Python Scripts

Each widget can have a Python script that is executed whenever one of its script variables changes. This allows, for example, to enable or disable widgets or to change their appearance depending on process values.

9.1. Script editor

Select a widget in edit mode and open the *Python Script* dock (or select **Edit script...** from the context menu of the widget).

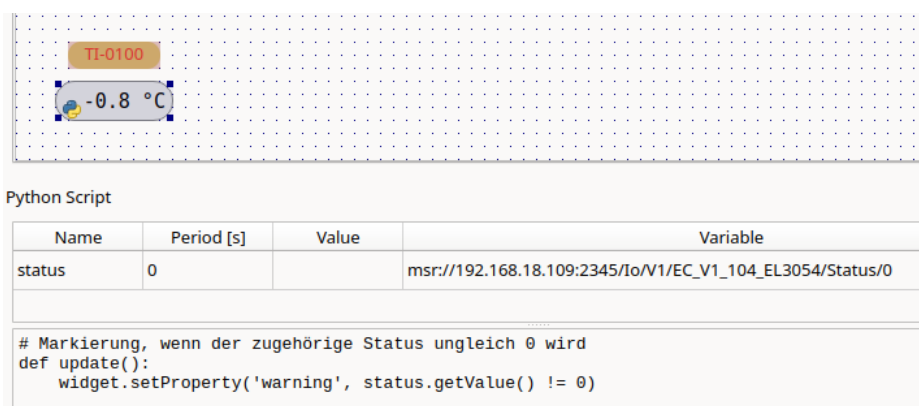


Figure 9.1.: Python script editor

The table at the top contains the *script variables*. Drag process variables from the data source tree into the table. The **Name** column defines the name under which the variable is available in the script; **Period** [s] defines the sample time.

Note

Within one connection to a process, a signal can be transmitted either event-based (period 0) or periodically (period > 0) (see [Browsing a data source](#)).

The script must define a function `update()`, which is called when a variable changes. Within the script, `widget` refers to the widget the script belongs to.

```
# Enable widget based on the value of the 'enable' variable
def update():
    widget.setEnabled(enable.getValue())
```

Right-click into the editor to insert templates.

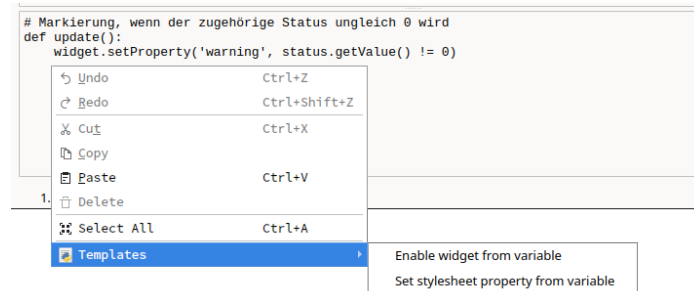


Figure 9.2.: Script templates

9.2. Combining scripts and stylesheets

A script can set dynamic properties, which are evaluated by the stylesheet:

```
def update():
    widget.setProperty('highlight', highlight.getValue())

QWidget[highlight=true] {
    background-color: yellow;
}
```

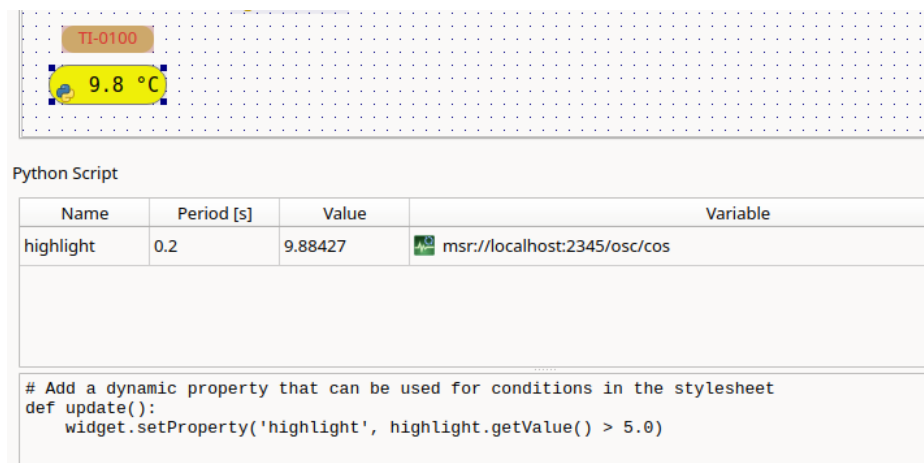


Figure 9.3.: Highlighting via script and stylesheet

9.3. Error messages

Syntax errors are displayed with the line number. If no callable `update()` is found, this is indicated as well.

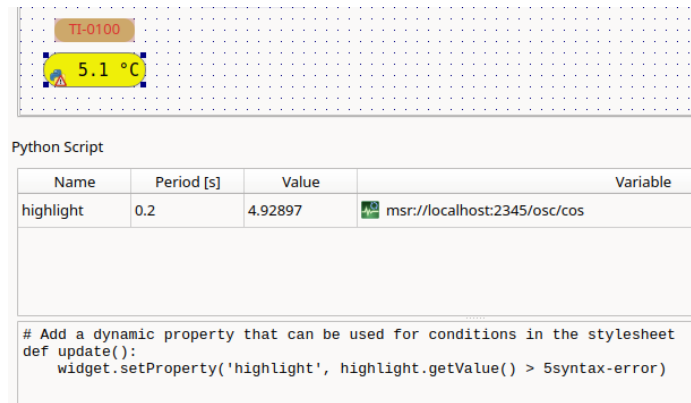


Figure 9.4.: Script error

9.4. Python shell

The *Python Shell* dock provides an interactive Python interpreter, which shares the environment with the scripts. Outputs of `print()` in scripts appear here.

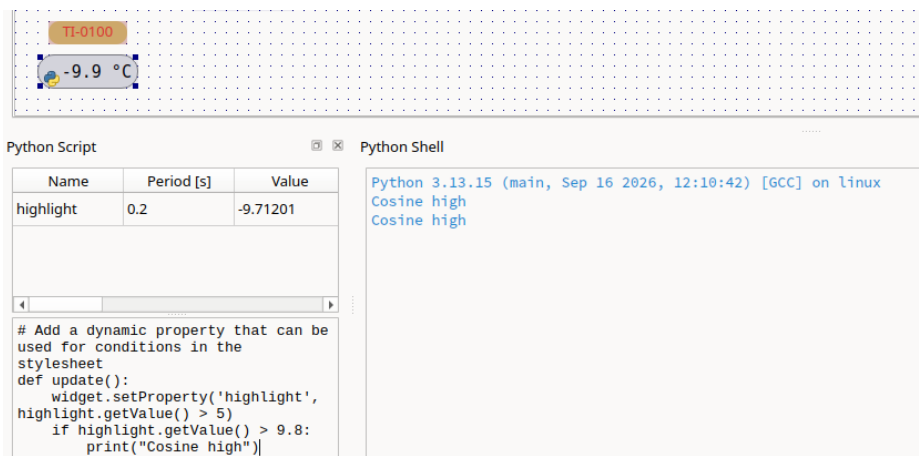


Figure 9.5.: Python shell

See [the Python API reference](#) for all available objects and functions.

10. Example Layout

Testmanager comes with an example layout, which is designed for the example process of PdServ. It demonstrates most of the features described in this manual and is a good starting point for own layouts.

10.1. Running the example

Start the PdServ example process (only available on Linux):

```
pdserv-example-st
```

Then open the example layout via **File** → **Load...** It is installed as `share/testmanager/example.tml`, on Linux usually at the path below. Alternatively, pass it on the command line:

```
testmanager --connect /usr/share/testmanager/example.tml
```

The layout refers to the data source `msr://localhost:2345`. If the example process runs on another machine, use [Replace data source](#).

10.2. What the example shows

- **Control:** An LED and a text widget show the state of the `/osc/enable` parameter. The two push buttons **I** and **O** switch it on and off. They are enabled and disabled by Python scripts depending on the current value, so only the meaningful button can be pressed. **Reset** sets the `/osc/reset` parameter.
- **Frequency:** Radio buttons and a spin box set `/osc/omega`.
- **Amplitude:** Radio buttons and a spin box set `/osc/amplitude/Setpoint`. The check box sets `/osc/amplitude/Limit`.
- **Messages:** The check box sets `/Event/State`, which makes the process emit a message, see [Messages](#).

10. Example Layout

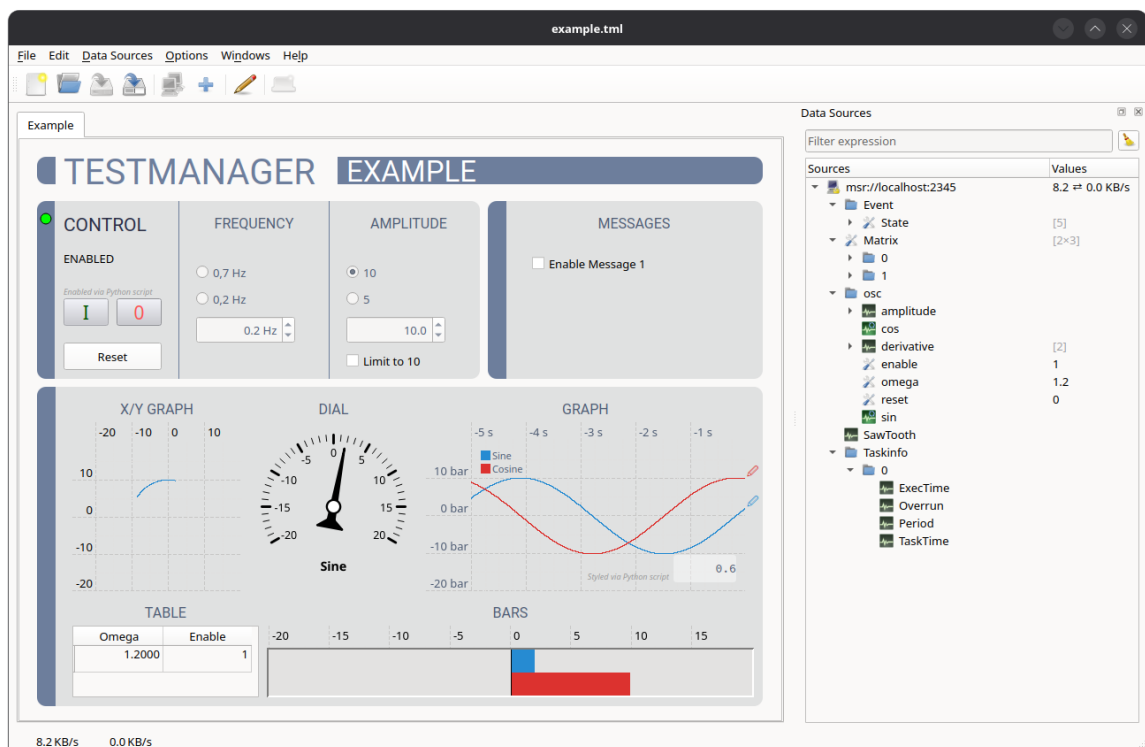


Figure 10.1.: The example layout, connected to pdserv-example-st

- **X/Y graph, Dial and Graph:** Different visualisations of the signals `/osc/sin` and `/osc/cos`. The graph shows a legend with the aliases *Sine* and *Cosine* and a digital display of the current value.
- **Table:** A table view showing and editing the parameters `/osc/omega` and `/osc/enable`.
- **Bars:** A bar graph showing both signals.

The example also contains several techniques, which are worth a closer look in edit mode:

- **Stylesheets:** The global stylesheet defines the colours and fonts of the whole layout. See [Stylesheets](#).
- **Dynamic properties:** A script sets the property `negative` depending on the sign of the sine signal, which is evaluated by a stylesheet rule:

```
def update():  
    widget.setProperty('negative', sin.getValue() < 0)
```

- **Moving widgets:** The pens at the right edge of the graph follow the current values. Their scripts change the `geometry` property:

```
def update():  
    widget.setProperty('geometry',  
        (820, 410 - sin.getValue() * 4.3, 20, 20))
```

See [Python Scripts](#) for details.

11. Parameter Sets

A *parameter set* is a named collection of parameters. Parameter sets are stored in the layout. They allow to save the current values of the parameters to a file and to restore them later.

11.1. Creating parameter sets

Open the *Parameter Sets* dock and create a new set with **New set**. Add parameters by dragging them from the data source tree into the set. When dragging a folder or a whole data source, all parameters below it are added. The context menu offers **Add all parameters** of all connected data sources.

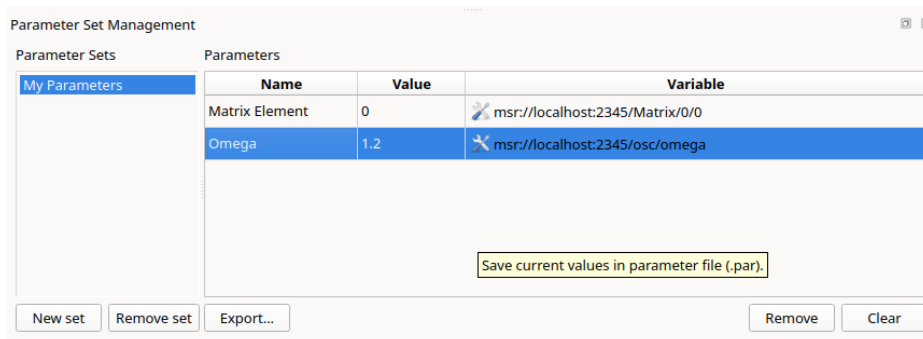


Figure 11.1.: Parameter sets dock

The number of parameters of a set is shown in its title and tooltip.

11.2. Exporting

Export... saves the current values of all parameters of the set to a parameter file (*.par), together with a description.

11. Parameter Sets

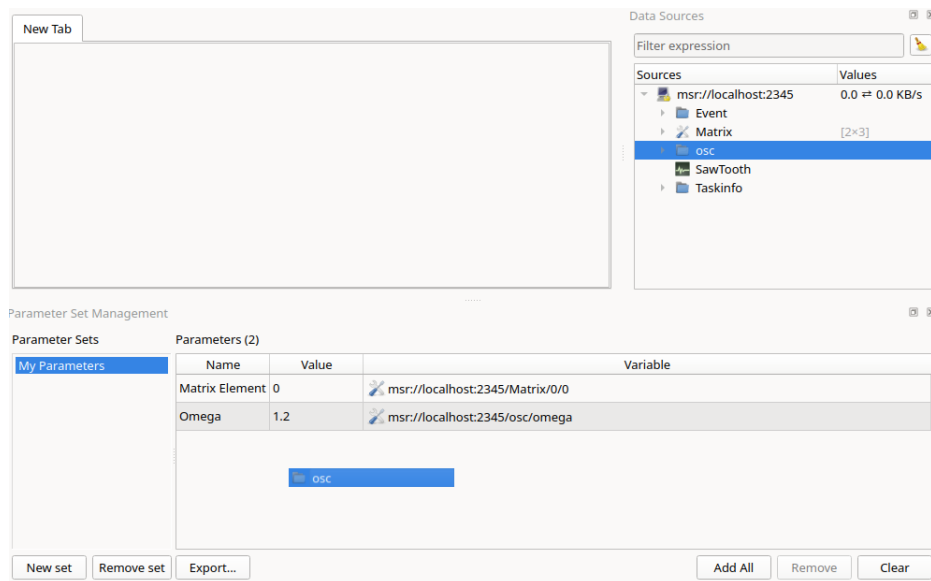


Figure 11.2.: Adding a folder via drag and drop

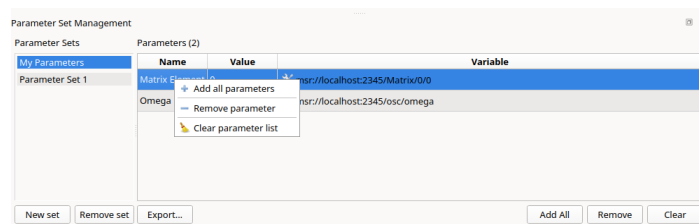


Figure 11.3.: Context menu of a parameter set

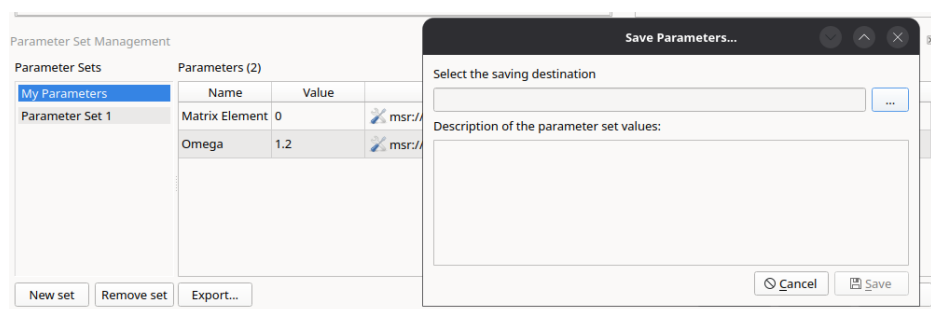


Figure 11.4.: Saving a parameter file

11.3. Importing

Load Parameter File... opens a parameter file and shows the stored and current values side by side, together with the meta data (creator, creation date, description). **Apply** writes the stored values to the process.

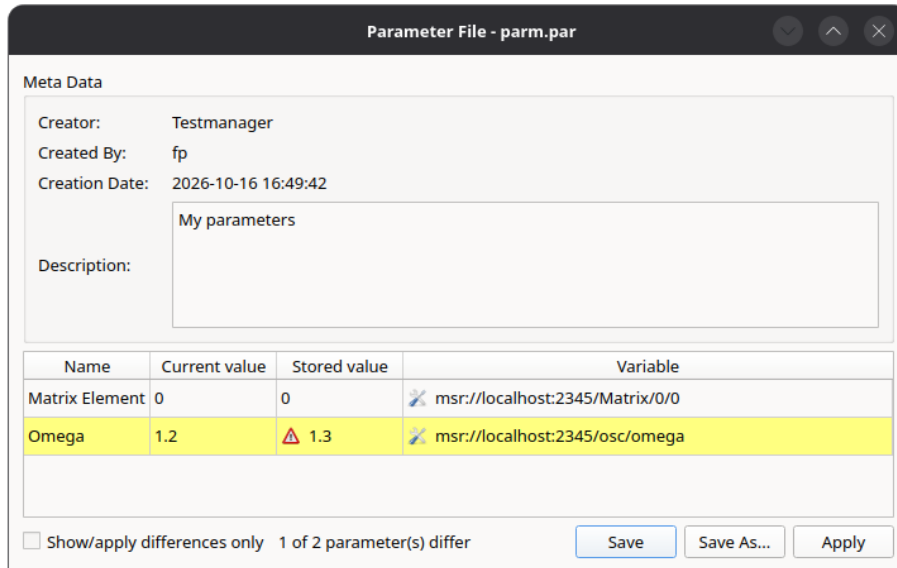


Figure 11.5.: Loading a parameter file

With **Show/apply differences only**, only the parameters whose stored value differs from the current value are shown and written.

Vector and matrix parameters are displayed and edited as tables.

11.4. Parameter set widget

The *ParameterSetWidget* allows operators to load parameter files directly from the layout.

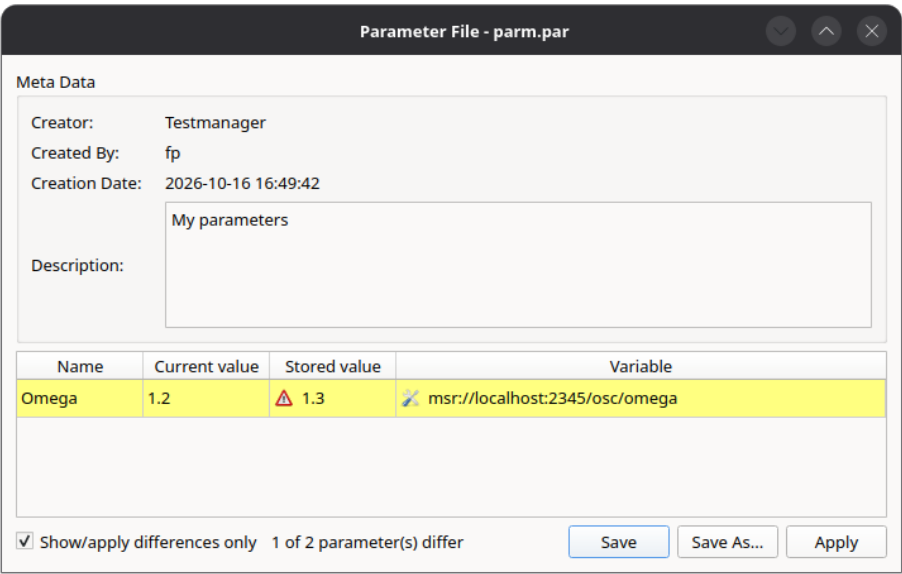


Figure 11.6.: Showing differences only

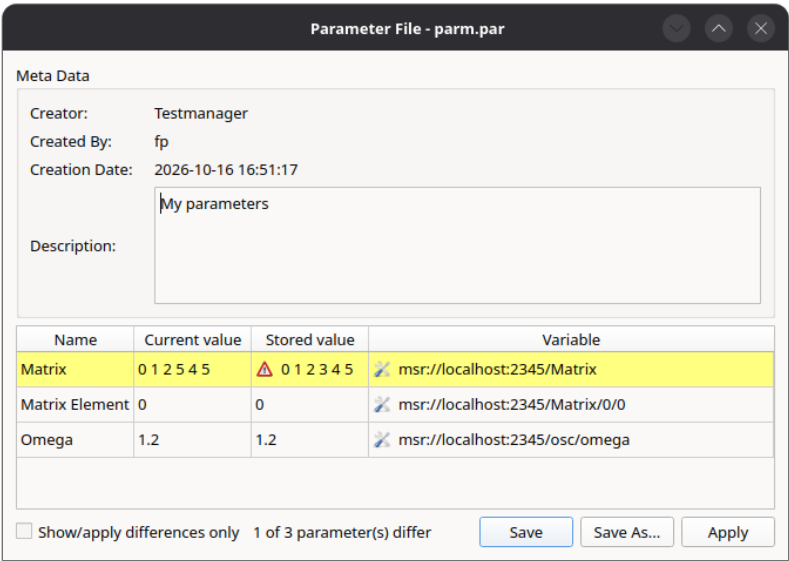


Figure 11.7.: Multi-dimensional parameter

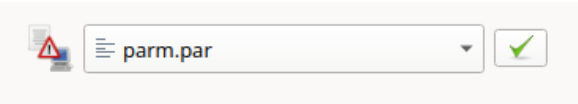


Figure 11.8.: Parameter set widget

12. Broadcasts

Broadcasts are text messages sent to a process and forwarded to all connected clients. They can be used, for example, to document actions in the process log.

Note

Broadcasts can also be used to log events in the measuring data – for example using EtherLab's [Data Logging Service \(DLS\)](#).

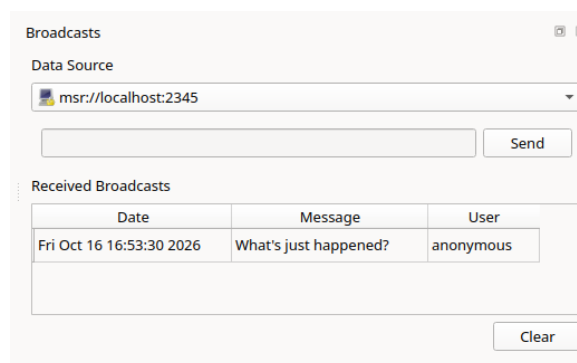
The *SendBroadcast* widget sends a broadcast when clicked.

The image shows a simple web widget for sending broadcasts. It consists of a text input field with the placeholder text "What's just happened?" and a "Send" button to its right.

What's just happened?	Send
-----------------------	------

Figure 12.1.: SendBroadcast widget

The *Broadcasts* dock lists all received broadcasts together with their data source. **Clear** empties the list.

The image shows a "Broadcasts" dock window. At the top, it has a "Data Source" dropdown menu currently set to "msr://localhost:2345". Below this is another text input field and a "Send" button. The main part of the dock is a table titled "Received Broadcasts". The table has three columns: "Date", "Message", and "User". It contains one row of data: "Fri Oct 16 16:53:30 2026", "What's just happened?", and "anonymous". At the bottom right of the dock is a "Clear" button.

Broadcasts		
Data Source		
msr://localhost:2345		
		Send
Received Broadcasts		
Date	Message	User
Fri Oct 16 16:53:30 2026	What's just happened?	anonymous
		Clear

Figure 12.2.: Received broadcasts

13. Messages

Processes can emit *messages* (information, warnings, errors, critical errors), for example when a limit is exceeded.

13.1. Configuration

The texts of the messages are defined in a *plain messages file*, which is configured per data source via **Configure messages...** in the context menu of the data source. The path is stored relative to the layout file. Translations are selected according to the language of the user interface.

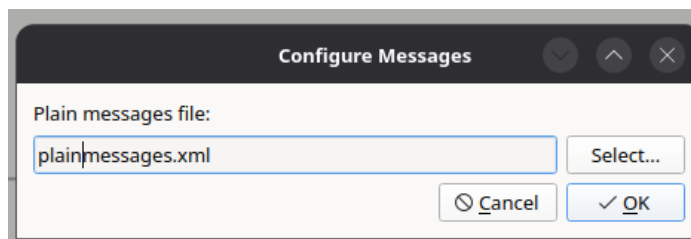


Figure 13.1.: Configure messages

13.2. Displaying messages

Show messages... in the context menu of a data source shows the messages of that data source.

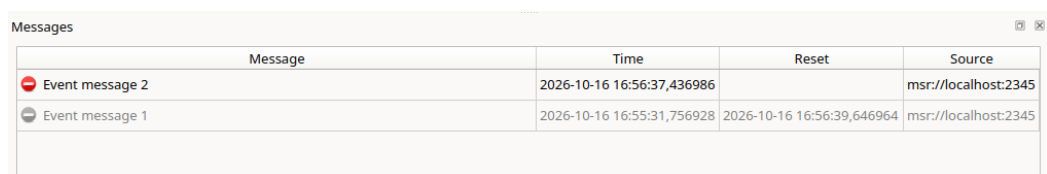
The *Messages* dock combines the messages of all connected data sources. With **Show Historic Messages**, messages that are no longer active are displayed as well.



A screenshot of a window titled "Messages" with a standard macOS-style title bar (red, yellow, green buttons). The window contains a table with three columns: "Message", "Time", and "Reset". The first row of the table contains the text "Event message 1" with a red stop icon to its left, the timestamp "2026-10-16 16:55:31,756928", and an empty "Reset" cell.

Message	Time	Reset
Event message 1	2026-10-16 16:55:31,756928	

Figure 13.2.: Messages of a data source



A screenshot of a window titled "Messages" with a standard macOS-style title bar. The window contains a table with four columns: "Message", "Time", "Reset", and "Source". The first row shows "Event message 2" with a red stop icon, the timestamp "2026-10-16 16:56:37,436986", an empty "Reset" cell, and the source "msr://localhost:2345". The second row shows "Event message 1" with a grey stop icon, the timestamp "2026-10-16 16:55:31,756928", the timestamp "2026-10-16 16:56:39,646964", and the source "msr://localhost:2345".

Message	Time	Reset	Source
Event message 2	2026-10-16 16:56:37,436986		msr://localhost:2345
Event message 1	2026-10-16 16:55:31,756928	2026-10-16 16:56:39,646964	msr://localhost:2345

Figure 13.3.: Messages dock

14. Replacing Data Sources

A layout references process variables by URL, including host and port. To use a layout with another process, for example when moving from a development machine to the target system, select **Replace...** from the context menu of the data source.

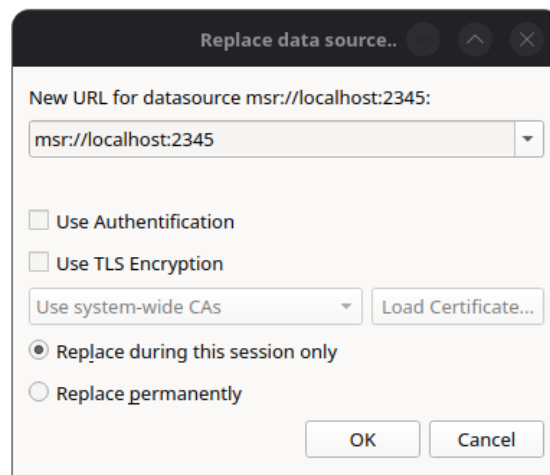


Figure 14.1.: Replace data source

- **Replace during this session only** keeps the layout file unchanged. This is useful to test a layout against a simulation.
- **Replace permanently** changes the URLs in the layout. Save the layout afterwards.

The dialog also allows to enable TLS encryption and authentication for the new data source.

A. Keyboard Shortcuts

Shortcut	Action
Ctrl+N	New layout
Ctrl+O	Load layout
Ctrl+S	Save layout
Ctrl+C	Copy selected widgets
Ctrl+V	Paste widgets
Del	Hide legend (legend menu)

B. Python API

B.1. Script variables

Each script variable is available under the name given in the script variable table.

Method	Description
<code>getValue()</code>	Current value (a number, or a NumPy array for vectors)

B.2. The widget object

Method	Description
<code>setEnabled(enabled)</code>	Enables or disables the widget
<code>setProperty(name, value)</code>	Sets a (dynamic) Qt property of the widget

B.3. The `tm` module

```
import tm
```

Function	Description
<code>tm.tabCount()</code>	Number of tabs of the current layout
<code>tm.maximize(max)</code>	Maximizes (or restores) the main window

