Testmanager 1.2.6
Benutzerhandbuch

Florian Pose, fp@igh.de
Ingenieurgemeinschaft IgH

Essen, 5. November 2026
Revision f8283c9c

Inhaltsverzeichnis

1. Einführung **1**
1.1. Hauptfenster 2

2. Installation **5**
2.1. Linux 5
2.2. Windows 6
2.3. Bauen aus dem Quellcode 7
2.4. Kommandozeile 7

3. Datenquellen **9**
3.1. Hinzufügen einer Datenquelle 9
3.2. Überwachen der Verbindung 11
3.3. Authentifizierung 12
3.4. Durchsuchen einer Datenquelle 12

4. Layouts und Editiermodus **15**
4.1. Layout-Dateien 15
4.2. Editiermodus 16
4.3. Eigenschafts-Editor 18
4.4. Tabs 18

5. Widgets **21**
5.1. Dial 22
5.2. Tabellenansicht 23
5.3. Text 23

6. Signale und Parameter verbinden **25**
6.1. Drag and Drop 25
6.2. Prozessdaten-Verbindungen 25
6.3. Vektorelemente auswählen 27
6.4. Dock „Datenverbindungen“ 28

7. Legende **29**

8. Stylesheets **31**
8.1. Ebenen . . . 31
8.2. Stylesheet-Editor . . . 31
8.3. Selektoren . . . 33
8.4. Ressourcen . . . 33
8.5. Dynamische Eigenschaften . . . 33

9. Python-Scripts **35**
9.1. Script-Editor . . . 35
9.2. Scripts und Stylesheets kombinieren . . . 36
9.3. Fehlermeldungen . . . 36
9.4. Python-Kommandozeile . . . 38

10. Beispiel-Layout **39**
10.1. Das Beispiel ausführen . . . 39
10.2. Was das Beispiel zeigt . . . 39

11. Parametersätze **43**
11.1. Parametersätze anlegen . . . 43
11.2. Exportieren . . . 43
11.3. Importieren . . . 45
11.4. Parametersatz-Widget . . . 45

12. Broadcasts **47**

13. Meldungen **49**
13.1. Konfiguration . . . 49
13.2. Meldungen anzeigen . . . 49

14. Datenquellen ersetzen **51**

A. Tastenkürzel **53**

B. Python-API **55**
B.1. Skriptvariablen . . . 55
B.2. Das Objekt `widget` . . . 55
B.3. Das Modul `tm` . . . 55

Abbildungsverzeichnis

1.1. Hauptfenster des Testmanagers mit Beispiel-Layout 1
1.2. Hauptfenster im Editiermodus 2
1.3. Statusleiste mit Netzwerkdurchsatz 3

2.1. Download-Schaltfläche auf GitLab 6
2.2. Entpacktes ZIP-Archiv mit testmanager.exe 7
2.3. Über-Dialog 7

3.1. Verbindungsdialog 10
3.2. Verbindungsdialog mit TLS-Optionen 10
3.3. Tooltip einer Datenquelle 11
3.4. Kontextmenü einer Datenquelle 11
3.5. Anmeldedialog 12
3.6. Datenquellen-Baum 13
3.7. Gefilterter Datenquellen-Baum 13
3.8. Kontextmenü einer Variable 14

4.1. Änderungen speichern? 15
4.2. Änderung des Layout-Dateiformats 15
4.3. Tab im Editiermodus 16
4.4. Kontextmenü der Arbeitsfläche 17
4.5. Kontextmenü eines Widgets 17
4.6. Markierte nicht verbundene Widgets 18
4.7. Eigenschafts-Editor 19
4.8. Kontextmenü eines Tab-Kopfes 19
4.9. Dialog „Reiter editieren“ 19
4.10. Abgelöster Tab 20

5.1. Alle Widgets auf einen Blick 21
5.2. Konfigurationsdialog des Dial 22
5.3. Konfigurationsdialog der Tabellenansicht 23
5.4. Konfigurationsdialog des Text-Widgets 23

6.1. Verbinden einer Variable per Drag and Drop 25
6.2. Tooltip mit verbundenen Variablen 26

6.3. Dialog „Datenverbindungen“ . . . 26
6.4. Anwenden eines Werts auf alle Verbindungen . . . 27
6.5. Variablen-Selektor . . . 28
6.6. Dock „Datenverbindungen“ . . . 28

7.1. Graph mit Legende . . . 29
7.2. Kontextmenü der Legende . . . 29

8.1. Das gleiche Widget ohne und mit Stylesheet . . . 31
8.2. Stylesheet-Editor . . . 32
8.3. Stylesheet-Vorlagen . . . 32
8.4. Einfügen einer Farbe . . . 32
8.5. Einfügen einer Ressource . . . 33

9.1. Python-Script-Editor . . . 35
9.2. Script-Vorlagen . . . 36
9.3. Hervorhebung per Script und Stylesheet . . . 37
9.4. Script-Fehler . . . 37
9.5. Python-Kommandozeile . . . 38

10.1. Das Beispiel-Layout, verbunden mit pdserv-example-st . . . 40

11.1. Dock „Parametersätze“ . . . 43
11.2. Hinzufügen eines Ordners per Drag & Drop . . . 44
11.3. Kontextmenü eines Parametersatzes . . . 44
11.4. Speichern einer Parameterdatei . . . 44
11.5. Laden einer Parameterdatei . . . 45
11.6. Nur Unterschiede anzeigen . . . 46
11.7. Mehrdimensionaler Parameter . . . 46
11.8. Parametersatz-Widget . . . 46

12.1. SendBroadcast-Widget . . . 47
12.2. Empfangene Broadcasts . . . 47

13.1. Meldungen konfigurieren . . . 49
13.2. Meldungen einer Datenquelle . . . 50
13.3. Dock „Meldungen“ . . . 50

14.1. Datenquelle ersetzen . . . 51

1. Einführung

Der Testmanager ist eine grafische Anwendung zur Visualisierung und Änderung der Signale und Parameter von EtherLab-Steuerungsprozessen. Er verbindet sich über das MSR-Protokoll mit einem oder mehreren Prozessen. Dabei wird auf Client-Seite die Bibliothek PdCom und auf Prozess-Seite PdServ verwendet.

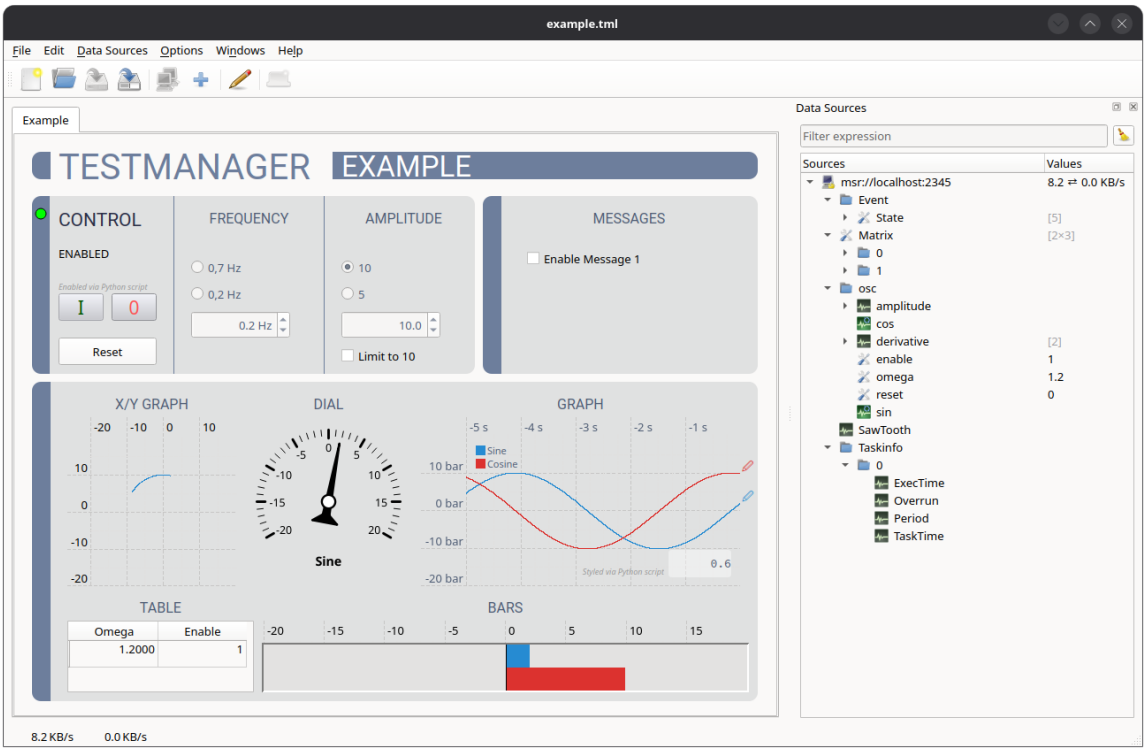

Abbildung 1.1.: Hauptfenster des Testmanagers mit Beispiel-Layout

Die Signale und Parameter der verbundenen Prozesse werden in einem Baum aufgelistet. Sie können mit Bedien- und Visualisierungs-Widgets (Balkenanzeigen, Zeigerinstrumente, LEDs, Zeitverläufe, Schaltflächen, Eingabefelder, ...) verknüpft werden, indem sie einfach auf die Widgets gezogen werden. Auf diese Weise entsteht eine Bedienoberfläche ohne Programmierung. Die Anordnung der Widgets wird in einer *Layout-Datei* gespeichert.

Weitere Funktionen sind:

- Beliebig viele Tabs, die als separate Fenster abgelöst werden können.
- Stylesheets auf globaler, Tab- und Widget-Ebene.
- Python-Skripte, die auf Prozesswerte reagieren.
- Parametersätze, die gespeichert, verglichen und wiederhergestellt werden können.
- Anzeige von Prozessmeldungen und Broadcasts.
- Verschlüsselte (TLS) und authentifizierte Verbindungen.

> **Tipp**
> Viele Themen dieses Handbuchs werden auch in kurzen Videos behandelt.

1.1. Hauptfenster

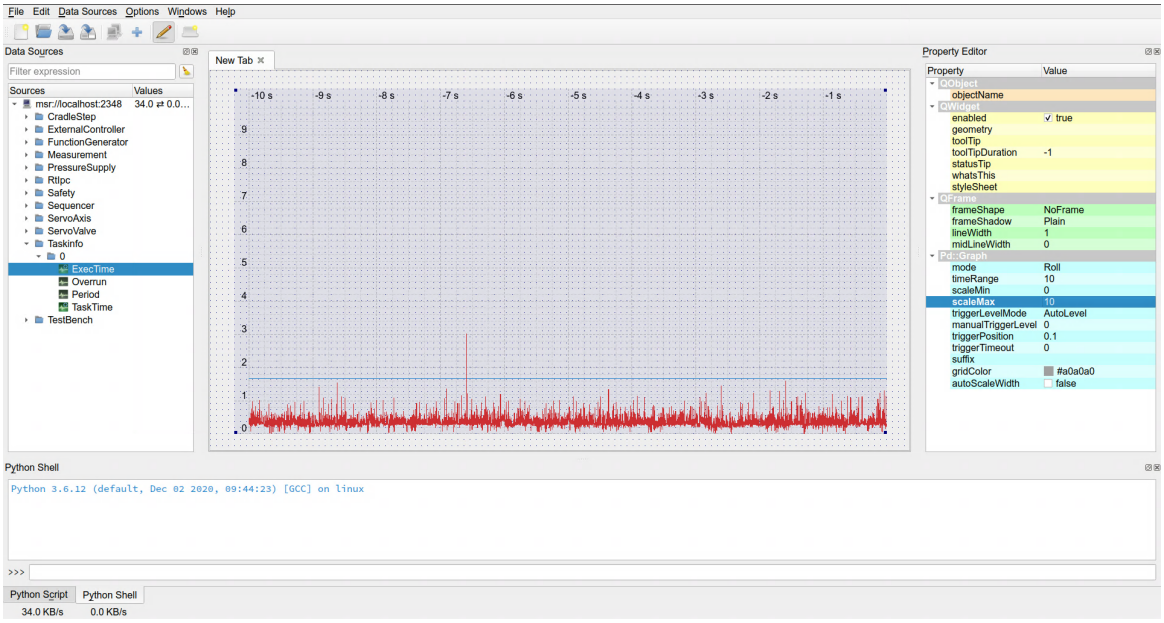

Abbildung 1.2.: Hauptfenster im Editiermodus

Das Hauptfenster besteht aus den folgenden Bereichen:

- **Menü- und Werkzeugleiste** oben, mit Aktionen für Layout-Dateien, zum Hinzufügen von Tabs und zum Umschalten des Editiermodus.
- **Tabs** in der Mitte. Jeder Tab ist eine Arbeitsfläche, die die Widgets enthält.
- **Docks** rund um die Tabs, die verschoben, übereinander gestapelt (wie im unteren Bereich der Abbildung) oder abgelöst werden können:

  - *Datenquellen*: Baum der verbundenen Prozesse und ihrer Variablen,
  - *Eigenschafts-Editor*: Eigenschaften des ausgewählten Widgets,
  - *Datenverbindungen*: Prozessdaten-Verbindungen der ausgewählten Widgets,
  - *Python-Script* und *Python-Kommandozeile*,
  - *Parametersätze*,
  - *Broadcasts* und *Meldungen*.
- **Statusleiste** unten.

Das Menü **Fenster** blendet die einzelnen Docks und die Werkzeugleiste ein oder aus.

Die Statusleiste zeigt den gesamten Netzwerkdurchsatz aller Datenquellen an, links die empfangenen und rechts die gesendeten Daten. Der Durchsatz einer einzelnen Datenquelle wird in der Spalte *Werte* des Datenquellen-Baums angezeigt.

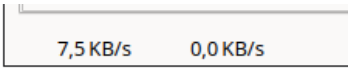

Abbildung 1.3.: Statusleiste mit Netzwerkdurchsatz

2. Installation

2.1. Linux

Pakete für verschiedene Distributionen werden über den openSUSE Build Service bereitgestellt. Sie ziehen die benötigten Bibliotheken PdCom, QtPdCom und QtPdWidgets automatisch mit.

Das Paket registriert außerdem den MIME-Typ für Layout-Dateien (`*.tml`) und einen Desktop-Eintrag, sodass Layout-Dateien per Doppelklick in einem Dateimanager geöffnet werden können.

2.1.1. openSUSE (RPM-basiert)

Fügen Sie das EtherLab-RPM-Repository hinzu:

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

Sie werden nun gefragt, ob Sie dem Repository-Schlüssel vertrauen. Installieren Sie anschließend den Testmanager und das PdServ-Beispiel (das später zur Demonstration verwendet wird):

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

2.1.2. Debian (deb-basiert)

Fügen Sie das EtherLab-Repository zu sources.list hinzu:

```
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
```

Installieren Sie anschließend den Testmanager und das PdServ-Beispiel (das später zur Demonstration verwendet wird):

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

2.2. Windows

Für Windows wird der Testmanager als ZIP-Archiv bereitgestellt, das von GitLab heruntergeladen werden kann.

Gehen Sie dazu auf GitLab und klicken Sie auf die blaue Schaltfläche „Download for Windows“:

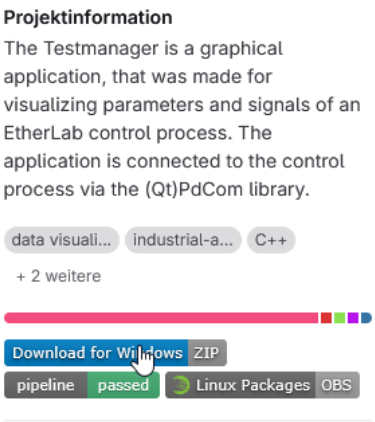

Abbildung 2.1.: Download-Schaltfläche auf GitLab

Es gibt kein Installationsprogramm. Stattdessen können Sie das heruntergeladene Archiv einfach entpacken und das Programm aus dem entpackten Ordner starten:

1. Laden Sie das ZIP-Archiv herunter.
2. Entpacken Sie das komplette Archiv in ein lokales Verzeichnis.
3. Starten Sie `testmanager.exe` aus dem entpackten Verzeichnis.

> **Wichtig**
> Starten Sie `testmanager.exe` nicht direkt aus dem ZIP-Archiv heraus. Der Testmanager benötigt die anderen Dateien des Archivs (Bibliotheken, mitgeliefertes Python), die erst nach dem Entpacken verfügbar sind.

Sie können dann zum Beispiel eine Verknüpfung erstellen, indem Sie das Programm (`testmanager.exe`) mit der rechten Maustaste auf Ihren Desktop ziehen.

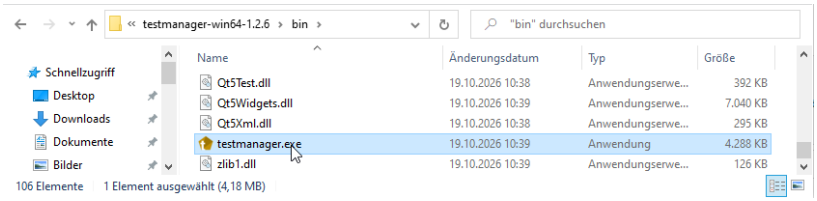

Abbildung 2.2.: Entpacktes ZIP-Archiv mit testmanager.exe

Hinweis

Wenn der Testmanager nicht startet, wird eine Fehlermeldung angezeigt. Bitte geben Sie deren Text zusammen mit der Version aus **Hilfe → Über den Testmanager** bei jeder Support-Anfrage an.

Abbildung 2.3.: Über-Dialog

2.3. Bauen aus dem Quellcode

Anweisungen zum Bauen finden Sie in der Datei `README.md` im Quellbaum.

2.4. Kommandozeile

```
testmanager [OPTIONS] [LAYOUT]
```

Option	Beschreibung
`LAYOUT`	Layout-Datei (`*.tml`), die beim Start geladen wird.
`-c`, `--connect`	Sofort verbinden und versuchen, verbunden zu bleiben.
`-n`, `--new`	Mit einem leeren Layout starten.
`-h`, `--help`	Hilfe zur Verwendung ausgeben.

Option	Beschreibung

3. Datenquellen

Eine *Datenquelle* ist ein Echtzeitprozess, der Signale und Parameter im Netzwerk bereitstellt. Voraussetzung für die Verbindung mit dem Testmanager ist, dass das Protokoll von der verwendeten PdCom-Bibliothek unterstützt wird (z. B. das MSR-Protokoll). Eine typische Datenquelle ist eine Echtzeitanwendung, die mit PdServ-Anbindung erstellt wurde, zum Beispiel ein mit EtherLab generiertes Simulink-Modell oder eine C/C++-Anwendung, die die PdServ-Bibliothek verwendet.

> **Tipp**
> Wenn PdServ installiert ist, startet der Befehl `pdserv-example-st` einen kleinen Beispielprozess, mit dem alle in diesem Handbuch beschriebenen Funktionen ausprobiert werden können. Eine vollständige Anleitung finden Sie im Beispiel-Kapitel.

3.1. Hinzufügen einer Datenquelle

Wählen Sie **Datenquellen → Datenquelle hinzufügen...** oder klicken Sie mit der rechten Maustaste in den Datenquellen-Baum und wählen Sie **Datenquelle hinzufügen...**. Geben Sie die URL des Prozesses ein:

URL-Schema	Standard-Port	Beschreibung
`msr://`	2345	Unverschlüsselte Verbindung
`msrs://`	4523	Per TLS verschlüsselte Verbindung

Zum Beispiel verbindet `msr://localhost` mit einem Prozess auf dem lokalen Rechner. Prozessvariablen werden über URLs der Form `msr://host:port/path/to/variable` referenziert.

Bei verschlüsselten Verbindungen kann das Server-Zertifikat entweder ignoriert oder gegen ein eigenes CA-Zertifikat geprüft werden. Diese Einstellungen werden pro Host und Port gespeichert.

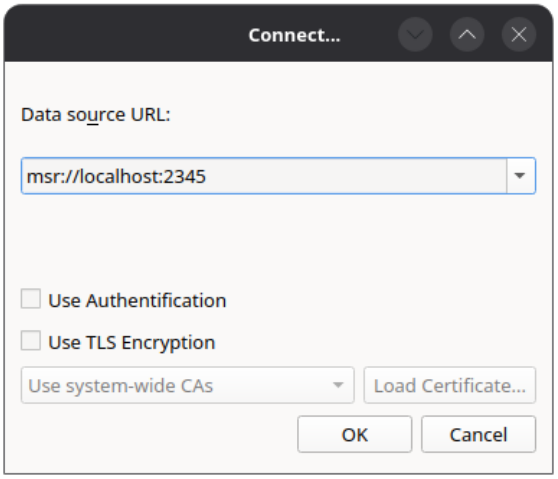

Abbildung 3.1.: Verbindungsdialog

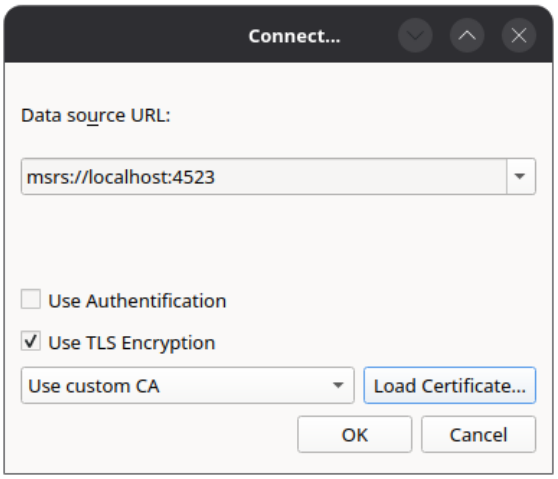

Abbildung 3.2.: Verbindungsdialog mit TLS-Optionen

3.2. Überwachen der Verbindung

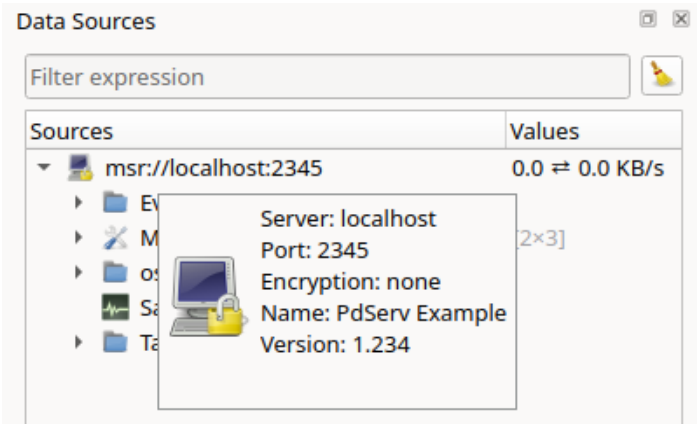

Abbildung 3.3.: Tooltip einer Datenquelle

Das Kontextmenü einer Datenquelle bietet die folgenden Aktionen: **Verbinden**, **Trennen**, **Anmelden**, **Abmelden**, **Entfernen**, **Ersetzen...**, **Meldungen anzeigen...** und **Meldungen konfigurieren...**.

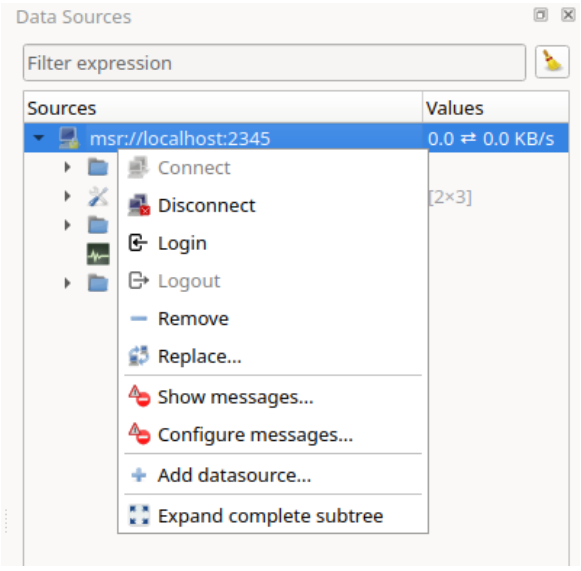

Abbildung 3.4.: Kontextmenü einer Datenquelle

Datenquellen → Datenquellen verbinden verbindet alle Datenquellen, die im aktuellen Layout referenziert werden.

3.3. Authentifizierung

Wenn der Prozess eine Authentifizierung erfordert, wählen Sie **Anmelden** im Kontextmenü der Datenquelle und geben Sie Benutzername und Passwort ein.

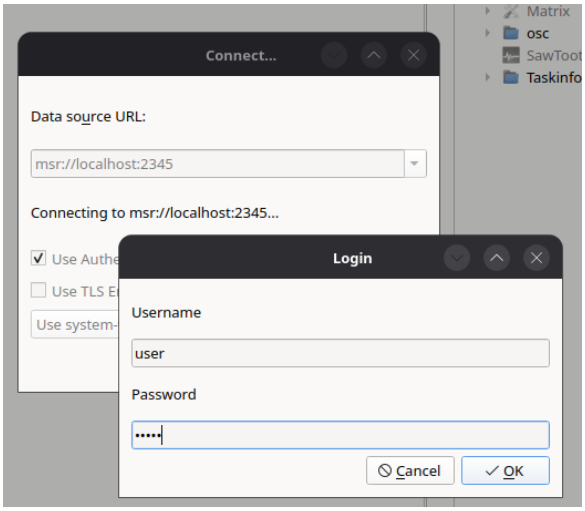

Abbildung 3.5.: Anmeldedialog

3.4. Durchsuchen einer Datenquelle

Nach dem Verbinden zeigt das Dock *Datenquellen* einen Baum aller Signale und Parameter des Prozesses.

- *Signale* sind vom Prozess berechnete Werte. Sie können nur gelesen werden.
- *Parameter* können gelesen und geschrieben werden.

Das Symbol vor einer Variable zeigt ihre Art an:

Symbol	Bedeutung
	Signal, derzeit nicht abonniert
	Signal, periodisch abonniert
	Signal, ereignisbasiert abonniert
	Parameter

Jede Variable hat eine *Abtastzeit*. Der Wert `0` bedeutet, dass die Variable nicht

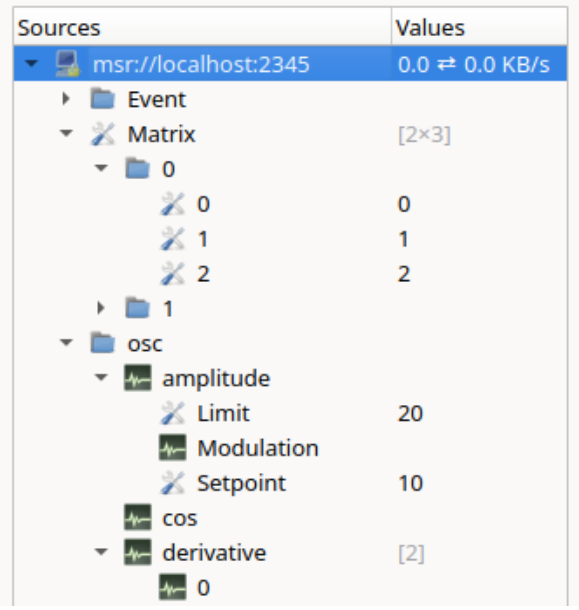

Abbildung 3.6.: Datenquellen-Baum

periodisch abgetastet, sondern *ereignisbasiert* übertragen wird, d. h. nur dann, wenn sich ihr Wert ändert. Parameter sind immer ereignisbasiert.

> **Wichtig**
> Innerhalb einer Verbindung zu einem Prozess kann ein Signal entweder ereignisbasiert (Periode `0`) *oder* periodisch (Periode > 0) übertragen werden. Verwenden Sie für ein Signal daher einheitlich entweder ereignisbasierte oder periodische Übertragung.

Ein Filterausdruck oberhalb des Baums schränkt die angezeigten Variablen ein.

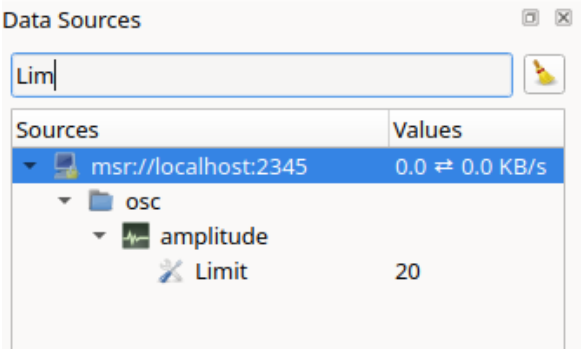

Abbildung 3.7.: Gefilterter Datenquellen-Baum

Das Kontextmenü einer Variable bietet:

- **Variablen-URL kopieren**: Kopiert die URL in die Zwischenablage.
- **Abonnenten auswählen**: Wählt alle Widgets aus, die mit der Variable verbunden sind.
- **Alles ausklappen**.

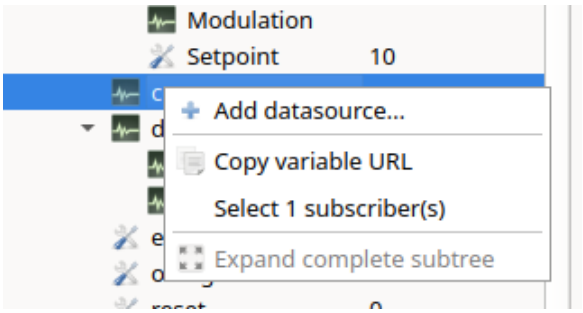

Abbildung 3.8.: Kontextmenü einer Variable

4. Layouts und Editiermodus

4.1. Layout-Dateien

Die Anordnung von Tabs und Widgets, ihre Verbindungen zu Prozessvariablen, Stylesheets, Skripte und Parametersätze werden in einer *Layout-Datei* mit der Endung `.tml` gespeichert. Verwenden Sie **Datei → Neu**, **Öffnen...**, **Speichern** und **Speichern als...**, um Layouts zu verwalten. Zuletzt verwendete Layouts sind unter **Datei → Zuletzt bearbeitete Dateien** aufgelistet.

Beim Schließen des Testmanagers oder beim Laden eines anderen Layouts fragt der Testmanager, ob ungespeicherte Änderungen gesichert werden sollen.

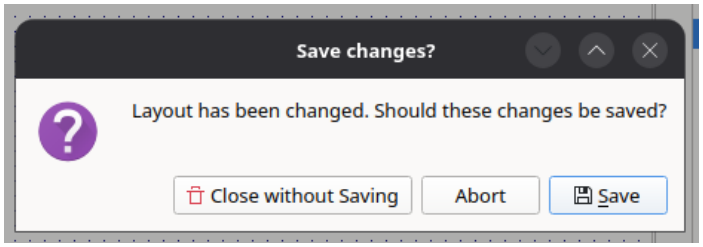

Abbildung 4.1.: Änderungen speichern?

Layout-Dateien älterer Versionen werden beim Laden automatisch konvertiert. Der Testmanager weist darauf hin, da ältere Versionen die Datei nach dem Speichern nicht mehr lesen können.

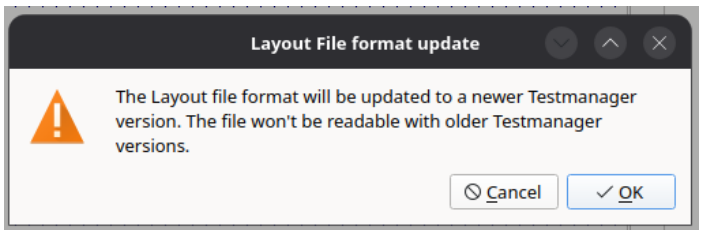

Abbildung 4.2.: Änderung des Layout-Dateiformats

> **Tipp**
> Ressourcen wie Bilder und Meldungsdateien werden relativ zum Verzeichnis der Layout-Datei referenziert. Bewahren Sie sie zusammen mit dem Layout auf, zum Beispiel in einem versionskontrollierten Verzeichnis.

4.2. Editiermodus

Der Testmanager hat zwei Modi: Im *Bedienmodus* können die Widgets verwendet werden, um Prozesswerte zu beobachten und zu ändern. Im *Editiermodus* können Widgets hinzugefügt, verschoben, in der Größe verändert und konfiguriert werden. Schalten Sie den Editiermodus mit **Editiermodus** in der Werkzeugleiste oder im Menü **Optionen** um.

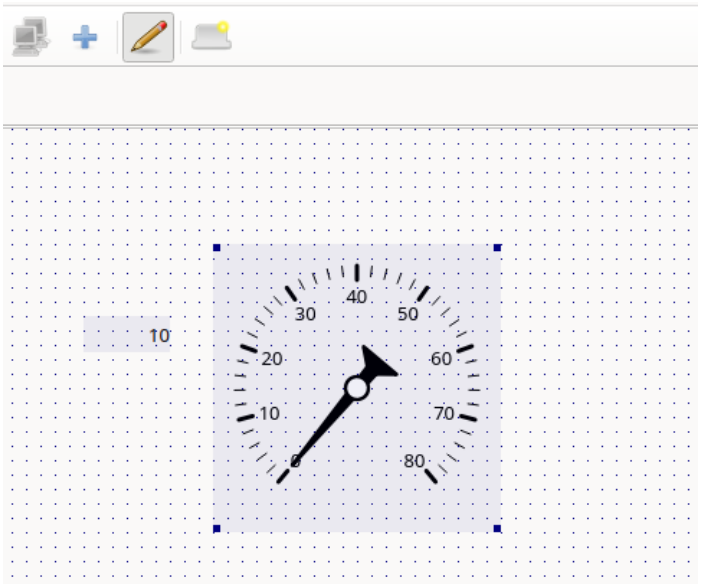

Abbildung 4.3.: Tab im Editiermodus

Klicken Sie mit der rechten Maustaste auf einen leeren Bereich der Arbeitsfläche, um Widgets hinzuzufügen, Widgets aus der Zwischenablage einzufügen, das Stylesheet des Tabs zu bearbeiten oder alle Widgets zu markieren, die noch nicht mit Prozessvariablen verbunden sind.

Klicken Sie mit der rechten Maustaste auf ein Widget, um es zu kopieren oder zu entfernen, seine Prozessdaten-Verbindungen zu bearbeiten, zu seinen Variablen zu springen, es auszurichten, seine Stapelreihenfolge zu ändern, sein Stylesheet oder Skript zu bearbeiten oder seine Legende anzuzeigen.

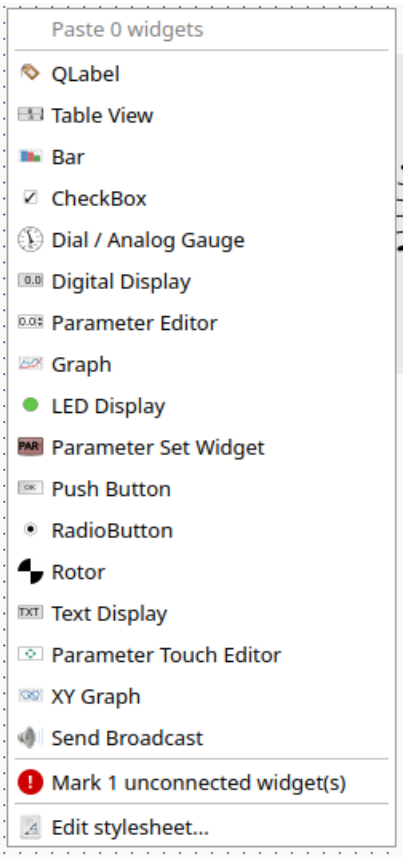

Abbildung 4.4.: Kontextmenü der Arbeitsfläche

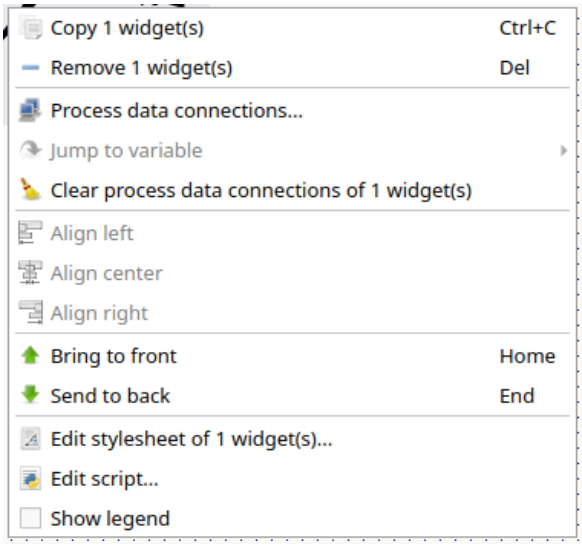

Abbildung 4.5.: Kontextmenü eines Widgets

4.2.1. Nicht verbundene Widgets

Im Editiermodus werden Widgets, die noch mit keiner Prozessvariable verbunden sind, markiert (siehe Abbildung unten). Die Statusleiste meldet die Anzahl der nicht verbundenen Widgets sowie der Widgets, deren Variablen in den verbundenen Datenquellen nicht gefunden werden konnten.

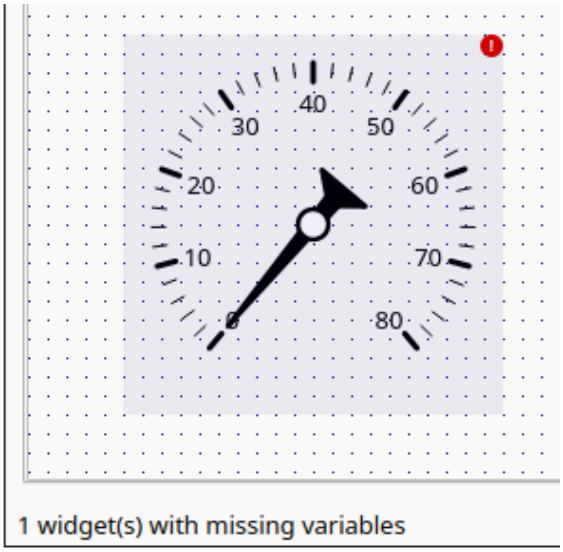

Abbildung 4.6.: Markierte nicht verbundene Widgets

4.3. Eigenschafts-Editor

Das Dock *Eigenschafts-Editor* zeigt die Eigenschaften des ausgewählten Widgets. Änderungen werden sofort übernommen. Klicken Sie mit der rechten Maustaste auf eine Eigenschaft und wählen Sie **Zurücksetzen**, um ihren Standardwert wiederherzustellen.

4.4. Tabs

Ein Layout besteht aus einem oder mehreren Tabs. Verwenden Sie **Reiter hinzufügen**, um einen neuen Tab zu erstellen. Über das Kontextmenü des Tab-Kopfes kann ein Tab umbenannt, abgelöst oder entfernt werden. Ein abgelöster Tab wird in einem separaten Fenster angezeigt, was bei Aufbauten mit mehreren Bildschirmen nützlich ist.

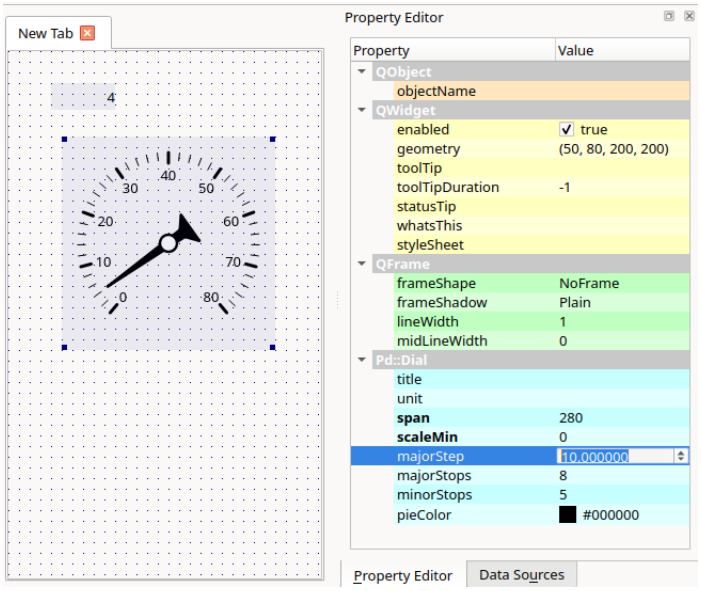

Abbildung 4.7.: Eigenschafts-Editor

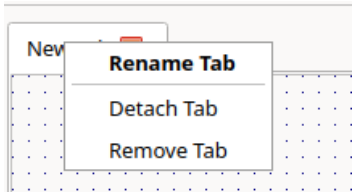

Abbildung 4.8.: Kontextmenü eines Tab-Kopfes

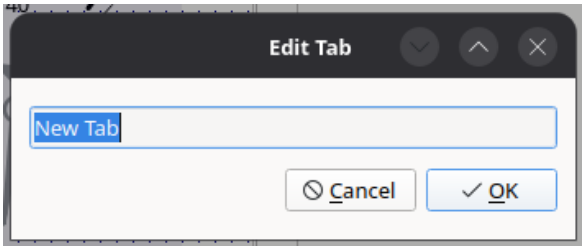

Abbildung 4.9.: Dialog „Reiter editieren“

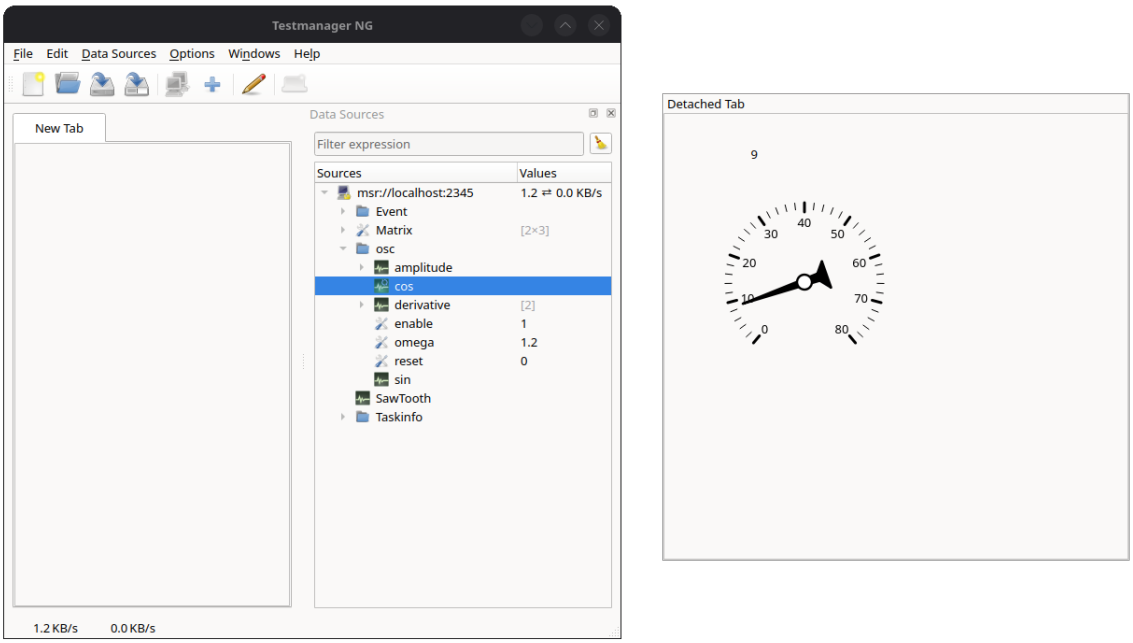

Abbildung 4.10.: Abgelöster Tab

5. Widgets

Der Testmanager verwendet die Widgets der Bibliothek QtPdWidgets.

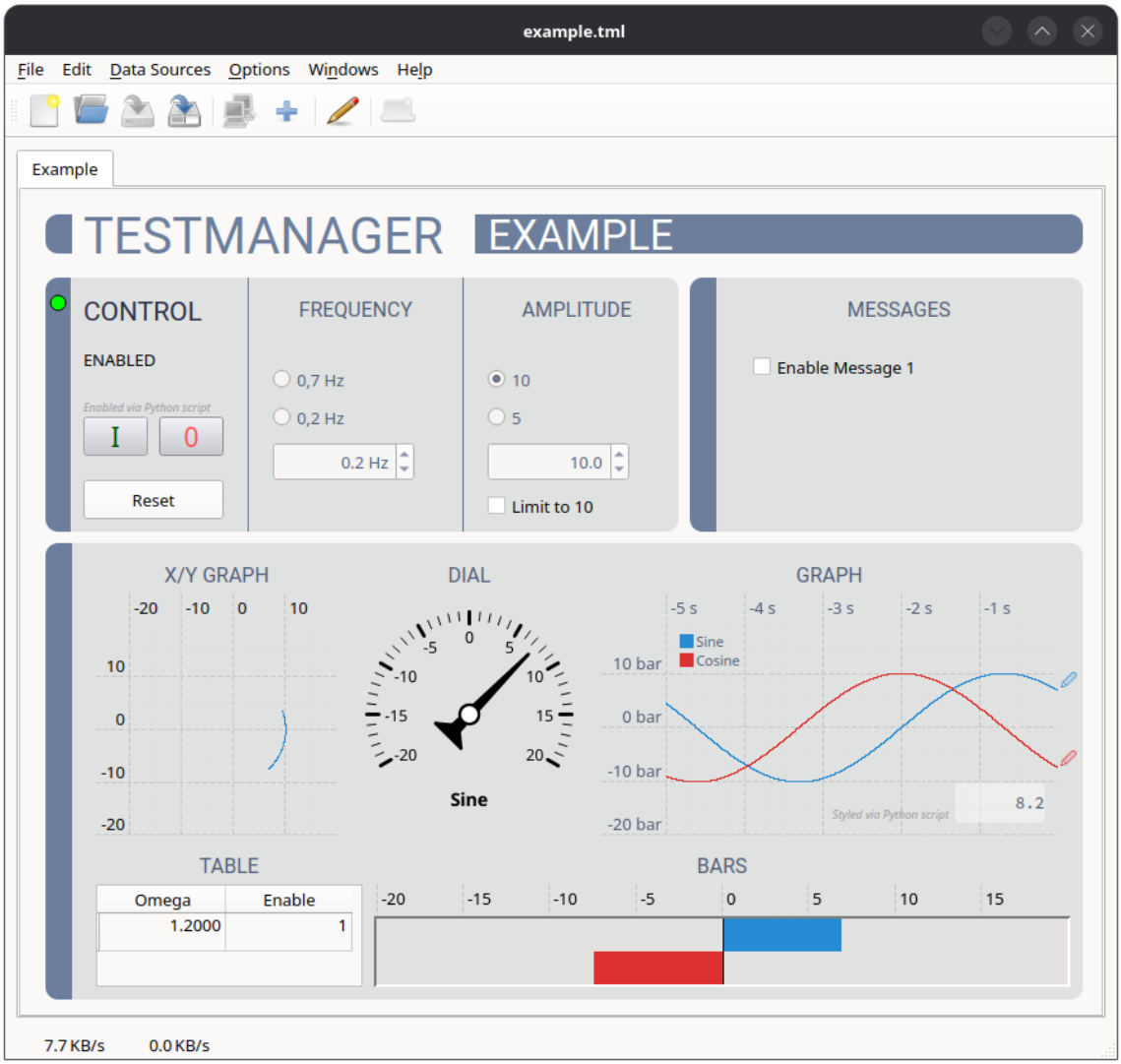

Abbildung 5.1.: Alle Widgets auf einen Blick

Widget	Zweck	Variablen
Bar	Balkendiagramm	Signale, Vektoren
CheckBox	Booleschen Parameter schalten	Parameter
Dial	Analoges Zeigerinstrument	Signal, Parameter
Digital	Numerische Anzeige	Signal
DoubleSpinBox	Numerische Eingabe	Parameter
Graph	Zeitverlaufsdiagramm	Signale
XYGraph	X/Y-Diagramm	Zwei Signale
Led	Boolesche Anzeige	Signal
ParameterSetWidget	Parametersätze anwenden	–
PushButton	Parameter beim Klicken setzen	Parameter
QLabel	Statischer Text oder Bild	–
RadioButton	Wert eines Parameters auswählen	Parameter
Rotor	Rotierende Anzeige	Signale
SendBroadcast	Broadcast-Nachricht senden	–
TableView	Tabelle mit Vektorwerten	Signale, Parameter
Text	Text abhängig von einem Wert	Signal
TouchEdit	Numerische Eingabe für Touchscreens	Parameter

5.1. Dial

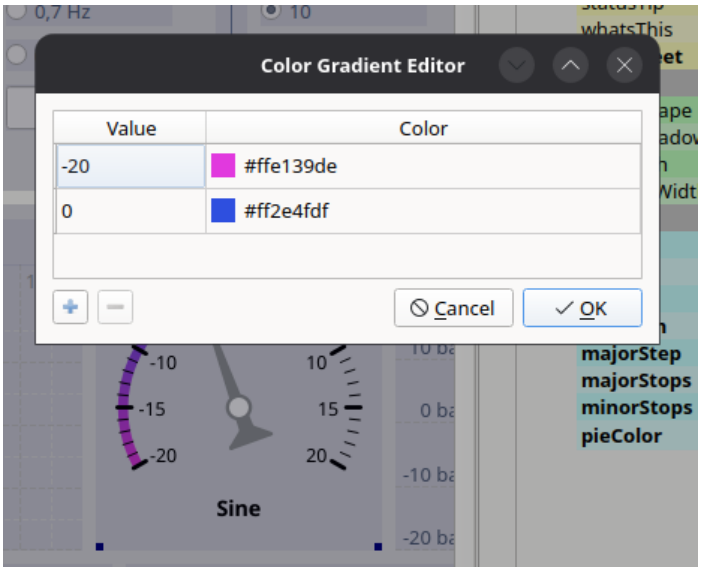

Abbildung 5.2.: Konfigurationsdialog des Dial

5.2. Tabellenansicht

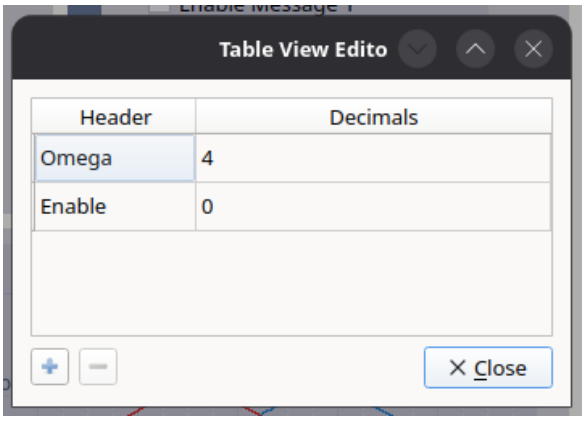

Abbildung 5.3.: Konfigurationsdialog der Tabellenansicht

5.3. Text

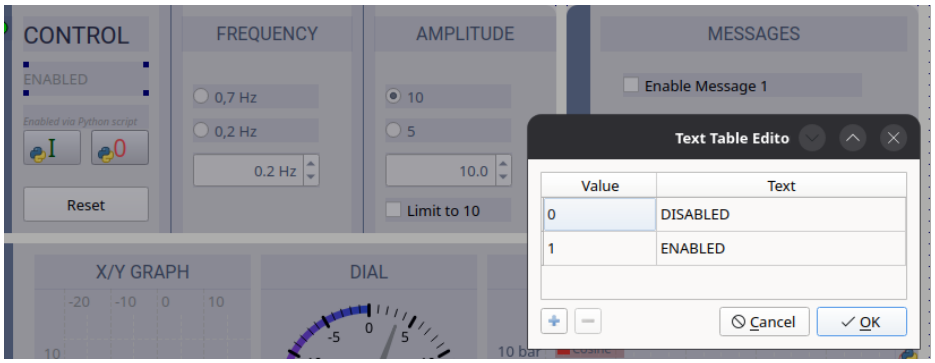

Abbildung 5.4.: Konfigurationsdialog des Text-Widgets

6. Signale und Parameter verbinden

6.1. Drag and Drop

Um eine Prozessvariable mit einem Widget zu verbinden, ziehen Sie sie im Editiermodus aus dem Datenquellen-Baum auf das Widget. Ein Tooltip zeigt an, ob die Variablen hinzugefügt werden oder die bestehenden Verbindungen ersetzen.

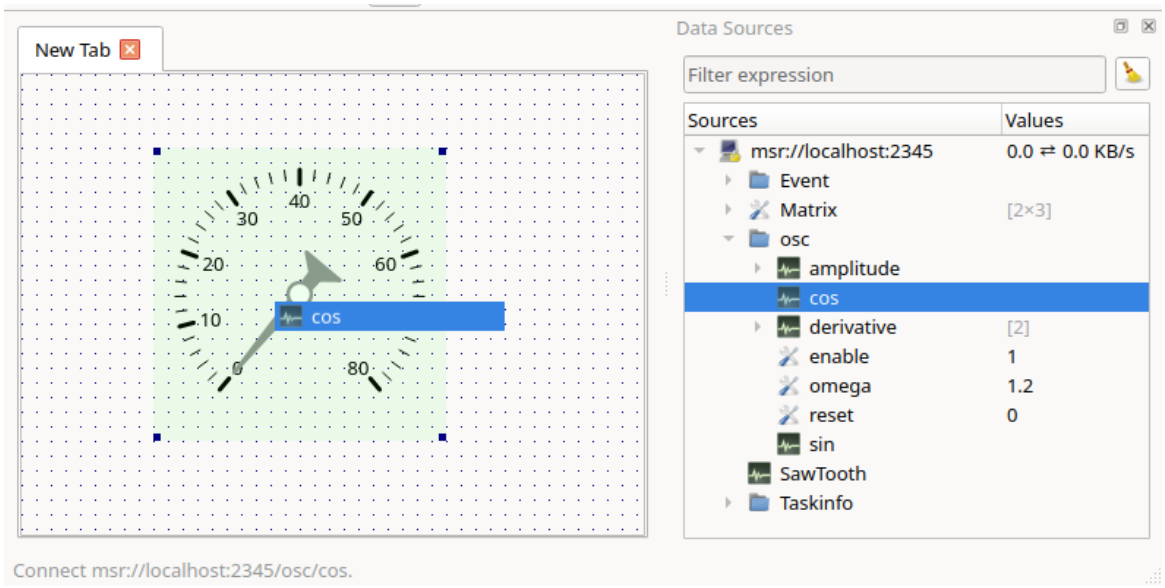

Abbildung 6.1.: Verbinden einer Variable per Drag and Drop

Wenn Sie den Mauszeiger über ein Widget bewegen, listet ein Tooltip alle verbundenen Variablen auf.

6.2. Prozessdaten-Verbindungen

Wählen Sie **Datenverbindungen...** aus dem Kontextmenü eines Widgets, um dessen Verbindungen im Detail einzustellen.

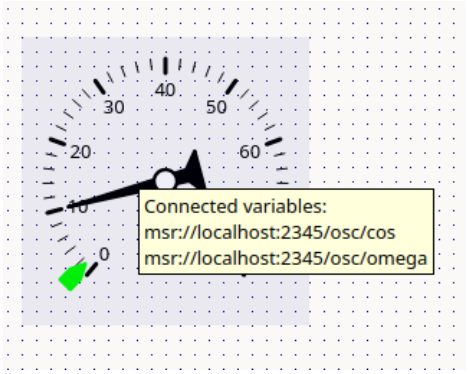

Abbildung 6.2.: Tooltip mit verbundenen Variablen

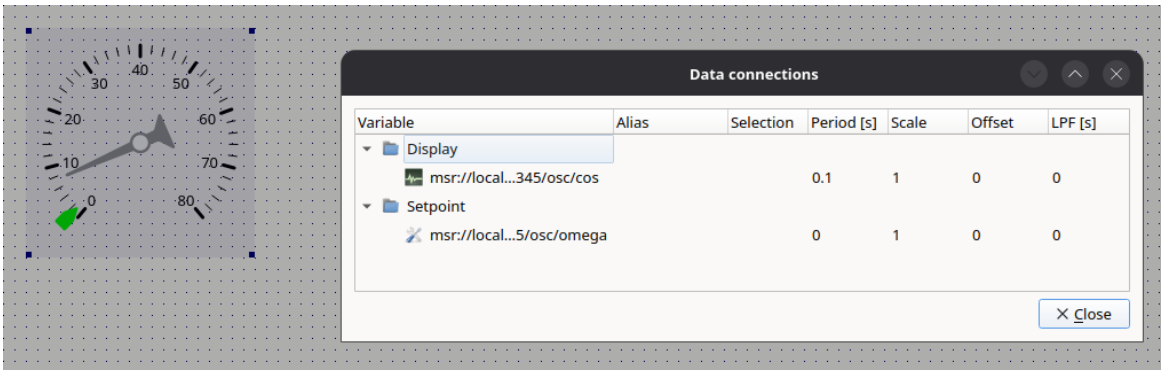

Abbildung 6.3.: Dialog „Datenverbindungen“

Spalte	Beschreibung
Alias	Anstelle des Variablenpfads angezeigter Name (z. B. in Legenden)
Feld	Element einer Vektor- oder Matrix-Variable
Periode [s]	Abtastzeit; `0` bedeutet ereignisbasierte Übertragung
Skalierung	Faktor, mit dem der Wert multipliziert wird
Versatz	Offset, der zum skalierten Wert addiert wird
Filterkonstante [s]	Zeitkonstante eines Tiefpassfilters erster Ordnung
Farbe	Farbe der Variable in Diagrammen

Wichtig

Innerhalb einer Verbindung zu einem Prozess kann ein Signal entweder ereignisbasiert (Periode `0`) *oder* periodisch (Periode > 0) übertragen werden. Verwenden Sie für ein Signal daher einheitlich entweder ereignisbasierte oder periodische Übertragung.

Klicken Sie mit der rechten Maustaste auf eine Spalte, um einen Wert auf alle Verbindungen oder auf die ausgewählten Verbindungen und deren Subelemente anzuwenden.

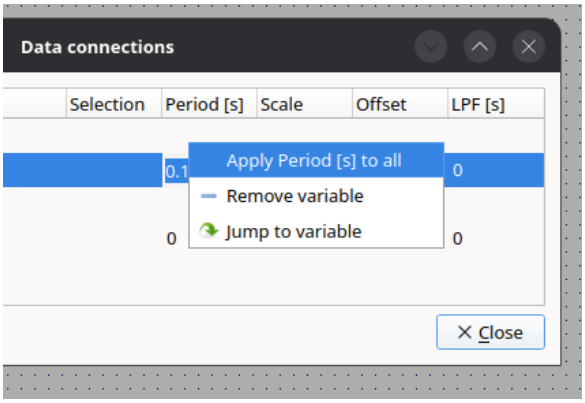

Abbildung 6.4.: Anwenden eines Werts auf alle Verbindungen

6.3. Vektorelemente auswählen

Wird eine Vektor- oder Matrix-Variable mit einem Widget verbunden, das einen Skalar erwartet, kann das Element im Dialog *Variablen-Selektor* ausgewählt werden.

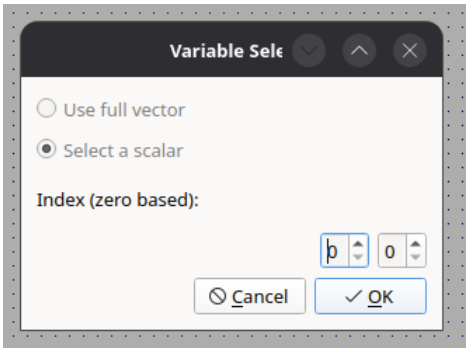

Abbildung 6.5.: Variablen-Selektor

6.4. Dock „Datenverbindungen“

Das Dock *Datenverbindungen* zeigt die Verbindungen der ausgewählten Widgets. Mit **Zur Variable springen** finden Sie eine verbundene Variable im Datenquellen-Baum.

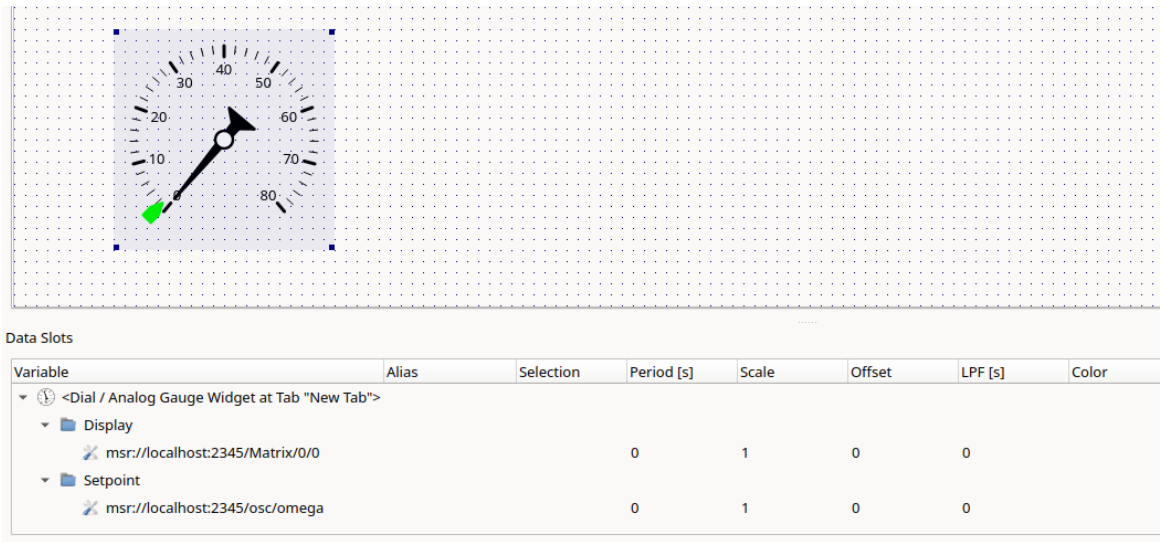

Abbildung 6.6.: Dock „Datenverbindungen“

7. Legende

Jedes Widget kann eine *Legende* anzeigen, die die Namen (oder Aliase) und Farben der verbundenen Variablen darstellt. Aktivieren Sie sie über **Legende anzeigen** im Kontextmenü des Widgets.

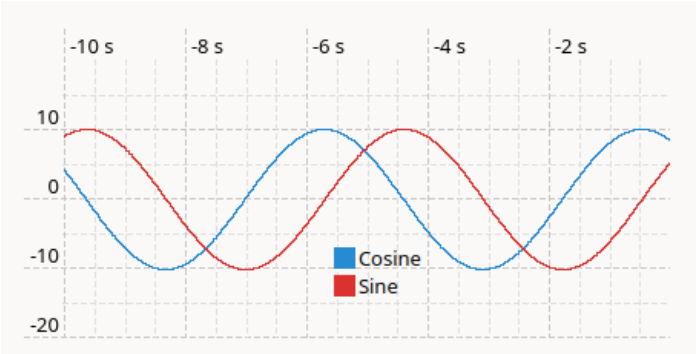

Abbildung 7.1.: Graph mit Legende

Über das Kontextmenü der Legende können Sie diese verbergen, die Textausrichtung ändern und ihr Stylesheet bearbeiten.

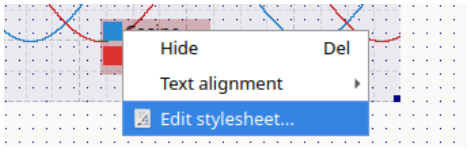

Abbildung 7.2.: Kontextmenü der Legende

8. Stylesheets

Das Erscheinungsbild von Testmanager-Layouts lässt sich mit Qt-Stylesheets anpassen, deren Syntax CSS ähnelt.

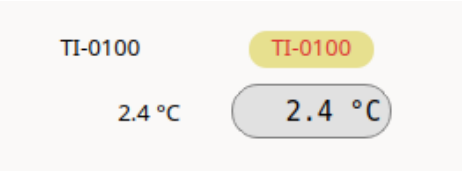

Abbildung 8.1.: Das gleiche Widget ohne und mit Stylesheet

8.1. Ebenen

Stylesheets können auf drei Ebenen definiert werden:

Ebene	Wo	Gilt für
Global	**Optionen → Globales Stylesheet…**	Alle Tabs des Layouts
Tab	Kontextmenü der Arbeitsfläche	Alle Widgets des Tabs
Widget	Kontextmenü eines oder mehrerer Widgets	Die ausgewählten Widgets

Spezifischere Stylesheets überschreiben allgemeinere.

8.2. Stylesheet-Editor

Die Werkzeugleiste des Editors hilft beim Einfügen von:

- **Ressourcen** (Bildern) für `background-image`, `border-image` und `image`,
- **Farben** für Farbeigenschaften (mit Alphakanal),
- **Schriftarten**,
- **Vorlagen** mit häufig verwendeten Regeln.

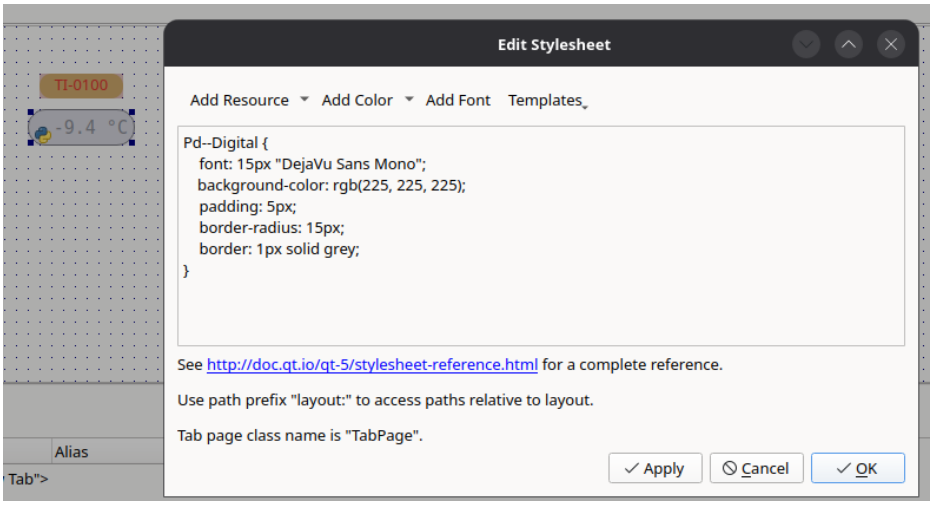

Abbildung 8.2.: Stylesheet-Editor

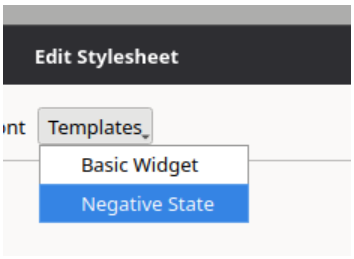

Abbildung 8.3.: Stylesheet-Vorlagen

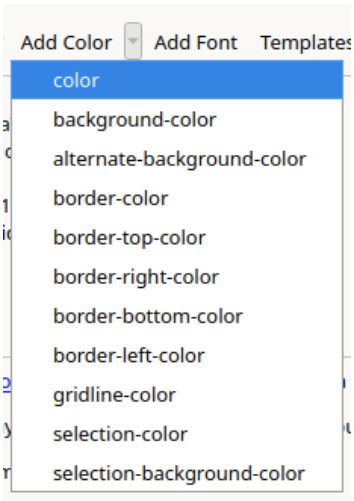

Abbildung 8.4.: Einfügen einer Farbe

8.3. Selektoren

Widgets werden über ihren Klassennamen ausgewählt. Da die Klassennamen der QtPdWidgets den Namensraum `Pd::` enthalten, der in Stylesheets nicht zulässig ist, muss der doppelte Doppelpunkt durch zwei Bindestriche ersetzt werden:

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

Der Testmanager stellt die folgenden speziellen Klassen bereit:

Klasse	Beschreibung
`TabPage`	Die Arbeitsfläche eines Tabs
`Legend`	Die Legende eines Widgets

8.4. Ressourcen

Über den Editor eingefügte Bilder werden mit dem Präfix `layout:` relativ zur Layout-Datei referenziert, sodass das Layout-Verzeichnis als Ganzes verschoben werden kann:

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

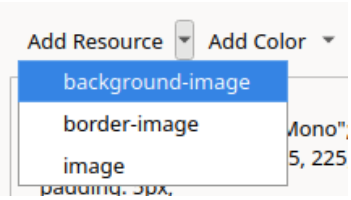

Abbildung 8.5.: Einfügen einer Ressource

8.5. Dynamische Eigenschaften

Regeln können von dynamischen Eigenschaften abhängen, die aus Python-Scripts gesetzt werden können (siehe Python-Scripts):

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

9. Python-Scripts

Jedes Widget kann ein Python-Script besitzen, das ausgeführt wird, sobald sich eine seiner Script-Variablen ändert. So lassen sich zum Beispiel Widgets abhängig von Prozesswerten aktivieren oder deaktivieren oder in ihrem Erscheinungsbild verändern.

9.1. Script-Editor

Wählen Sie im Editiermodus ein Widget aus und öffnen Sie das Dock *Python-Script* (oder wählen Sie **Script bearbeiten...** aus dem Kontextmenü des Widgets).

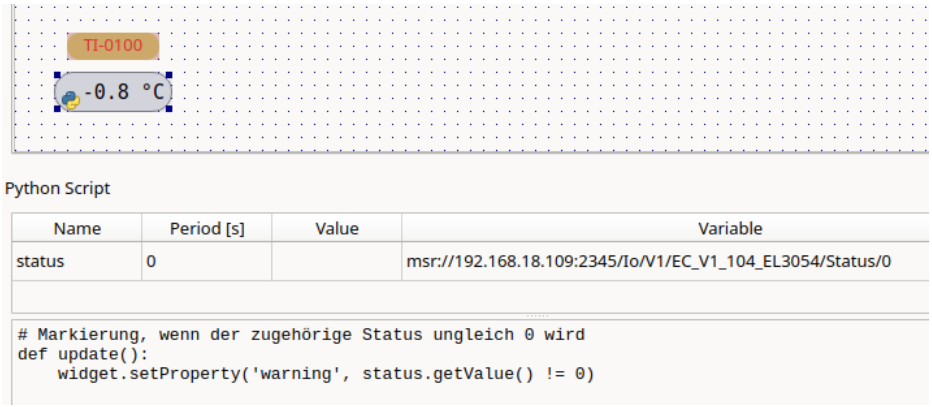

Abbildung 9.1.: Python-Script-Editor

Die Tabelle im oberen Bereich enthält die *Script-Variablen*. Ziehen Sie Prozessvariablen aus dem Datenquellen-Baum in die Tabelle. Die Spalte **Name** legt fest, unter welchem Namen die Variable im Script verfügbar ist; **Periode [s]** legt die Abtastzeit fest.

> **Hinweis**
> Ein Signal kann innerhalb einer Verbindung zu einem Prozess entweder ereignisbasiert (Periode `0`) *oder* periodisch (Periode > 0) übertragen werden (siehe Kapitel Datenquellen).

Das Script muss eine Funktion `update()` definieren, die aufgerufen wird, wenn sich eine Variable ändert. Innerhalb des Scripts verweist `widget` auf das Widget, zu dem das Script gehört.

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

Klicken Sie mit der rechten Maustaste in den Editor, um Vorlagen einzufügen.

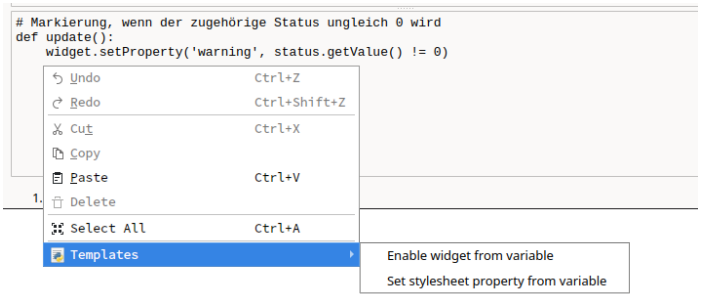

Abbildung 9.2.: Script-Vorlagen

9.2. Scripts und Stylesheets kombinieren

Ein Script kann dynamische Eigenschaften setzen, die vom Stylesheet ausgewertet werden:

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

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

9.3. Fehlermeldungen

Syntaxfehler werden mit der Zeilennummer angezeigt. Wird keine aufrufbare Funktion `update()` gefunden, wird dies ebenfalls angezeigt.

Abbildung 9.3.: Hervorhebung per Script und Stylesheet

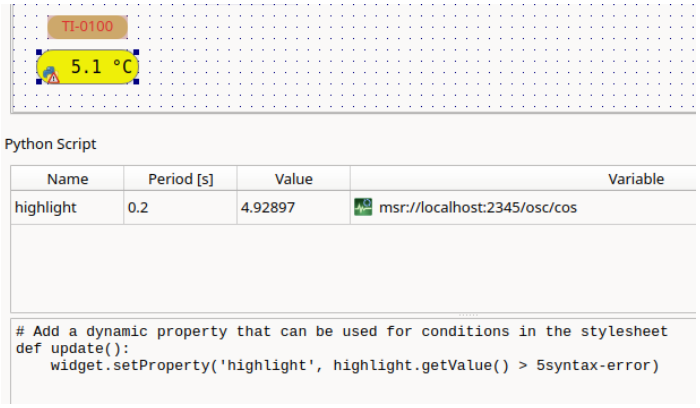

Abbildung 9.4.: Script-Fehler

9.4. Python-Kommandozeile

Das Dock *Python-Kommandozeile* stellt einen interaktiven Python-Interpreter bereit, der seine Umgebung mit den Scripts teilt. Ausgaben von `print()` in Scripts erscheinen hier.

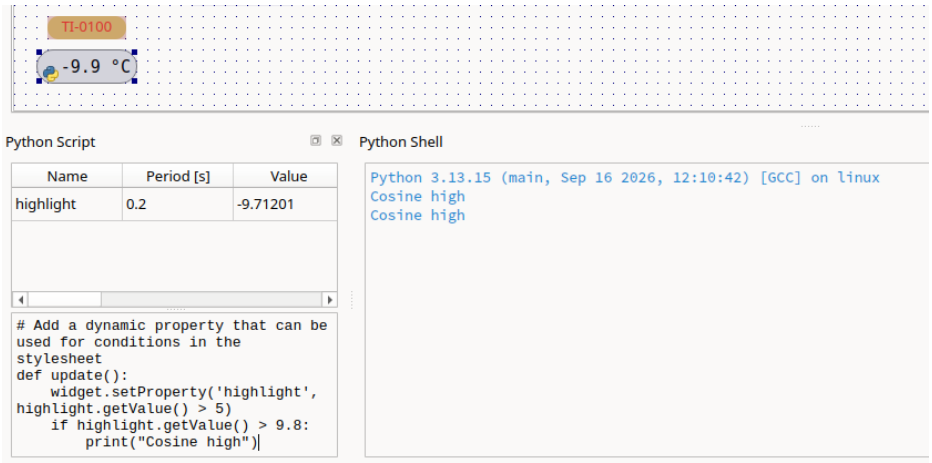

Abbildung 9.5.: Python-Kommandozeile

Alle verfügbaren Objekte und Funktionen finden Sie in der Python-API-Referenz.

10. Beispiel-Layout

Der Testmanager wird mit einem Beispiel-Layout ausgeliefert, das für den Beispielprozess von PdServ ausgelegt ist. Es demonstriert die meisten der in diesem Handbuch beschriebenen Funktionen und ist ein guter Ausgangspunkt für eigene Layouts.

10.1. Das Beispiel ausführen

Starten Sie den PdServ-Beispielprozess (nur unter Linux verfügbar):

```
pdserv-example-st
```

Öffnen Sie anschließend das Beispiel-Layout über **Datei → Öffnen...**. Es wird als `share/testmanager/example.tml` installiert, unter Linux in der Regel unter dem unten angegebenen Pfad. Alternativ können Sie es auf der Kommandozeile übergeben:

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

Das Layout verweist auf die Datenquelle `msr://localhost:2345`. Läuft der Beispielprozess auf einem anderen Rechner, verwenden Sie Datenquelle ersetzen.

10.2. Was das Beispiel zeigt

- **Steuerung:** Eine LED und ein Text-Widget zeigen den Zustand des Parameters `/osc/enable`. Die beiden Push-Buttons **I** und **0** schalten ihn ein und aus. Sie werden von Python-Scripts abhängig vom aktuellen Wert aktiviert und deaktiviert, sodass nur der sinnvolle Button gedrückt werden kann. **Reset** setzt den Parameter `/osc/reset`.
- **Frequenz:** Radio-Buttons und eine Spin-Box setzen `/osc/omega`.
- **Amplitude:** Radio-Buttons und eine Spin-Box setzen `/osc/amplitude/Setpoint`. Die Check-Box setzt `/osc/amplitude/Limit`.
- **Meldungen:** Die Check-Box setzt `/Event/State`, wodurch der Prozess eine Meldung ausgibt, siehe Meldungen.

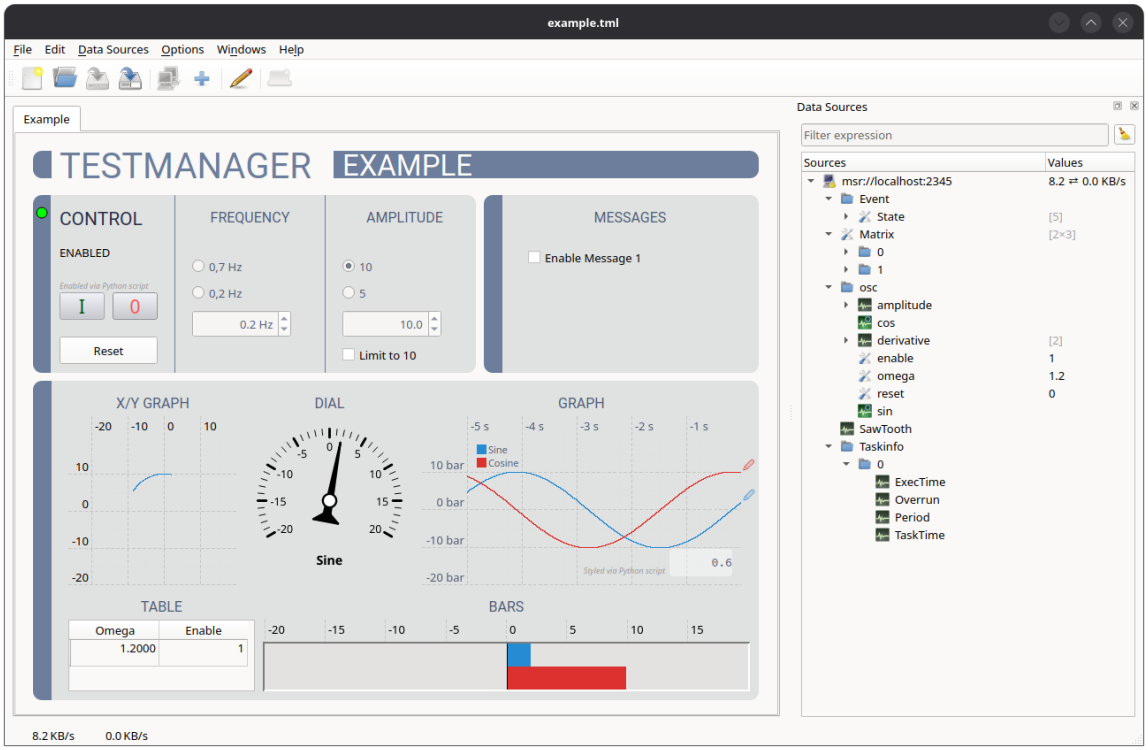

Abbildung 10.1.: Das Beispiel-Layout, verbunden mit pdserv-example-st

- **X/Y-Graph**, **Dial** und **Graph**: Verschiedene Darstellungen der Signale `/osc/sin` und `/osc/cos`. Der Graph zeigt eine Legende mit den Aliasen *Sine* und *Cosine* sowie eine Digitalanzeige des aktuellen Werts.
- **Tabelle:** Eine Tabellenansicht, die die Parameter `/osc/omega` und `/osc/enable` anzeigt und bearbeitbar macht.
- **Balken:** Ein Balkendiagramm, das beide Signale zeigt.

Das Beispiel enthält außerdem einige Techniken, die einen genaueren Blick im Editiermodus lohnen:

- **Stylesheets:** Das globale Stylesheet legt die Farben und Schriftarten des gesamten Layouts fest. Siehe Stylesheets.
- **Dynamische Eigenschaften:** Ein Script setzt die Eigenschaft `negative` abhängig vom Vorzeichen des Sinussignals, die von einer Stylesheet-Regel ausgewertet wird:

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

- **Bewegte Widgets:** Die Stifte am rechten Rand des Graphen folgen den aktuellen Werten. Ihre Scripts ändern die Eigenschaft `geometry`:

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

Details finden Sie unter Python-Scripts.

11. Parametersätze

Ein *Parametersatz* ist eine benannte Sammlung von Parametern. Parametersätze werden im Layout gespeichert. Sie ermöglichen es, die aktuellen Werte der Parameter in einer Datei zu sichern und später wiederherzustellen.

11.1. Parametersätze anlegen

Öffnen Sie das Dock *Parametersätze* und legen Sie mit **Neuer Satz** einen neuen Satz an. Fügen Sie Parameter hinzu, indem Sie sie aus dem Datenquellen-Baum in den Satz ziehen. Wird ein Ordner oder eine ganze Datenquelle gezogen, werden alle darunterliegenden Parameter hinzugefügt. Das Kontextmenü bietet **Alle Parameter hinzufügen** aller verbundenen Datenquellen an.

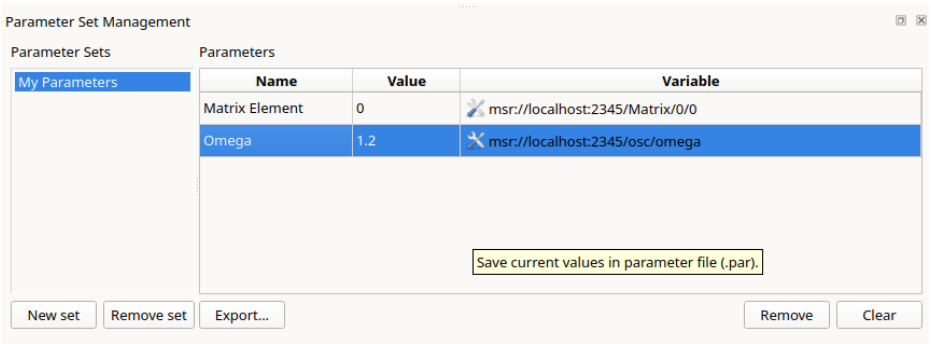

Abbildung 11.1.: Dock „Parametersätze“

Die Anzahl der Parameter eines Satzes wird in dessen Titel und Tooltip angezeigt.

11.2. Exportieren

Exportieren... speichert die aktuellen Werte aller Parameter des Satzes zusammen mit einer Beschreibung in einer Parameterdatei (`*.par`).

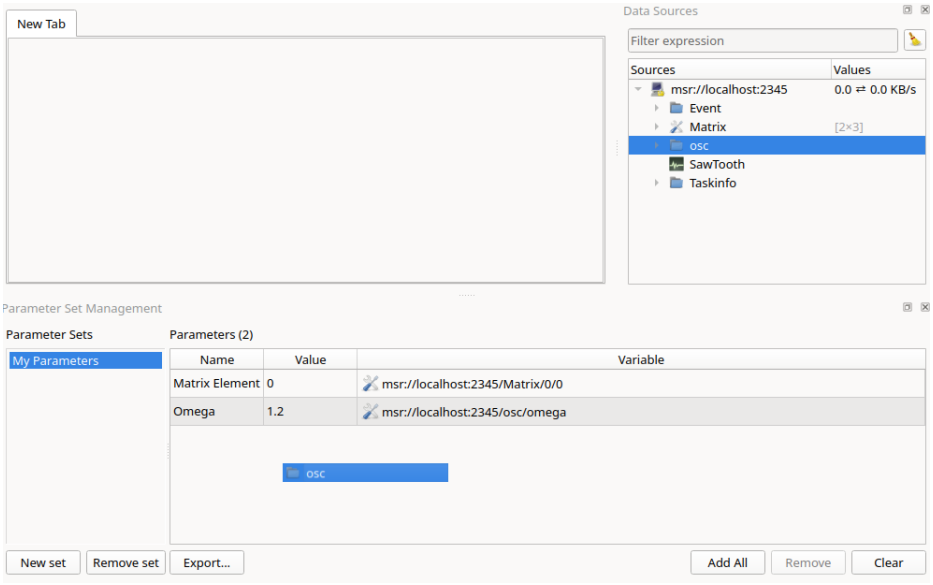

Abbildung 11.2.: Hinzufügen eines Ordners per Drag & Drop

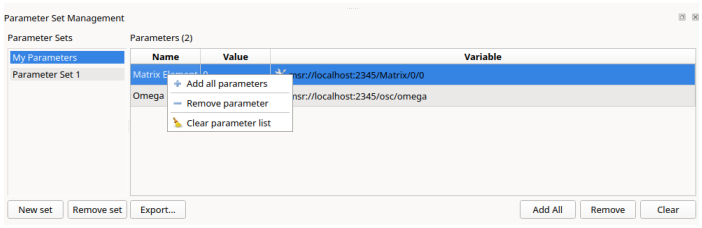

Abbildung 11.3.: Kontextmenü eines Parametersatzes

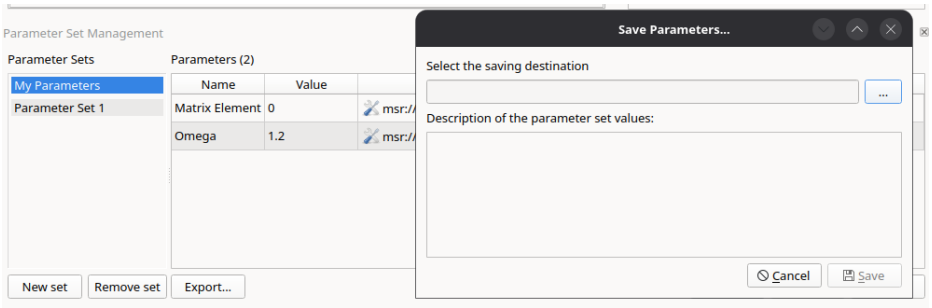

Abbildung 11.4.: Speichern einer Parameterdatei

11.3. Importieren

Parameterdatei Laden... öffnet eine Parameterdatei und zeigt die gespeicherten und die aktuellen Werte nebeneinander an, zusammen mit den Metadaten (Ersteller, Erstellungsdatum, Beschreibung). **Anwenden** schreibt die gespeicherten Werte in den Prozess.

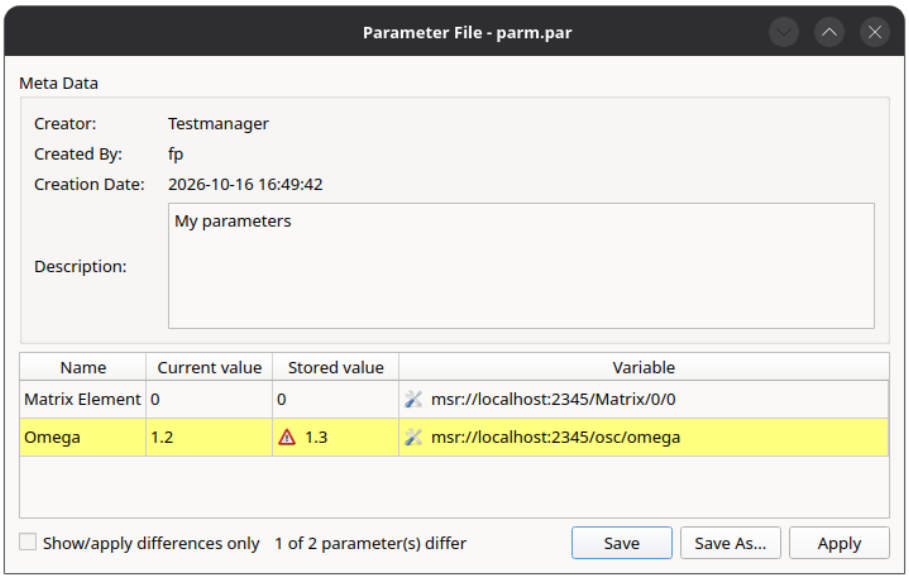

Abbildung 11.5.: Laden einer Parameterdatei

Mit **Nur Unterschiede anzeigen/anwenden** werden nur die Parameter angezeigt und geschrieben, deren gespeicherter Wert vom aktuellen Wert abweicht.

Vektor- und Matrixparameter werden als Tabellen angezeigt und bearbeitet.

11.4. Parametersatz-Widget

Das *ParameterSetWidget* ermöglicht es Bedienern, Parameterdateien direkt aus dem Layout heraus zu laden.

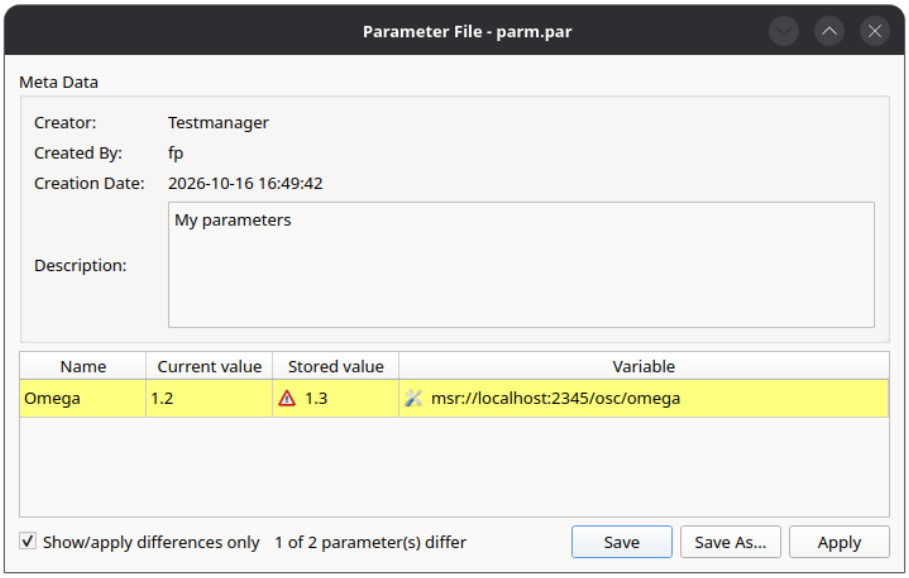

Abbildung 11.6.: Nur Unterschiede anzeigen

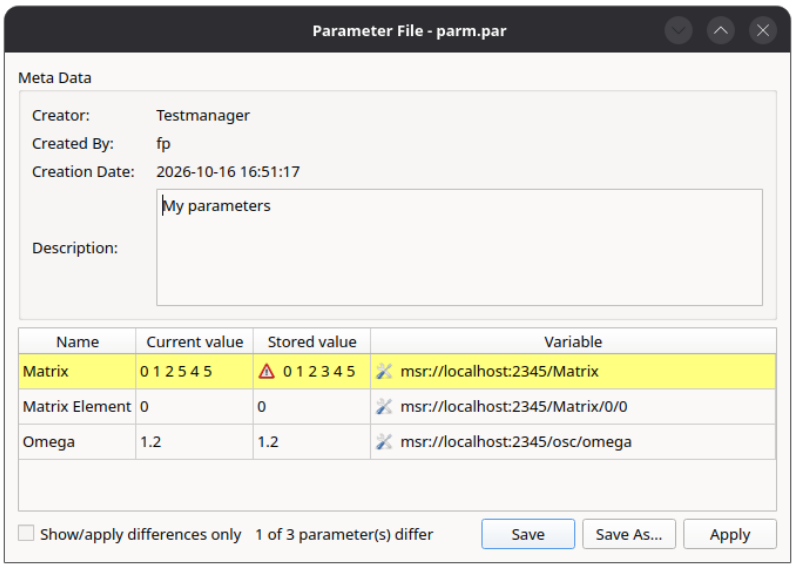

Abbildung 11.7.: Mehrdimensionaler Parameter

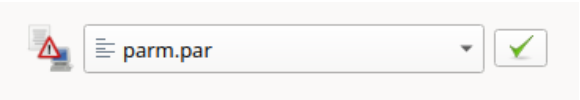

Abbildung 11.8.: Parametersatz-Widget

12. Broadcasts

Broadcasts sind Textnachrichten, die an einen Prozess gesendet und an alle verbundenen Clients weitergeleitet werden. Sie können zum Beispiel verwendet werden, um Aktionen im Prozessprotokoll zu dokumentieren.

> **Hinweis**
> Broadcasts können auch verwendet werden, um Ereignisse in den Messdaten zu protokollieren – zum Beispiel mit EtherLabs Data Logging Service (DLS).

Das Widget *SendBroadcast* sendet beim Anklicken einen Broadcast.

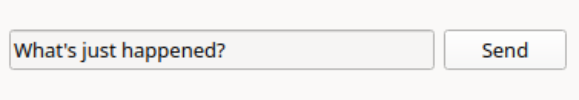

Abbildung 12.1.: SendBroadcast-Widget

Das Dock *Broadcasts* listet alle empfangenen Broadcasts zusammen mit ihrer Datenquelle auf. **Löschen** leert die Liste.

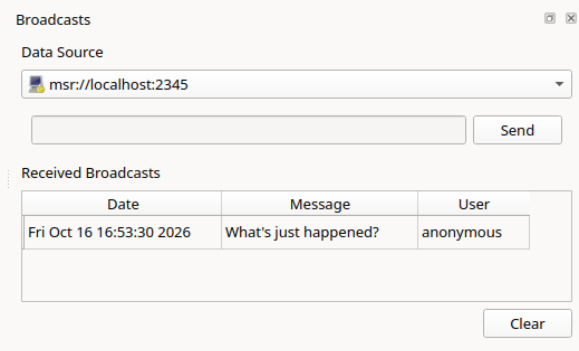

Abbildung 12.2.: Empfangene Broadcasts

13. Meldungen

Prozesse können *Meldungen* ausgeben (Informationen, Warnungen, Fehler, kritische Fehler), zum Beispiel wenn ein Grenzwert überschritten wird.

13.1. Konfiguration

Die Texte der Meldungen werden in einer *Plain-Messages-Datei* definiert, die für jede Datenquelle über **Meldungen konfigurieren...** im Kontextmenü der Datenquelle konfiguriert wird. Der Pfad wird relativ zur Layout-Datei gespeichert. Übersetzungen werden entsprechend der Sprache der Benutzeroberfläche ausgewählt.

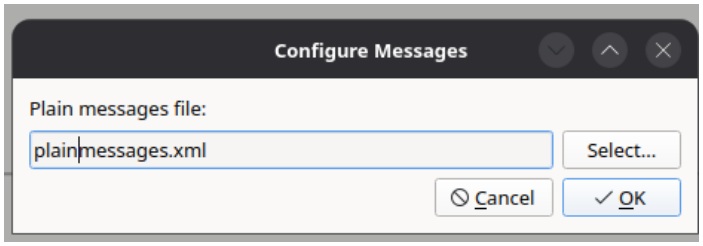

Abbildung 13.1.: Meldungen konfigurieren

13.2. Meldungen anzeigen

Meldungen anzeigen... im Kontextmenü einer Datenquelle zeigt die Meldungen dieser Datenquelle an.

Das Dock *Meldungen* fasst die Meldungen aller verbundenen Datenquellen zusammen. Mit **Zeige historische Meldungen** werden auch Meldungen angezeigt, die nicht mehr aktiv sind.

Abbildung 13.2.: Meldungen einer Datenquelle

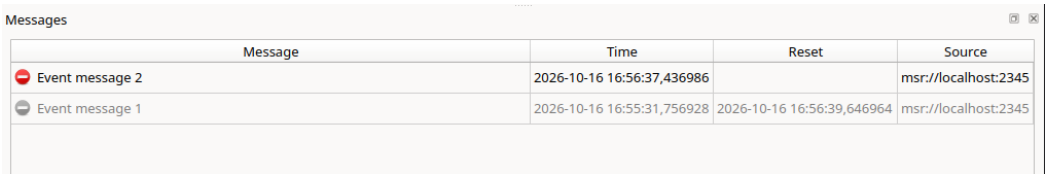

Abbildung 13.3.: Dock „Meldungen“

14. Datenquellen ersetzen

Ein Layout referenziert Prozessvariablen per URL, einschließlich Host und Port. Um ein Layout mit einem anderen Prozess zu verwenden, zum Beispiel beim Wechsel von einem Entwicklungsrechner auf das Zielsystem, wählen Sie **Ersetzen...** im Kontextmenü der Datenquelle.

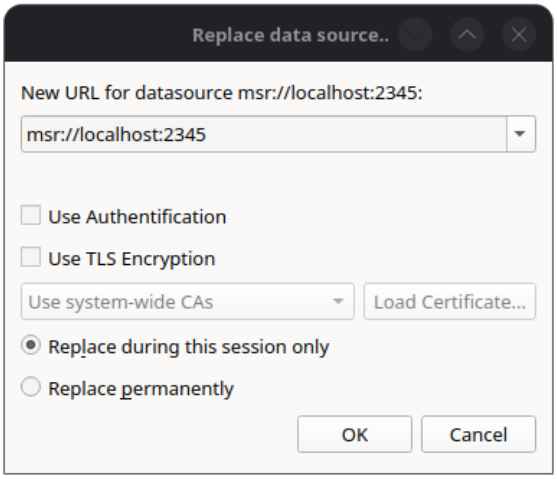

Abbildung 14.1.: Datenquelle ersetzen

- **Nur in dieser Sitzung ersetzen** lässt die Layout-Datei unverändert. Dies ist nützlich, um ein Layout gegen eine Simulation zu testen.
- **Dauerhaft ersetzen** ändert die URLs im Layout. Speichern Sie das Layout anschließend.

Der Dialog ermöglicht außerdem, TLS-Verschlüsselung und Authentifizierung für die neue Datenquelle zu aktivieren.

A. Tastenkürzel

Tastenkürzel	Aktion
Strg+N	Neues Layout
Strg+O	Layout laden
Strg+S	Layout speichern
Strg+C	Ausgewählte Widgets kopieren
Strg+V	Widgets einfügen
Entf	Legende verbergen (Menü der Legende)

B. Python-API

B.1. Skriptvariablen

Jede Skriptvariable ist unter dem Namen verfügbar, der in der Skriptvariablen-Tabelle angegeben ist.

Methode	Beschreibung
`getValue()`	Aktueller Wert (eine Zahl oder bei Vektoren ein NumPy-Array)

B.2. Das Objekt `widget`

Methode	Beschreibung
`setEnabled(enabled)`	Aktiviert oder deaktiviert das Widget
`setProperty(name, value)`	Setzt eine (dynamische) Qt-Eigenschaft

B.3. Das Modul `tm`

```
import tm
```

Funktion	Beschreibung
`tm.tabCount()`	Anzahl der Tabs des aktuellen Layouts
`tm.maximize(max)`	Maximiert das Hauptfenster (oder stellt es wieder her)