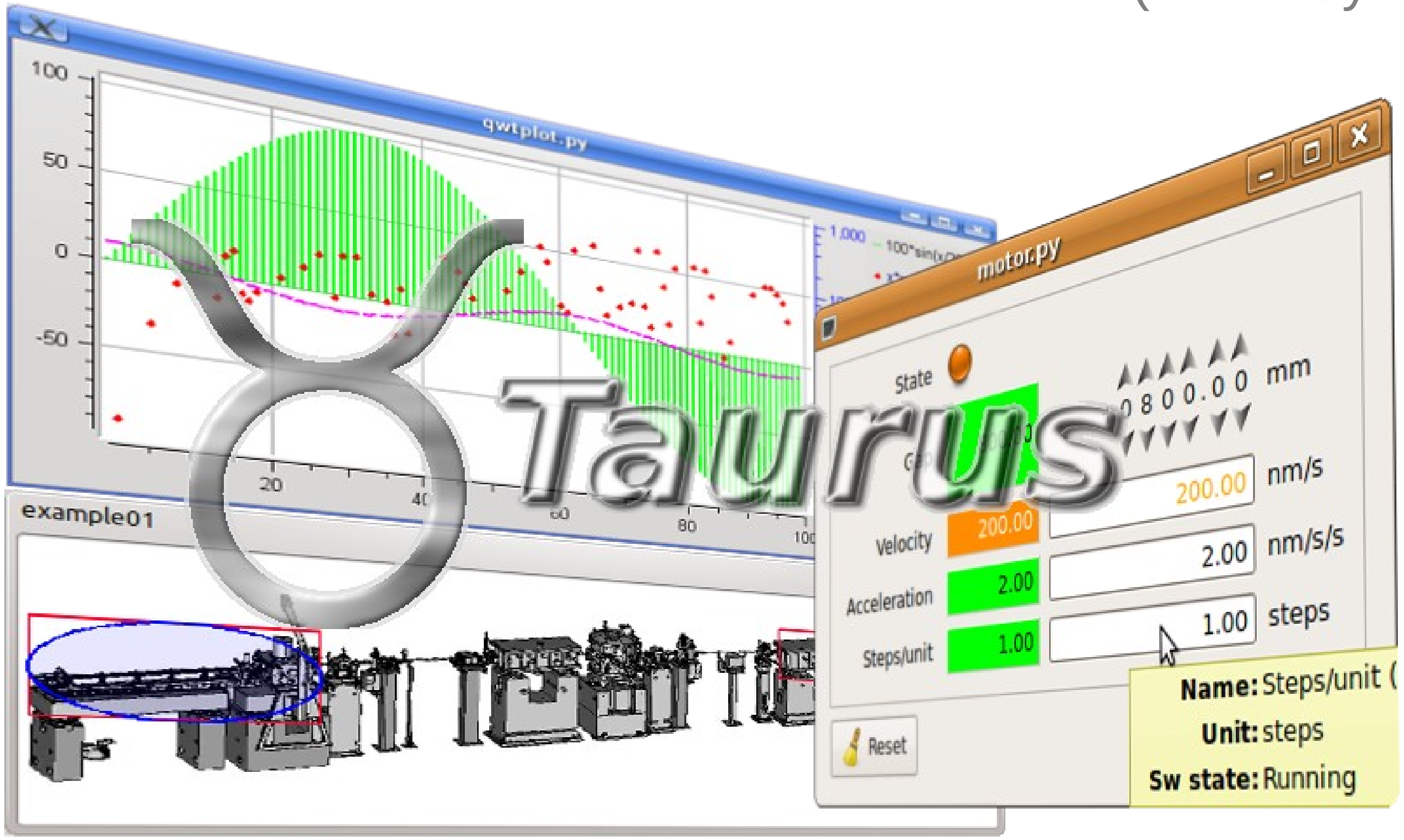
Taurus Tutorial

by Zbigniew Reszela

(ALBA Synchrotron, Spain)

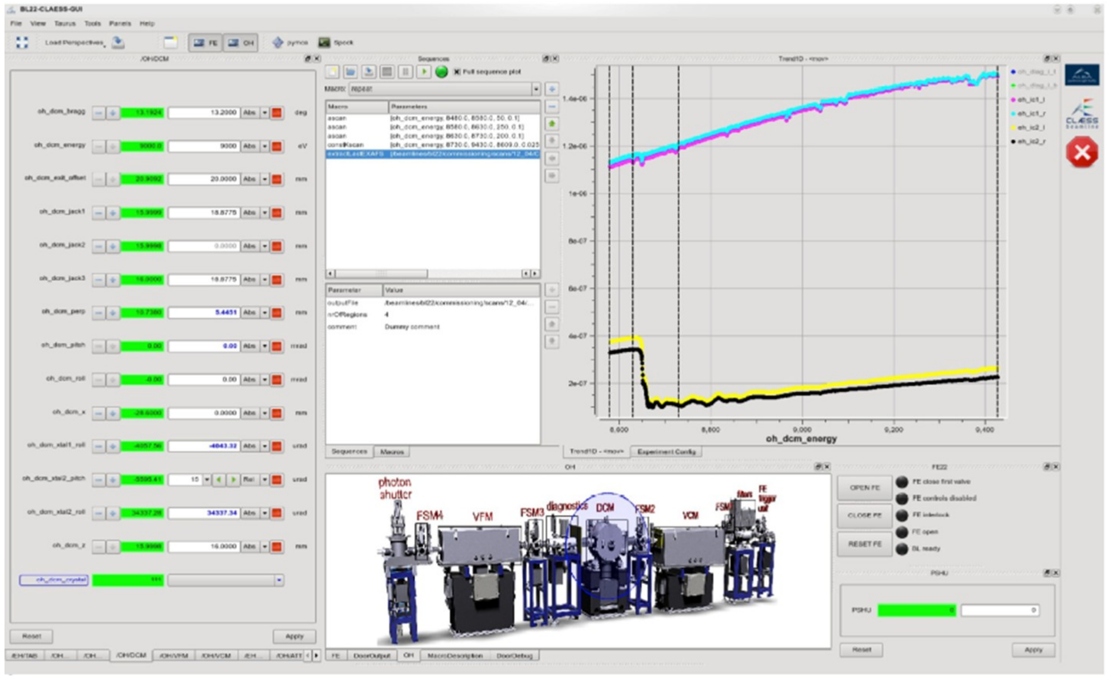


Also by:

- Martí Caixal
- Guifré Cuní
- Emilio Morales
- Miquel Navarro
- Jose Ramos
- Sergi Rubio

Contents

Taurus is...



*“**Taurus** is a **python** framework for control and data acquisition **CLIs** and **GUIs** in scientific/industrial environments. It supports multiple control systems or data sources: **Tango**, **EPICS**, ... New control system libraries can be integrated through plugins.”*

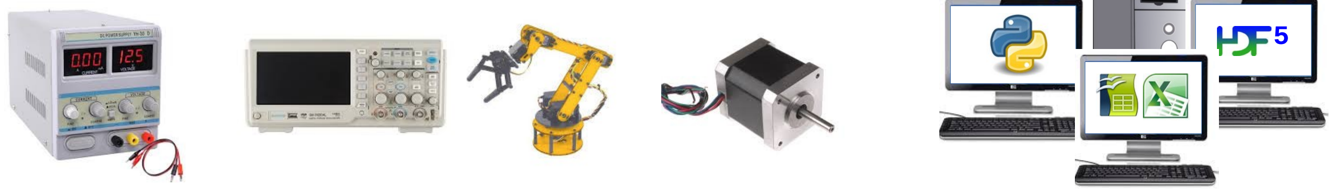
- Widely used
- Production-ready
- Well supported
- Actively developed
- Free/Open Source
- Community-driven
- Modular
- Multi-platform
- Based on Python and Qt
- Easy to install

Structure of Taurus

TaurusGUIs

TaurusGUIs

External Hardware and data sources

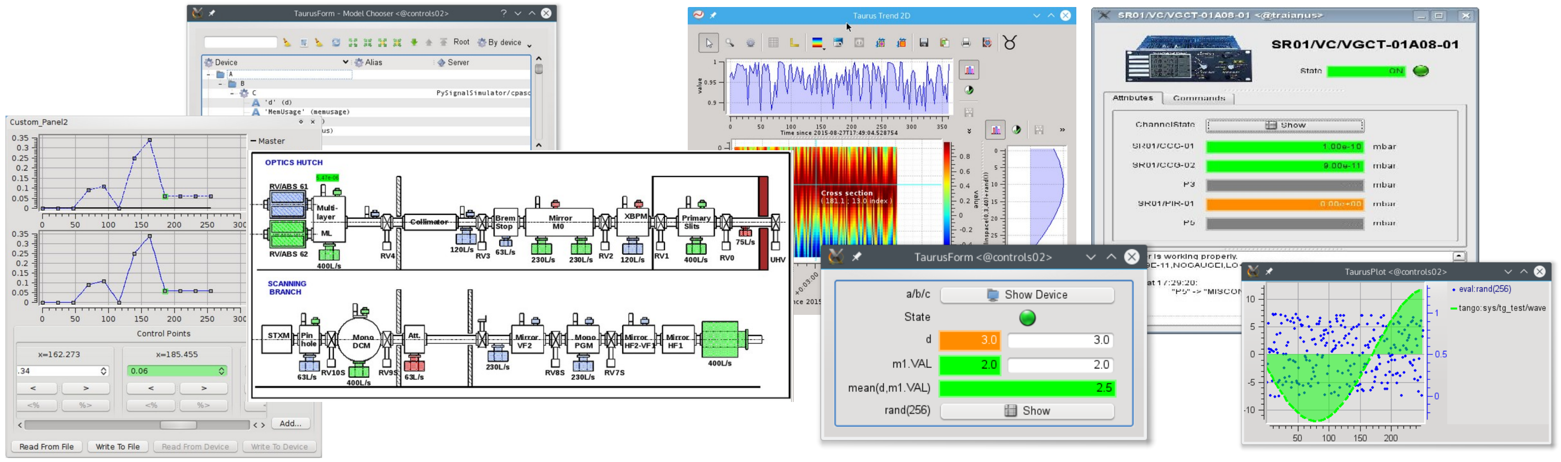


Structure of Taurus

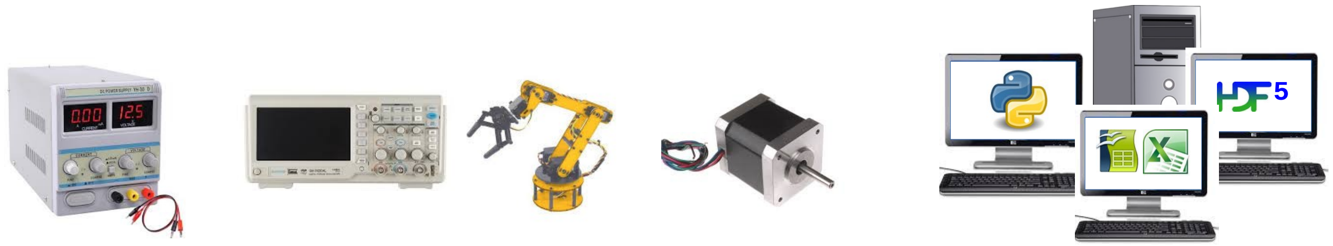
TaurusGUIs

TaurusGUIs

Taurus Qt Widgets



External Hardware and data sources



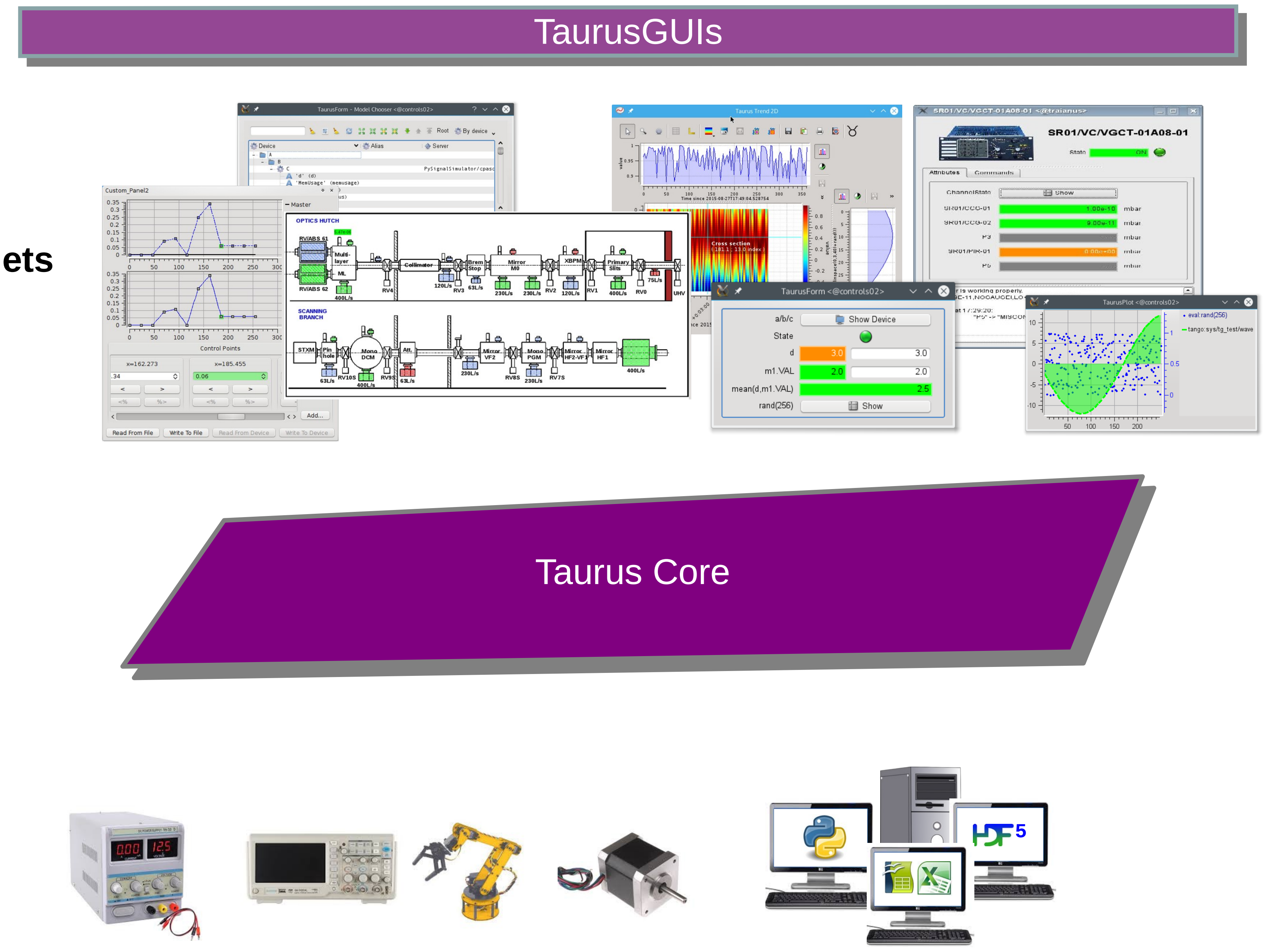
Structure of Taurus

TaurusGUIs

Taurus Qt Widgets

Taurus Core

External Hardware and data sources



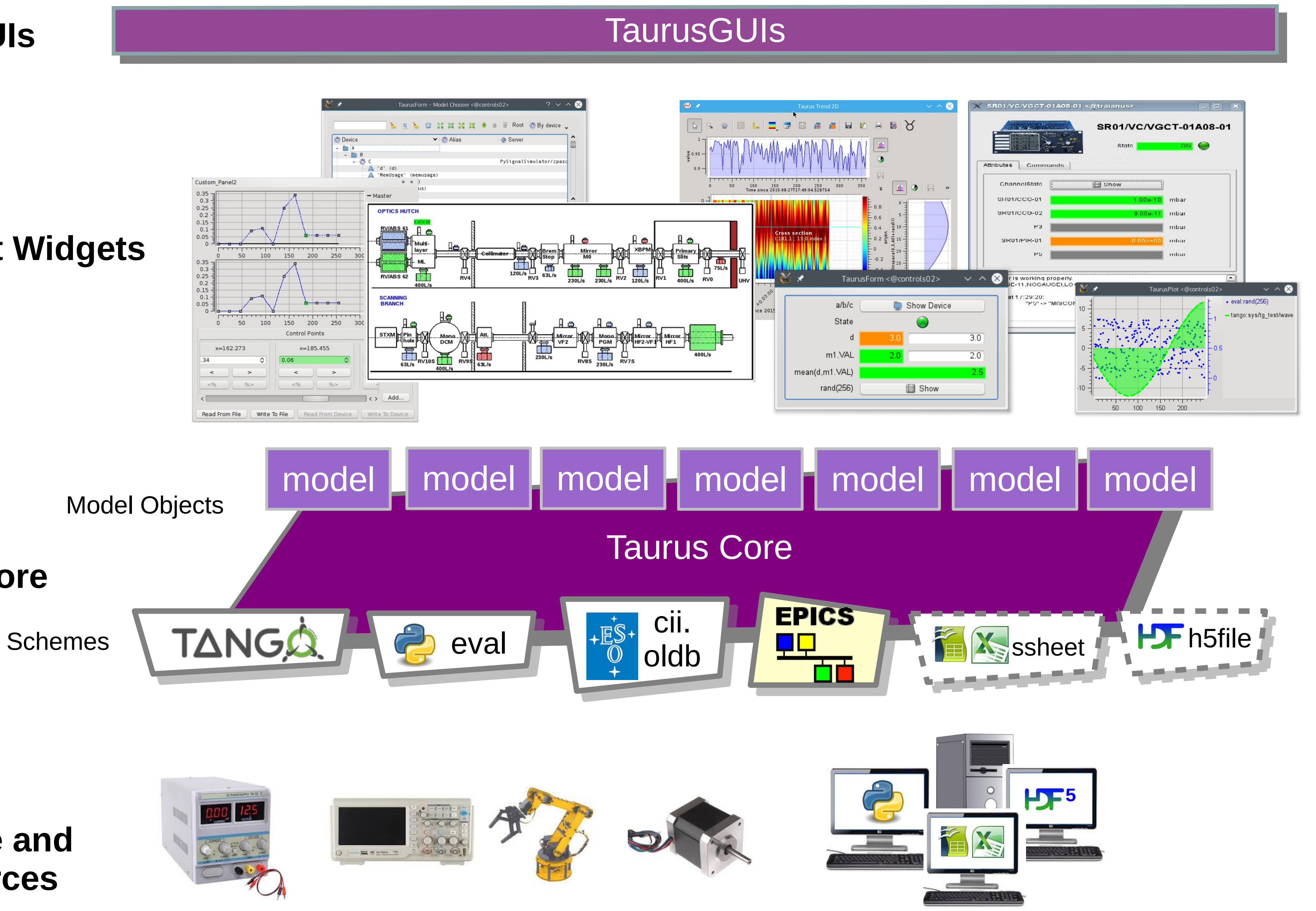
Structure of Taurus

TaurusGUIs

Taurus Qt Widgets

Taurus Core

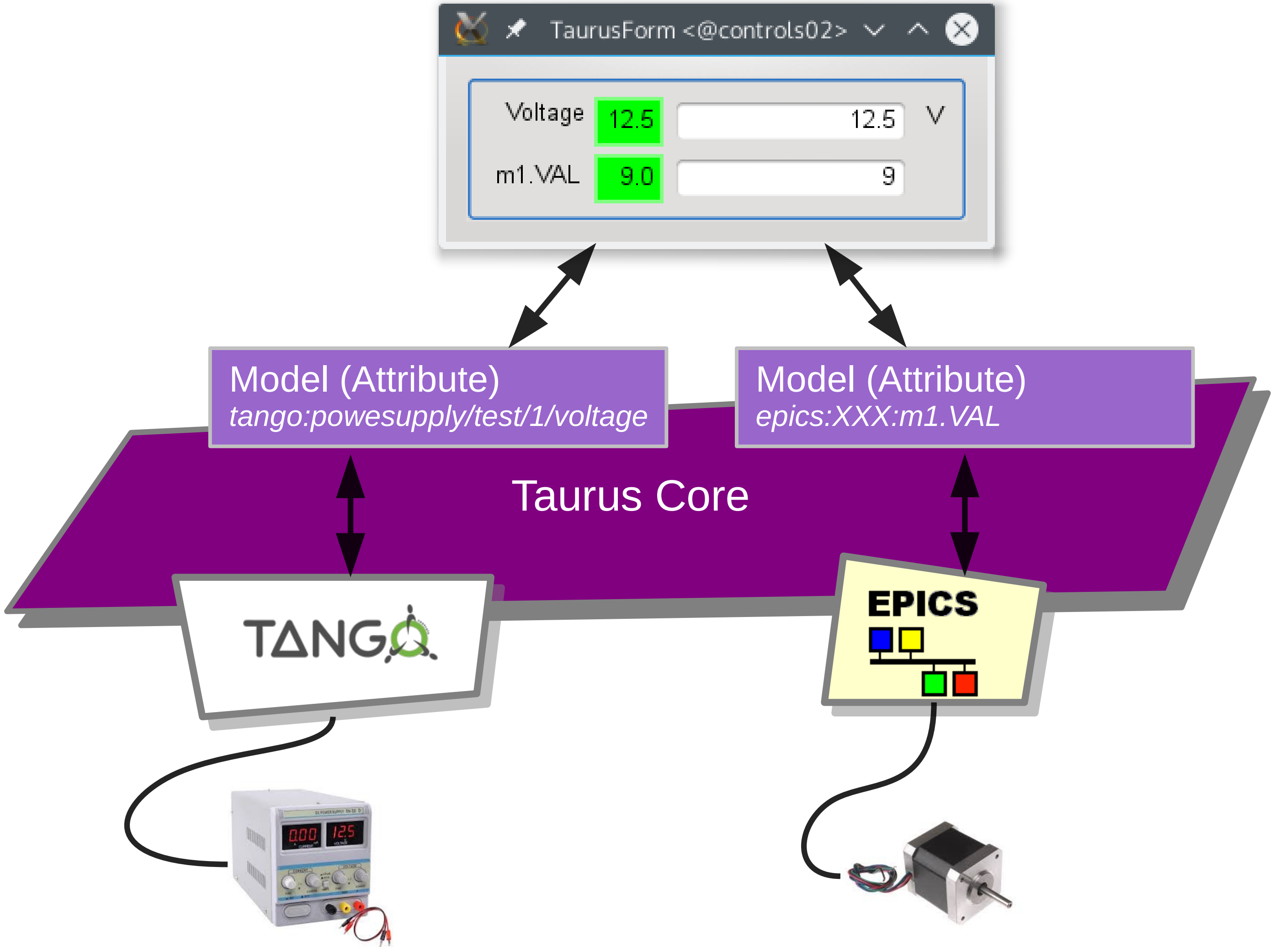
External Hardware and data sources



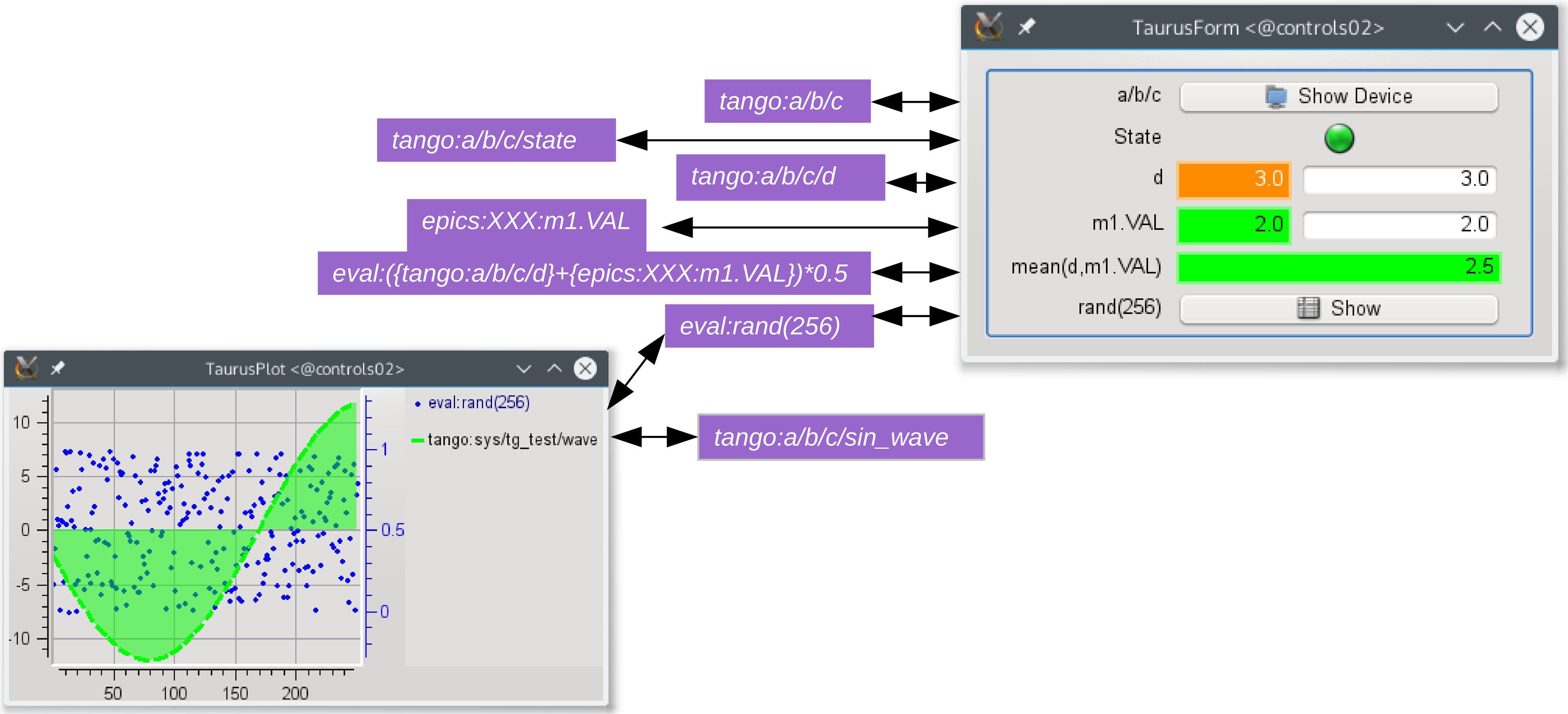
Model-View-controller

Taurus Qt Widgets (View & Controller)

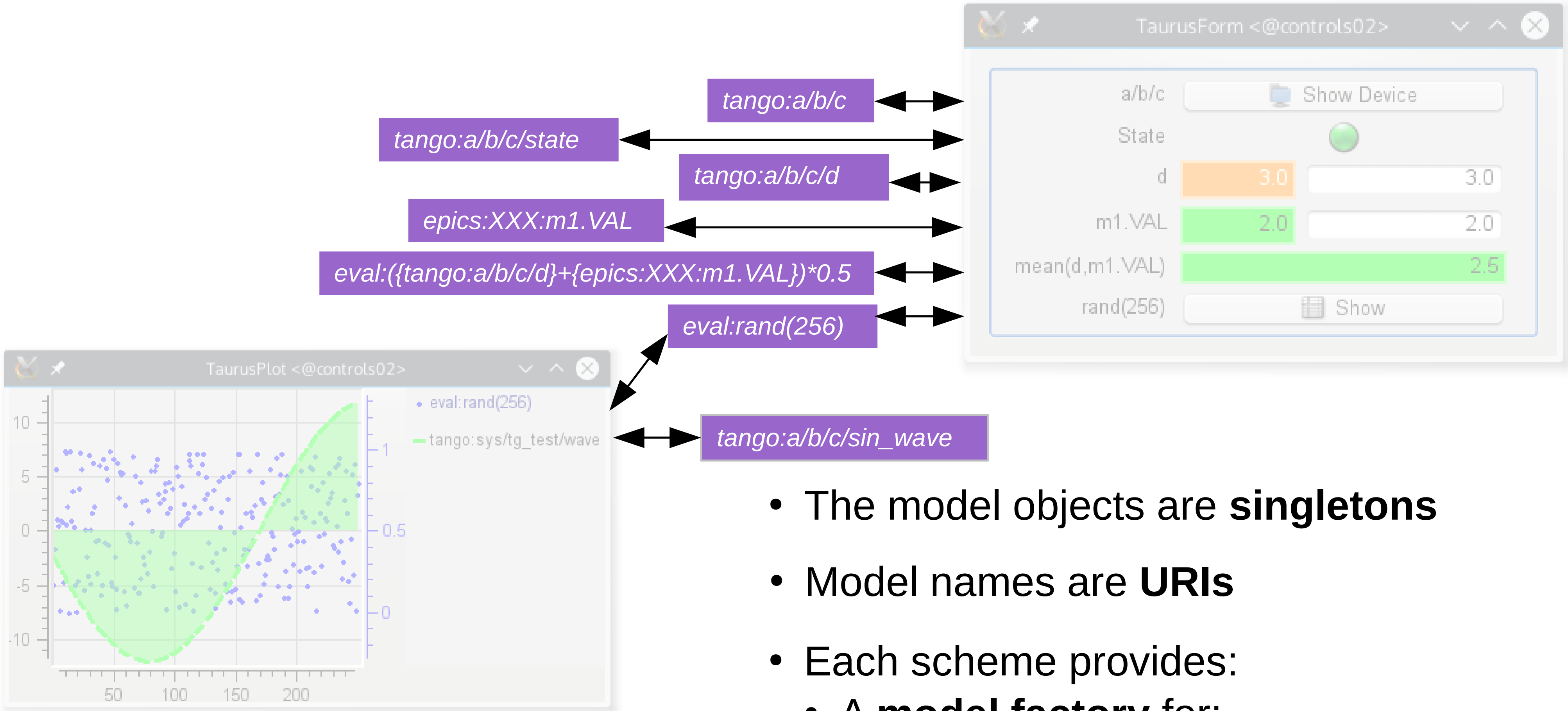
Taurus Core (Model)



Model-View-controller

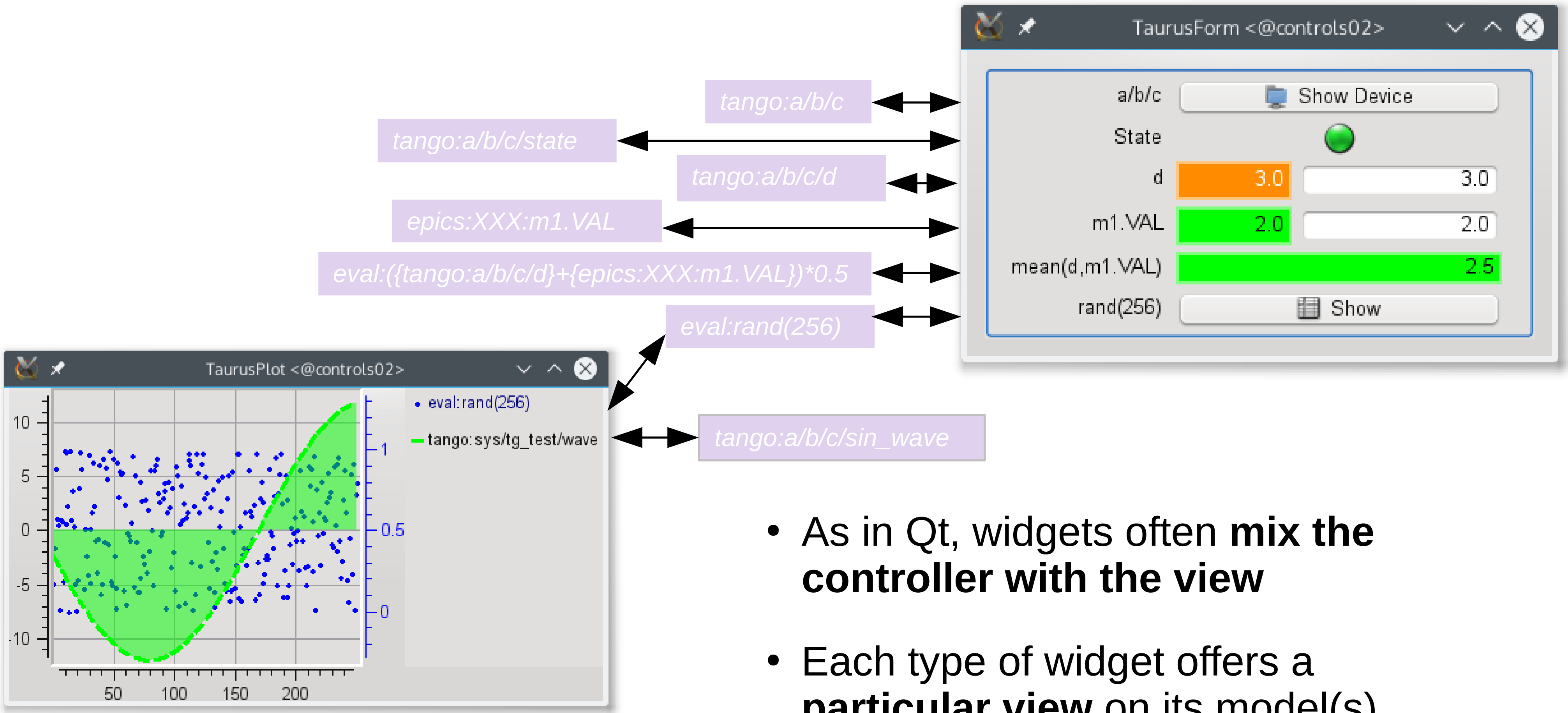


Model-View-controller



- The model objects are **singletons**
- Model names are **URIs**
- Each scheme provides:
  - A **model factory** for:
    - Authority
    - Device
    - Attribute
  - **Model name validators**

Model-View-controller



- As in Qt, widgets often **mix the controller with the view**
- Each type of widget offers a **particular view** on its model(s)
- All functionality is enabled by just **attaching** the widget to a model (i.e. providing its URI)

Examples of model names

scheme:authority/path?query#fragment

#	Model name (URI)	Scheme	Model type	Represented source of data/control object
1	*tango://foo:1234*	TANGO	Authority	**Tango** database listening to port *1234* of host *foo*
2	*tango://foo:1234/a/b/c*	TANGO	Device	**Tango** Device *a/b/c* registered in database *foo*
3	*tango:a/b/c/state*	TANGO	Attribute	**Tango** attribute *state* of device #2
4	*tango:a/b/c/d#units*	TANGO	Attribute	**Tango** attribute *d* of device #2 (*units* fragment)
5	*ca:XXX:m1.VAL*	EPICS	Attribute	**EPICS** process variable *XXX:m1.VAL*
6	*eval:({tango:a/b/c/d}+{epics:XXX:m1.VAL})*0.5*	eval	Attribute	**Calculated** average of the values of #4 and #5
7	*eval:rand(256)*	eval	Attribute	Random **generated** array of 256 values
8	*msenv://foo:1234/macroserver/bar/1/ScanDir*	[MSenv]	Attribute	*ScanDir* variable from **Sardana's environment**
9	*h5file:/mydir/myfile.hdf5*	HDF5	Device	File in **HDF5** format saved at */mydir/myfile*
10	*h5file:/mydir/myfile.hdf5:data/energy*	HDF5	Attribute	**HDF5** dataset *energy* of group *data* from file #9
11	*ssheet:myfile.ods:Sheet1.A1*		Attribute	Contents of cell A1 of Sheet1 of *myfile.ods* **spreadsheet**

Other suggested schemes:

Archiving, SQL, Icat, ASCII tables, ...

Installation

- PyPI
- Debian Linux
- **conda**

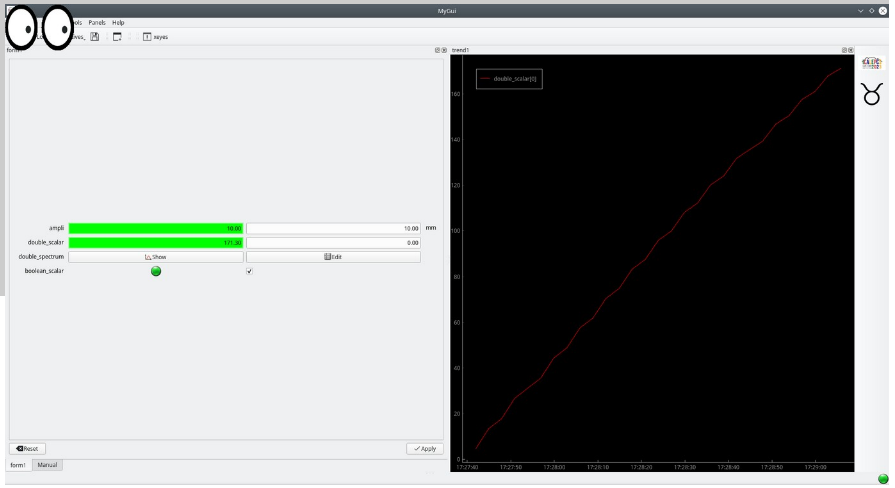
```
conda create -y -c conda-forge -n taurus_tutorial \
    taurus \
    taurus_pyqtgraph \
    qt=5.12 pytango=9.3 \   # see taurus#1233
    guidata=2.3.1 \         # guidata and guiqwt incompatibilities
    tango-test              # DS for demo purposes
conda activate taurus_tutorial
pip install pyhdbpp
```

More on: https://taurus-scada.org/users/getting_started.html

Contents

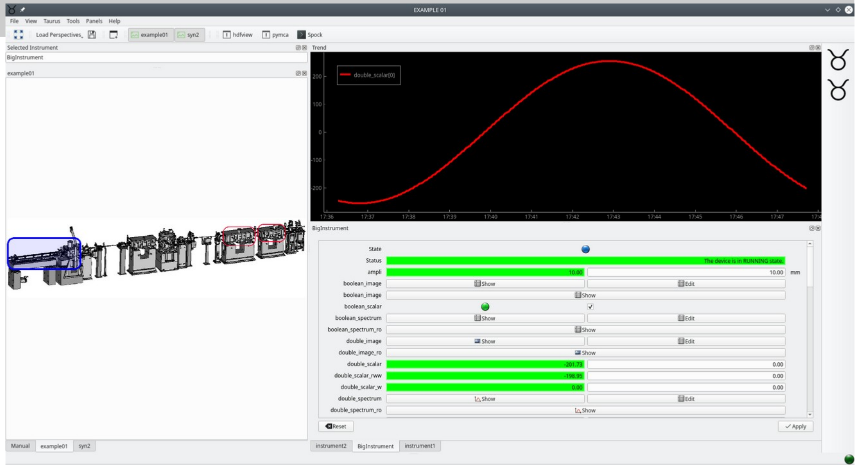
TaurusGUI – New GUI

```
# Start the TaurusGUI wizard
taurus newgui
 # next
 # select directory e.g. mygui → next
 # select GUI name: MyGui → next
 # select logo → next
 # skip: select synoptic → next
 # create panels editor e.g.
 #   create a form and a trend and connect to sys/tg_test/1 attrs
 # create external applications launcher e.g. xeyes
 # skip: configure monitor
 # install the project
# Start the application
mygui
# Dock the panels in the main window
# Launch the external application
```



TaurusGUI – example01

```
taurus gui example01
 # ignore errors, some extra dependencies are missing
 # unlock the view (Menu → View → Lock View)
 # dock panels in the main window
 # navigate over the lab with active synoptic and panels
 # save current view as a perspective
 # add panels: form and trend/plot and drag and drop models:
 # - from the same application
 # - from mygui (see previous slide) application
 # save current view as another perspective (save temp. panels)
 # switch between perspectives
 # restart application (save temp. panels on exit, save settings)
 # access perspectives
```



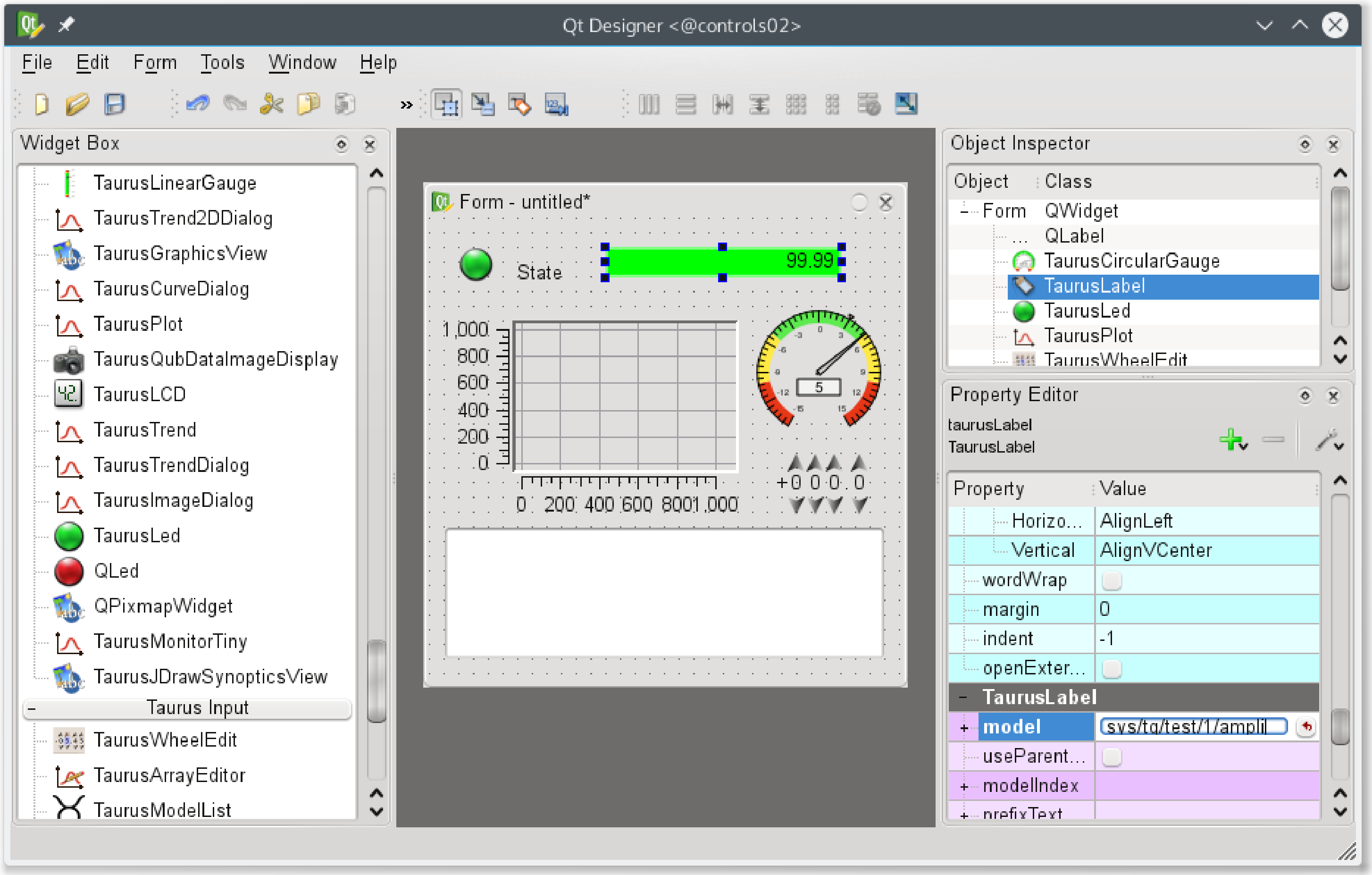
Contents

TaurusDesigner (Qt Designer)

GUIs can be created using the Qt Designer (*)

The Taurus widgets are available in the catalogue

The model name can be set as a Qt property



Object	Class
Form	QWidget
...	QLabel
	TaurusCircularGauge
	TaurusLabel
	TaurusLed
	TaurusPlot
	TaurusWheelEdit

tauruslabel
TaurusLabel

Property	Value
Horizo...	AlignLeft
Vertical	AlignVCenter
wordWrap	☐
margin	0
indent	-1
openExter...	☐
TaurusLabel	
model	sys/tg/test/1/ampli
useParent...	☐
modelIndex	
prefixText	

TaurusDesigner (Qt Designer)

```
taurus designer
 # Create a Widget with:
 # - Vertical Layout
 #   - TaurusTrend2DDialog connected to sys/tg_test/1/wave
 #   - Horizontal Layout
 #     - QLabel
 #     - TaurusValueLineEdit connected to sys/tg_test/1/ampli
 #   - Horizontal Layout
 #     - TaurusLed – State
 #     - TaurusCommandButton - connected to sys/tg_test/1
 #                           - configured with SwitchState()
 #
 # Store the widget as MyWidget.ui in the following place:
 #   MyWidget
 #    └── ui
 #        └── MyWidget.ui
```

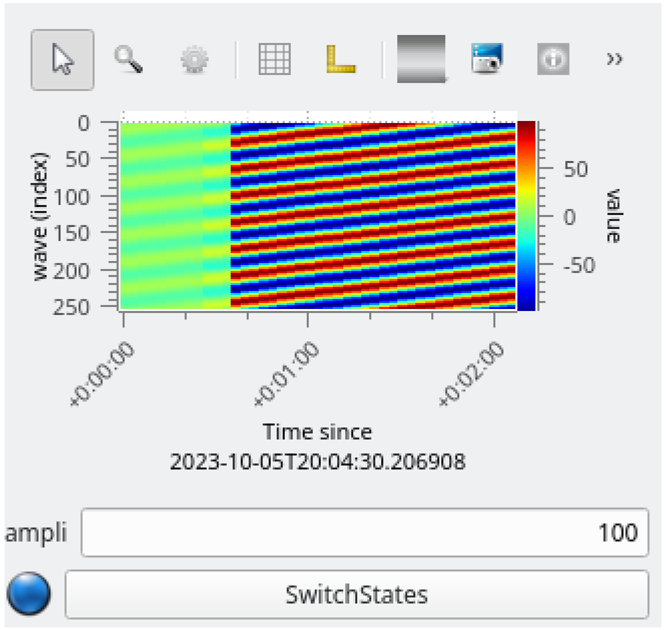
TaurusDesigner (Qt Designer)

https://taurus-scada.org/devel/api/taurus.qt.qtgui.util.html#taurus.qt.qtgui.util.UILoadable

```
# Create the following Python module:
from taurus.qt.qtgui.application import TaurusApplication
from taurus.external.qt import Qt
from taurus.qt.qtgui.util.ui import UILoadable

@UILoadable
class MyWidget(Qt.QWidget):

    def __init__(self, parent=None):
        Qt.QWidget.__init__(self, parent)
        self.loadUi()
        self.label.setText("ampli")

if __name__ == "__main__":
    import sys

    app = TaurusApplication()
    w = MyWidget()
    w.show()
    sys.exit(app.exec_())
```

TaurusDesigner (Qt Designer)

```
# Store the Python module in:
# MyWidget
#    ├── MyWidget.py
#    └── ui
#        └── MyWidget.ui
#
# Launch the application:
python MyWidget.py
```



Contents

How to “Taurus-ify” a Qt widget

- Create a widget that inherits both from a **`QWidget`** (or a QWidget-derived class) and from the `taurus.qt.qtgui.base.`**`TaurusBaseComponent`** *mixin* class.

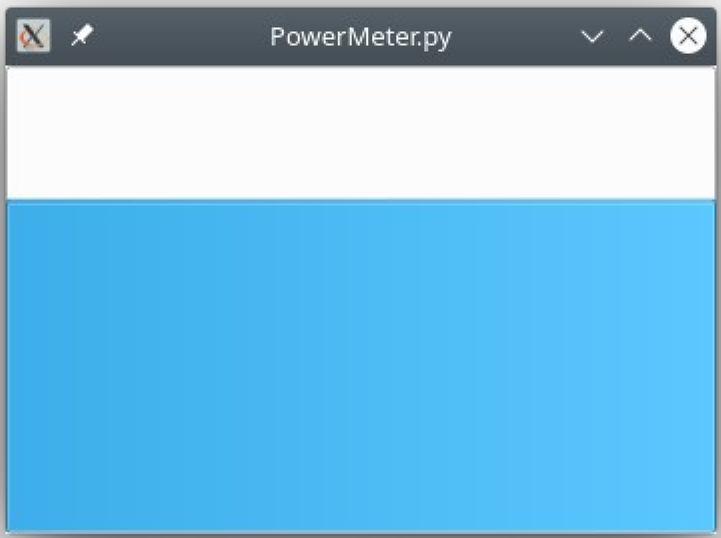
- These Taurus *mixin* class provide several APIs that are expected from Taurus widgets, such as:
  - model support API
  - configuration API
  - logger API
  - formatter API

- All you needs to do is to implement the **`handleEvent()`** method that will be called whenever the attached taurus model is updated.

How to “Taurus-ify” a Qt widget

- Store in `PowerMeter.py` and run it: **`python PowerMeter.py`**

```
from taurus.external.qt import Qt
from taurus.qt.qtgui.base import TaurusBaseComponent
from taurus.qt.qtgui.application import TaurusApplication


class PowerMeter(Qt.QProgressBar, TaurusBaseComponent):
    """A Taurus-ified QProgressBar"""


    # setFormat() defined by both TaurusBaseComponent and QProgressBar. Rename.
    setFormat = TaurusBaseComponent.setFormat
    setBarFormat = Qt.QProgressBar.setFormat


    def __init__(self, parent=None, value_range=(0, 100)):
        super(PowerMeter, self).__init__(parent=parent)
        self.setOrientation(Qt.Qt.Vertical)
        self.setRange(*value_range)
        self.setTextVisible(False)


    def handleEvent(self, evt_src, evt_type, evt_value):
        """reimplemented from TaurusBaseComponent"""
        try:
            self.setValue(int(evt_value.rvalue.m))
        except Exception as e:
            self.info("Skipping event. Reason: %s", e)


if __name__ == "__main__":
    import sys


    app = TaurusApplication()
    w = PowerMeter()
    w.setModel("eval:Q(60+20*rand())")
    w.show()
    sys.exit(app.exec_())
```

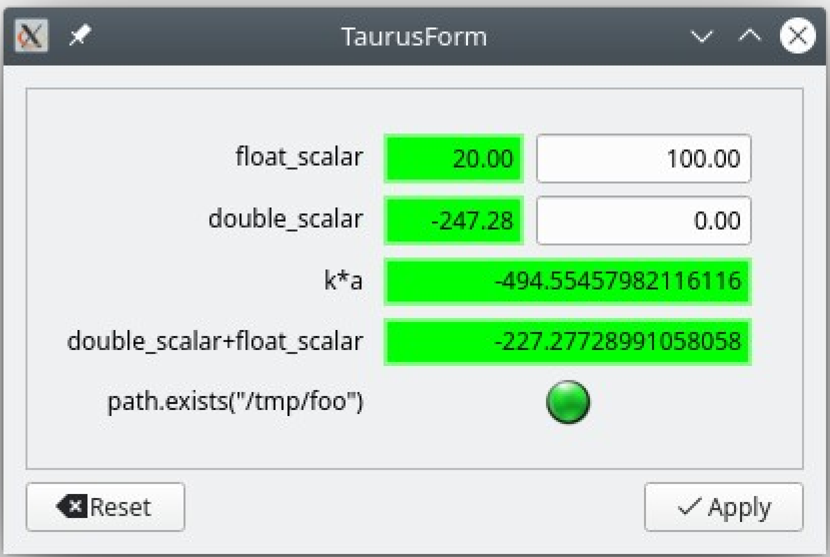


See more on: https://taurus-scada.org/devel/custom_widgets.html

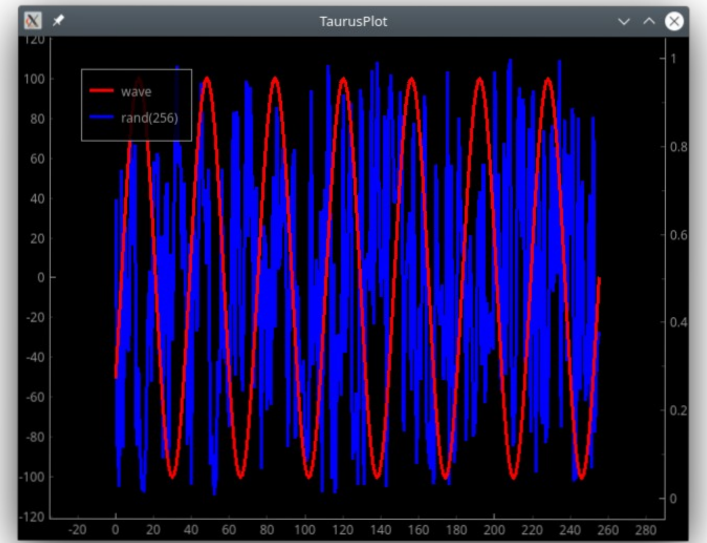
Bonus: Taurus Evaluation Scheme

https://taurus-scada.org/devel/api/taurus.core.evaluation.html#module-taurus.core.evaluation

```
# model name: eval:[//<authority>][@<evaluator>/][<subst>;]<expr>

taurus form \
sys/tg_test/1/float_scalar sys/tg_test/1/double_scalar \
'eval:{sys/tg_test/1/double_scalar}*2' \
'eval:k=2;a={sys/tg_test/1/double_scalar};k*a' \
'eval:{sys/tg_test/1/double_scalar}+{sys/tg_test/1/float_scalar}' \
'eval:@os.*/path.exists("/tmp/foo")'

taurus plot sys/tg_test/1/wave 'eval:rand(256)'
```


TaurusForm

float_scalar	20.00	100.00
double_scalar	-247.28	0.00
k*a	-494.55457982116116	
double_scalar+float_scalar	-227.27728991058058	
path.exists("/tmp/foo")	●	

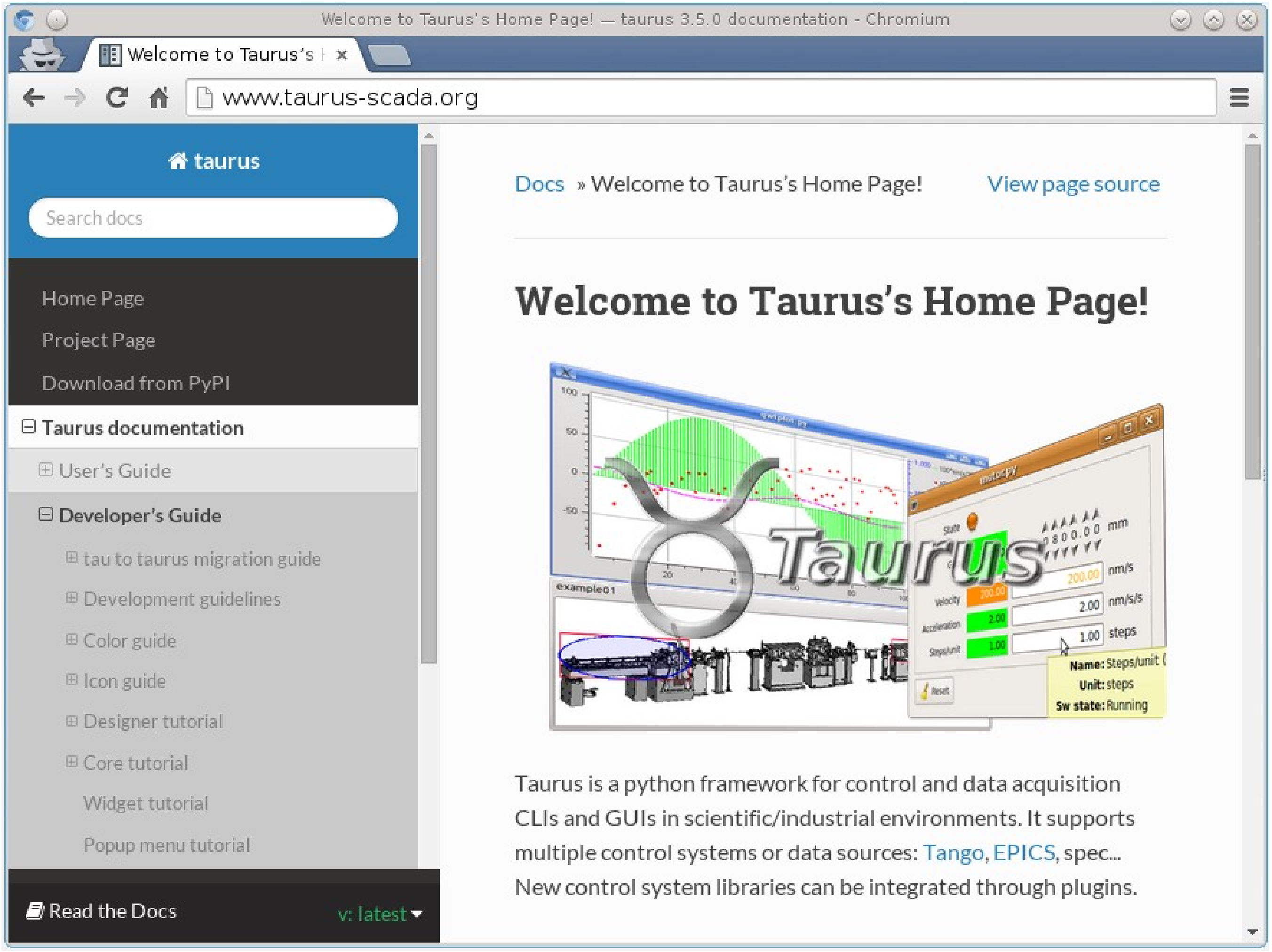


Home Page

http://www.taurus-scada.org

Access to:

- Documentation
- Releases
- Git repository
- Mailing lists
- Bugs & Requests tracker
- Enhancement Proposals
- ...



Welcome to Taurus's Home Page! — taurus 3.5.0 documentation - Chromium

www.taurus-scada.org

Welcome to Taurus's Home Page!

Taurus is a python framework for control and data acquisition CLIs and GUIs in scientific/industrial environments. It supports multiple control systems or data sources: Tango, EPICS, spec... New control system libraries can be integrated through plugins.