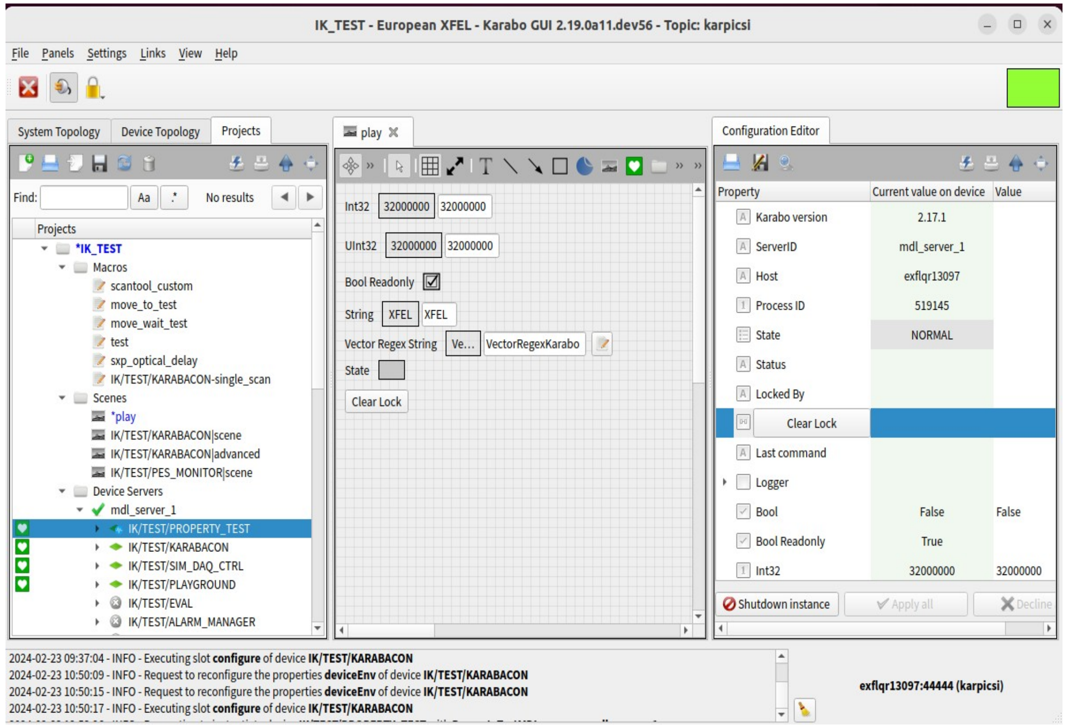
# Session 4 : GUI Extensions

## Karabo GUI

- Control and designer modes.
- In designer mode: drag and drop device properties to the scene.
- Default widgets based on the data type are displayed.
- Contains 3 items: property name, display and edit widgets.
- Scenes are stored in project or provided by devices.



```
Projects
  *IK_TEST
    Macros
      scantool_custom
      move_to_test
      move_wait_test
      test
      sxp_optical_delay
      IK/TEST/KARABACON-single_scan
    Scenes
      *play
      IK/TEST/KARABACON|scene
      IK/TEST/KARABACON|advanced
      IK/TEST/PES_MONITOR|scene
    Device Servers
      mdl_server_1
        IK/TEST/PROPERTY_TEST
        IK/TEST/KARABACON
        IK/TEST/SIM_DAQ_CTRL
        IK/TEST/PLAYGROUND
        IK/TEST/EVAL
        IK/TEST/ALARM_MANAGER
```

```
Int32              32000000  32000000
UInt32             32000000  32000000
Bool Readonly      ☑
String             XFEL  XFEL
Vector Regex String  Ve...  VectorRegexKarabo
State
Clear Lock
```

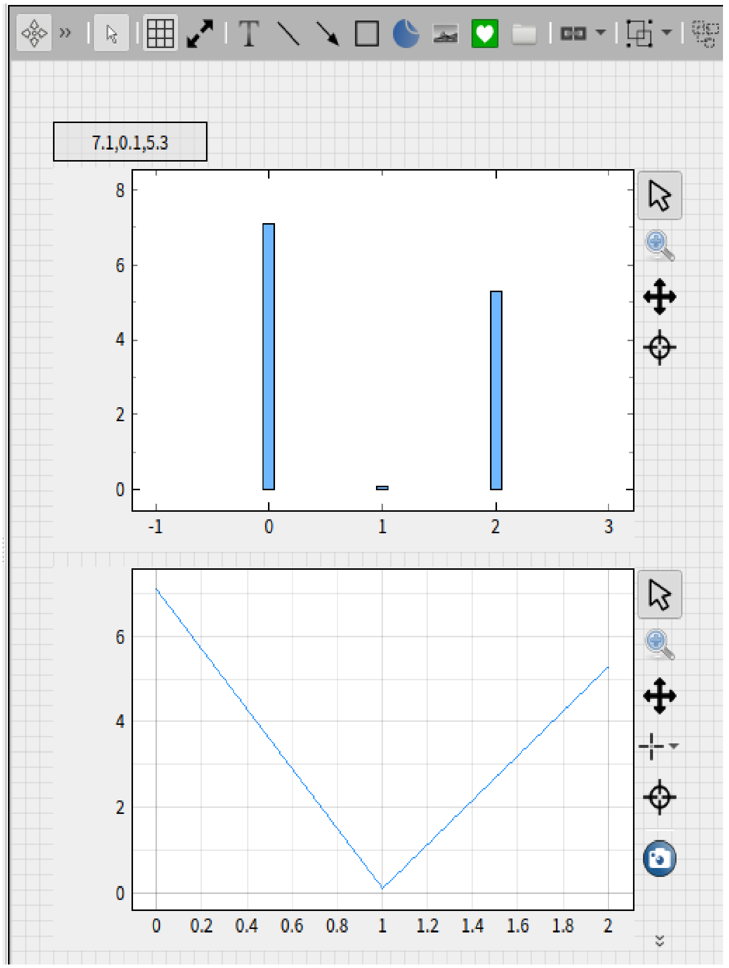
| Property | Current value on device | Value |
|---|---|---|
| Karabo version | 2.17.1 | |
| ServerID | mdl_server_1 | |
| Host | exflqr13097 | |
| Process ID | 519145 | |
| State | NORMAL | |
| Status | | |
| Locked By | | |
| Clear Lock | | |
| Last command | | |
| Logger | | |
| Bool | False | False |
| Bool Readonly | True | |
| Int32 | 32000000 | 32000000 |

```
2024-02-23 09:37:04 - INFO - Executing slot configure of device IK/TEST/KARABACON
2024-02-23 10:50:09 - INFO - Request to reconfigure the properties deviceEnv of device IK/TEST/KARABACON
2024-02-23 10:50:15 - INFO - Request to reconfigure the properties deviceEnv of device IK/TEST/KARABACON
2024-02-23 10:50:17 - INFO - Executing slot configure of device IK/TEST/KARABACON
```

exflqr13097:44444 (karpicsi)

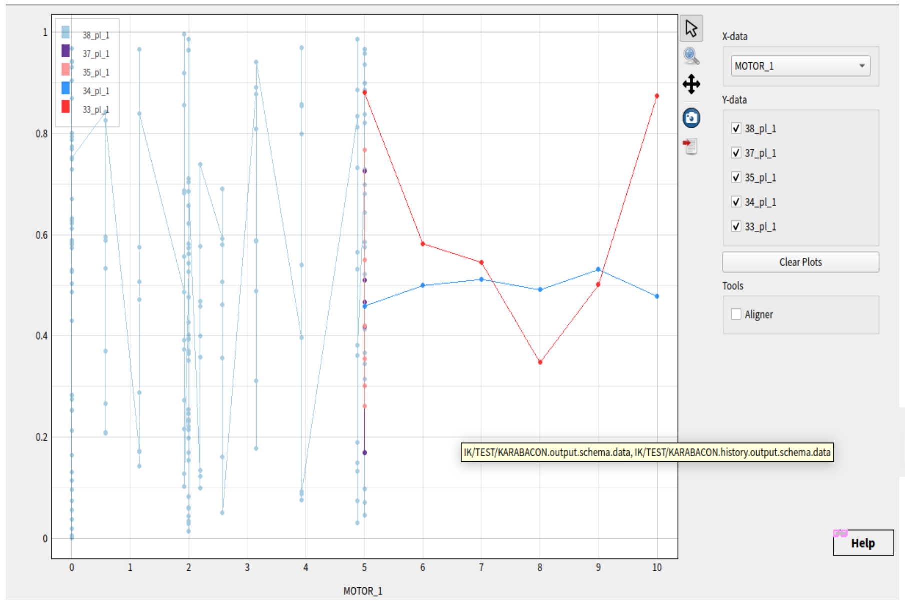
## Karabo GUI

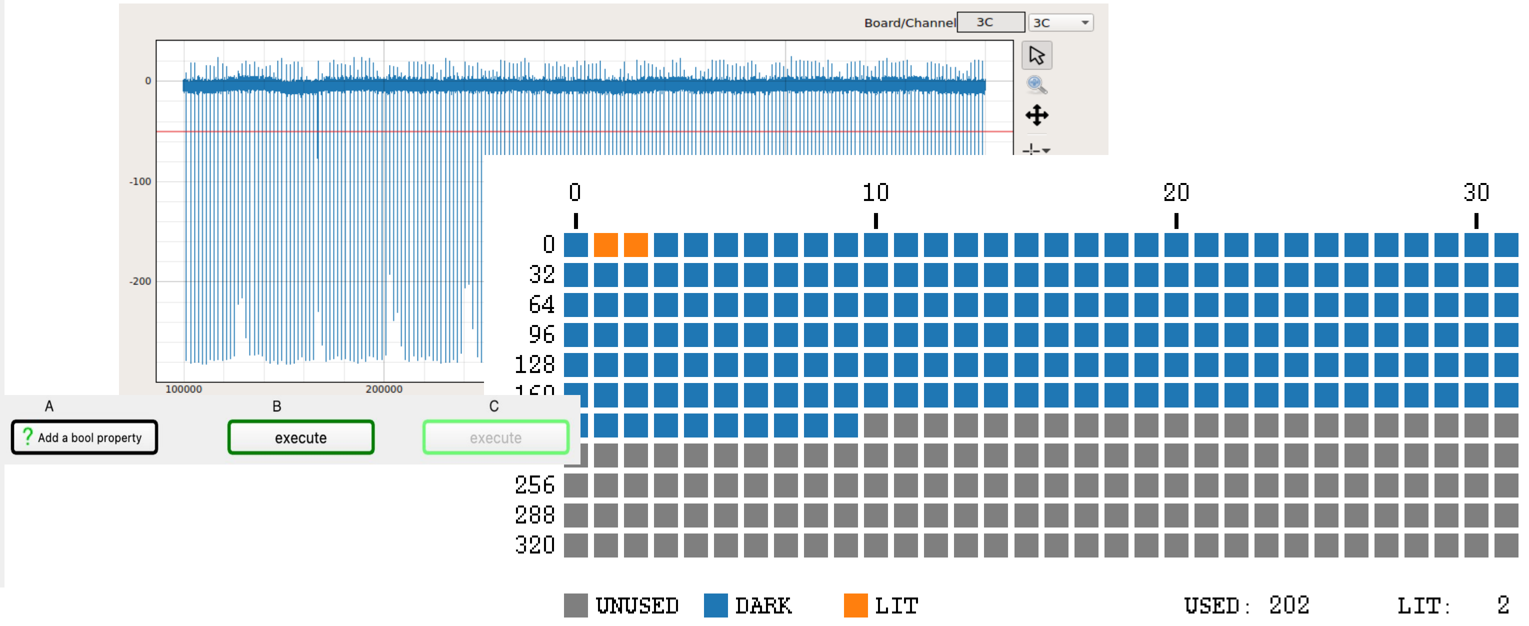
- Large set of built-in widgets.
- Widgets can be changed by right clicking on the widget and selecting **Change Widget**.
- Based on the data type, display types changes.
- Switching between widgets is always possible.



## GUI Extensions

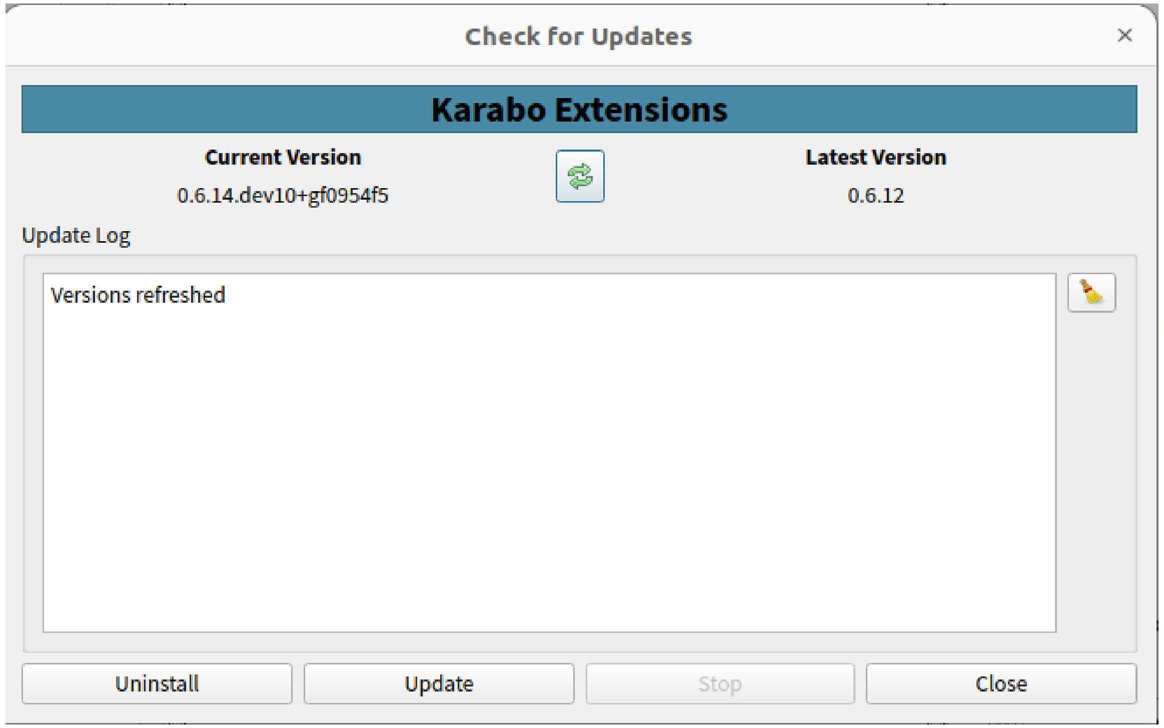
- Karabo GUI Extensions (further extensions) is a python package that compliments Karabo GUI by adding custom tailor made widgets.
- It is **not** part of Karabo framework and does not follow karabo release/deployment cycle and can be updated at any point.
- Gitlab link: https://git.xfel.eu/karaboDevices/guiextensions


38_pl_1
37_pl_1
35_pl_1
34_pl_1
33_pl_1
MOTOR_1
X-data
Y-data
Clear Plots
Tools
Aligner
Help
IK/TEST/KARABACON.output.schema.data, IK/TEST/KARABACON.history.output.schema.data



## Solution: Update gui extensions

- As the extensions are frequently updated one might miss the latest widgets.
- Use **Help** → **Check for Updates** to update extensions.
- Close and open karabo gui to reload extensions.


Check for Updates

Karabo Extensions

| Current Version | Latest Version |
|---|---|
| 0.6.14.dev10+gf0954f5 | 0.6.12 |

Update Log

Versions refreshed

## Karabo GUI development environment

- Karabo GUI is installed via conda package manager and dependency tool.
- It is written in Python by using PyQt and pyqtgraph libraries.
- To prepare the development environment of gui extensions:

  1. Clone guiextensions repository
  2. Activate conda environment of the Karabo GUI.
  3. Install guiextensions python package.

## Hands on #1 : prepare developer environment

- Open terminal
- **source /*opt*/miniconda/bin/activate**
- **conda activate karabogui**
- **cd**
- **cd karabo/devices/guiextensions**
- **git checkout workshop**
- **pip install -e .**
- **karabo-gui** (might take few sec.)

## Hands on #2: Change widget to gui extension

- Open karabo from terminal and connect with **admin** rights.
- Open project **SESSION_4**.
- Instantiate the middlelayer device **KARABO_TEST/MDL/PROPERTY_TEST**.
- Create a scene **SESSION_4_SCENE**.
- Drag and drop a **Float (Min / Max)** property on the scene.
- Ungroup widgets and change the second widget to **Workshop Example 1**.
- Save project.

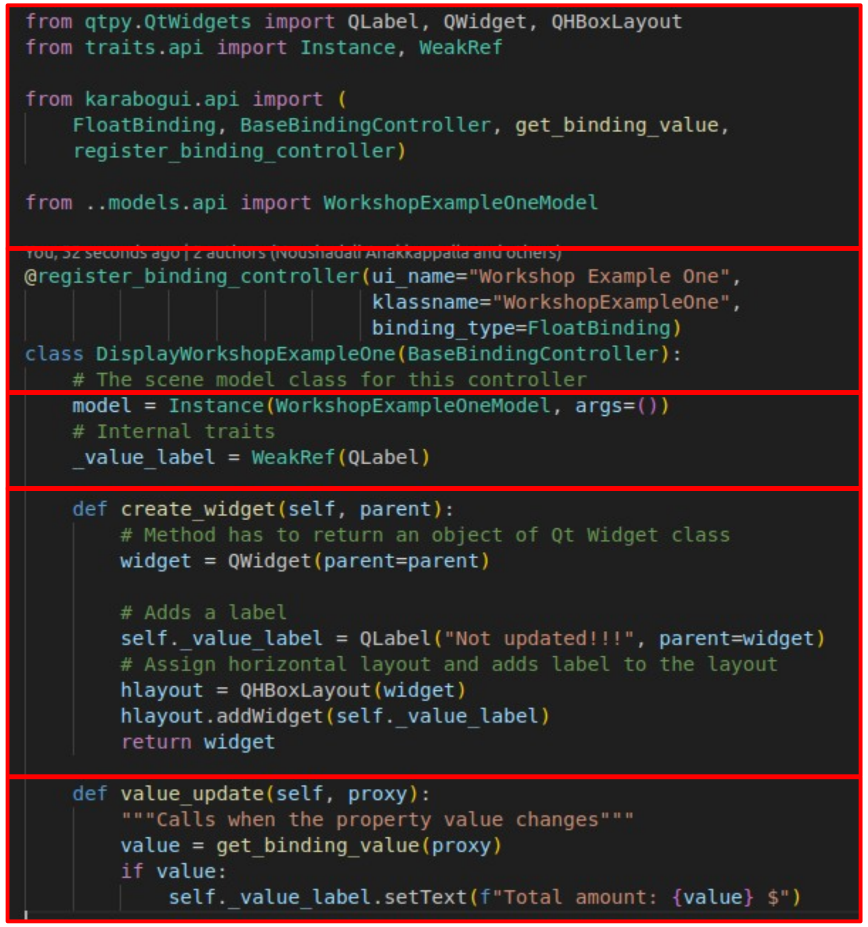
## GUI Extensions: Building blocks

- Extension is defined as an entry point in the **setup.py**
- It has to have a **data model**
- And representation **widget**.

## Hands on #3: explore gui extension

- Open visual code.
- Open folder **karabo/devices/guiextensions**
- Open file **src/extensions/workshop/display_example_one.py**

## GUI Extensions: Building blocks

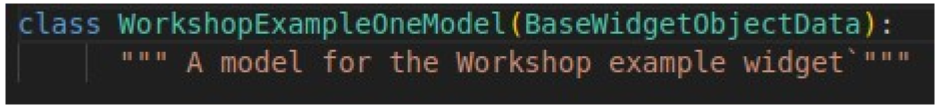


```
from qtpy.QtWidgets import QLabel, QWidget, QHBoxLayout
from traits.api import Instance, WeakRef

from karabogui.api import (
    FloatBinding, BaseBindingController, get_binding_value,
    register_binding_controller)

from ..models.api import WorkshopExampleOneModel

You, 32 seconds ago | 2 authors (Noushadali Anakkappalla and others)
@register_binding_controller(ui_name="Workshop Example One",
                             klassname="WorkshopExampleOne",
                             binding_type=FloatBinding)
class DisplayWorkshopExampleOne(BaseBindingController):
    # The scene model class for this controller
    model = Instance(WorkshopExampleOneModel, args=())
    # Internal traits
    _value_label = WeakRef(QLabel)

    def create_widget(self, parent):
        # Method has to return an object of Qt Widget class
        widget = QWidget(parent=parent)

        # Adds a label
        self._value_label = QLabel("Not updated!!!", parent=widget)
        # Assign horizontal layout and adds label to the layout
        hlayout = QHBoxLayout(widget)
        hlayout.addWidget(self._value_label)
        return widget

    def value_update(self, proxy):
        """Calls when the property value changes"""
        value = get_binding_value(proxy)
        if value:
            self._value_label.setText(f"Total amount: {value} $")
```

- Class is decorated with **register_binding_controller**.
  - **ui_name** appears in the gui.
  - **klassname** should be the same as in setup.py
  - **binding_type** defines data type that the controller will accept.
- Class has to be inherited from **BaseBindingController**.
- Has a model (for storing attributes in project) and internal objects.
- Mandotary methods that needs to be implemented:
  - **create_widget** returns PyQt widget object,
  - **value_update** callback when value proxy changes.

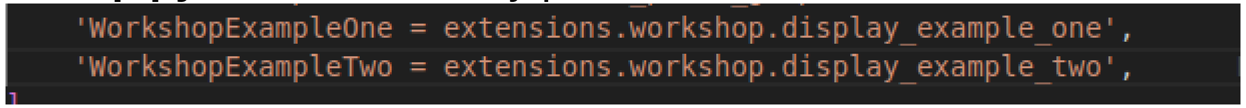
## GUI Extensions: Building blocks

- **Model** can be used to store a gui related configuration in the Karabo project.



```
class WorkshopExampleOneModel(BaseWidgetObjectData):
        """ A model for the Workshop example widget`"""
```

- **Setup.py** contains all entry points:



```
    'WorkshopExampleOne = extensions.workshop.display_example_one',
    'WorkshopExampleTwo = extensions.workshop.display_example_two',
```

- **pip install -e .** to links the package. Necessary if the entry points in the setup.py change. No need to do pip install if the extension code changes.

## Note about libraries: Qt, PyQt, qtpy and traits

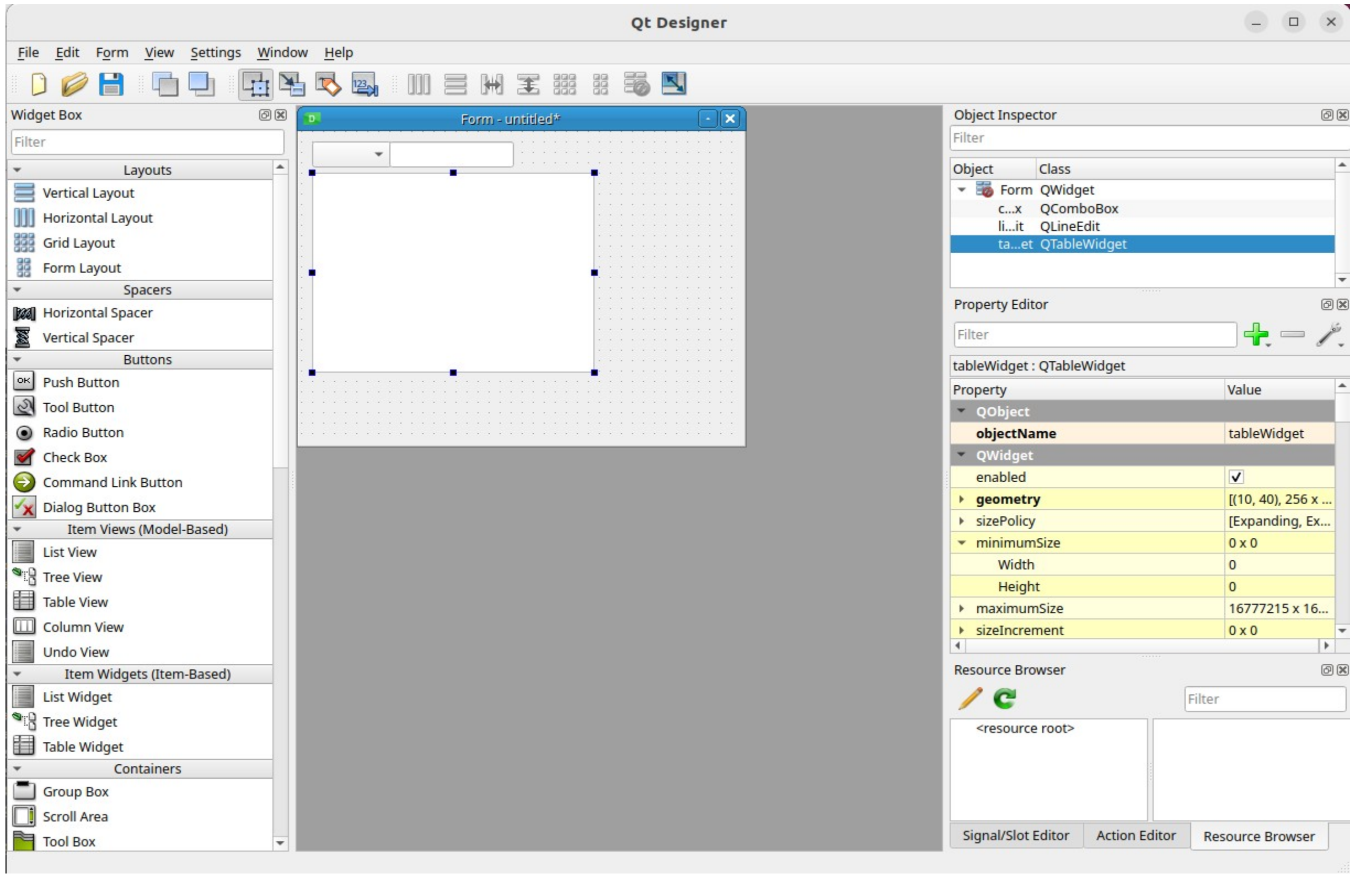
- Qt is graphical library written is c++, PyQt is wrapper of Qt and qtpy handles various PyQt versions.
- QWidget is a base class of all user interface objects (QLabel, QlineEdit, QPushButton, etc.).
- Widgets are grouped in layout(s) (QHBoxLayout, QVBoxLayout, QgridLayout).
- Traits package is used to ensure data validation.

**Hint:**

- PyQt provides **designer** tool.
- Allows to explore Qt library and create *ui files for complex widgets and layouts.

## Designer

- Activate conda karabo environment and launch **designer**.



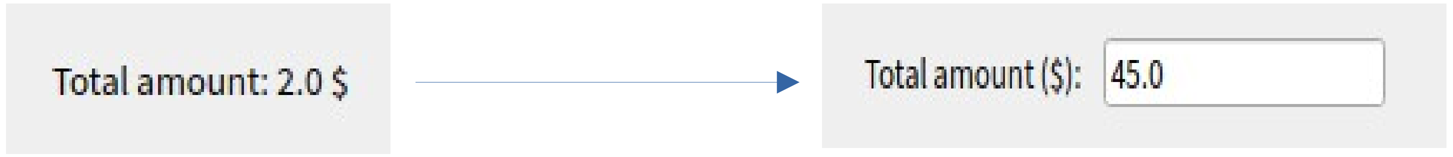
| Object | Class |
|---|---|
| Form | QWidget |
| c...x | QComboBox |
| li...it | QLineEdit |
| ta...et | QTableWidget |

tableWidget : QTableWidget

| Property | Value |
|---|---|
| QObject | |
| objectName | tableWidget |
| QWidget | |
| enabled | ✓ |
| geometry | [(10, 40), 256 x ... |
| sizePolicy | [Expanding, Ex... |
| minimumSize | 0 x 0 |
| Width | 0 |
| Height | 0 |
| maximumSize | 16777215 x 16... |
| sizeIncrement | 0 x 0 |

## Hands on #4

- Currently **display_example_one.py** contains one label that displays a text with attribute value.
- Modify widget by adding a value input widget that displays the value.



## Hands on #4 : steps and solution

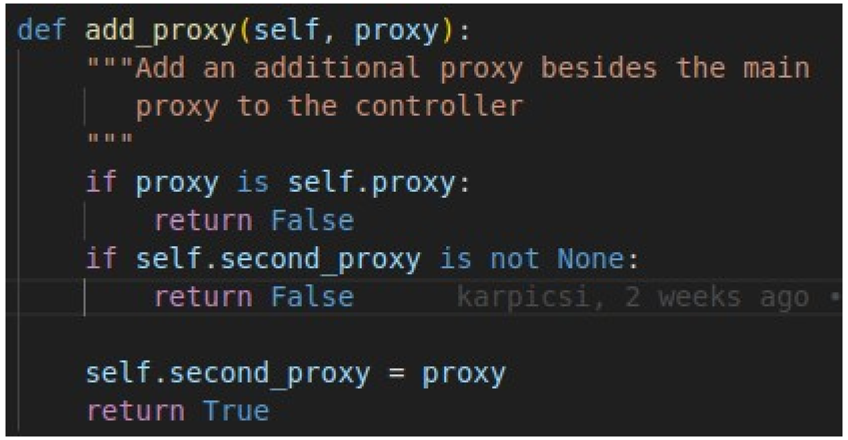
- Add **QLineEdit** (widget for value input) to the imports (line 1).
- Add _**value_ledit = WeakRef(QLineEdit)** to internal objects (After line 18)
- Add line edit to the widget and layout (after line 25 and 28):

  - **self._value_ledit = QlineEdit(parent=widget)**

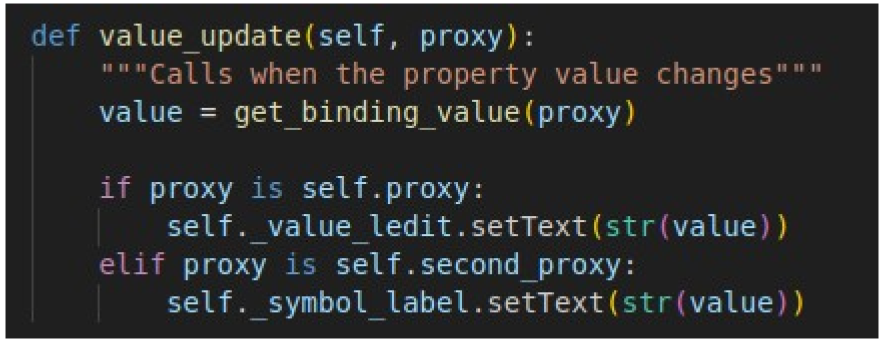
  - **hlayout.addWidget(self._value_ledit)**
- Set label and value_ledit text (after line 34):
  - **self._value_ledit.setText(str(value))**
  - Solution: **git checkout workshop_done** (close/open GUI)
  - **git diff workshop workshop_done**

## Property proxies

- Property proxy links device attributes with widgets.
- It is not the device proxies used in the middlelayer context.
- All karabo data types are supported. Including nodes.
- Reimplement **add_proxy** to accept multiple proxies.
- **add_proxy** on success should return True.
- **value_update** will be called if any proxy has been changed.



```
def add_proxy(self, proxy):
    """Add an additional proxy besides the main
      proxy to the controller
    """
    if proxy is self.proxy:
        return False
    if self.second_proxy is not None:
        return False        karpicsi, 2 weeks ago •

    self.second_proxy = proxy
    return True
```



```
def value_update(self, proxy):
    """Calls when the property value changes"""
    value = get_binding_value(proxy)

    if proxy is self.proxy:
        self._value_ledit.setText(str(value))
    elif proxy is self.second_proxy:
        self._symbol_label.setText(str(value))
```

## Hands on # 5: Example with two proxies

- Drag and drop **Float** property to the scene.
- Change widget to **Workshop Example 2**.
- Drag and drop a **String** property on top of the widget.
- Text appears in the label. Hovering over the widget shows connected proxies.
- Open **display_example_two.py** in visual code.

## Hands on #6

- Modify display_workshop_example_two.py.
- Implement basic interactive gui. If the float value is above 100:
  - Disable value self._value_edit (use **setEnabled()**).
  - Informe user that manual mode is disabled: Set tool tip of the **self._value_edit**.

## Hands on #6 : steps and result

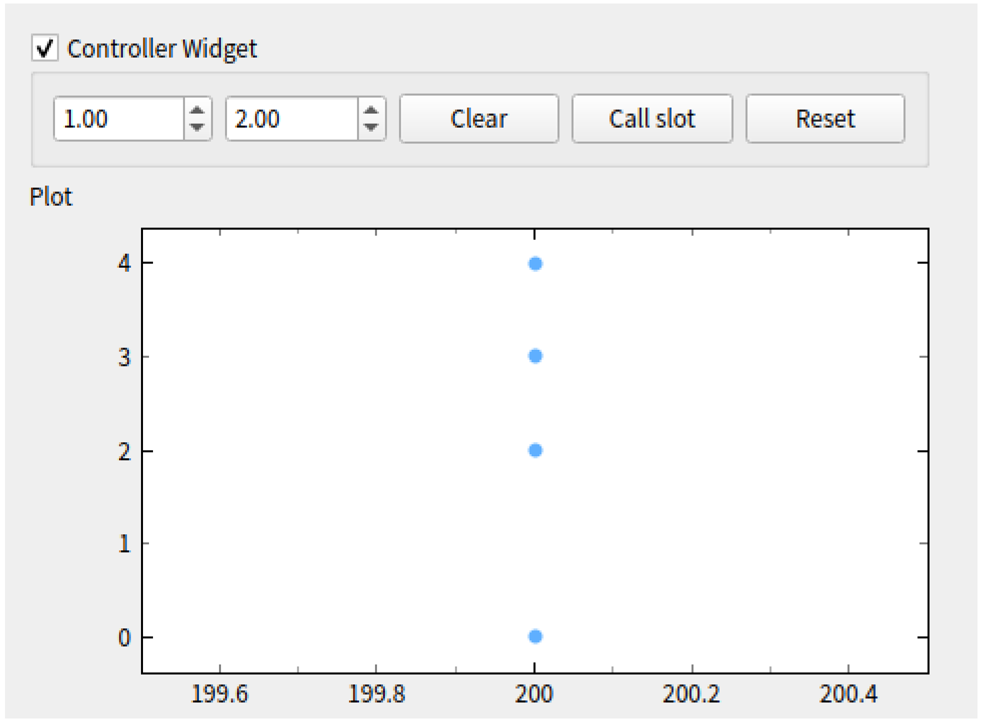
- Add code to the **value_update**:

```
self._value_ledit.setEnabled(value < 100)

tool_tip = "Manual mode is disabled" if value > 0 else ""

self._value_ledit.setToolTip(tool_tip)
```

- Solution: **git checkout workshop_done** (close/open GUI)
- Difference: **git diff workshop workshop_done**

## An advanced example

- Workshop example 3
- Nested layouts.
- Usage of model attributes (Traits package).
- Qt signals and slots.
- Setting attributes via proxy.
- Handling state_update.
- Calling karabo slots and handling response.
- KaraboPlotView class.



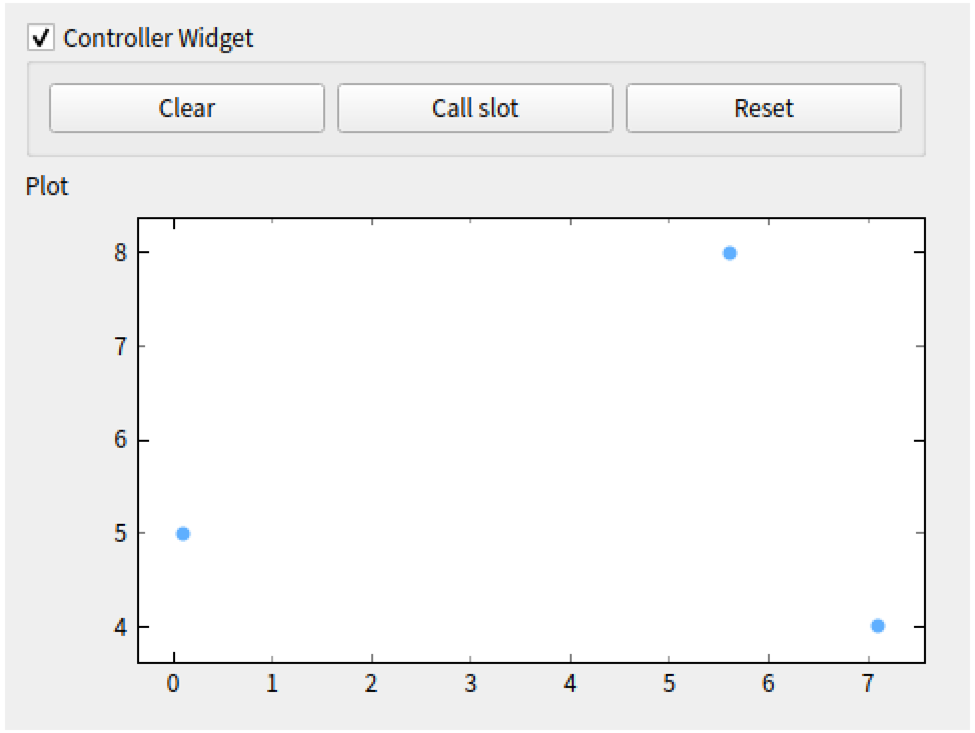
## Hands on #7

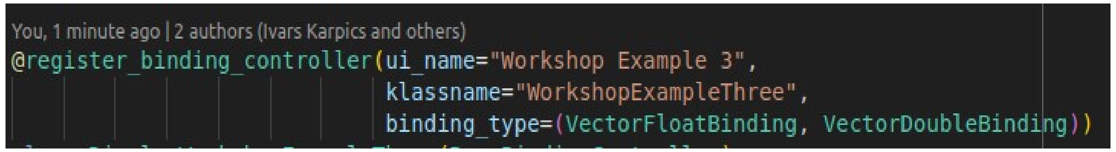
- Drag and drop Float attribute on the scene.
- Change widget to Workshop Example 3.
- Drag and drop a second Float attribute on the scene.

## Hands on #8

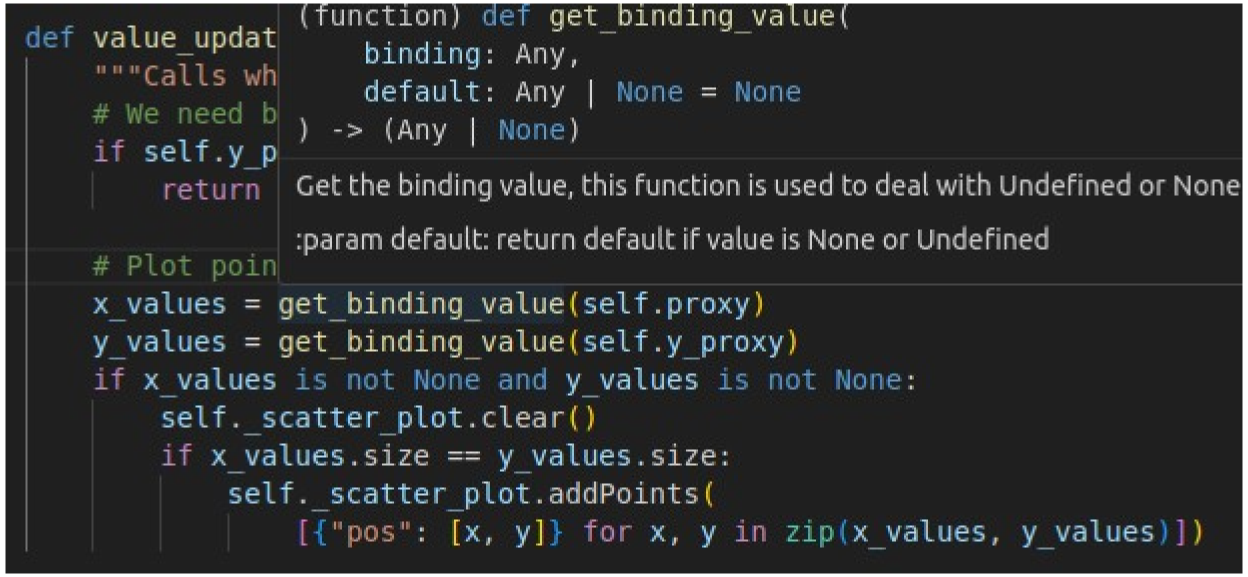
- Modify Workshop example 3
- Remove spinbox items
- biding_type should accept VectorDouble and VectorFloat attributes.
- on_value change verify that vectors have the same length.
- On each value update clean scatter graph and plot new points.

## Hands on #8 : result




You, 1 minute ago | 2 authors (Ivars Karpics and others)

```
@register_binding_controller(ui_name="Workshop Example 3",
                             klassname="WorkshopExampleThree",
                             binding_type=(VectorFloatBinding, VectorDoubleBinding))
```


```
def value_updat
    """Calls wh
    # We need b
    if self.y_p
        return

    # Plot poin
    x_values = get_binding_value(self.proxy)
    y_values = get_binding_value(self.y_proxy)
    if x_values is not None and y_values is not None:
        self._scatter_plot.clear()
        if x_values.size == y_values.size:
            self._scatter_plot.addPoints(
                [{"pos": [x, y]} for x, y in zip(x_values, y_values)])
```

```
(function) def get_binding_value(
    binding: Any,
    default: Any | None = None
) -> (Any | None)
```

Get the binding value, this function is used to deal with Undefined or None

:param default: return default if value is None or Undefined

## Further reading

- Karabo scenes: https://rtd.xfel.eu/docs/karabo/en/latest/library/gui_scene_development.html
- GUI Extensions: https://rtd.xfel.eu/docs/gui-extensions/en/latest/install_latest_version.html
- Qt: https://doc.qt.io/qt-5/qtwidgets-index.html , https://doc.qt.io/qt-5/qtwidgets-module.html
- PyQt graph: https://www.pyqtgraph.org/

## Take Away

- Karabo GUI has a rich set of built-in widgets.
- Most of them are adjustable to fit user needs.
- Missing gui appearance and complex widgets might be achieved via gui extensions.
- Use existing set of Karabo GUI widgets, ask for support/feature or develop your gui extensions.