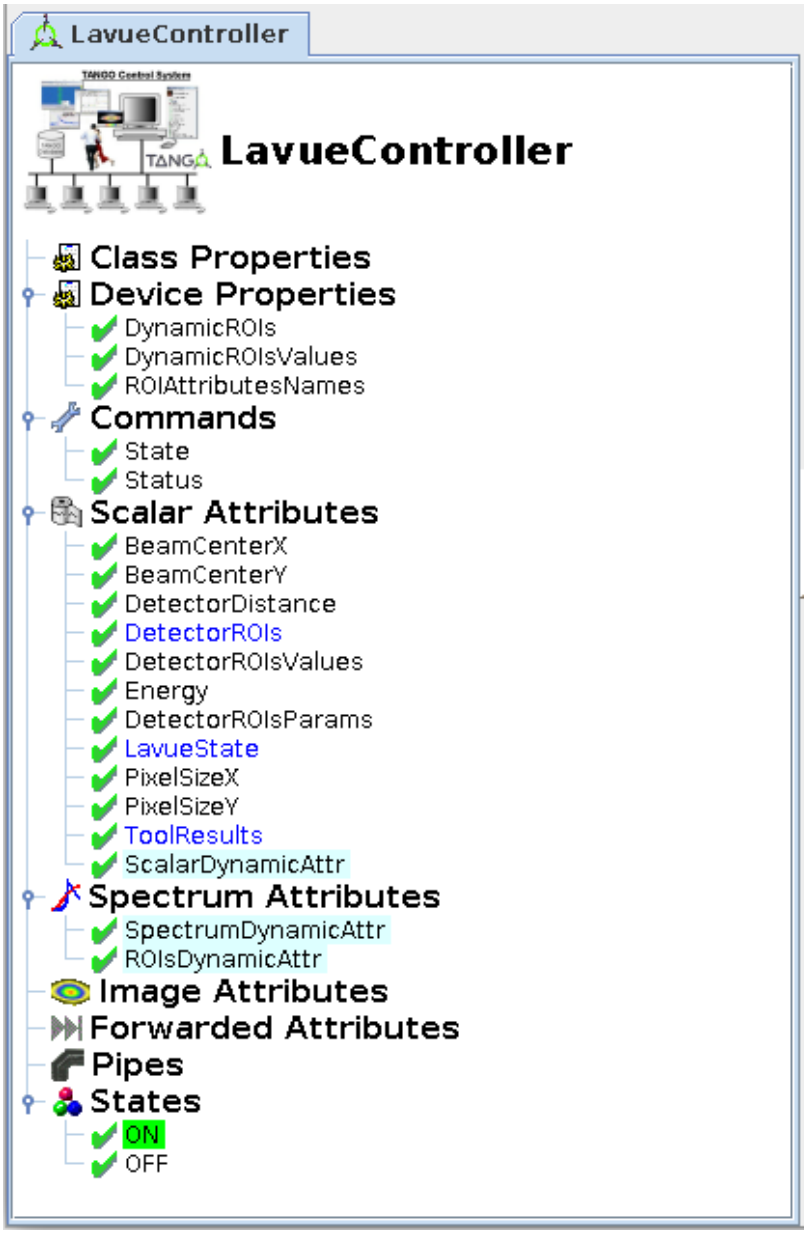
LaVue - LavueController server

LavueController server allows for communication with LaVue GUI via tango interface, e.g. with user scripts or macros



Attributes

The user can **pass to** and **read back** from the LaVue GUI the following attributes:

- **BeamCenterX:** x-coordinate of the beam center in *pixels*, e.g. used in `Angle/Q Tool`
- **BeamCenterY:** y-coordinate of the beam center in *pixels*, e.g. used in `Angle/Q Tool`
- **PixelSizeX:** x-size of the detector pixel in *micrometers*, e.g. used in `Angle/Q Tool`
- **PixelSizeY:** y-size of the detector pixel in *micrometers*, e.g. used in `Angle/Q Tool`
- **DetectorDistance:** detector distance from the sample in *mm*, e.g. used in `Angle/Q Tool`
- **Energy:** beam energy in *eV*, e.g. used in `Angle/Q Tool`
- **DetectorROIs:** JSON dictionary with Regions Of Interests ranges, e.g.
  `{"pilatusrois": [[67, 131, 124, 153], [67, 69, 117, 119], [134, 129, 184, 179], [125, 72, 175, 122]]}`

Moreover the user can **read** from the LaVue GUI:

- **DetectorROIsValues**: JSON dictionary with Regions Of Interests sums, e.g.
  `{"pilatusrois": [8167.0, 2262.0, 478.0, 1069.0]}`
- **DetectorROIsParams**: JSON list of image transformations performed by lavue, e.g.
  `["transpose", "flip-left-right", "flip-up-down"]`
- **ToolResults:** JSON dictionary with tool results, i.e. 1d diffractogram plot or position of peaks

Finally, the user can change state of lavue by **writing** to

- **LavueState:** JSON dictionary with lavue configuration with parameters corresponding to command-line parameters of lavue (to display them: `lavue -h`). The currently supported commands are: `source, configuration, start, stop, imagefile, offset, rangewindow, dsfactor, dsreduction, filters, mbuffer, maskfile, maskhighvalue, transformation, scaling, levels, autofactor, gradient, viewrange, tool, toolconfig, tangodevice, doordevice, analysisdevice, log`.

  e.g.

```
import tango
import json

lc = tango.DeviceProxy("p09/lavuecontroller/1")

# start the Test source
lc.LavueState = json.dumps({"source": "test", "start": True})

# stop an image source
lc.LavueState = json.dumps({"stop":True})

# set the hidra source with a server configured
lc.LavueState = json.dumps({"source": "hidra", "configuration": "haspilatus1m.desy.de"})

# start Tango Attribute image source with a tango attribute from TangoTest server
# lc.LavueState = json.dumps({"source": "tangoattr", "configuration": "sys/tg_test/1
/double_image_ro", "start": True})

# set ROI tool in the tool combobox
lc.LavueState = json.dumps({"tool": "roi"})

# set display intensity levels
lc.LavueState = json.dumps({"levels": "m20,20"})

# set level auto factor
lc.LavueState = json.dumps({"autofactor": 1})

# set auto levels
lc.LavueState = json.dumps({"autofactor": ""})

# set mask values above
lc.LavueState = json.dumps({"maskhighvalue": 30000})

# set flip-up-down transformation
lc.LavueState = json.dumps({"transformation": "flip-up-down"})

# set log intensity scaling
lc.LavueState = json.dumps({"scaling": "log"})

# set eiger image source with configuration defined by alias "eiger"
#   (otherwise you have to set as configuration the whole URL string)
lc.LavueState = json.dumps({"source":"http", "configuration":"eiger", "start":True})
```

You can **read** the current LaVue state via `LavueState`. To update the `viewrange` parameter write to `LavueState` an empty JSON dictionary or `{"__update__": true}`.

Properties

The **DynamicROIsValues** device property can be set to `true` or `false.` When it is true dynamic attributes with ROIs sums are added.

They names are defined by ROIs aliases from Lavue GUI.

The **DynamicROIs** device property can be set to `true` or `false.` When it is true dynamic attributes with ROIs bounds are added.

They names are defined by ROIs aliases from Lavue GUI.

The **ROIAttributesNames** device property contains names of dynamic attributes which will be created event if the corresponding ROIs aliases in Lavue GUI are missing.