HowToGUI

Release 1.0

Controls

Aug 16, 2023

CONTENTS

Contents:

CHAPTER

ONE

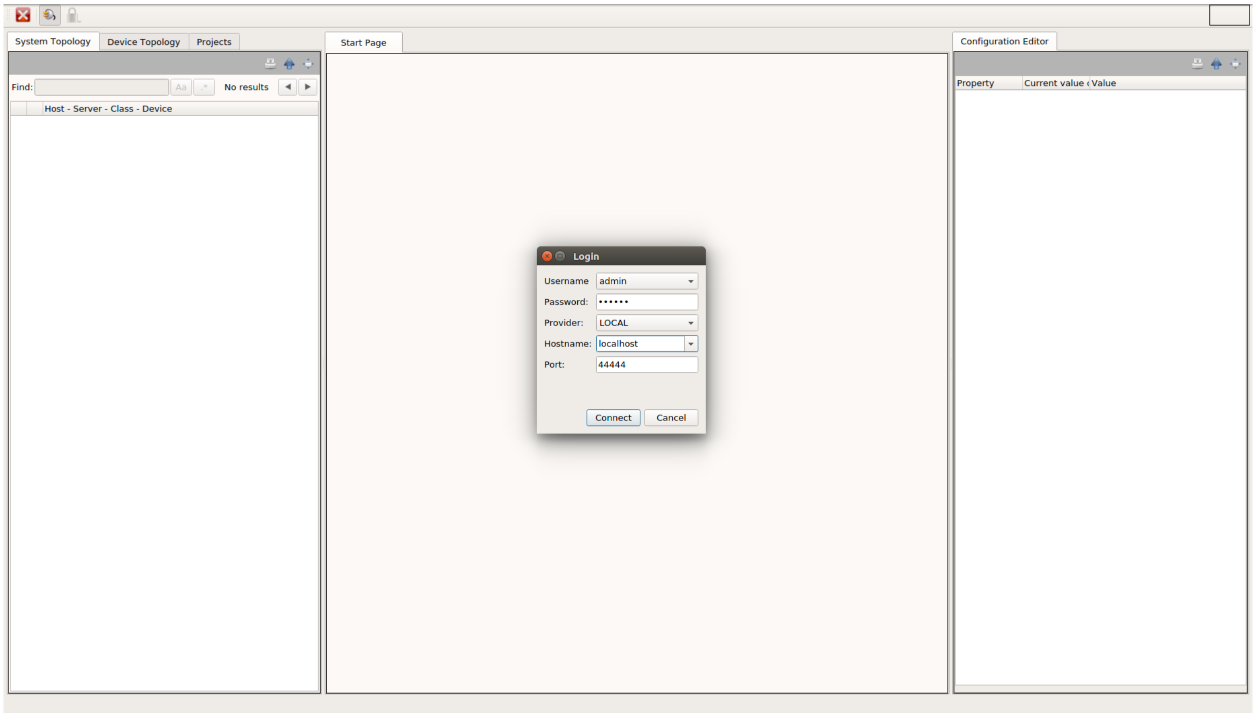
THE KARABO GUI

1.1 Getting Started

The GUI starts up into a non-connected state, meaning that you need to login to a specific GUI server with your login credentials. By doing so your access level is also determined and the appropriate options will be available to you.

```
karabo-gui
```

will open up this panel:



Username admin

Password: ••••••

Provider: LOCAL

Hostname: localhost

Port: 44444

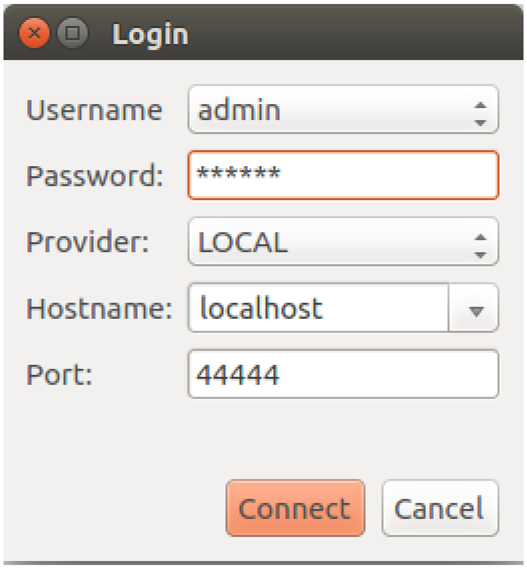
Fig. 1: The karabo GUI application directly after startup

You can see that most of the panels are still empty, as you are in a non-connected state. From left to right the following panels are available:

- The **navigation panel**, which gives you a live view of the full system topology and allows for filtering (`Left Area`)
- The **project panel**, giving you a logical view on the projects you have loaded (`Left Area`)

- The **scene panel**, which is used to display custom views (`Middle Area`)
- The **service panels**, giving access to logging, alarm service and an `ikarabo` console (`Middle Area`)
- The **configurator panel**, which lists *all* properties and slots available for a device as appropriate for your access level (`Right Area`)

The *connect dialog* automatically shows up after starting the GUI application. Alternatively, for connecting to a `GUI server`, the **connect to server** button can be clicked in the top left of the application. The connect dialog requires the operator credentials and the gui server information (host and port) for connecting. The `GUI server` selection field remembers the *5* last recently used gui servers configurations.



Username admin

Password: ******

Provider: LOCAL

Hostname: localhost

Port: 44444

Fig. 2: The GUI connection dialog.

After connecting to a GUI server, the GUI application will receive the Broker and Topology information showing a live view of the system.

Any of the listed panels may be detached and arranged separately on the screen. In the following each panel is introduced in more detail.

1.1.1 How to connect to the GUI Server

In order to use the Karabo GUI with the control network, we have to open an ssh tunnel towards the desired Karabo GUI server. This can be done as follows:

```
ssh desyusername@exflgateway -L LOCAL_FORWARDING_PORT:GUI_SERVER_ALIAS:GUI_SERVER_PORT
```

Hence, a tunnel forwarding from the GUI server of the XTD2/XTD9 installation looks like:

```
ssh desyusername@exflgateway -L 44444:sa1-br-sys-con-gui1:44444
```

Afterwards, launch the Karabo GUI on your local machine and connect to the GUI Server with the settings **localhost** and port **44444**.

Sometimes it is even beneficial if you make the forwarding port differ from 44444 (e.g. 44445) if you have a local running GUI server already running on port 44444.

```
ssh desyusername@exflgateway -L 44445:sa1-br-sys-con-gui1:44444
```

In case the connection is refused, contact the responsible control network responsibles once you filled the control network access form.

If you want to connect a GUI from outside the DESY network, you have to tunnel twice, once through bastion.desy.de to get into the DESY network and then through exflgateway, e.g.

```
ssh -L38080:localhost:38081 desyusername@bastion.desy.de -t ssh -L38081:sa1-br-sys-con-
↪gui1:44444 exflgateway
```

where 38080 is the local forwarding port and 38081 an intermediate port.

1.2 The Cinema

The Karabo GUI is capable to run in a `cinema` fashion, by just providing a project database `domain` and the corresponding `uuid` of the scene:

```
karabo-cinema DOMAIN UUID
```

Since uuids are not convenient to handle, a key-stroke `ctrl + c` is available from the project panel project_intro to copy the uuid to the `selection clipboard`. The selection clipboard can be pasted with the *middle mouse button*.

The cinema can connect directly by providing host and port. Furthermore, multiple scenes can be launched and a username can be provided.

```
karabo-cinema DOMAIN UUID -host HOSTNAME -port PORTNUMBER -username USERNAME
```

```
karabo-cinema LOCAL 7f9023e8-bae3-4352-beb9-b0c162097b60 1d057710-2cc3-4410-a587-
↪61ecb3c156c2 -host localhost -port 44444 -username expert
```

1.3 The Processing Lamp

A processing lamp is visible in the menu bar on the right side of the GUI on top of the configurator. It is used to visualize the difference between the arrival time of packet and its actual processing within the GUI client.

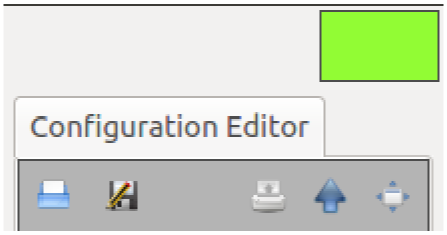


Fig. 3: The processing lamp

Color	status	description
	Fine	The Karabo GUI is up to date
	Warning	The updates are at least behind by **2s**
	Alarm	The updates are at least behind by **5s**

The delay in processing only happens in very rare cases, e.g. Project loading. Without activity in the GUI, the processing lamp will not be able to change its color.

Note: The operator is still able to send out changes when the processing delay is large, e.g. stop the movement of motors!

CHAPTER

TWO

THE NAVIGATION PANEL

The navigation panels gives you a live view of the system, in a tree hierarchy. This is in contrast to the project panel, which presents a logical view of the system, as grouped by components.

In its default view, the navigation panel shows the system topology in tree hierarchy showing the unified and alarm states of the running devices. This is automatically shown upon successfully connecting to the GUI server of the desired topic.

This is comprised of the following components, and will be further discussed on the following chapters:

1. Panel Bar
2. Search Bar
3. System Topology

2.1 System Topology

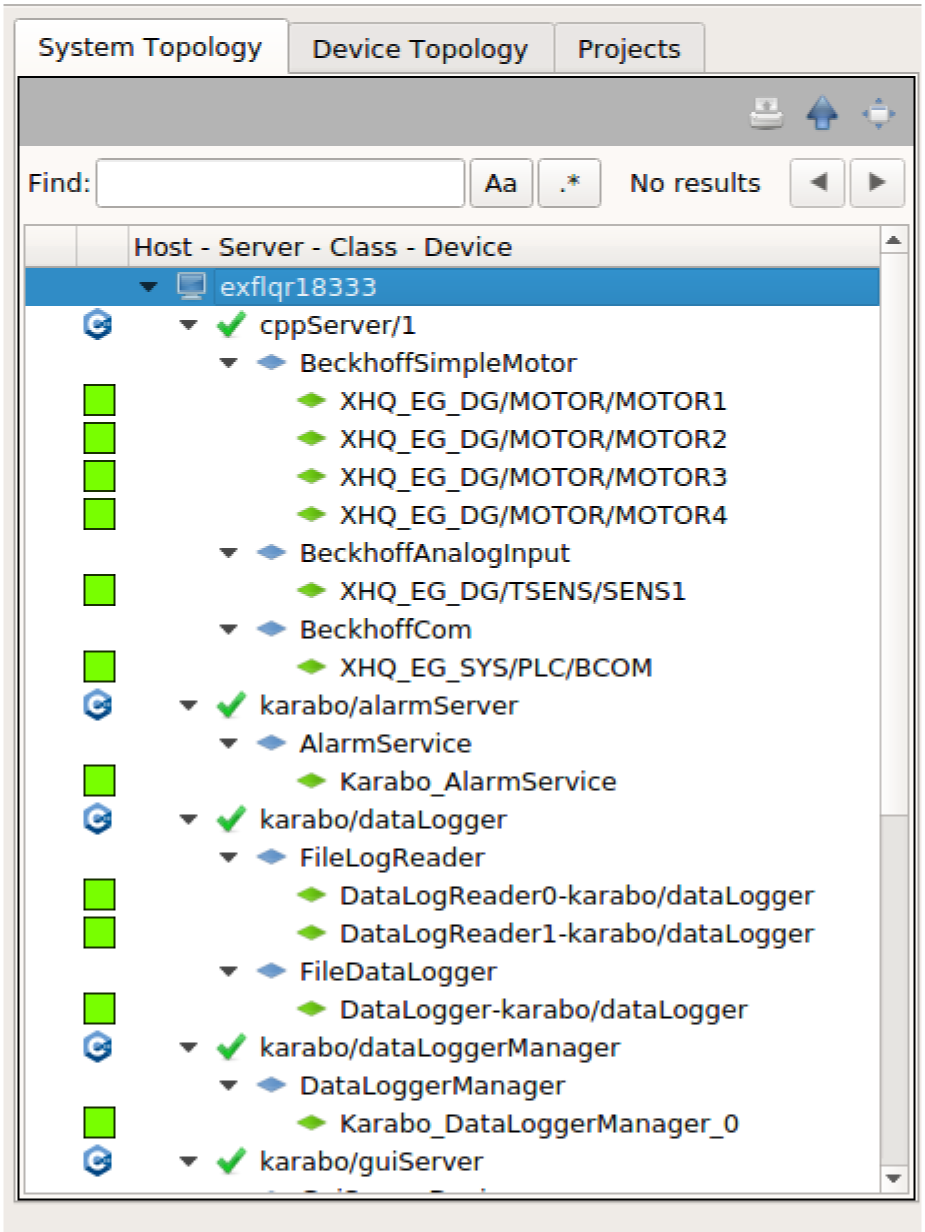
The system topology consists of Karabo components, namely: control servers, device servers, device classes and device instances.

- Control servers are the physical machine running the device servers.
- Device servers are the hosts of devices.
- Device classes are the templates of devices.
- Device instances are running devices on the device server.

These components are show in a tree structure:

```
control_server
├── device_server
    ├── device_class
        ├── device_instance
```

and can be seen in the Navigation Panel as follows:



```
exflqr18333
    ✓ cppServer/1
        BeckhoffSimpleMotor
🟩          XHQ_EG_DG/MOTOR/MOTOR1
🟩          XHQ_EG_DG/MOTOR/MOTOR2
🟩          XHQ_EG_DG/MOTOR/MOTOR3
🟩          XHQ_EG_DG/MOTOR/MOTOR4
        BeckhoffAnalogInput
🟩          XHQ_EG_DG/TSENS/SENS1
        BeckhoffCom
🟩          XHQ_EG_SYS/PLC/BCOM
    ✓ karabo/alarmServer
        AlarmService
🟩          Karabo_AlarmService
    ✓ karabo/dataLogger
        FileLogReader
🟩          DataLogReader0-karabo/dataLogger
🟩          DataLogReader1-karabo/dataLogger
        FileDataLogger
🟩          DataLogger-karabo/dataLogger
    ✓ karabo/dataLoggerManager
        DataLoggerManager
🟩          Karabo_DataLoggerManager_0
    ✓ karabo/guiServer
```

Fig. 1: The Karabo navigation panel. Icons next to the device instance id give you the **color code status** of the device, as well as the alarm condition of the device if an alarm is active.

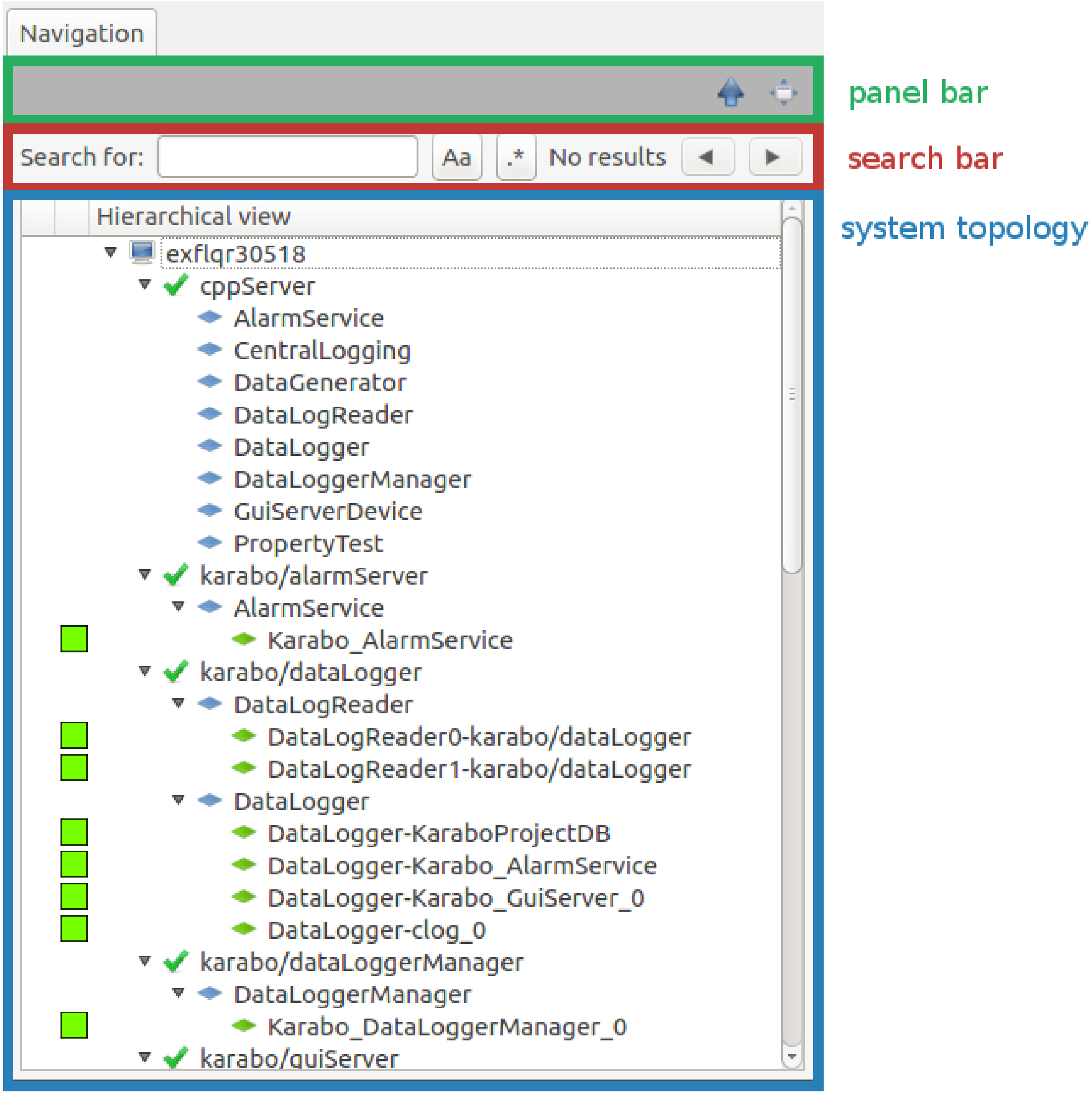
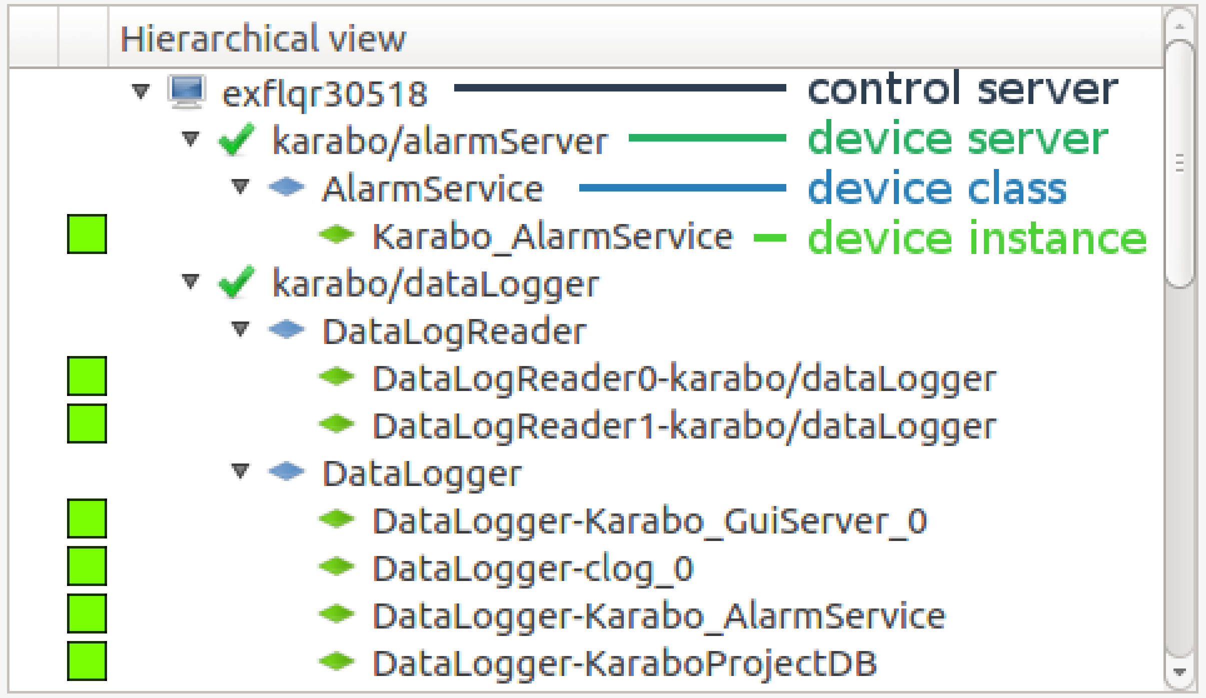


Fig. 2: A navigation panel is composed of the panel bar, the search bar, and the corresponding topology tree (`System Topology`).



2.1.1 🖥 Control Server

The control server is a machine or computer that hosts Karabo components, namely device servers, devices, clusters of brokers, and database servers. This usually has a name `exfl_____`. A system could have one or more control servers.

2.1.2 ✓ Device Server

Device servers are the hosts for Karabo devices. They are responsible for loading the device code and running the device in an event loop, i.e. keeping it live and available to the distributed system. Device servers come in three flavors: C++, Python and middle-layer device servers.

Context menu

Right-clicking will show a context menu, which are as follows:

1. Shutdown server

   This will restart the server, which takes a few seconds. When this option is selected, the server will be removed in the system topology but will then restart and reappear after the last server of the same control server.

2. About

   This will show a dialog that contains keys which describes the device server.

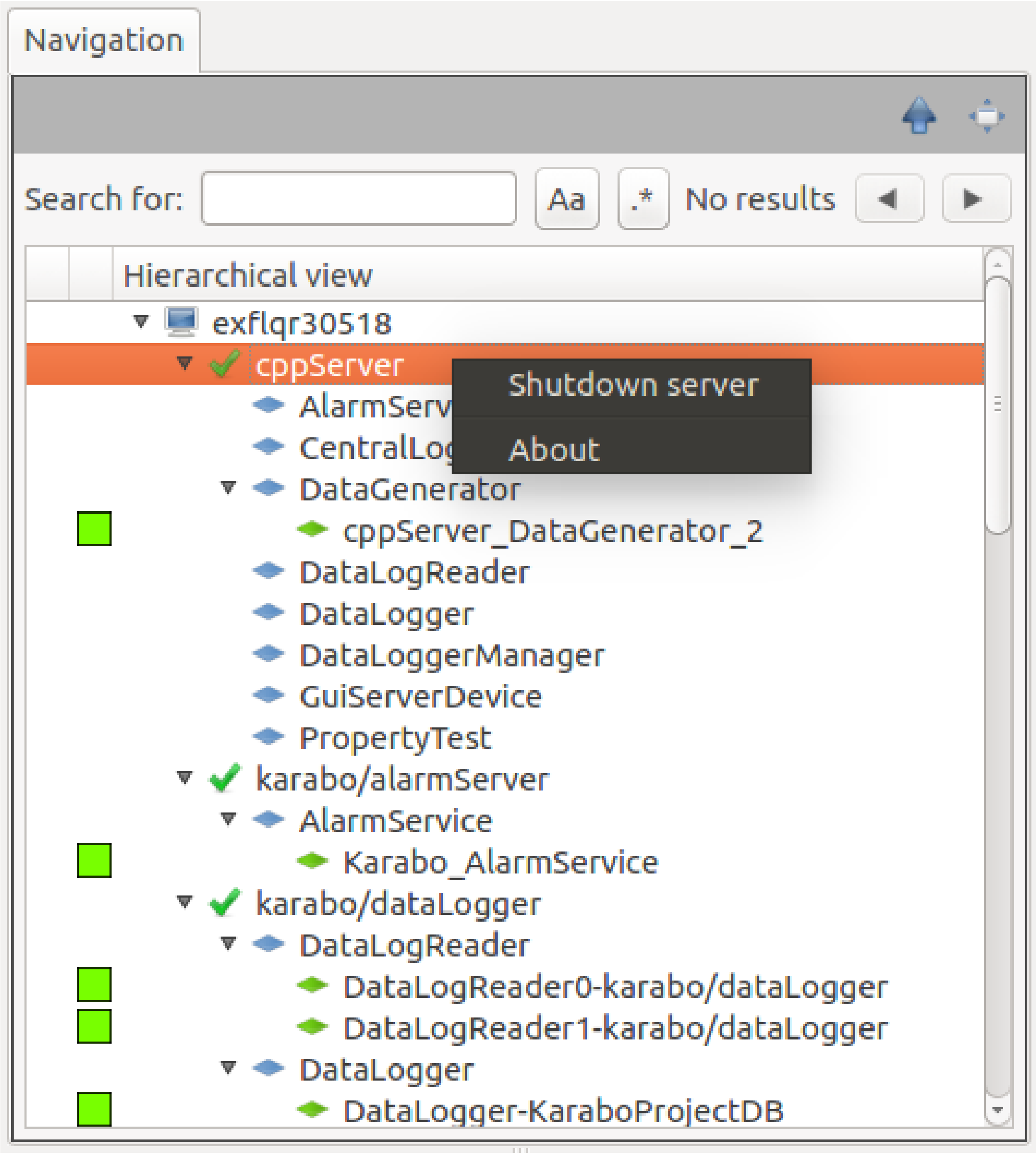
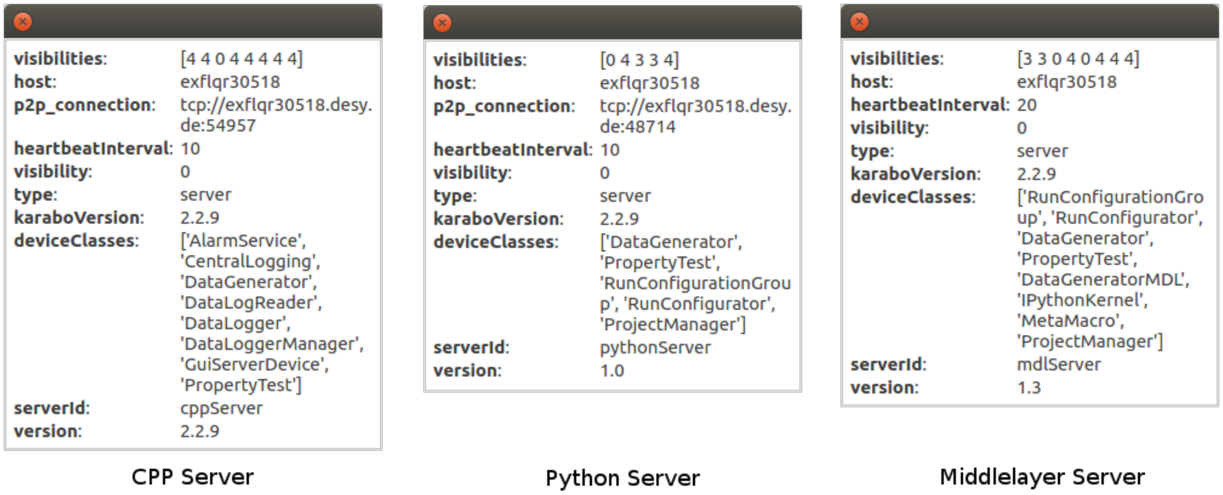


Fig. 3: The server context menu.

About

The contents of **About** varies for different servers classes, which can be seen as follows:



visibilities: [4 4 0 4 4 4 4 4]
host: exflqr30518
p2p_connection: tcp://exflqr30518.desy.de:54957
heartbeatInterval: 10
visibility: 0
type: server
karaboVersion: 2.2.9
deviceClasses: ['AlarmService', 'CentralLogging', 'DataGenerator', 'DataLogReader', 'DataLogger', 'DataLoggerManager', 'GuiServerDevice', 'PropertyTest']
serverId: cppServer
version: 2.2.9

CPP Server

visibilities: [0 4 3 3 4]
host: exflqr30518
p2p_connection: tcp://exflqr30518.desy.de:48714
heartbeatInterval: 10
visibility: 0
type: server
karaboVersion: 2.2.9
deviceClasses: ['DataGenerator', 'PropertyTest', 'RunConfigurationGroup', 'RunConfigurator', 'ProjectManager']
serverId: pythonServer
version: 1.0

Python Server

visibilities: [3 3 0 4 0 4 4 4]
host: exflqr30518
heartbeatInterval: 20
visibility: 0
type: server
karaboVersion: 2.2.9
deviceClasses: ['RunConfigurationGroup', 'RunConfigurator', 'DataGenerator', 'PropertyTest', 'DataGeneratorMDL', 'IPythonKernel', 'MetaMacro', 'ProjectManager']
serverId: mdlServer
version: 1.3

Middlelayer Server

Fig. 4: The About dialog of the different device server classes.

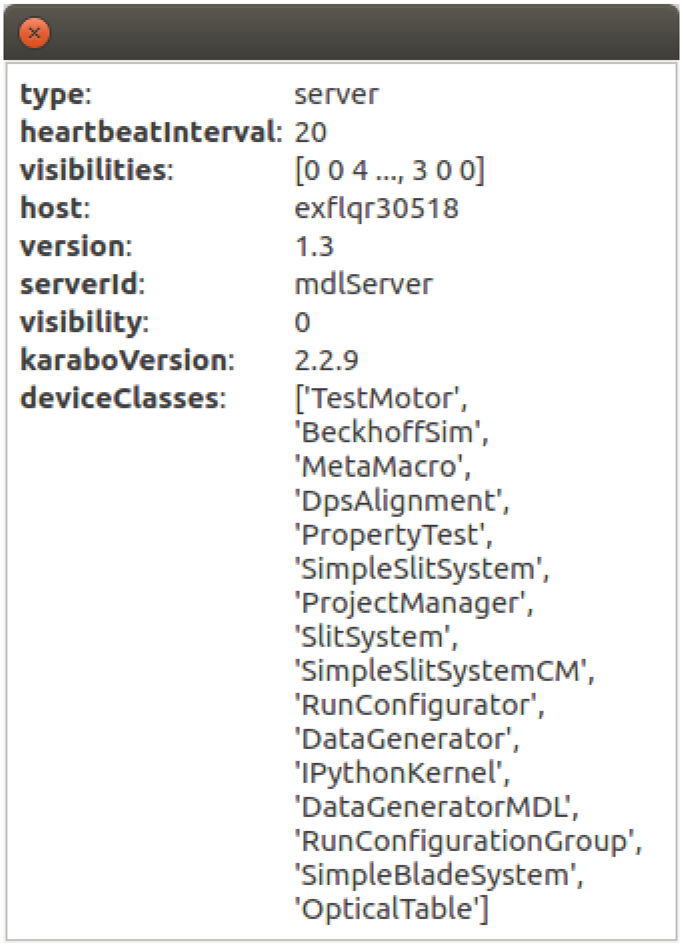
Below is a more detailed information about each keys.

key	type	description
`type`	string	Karabo component type: `host`: control server `server`: device server `class`: device class `device`: device instance
`serverID`	string	The ID of the device server.
`version`	string	The version of the device server.
`host`	string	The ID of the control server where the server is located.
`visibility`	integer	User access level who can see the server: `0`: observer *(default)* `1`: user `2`: operator `3`: expert `4`: admin `5`: god
`heartbeatInterval`	integer	Interval in seconds at which the server sends heartbeats to distributed systems
`karaboVersion`	string	Current version of the Karabo Framework.
`p2p_connection`	string	Complete address of the host (`tcp://host:port`) for p2p connection
`deviceClasses`	vector string	List of devices classes that can be instantiated in the server.
`visibilities`	vector integer	List of visibility of the device classes.

2.1.3 Device Class

Device classes that can be run on the device server are shown at its children in the `System Topology`. They appear in the topology when a device of the respective class is online.

The list of available classes can be viewed via the server's **About**.

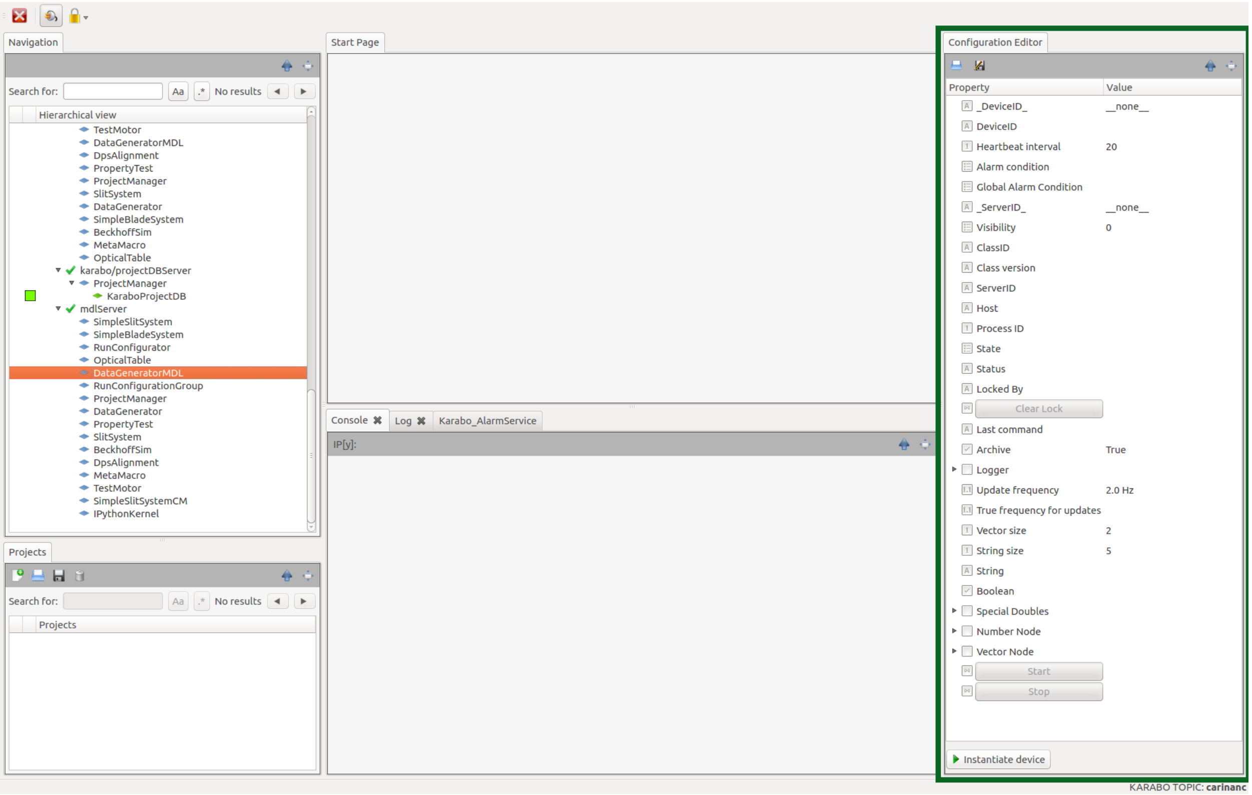


type:	server
heartbeatInterval:	20
visibilities:	[0 0 4 ..., 3 0 0]
host:	exflqr30518
version:	1.3
serverId:	mdlServer
visibility:	0
karaboVersion:	2.2.9
deviceClasses:	['TestMotor', 'BeckhoffSim', 'MetaMacro', 'DpsAlignment', 'PropertyTest', 'SimpleSlitSystem', 'ProjectManager', 'SlitSystem', 'SimpleSlitSystemCM', 'RunConfigurator', 'DataGenerator', 'IPythonKernel', 'DataGeneratorMDL', 'RunConfigurationGroup', 'SimpleBladeSystem', 'OpticalTable']

Fig. 5: The device classes are also listed in the server's About dialog.

Configuration

Clicking the **device class** will show the default device `Schema` in the Configuration Panel.



Property	Value
DeviceID	__none__
DeviceID	
Heartbeat interval	20
Alarm condition	
Global Alarm Condition	
ServerID	__none__
Visibility	0
ClassID	
Class version	
ServerID	
Host	
Process ID	
State	
Status	
Locked By	
Last command	
Archive	True
Logger	
Update frequency	2.0 Hz
True frequency for updates	
Vector size	2
String size	5
String	
Boolean	
Special Doubles	
Number Node	
Vector Node	

Fig. 6: The default device class configuration will be shown when clicking the device class in the Project Panel.

Context menu

Right-clicking will show a context menu, which are as follows:

1. Open configuration (**.xml*)

   This will load a device class configuration file.

2. Save configuration as (**.xml*)

   This will save the current device class configuration as an `.xml`.

See also:

Configuration Panel.

3. Open device scene

   This option will only appear when a device class has been instantiated for the session and if it has a default scene. If selected, the default scene for the device will be shown on the Scene Panel. The retrieved scene from a navigation panel will not be attached to the project.

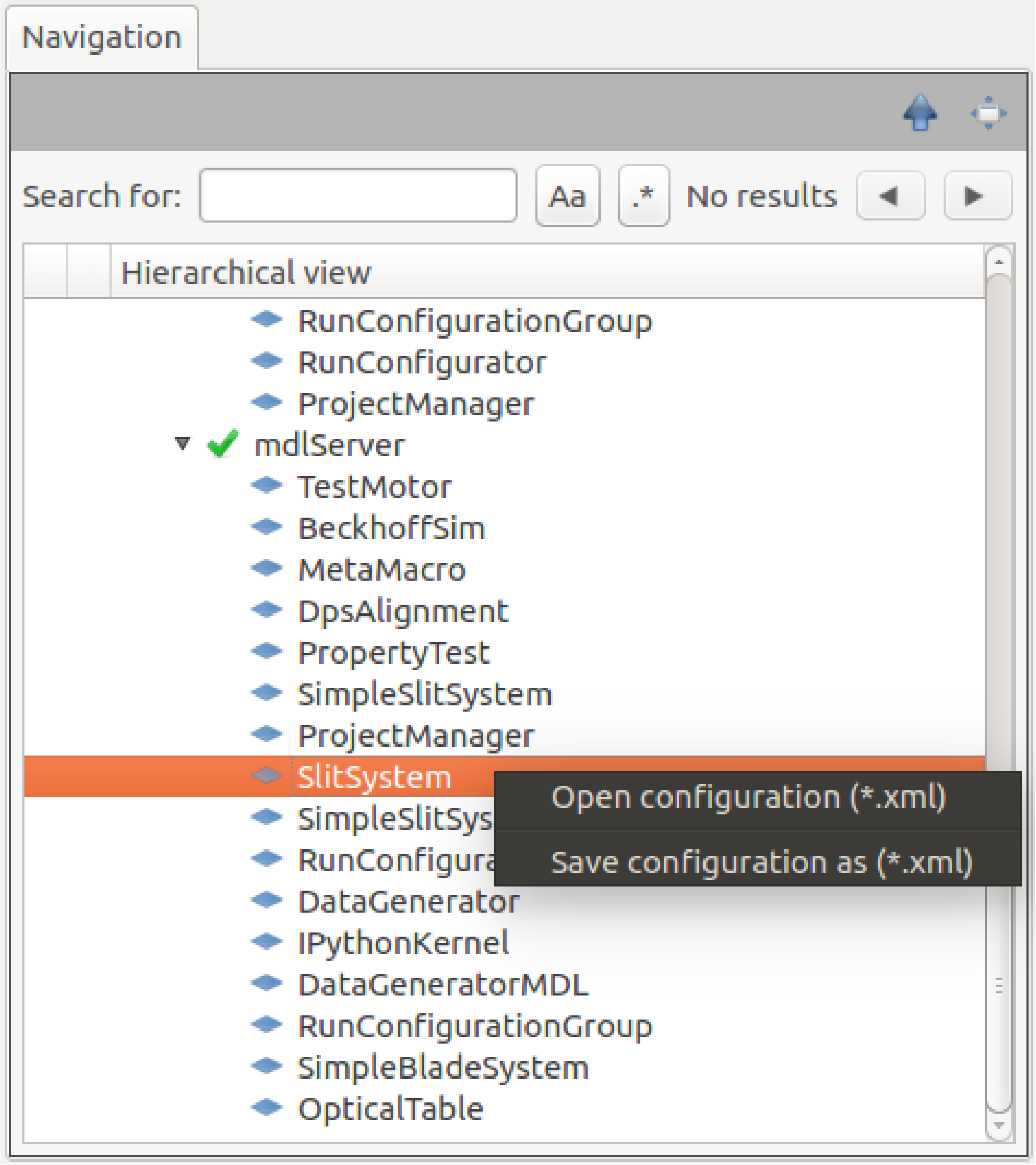
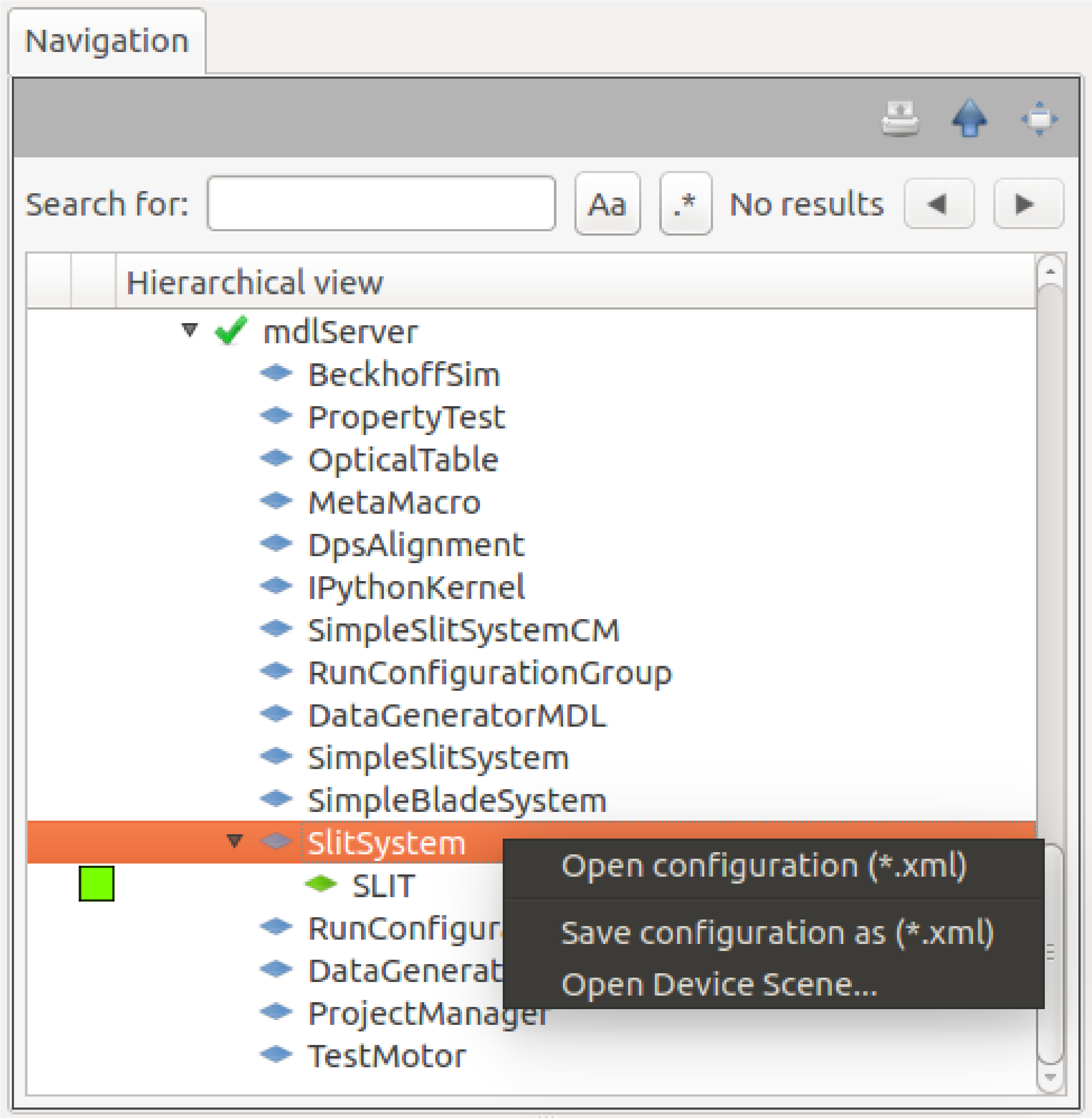


Fig. 7: The device class context menu.


Navigation
Search for:
Aa
.*
No results
Hierarchical view
mdlServer
BeckhoffSim
PropertyTest
OpticalTable
MetaMacro
DpsAlignment
IPythonKernel
SimpleSlitSystemCM
RunConfigurationGroup
DataGeneratorMDL
SimpleSlitSystem
SimpleBladeSystem
SlitSystem
SLIT
RunConfigur
DataGenerat
ProjectManager
TestMotor
Open configuration (*.xml)
Save configuration as (*.xml)
Open Device Scene...

See also:

Scene Panel.

2.1.4 ◆ Device Instance

When a device is successfully instantiated, it goes online and runs on the specified device server. A device instance is shown under the device class of the device server that hosts it. Its status can easily be perceived in the Navigation Panel with the help of some visual indications.

Instance status

The device instance icons change depending on its status.

icon	status	description
◆	Instantiated	The device is instantiated and is healthy.
◆⛔	Instance Error	The device is instantiated but has encountered error(s)
	Monitored	The device is instantiated and its configuration is being monitored via the scene or configuration panel.

Alarm levels

The alarm level of device instances can be seen in the first column of the system topology. This appears when one of its properties trigger an alarm. Below are the alarm indications and their meanings.

icon	**status**	**description**
	None	The device is working normally.
	Warning	A device property is in a value range where it should be monitored.
	Critical	A device property is in its critical value range.
	Interlock	A device property has triggered an interlock.

See also:

The Karabo SCADA Framework: Alarm System

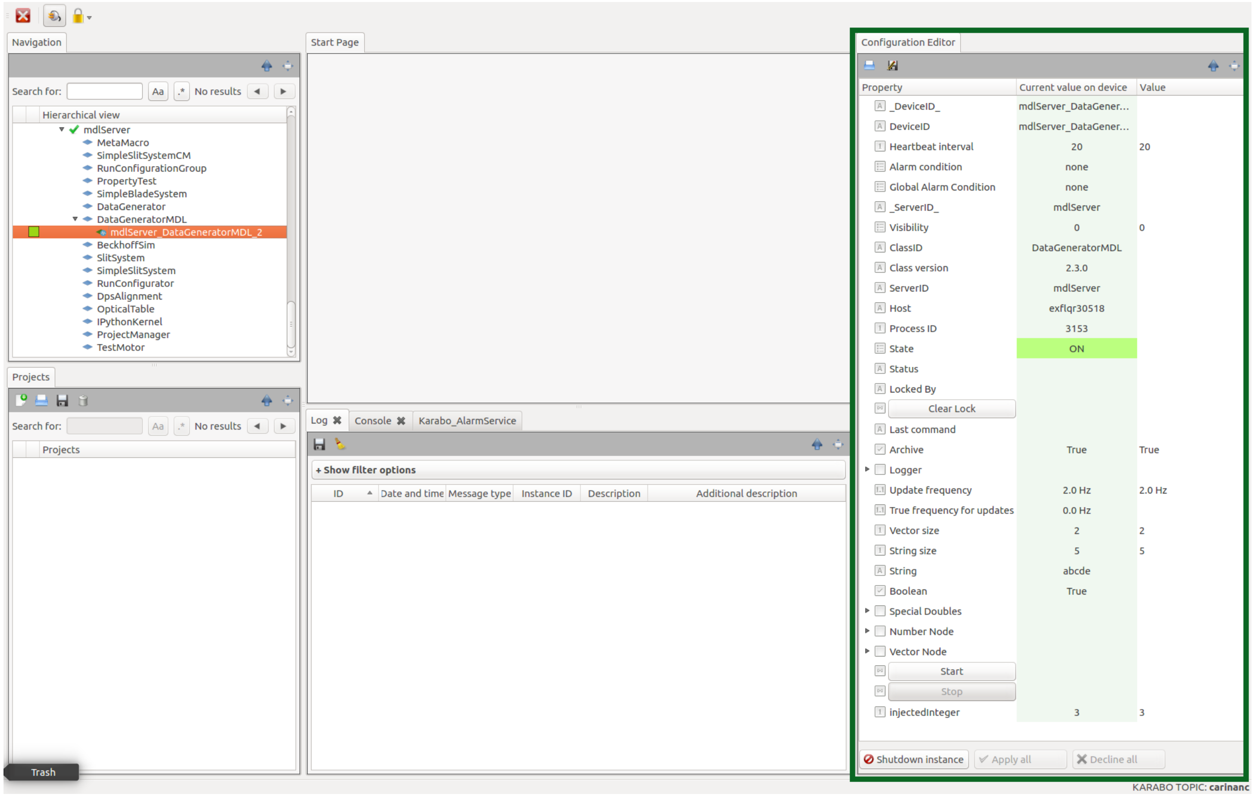
Device status

The device status can also be perceived from the status icons that appear on the second column of the system topology.

icon	status	description
	error	Hardware and/or pipeline-processing error(s) has been encountered.
	ok	The device did not encounter any errors

Configuration

Clicking the **device instance** will show the current device instance configuration in the Configuration Panel.



Property	Current value on device	Value
DeviceID	mdlServer_DataGener...	
DeviceID	mdlServer_DataGener...	
Heartbeat interval	20	20
Alarm condition	none	
Global Alarm Condition	none	
ServerID	mdlServer	
Visibility	0	0
ClassID	DataGeneratorMDL	
Class version	2.3.0	
ServerID	mdlServer	
Host	exflqr30518	
Process ID	3153	
State	ON	
Status		
Locked By		
Clear Lock		
Last command		
Archive	True	True
Logger		
Update frequency	2.0 Hz	2.0 Hz
True frequency for updates	0.0 Hz	
Vector size	2	2
String size	5	5
String	abcde	
Boolean	True	
Special Doubles		
Number Node		
Vector Node		
Start		
Stop		
injectedInteger	3	3

Fig. 8: The instance of the new device configuration.

This will also change its icon to , denoting that it is the current device configuration shown in the Configuration Panel.

Context menu

Right-clicking will show a context menu, which are as follows:

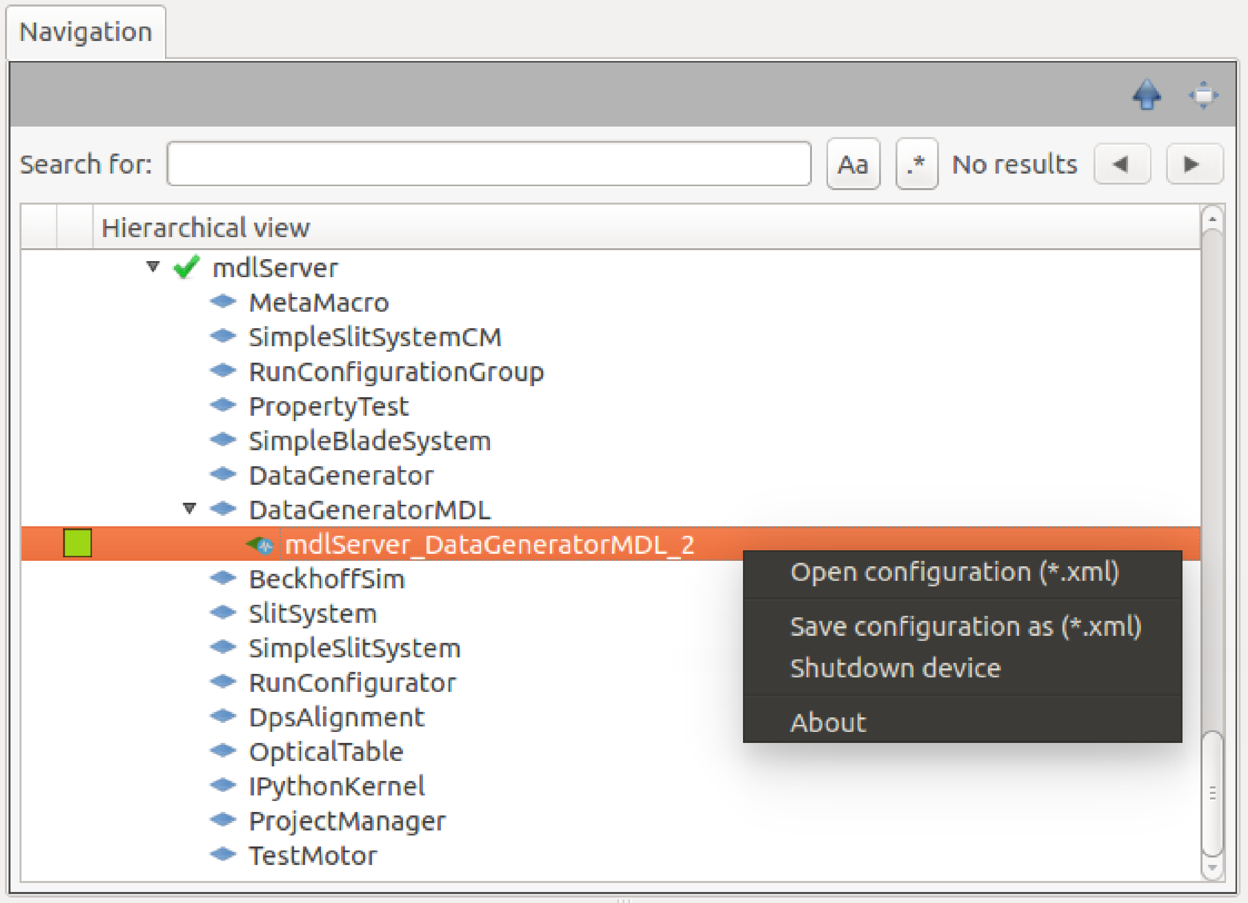


Fig. 9: The context menu of the device instance.

1. Open configuration (**.xml*)

   - This will load a device class configuration file.

2. Save configuration as (**.xml*)

   - This will save the current device class configuration as an `.xml`.

3. Shutdown device

   - This will shutdown the device instance and remove it from the system.

4. Open device scene

   This option will only appear for devices with available scenes. If selected, a dialog where the scene to be displayed can be selected.

5. About

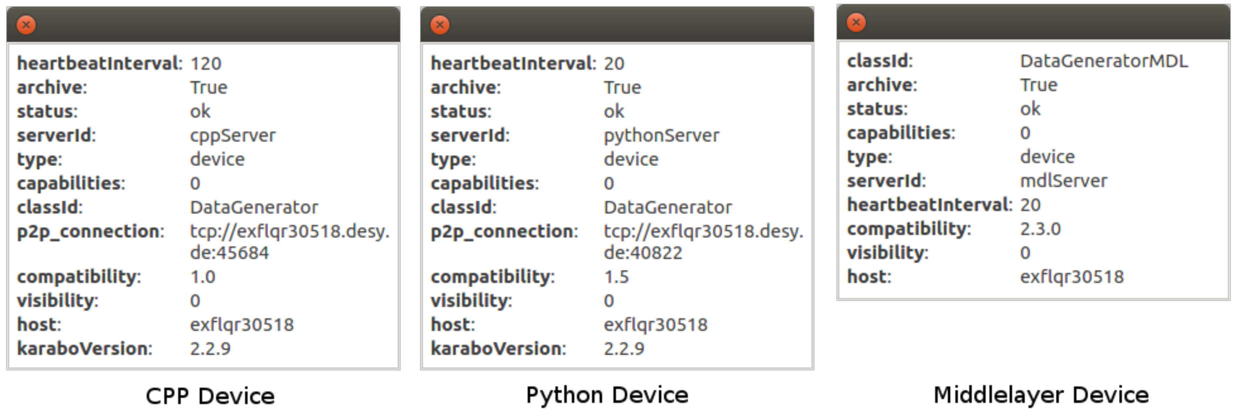
   This will show a dialog that contains keys which describes the device instance.

Opening Default Device Scene

Aside from displaying scenes from the context menu option, the default device scene can also be opened by **double-clicking** the device instance.

About

The contents of **About** varies for different instance classes, which can be seen as follows:



heartbeatInterval:	120
archive:	True
status:	ok
serverId:	cppServer
type:	device
capabilities:	0
classId:	DataGenerator
p2p_connection:	tcp://exflqr30518.desy.de:45684
compatibility:	1.0
visibility:	0
host:	exflqr30518
karaboVersion:	2.2.9

CPP Device

heartbeatInterval:	20
archive:	True
status:	ok
serverId:	pythonServer
type:	device
capabilities:	0
classId:	DataGenerator
p2p_connection:	tcp://exflqr30518.desy.de:40822
compatibility:	1.5
visibility:	0
host:	exflqr30518
karaboVersion:	2.2.9

Python Device

classId:	DataGeneratorMDL
archive:	True
status:	ok
capabilities:	0
type:	device
serverId:	mdlServer
heartbeatInterval:	20
compatibility:	2.3.0
visibility:	0
host:	exflqr30518

Middlelayer Device

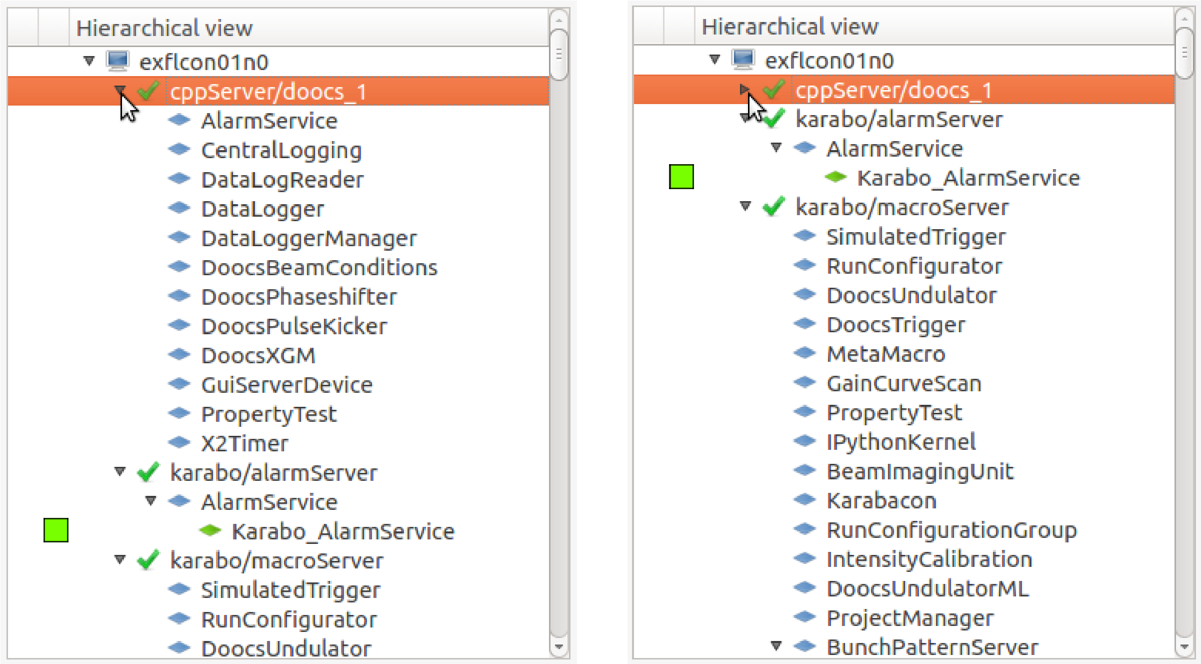
Fig. 10: The About dialog of the different device instance classes.

Below is a more detailed information about each keys.

key	type	description
heartbeatInterval	integer	Interval in seconds at which the server sends heartbeats to distributed systems
archive	bool	Whether the device is archived in the DataLogger
status	string	Description of the device status.
serverId	string	The ID of device server which the device is running on: cppServer pythonServer mdlServer
type	text	Karabo component type: host: control server server: device server class: device class device: device instance
capabilities	integer	What the device can do, a sum of: 1: provides scenes 2: provides macros 4: provides interfaces
classId	string	The ID of the device class of the instance.
p2p_connection	string	Complete address of the host (tcp://host:port)
compatibility	string	Earliest version of Karabo Framework the device is compatible with.
visibility	integer	User access level who can see the device: 0: observer *(default)* 1: user 2: operator 3: expert 4: admin 5: god
host	string	The ID of the control server where the device is running on.
karaboVersion	string	Current version of the Karabo Framework

2.1.5 Expanding/collapsing the tree

The system topology is expanded by default. This means all items are being shown in the Navigation Panel, resulting to a lengthy topology. A component can be collapsed, with its members being hidden from the list. The arrow icon beside the component can be clicked in order to do so.



Hierarchical view

- exflcon01n0
  - cppServer/doocs_1
    - AlarmService
    - CentralLogging
    - DataLogReader
    - DataLogger
    - DataLoggerManager
    - DoocsBeamConditions
    - DoocsPhaseshifter
    - DoocsPulseKicker
    - DoocsXGM
    - GuiServerDevice
    - PropertyTest
    - X2Timer
  - karabo/alarmServer
    - AlarmService
      - Karabo_AlarmService
  - karabo/macroServer
    - SimulatedTrigger
    - RunConfigurator
    - DoocsUndulator

Hierarchical view

- exflcon01n0
  - cppServer/doocs_1
  - karabo/alarmServer
    - AlarmService
      - Karabo_AlarmService
  - karabo/macroServer
    - SimulatedTrigger
    - RunConfigurator
    - DoocsUndulator
    - DoocsTrigger
    - MetaMacro
    - GainCurveScan
    - PropertyTest
    - IPythonKernel
    - BeamImagingUnit
    - Karabacon
    - RunConfigurationGroup
    - IntensityCalibration
    - DoocsUndulatorML
    - ProjectManager
    - BunchPatternServer

Fig. 11: A Karabo component with members (such as `cppServer/doocs_1`) can be expanded (left) and collapsed (right) by clicking on the arrow icon beside it.

The header of the system topology can also be double-clicked to expand/collapse all the components.

2.2 Search Bar

The **Search Bar** enables users to find the Karabo component of their interest in the system topology. This is helpful especially with complicated topologies such as instrument topics.

This is composed of a text field, navigation buttons, and case-sensitivity and regular expression toggle buttons.

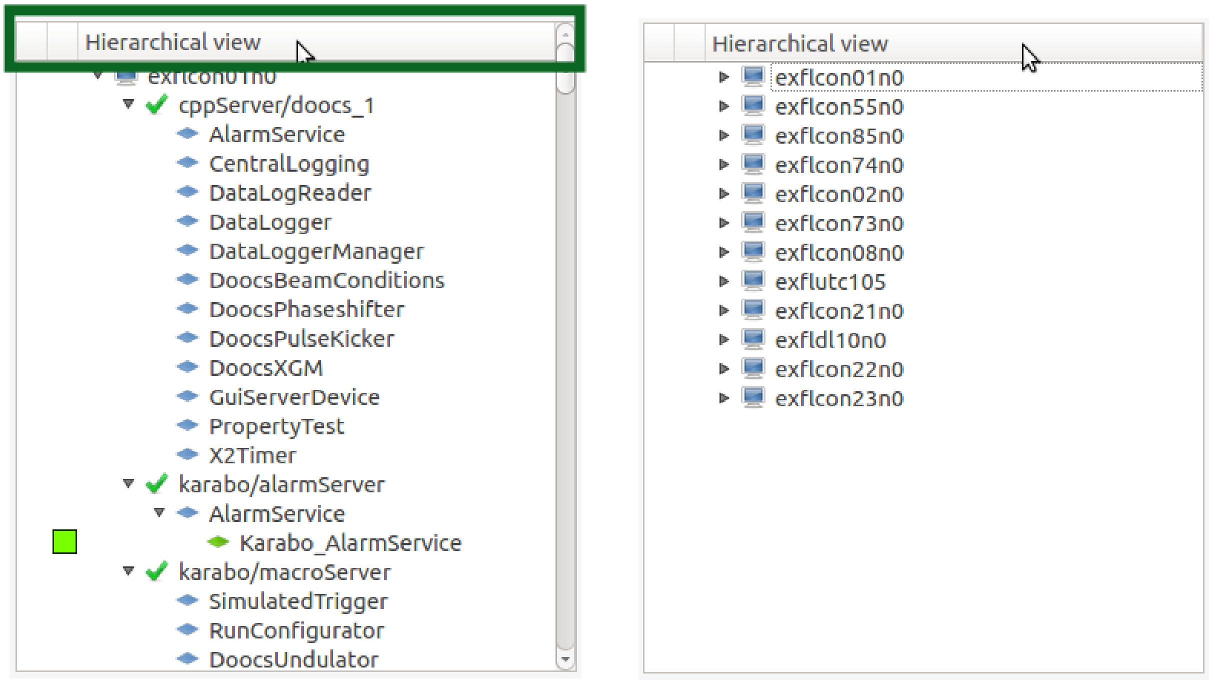
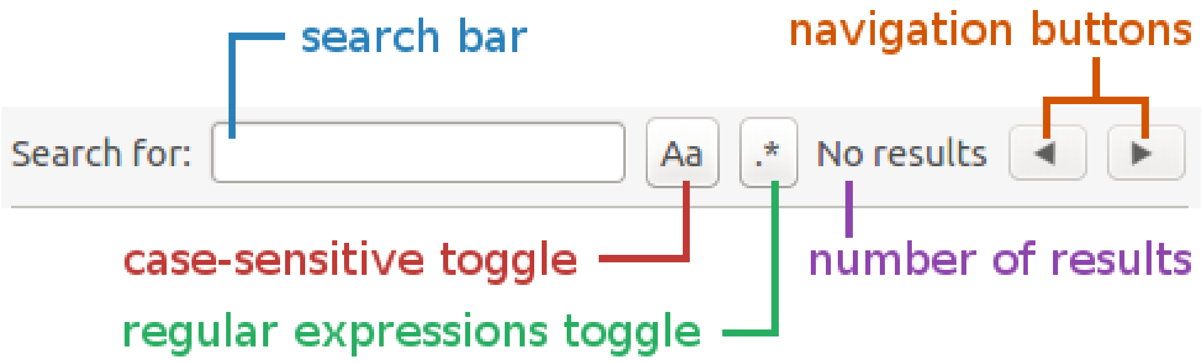


Fig. 12: Double-clicking the header (left) will collapse the tree, leaving a list of control servers (right). Double-clicking it again will expand all the components.



2.2.1 Searching Karabo components

In order to search for a Karabo component, the user can enter its ID in the search field. The search is being done incrementally, meaning that it is being done as-you-type. The number of results are reflected on the text on the right. The first result from the top will also be highlighted.

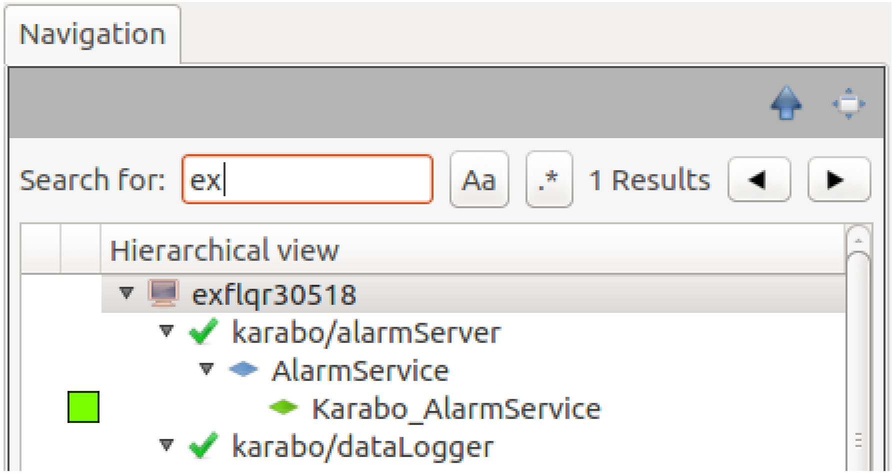


Fig. 13: Searching for `ex` in the system topology returns 1 result. It hits the control server `exflqr30518`.

2.2.2 Navigating through the results

The search results can be viewed via the navigation buttons. These buttons can be used to go to the previous or next result. When pressed, the consequent entry will be focused, therefore being highlighted and its configuration being shown in the Configuration Panel.

The navigation is done in an orderly manner, which means that the results are viewed from the top to the bottom of the topology.

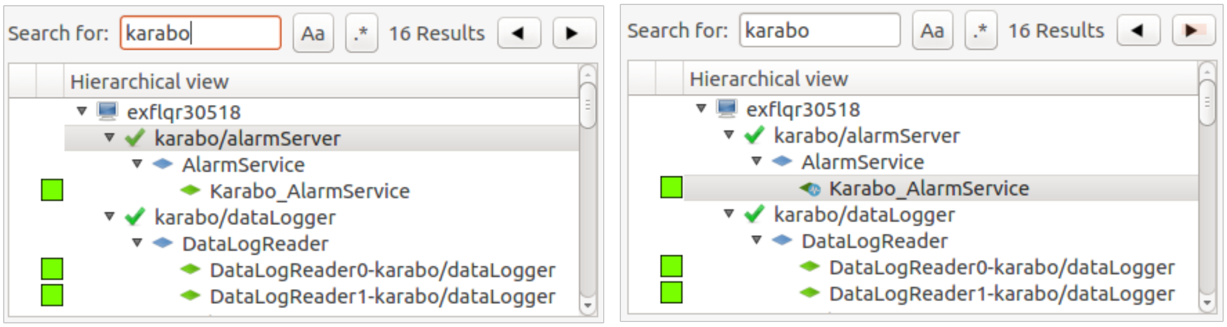
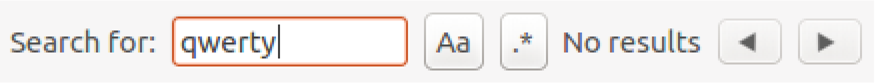


Fig. 14: Searching for `karabo` will hit the first component ID from the top that contains it, which is the *device server* `karabo/alarmServer` (left). Clicking on the **next** navigation button will then hit the device instance `Karabo_AlarmService`, which is the second entry from the top (right).

When the search returns no results, these buttons are disabled.

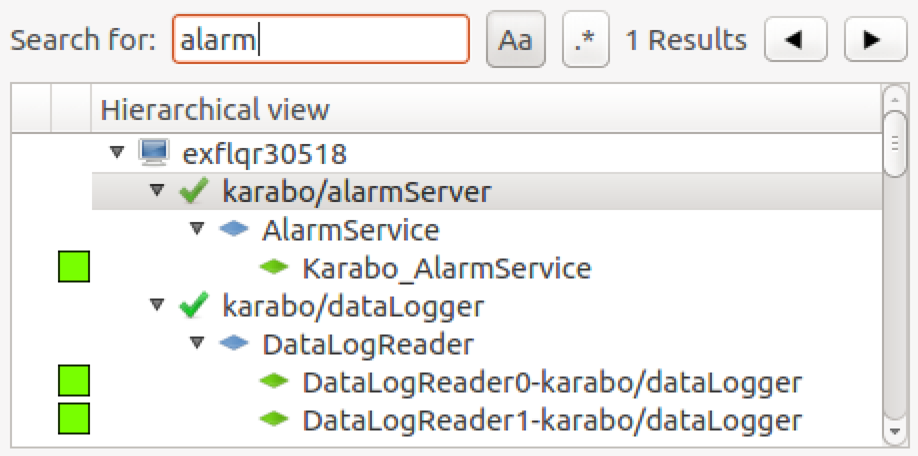


Search for: qwerty

Fig. 15: Navigation buttons are disabled when no results are found.

2.2.3 Case-sensitive search

The search filter is case-insensitive by default. Toggling the case-sensitive button will match the case of the input text.



Search for: alarm

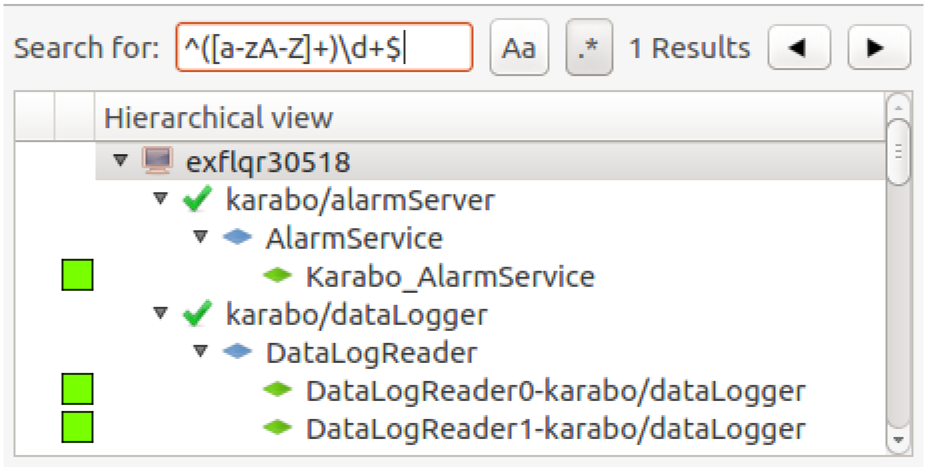
Fig. 16: The `alarm` search with case-sensitivity enabled hits the `karabo/alarmServer` but not `AlarmService` and `Karabo_AlarmService`.

2.2.4 Regular expressions

Regular expressions (regex) can also be utilized when searching. This is by toggling the regex button and entering a valid regex string.

Note: Karabo GUI uses Python's `re` module in managing the regex. Follow the link below for more information.

Python: Regular Expression HOWTO



Search for: ^([a-zA-Z]+)\d+$

1 Results

```
exflqr30518
  ✓ karabo/alarmServer
      AlarmService
🟩          Karabo_AlarmService
  ✓ karabo/dataLogger
      DataLogReader
🟩          DataLogReader0-karabo/dataLogger
🟩          DataLogReader1-karabo/dataLogger
```

Fig. 17: Searching for component IDs that start with letters and end with digits can be done by the regex `^([a-zA-Z]+)\d+$`. It successfully hits the control server `exflqr30518`.

CHAPTER

THREE

THE PROJECT PANEL

The project panel is the rightmost tab of the Navigation Panel, in the Karabo GUI. It is the main access point for interacting with karabo projects and devices in a hierarchical fashion.

By default, no project is loaded in the project panel.

3.1 Projects

A karabo project bundles device servers, device configurations, GUI scenes, macros, and subprojects. It has different categories, which will be explained in the following sections:

1. Macros
2. Scenes
3. Device Servers
   a. Devices
   b. Configurations
4. Subprojects

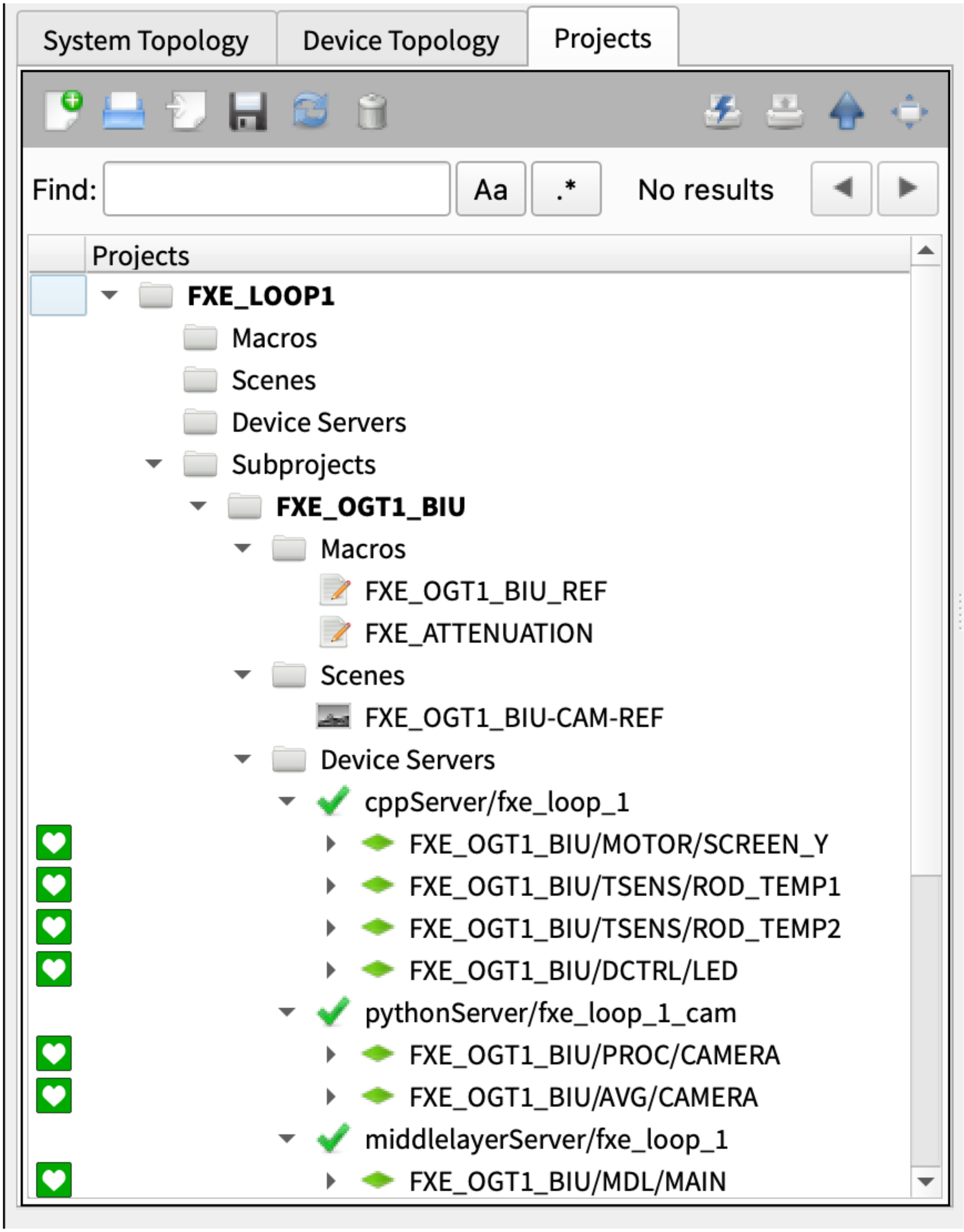
A project can be (a) created, (b) loaded, (c) saved, and (d) trashed.

3.1.1 Creating Projects

A project can be created by clicking the *create new (sub)project* button. A dialog of two fields will appear, which prompts the user to select the domain where the project will be saved in to, and to supply the project title. It is enforced that the title contains alphanumeric, dashes, and underscores only. Though it is best practice to come up with a unique project title, it is allowed to have projects with the same title since each project has a unique identifier (UUID).

Note: Aside from projects, macros and scenes have UUIDs as well! The values can be seen by hovering on these object on the *Project Panel*.

When a project is successfully created, a single folder with the designated title will be shown in the project panel. Note that this project is not yet saved in the domain. The unsaved changes is denoted by a * and a blue title text. The categories can be seen by unfolding the folder (by clicking the arrow icon), and are expected to be empty.



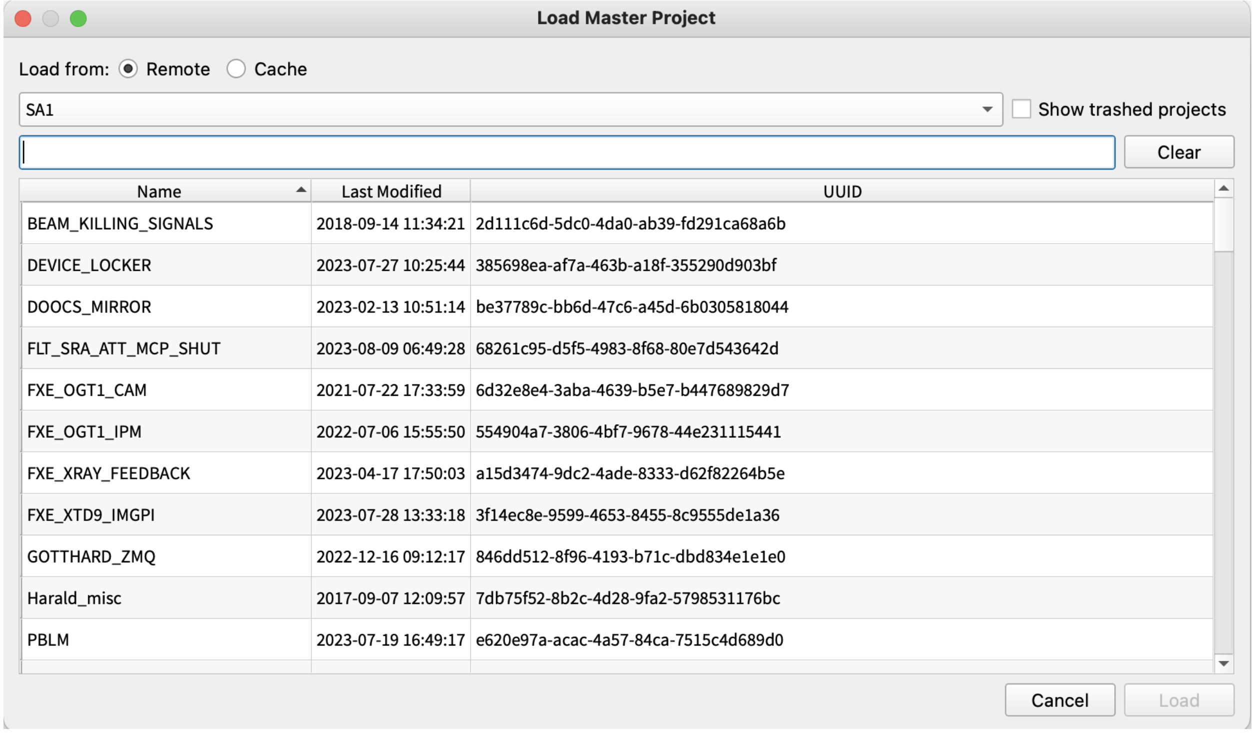
- FXE_LOOP1
  - Macros
  - Scenes
  - Device Servers
  - Subprojects
    - FXE_OGT1_BIU
      - Macros
        - FXE_OGT1_BIU_REF
        - FXE_ATTENUATION
      - Scenes
        - FXE_OGT1_BIU-CAM-REF
      - Device Servers
        - cppServer/fxe_loop_1
          - FXE_OGT1_BIU/MOTOR/SCREEN_Y
          - FXE_OGT1_BIU/TSENS/ROD_TEMP1
          - FXE_OGT1_BIU/TSENS/ROD_TEMP2
          - FXE_OGT1_BIU/DCTRL/LED
        - pythonServer/fxe_loop_1_cam
          - FXE_OGT1_BIU/PROC/CAMERA
          - FXE_OGT1_BIU/AVG/CAMERA
        - middlelayerServer/fxe_loop_1
          - FXE_OGT1_BIU/MDL/MAIN

Fig. 1: The **FXE_LOOP1** project. It contains various objects that can interact with FXE-related devices.

3.1.2 Loading Projects

Load an Existing Project

An existing project can be loaded using the “Load Master Project” dialog. The dialog is opened by clicking the “Load an Existing Project” button in the toolbar.



Name	Last Modified	UUID
BEAM_KILLING_SIGNALS	2018-09-14 11:34:21	2d111c6d-5dc0-4da0-ab39-fd291ca68a6b
DEVICE_LOCKER	2023-07-27 10:25:44	385698ea-af7a-463b-a18f-355290d903bf
DOOCS_MIRROR	2023-02-13 10:51:14	be37789c-bb6d-47c6-a45d-6b0305818044
FLT_SRA_ATT_MCP_SHUT	2023-08-09 06:49:28	68261c95-d5f5-4983-8f68-80e7d543642d
FXE_OGT1_CAM	2021-07-22 17:33:59	6d32e8e4-3aba-4639-b5e7-b447689829d7
FXE_OGT1_IPM	2022-07-06 15:55:50	554904a7-3806-4bf7-9678-44e231115441
FXE_XRAY_FEEDBACK	2023-04-17 17:50:03	a15d3474-9dc2-4ade-8333-d62f82264b5e
FXE_XTD9_IMGPI	2023-07-28 13:33:18	3f14ec8e-9599-4653-8455-8c9555de1a36
GOTTHARD_ZMQ	2022-12-16 09:12:17	846dd512-8f96-4193-b71c-dbd834e1e1e0
Harald_misc	2017-09-07 12:09:57	7db75f52-8b2c-4d28-9fa2-5798531176bc
PBLM	2023-07-19 16:49:17	e620e97a-acac-4a57-84ca-7515c4d689d0

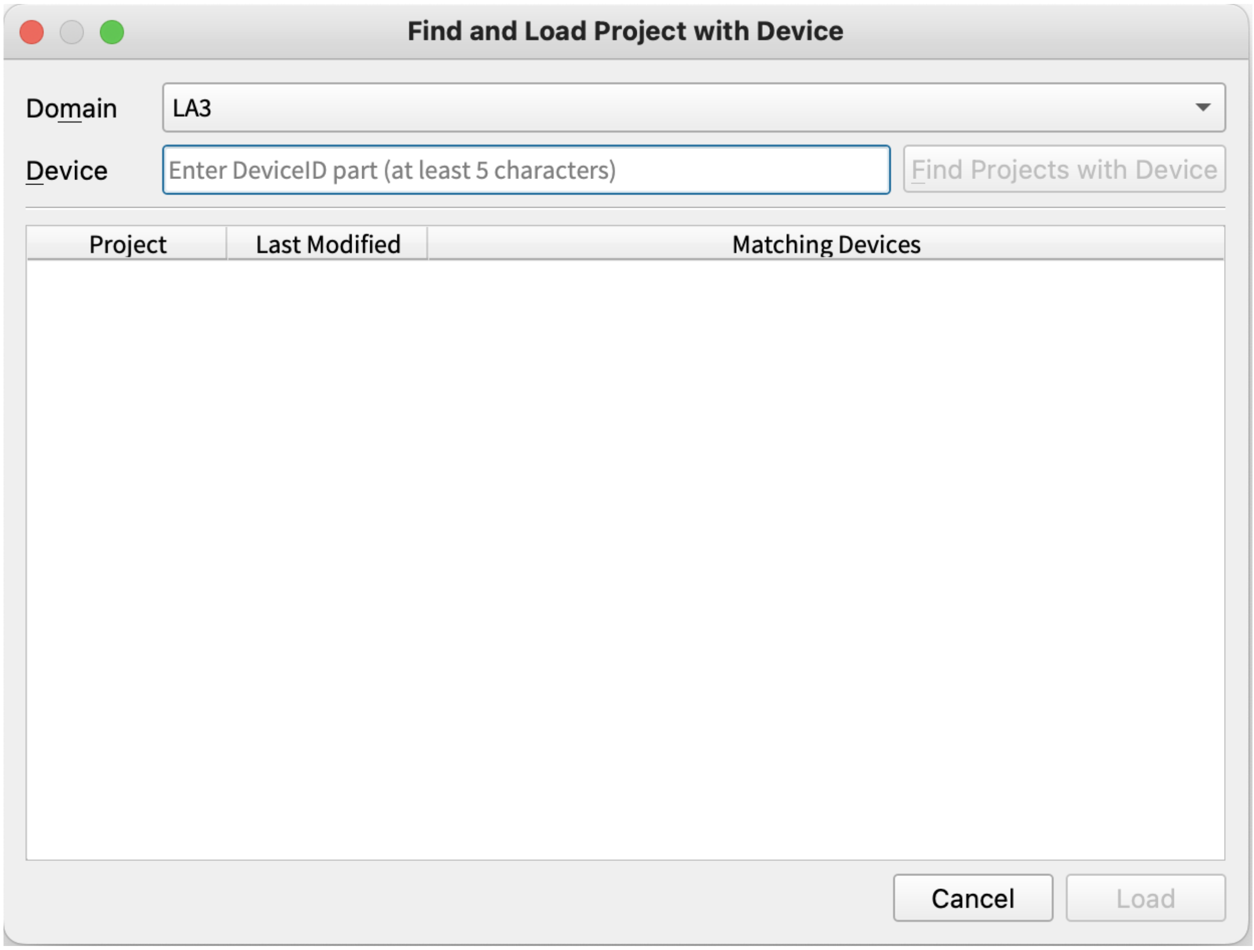
Fig. 2: The **Load Project** dialog showing a selection of the available project in the domain, e.g. *SA1*.

Note: The selection option **cache** will allow loading a project from the local disk. It is the latest updated project version from the project database.

In this dialog we can freely select our top-level project to be opened. We can either click a project name directly or search for a project name in the search bar. Technically, after selecting a project our GUI client will request the project data from the project database, routed via the GUI server. During this period we cannot operate with our client, it goes into a busy mode and several options or panels are disabled. This is notified with a spinner widget in the project bar.

Find and Load Project with Device

The “Find and Load Project with Device” dialog provides an efficient and user-friendly way to locate specific projects associated with devices and seamlessly load them into the Project Panel. The “Find and Load Project with Device” button in the toolbar opens this dialog.



Domain LA3

Device Enter DeviceID part (at least 5 characters)

3.1.3 Saving Projects

There is unsaved changes in the project when the project title is denoted by a * and a blue text. Karabo objects with unsaved changes would also have the same indication.

Changes in the project can be saved in repository by clicking the **Save Project Snapshot** button. When the project is successfully saved, the unsaved indication will be removed, returning to a project title/karabo objects with black text.

Alternatively, saving can also be done by selecting the **Save** option on the project context menu (by right-clicking the Project folder).

The users will also be prompted to save the project with unsaved changes when:

- closing the project;
- creating/loading a new project;
- disconnecting from the server; and,
- closing the Karabo GUI.

3.1.4 Trashing Projects

Unwanted projects can be marked as trashed via the **Trash Project** button. Trashing a project will result to a purple project folder icon, indicating that it is moved in a different repository, namely for trashed projects. This can be undone by clicking the **Trash Project** button again.

Trashed projects can still be retrieved via the **Load Project** dialog, with *Showed trashed projects* selected.

Alternatively, trashing/untrashing can also be done by selecting the **Move to trash/Restore from trash** option on the project context menu (by right-clicking the Project folder).

Note that such changes are saved automatically.

3.1.5 Renaming Projects

It is possible to rename an existing project with the following steps:

1. Right-click the **Project** folder;
2. Select **Rename** menu. A dialog will appear.
3. Supply the new project title.

The new title will be then reflected on the project panel upon successfully renaming the project. Note that such changes are not saved automatically, thus the unsaved changes indication.

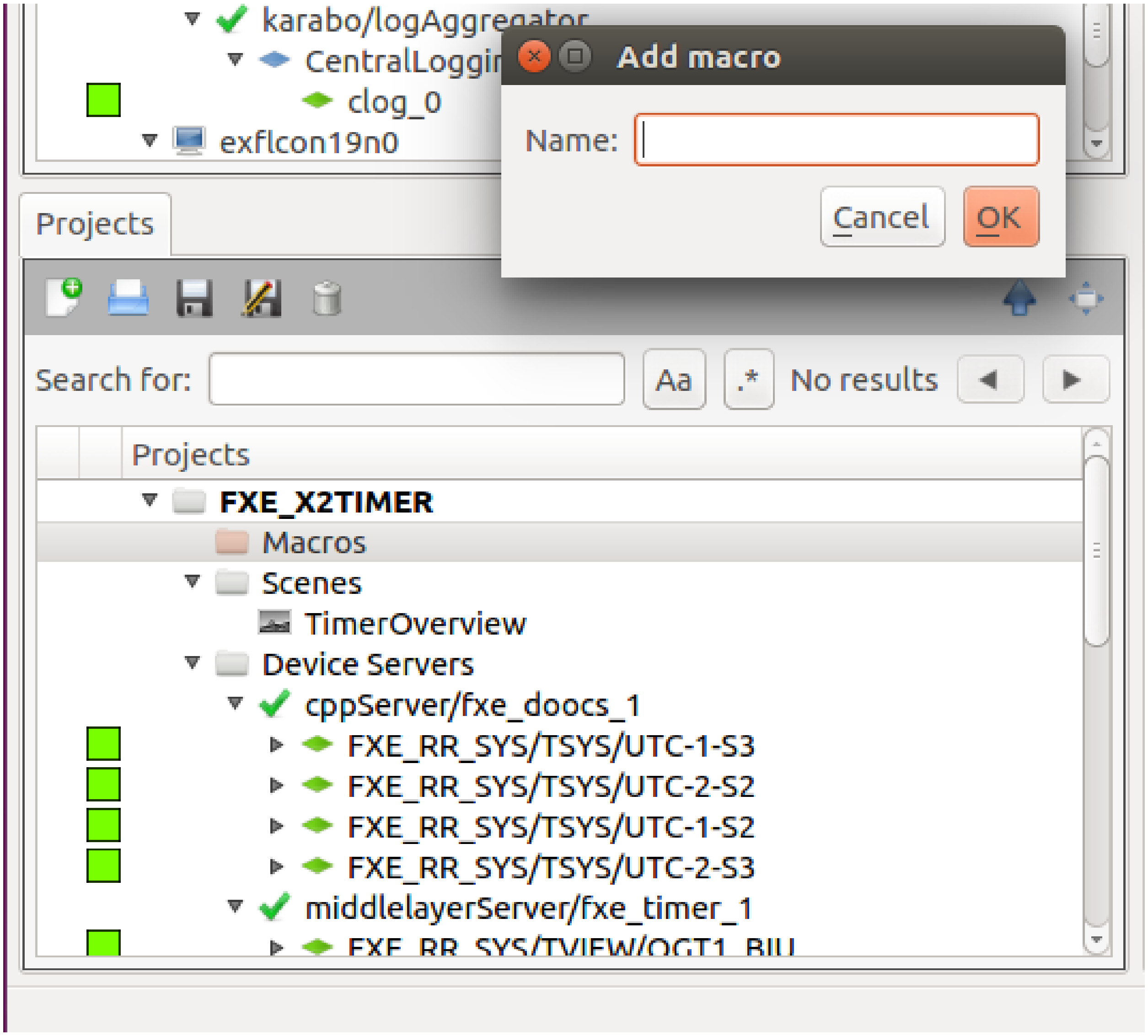
3.2 Macros

A macro is a Python script that helps communicate with Karabo devices and automate recurring tasks. In the project panel, macros can be (a) added/created, (b) loaded from local, and (c) loaded from device.

3.2.1 Creating Macros

A new macro can be created with the following steps:

1. Right-click the **Macros** folder
2. Select **Add macro** option. A dialog will show up. See figure below.
3. Fill the dialog with a `macro-name`. Note that the name is limited to alphanumeric characters, dashes, and underscores.



By double-clicking on the newly listed macro, its default code will appear in the **Macro Panel**, which is located in the upper-middle panel of Karabo GUI.

```
from karabo.middlelayer import Macro, MacroSlot, String

class Example(Macro):
    name = String(defaultValue="Karabo")

```

(continues on next page)

(continued from previous page)

```
    @MacroSlot()
    def execute(self):
        print("Hello {}!".format(self.name))
```

The above sample code will be generated, which can be freely edited in the Macro Panel. Similarly, as described in HowToMiddlelayer, the macro consists of:

- A **Macro** class
- Properties (**Karabo descriptors**), e.g. String, Float, Double …
- Slots to execute public functions

3.2.2 Loading Macros from Local Machines

Macros can be also loaded from local machines as it is a Python script. This can be done by:

1. Right-click the **Macros** folder
2. Select **Load macro…** menu. A dialog asking for a Python file (`*.py`) will show up.
3. Select the macro file to be loaded in the project.

Upon successfully loading the macro, it will be added under the **Macros** folder with the file name as the macro name. Note that such changes are not saved automatically, thus the *unsaved changes* indication.

3.2.3 Loading Macros from Devices

Some devices can also provide macros, similarly with providing scenes. These can be loaded in the project by the following steps:

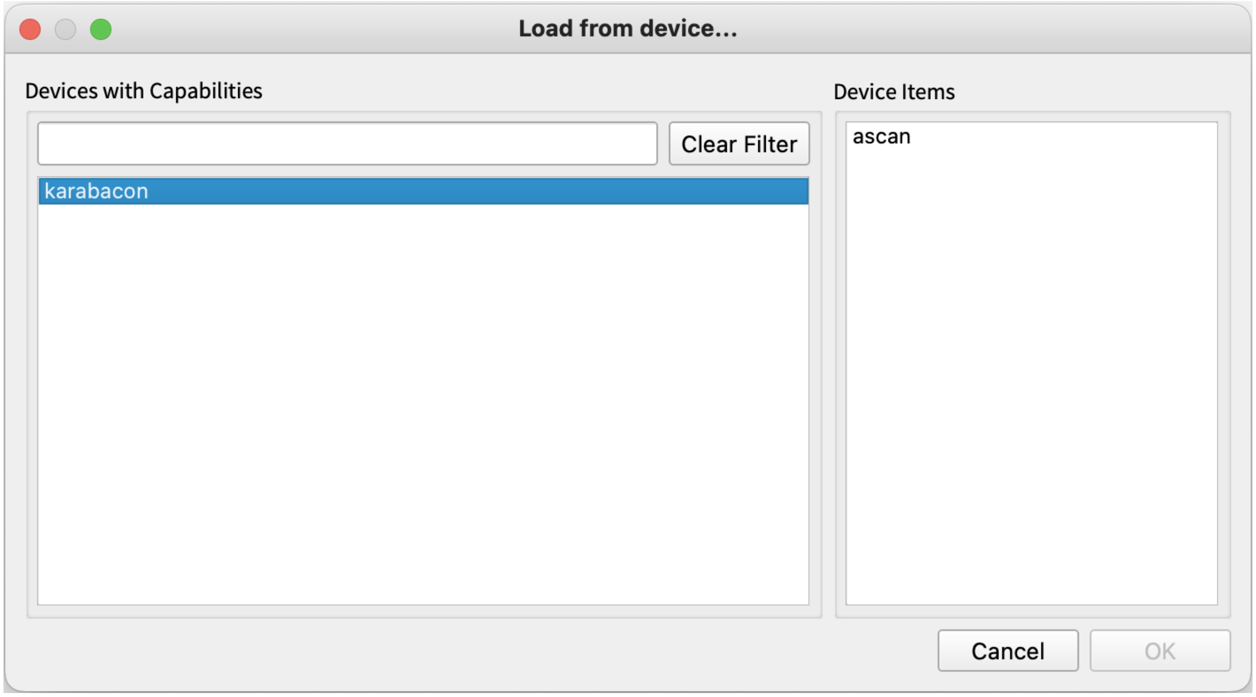
1. Right-click the **Macros** folder
2. Select **Load from device…** menu. A dialog will show up. See figure below.
3. Select the desired device and macro from the *Device with Capabilities* and *Device Items*, respectively.

By default, Macros are listed in the Projects Panel in the order they were created. However, the order can be changed using the “Arrange Macros” right click context menu. This opens a dialog with a list of Macros in the project and options to move them up or down.

3.2.4 Working with Macros

The macro has the following context menu options:

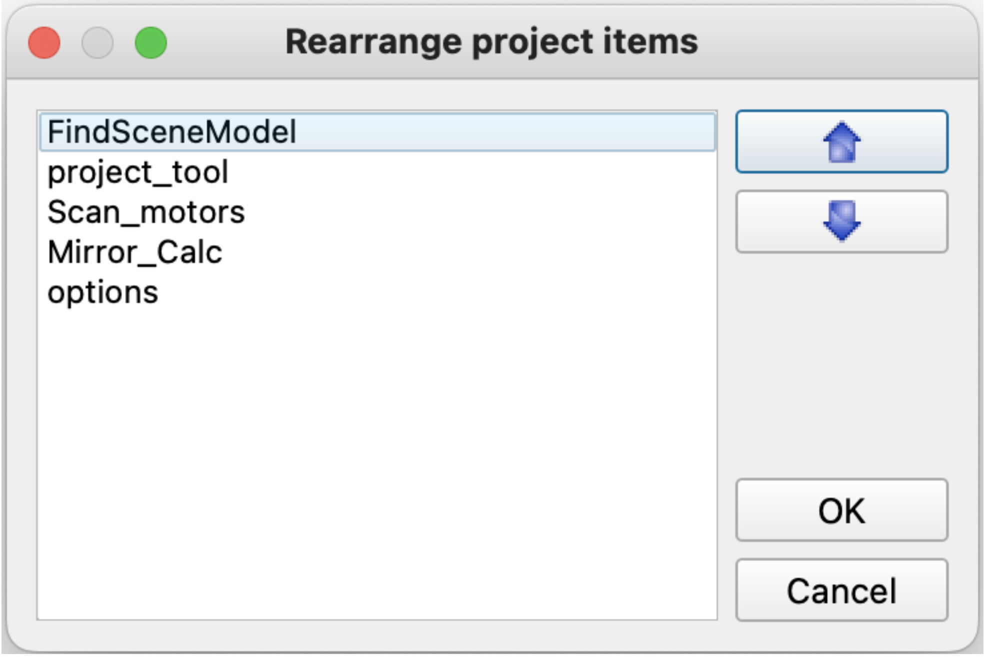
1. Edit
2. Duplicate
3. Delete
4. Save to file
5. Run



karabacon

ascan

Fig. 3: The **Load macro from device** dialog shows that the scantool KARABACON has two macros available.



FindSceneModel
project_tool
Scan_motors
Mirror_Calc
options

Renaming Macros

Renaming macros can be done with the following steps:

1. Right-click the **Macro** object. A context menu will appear.
2. Select **Edit** option. The dialog **Edit macro** will show up.
3. Supply the new macro **Name**.

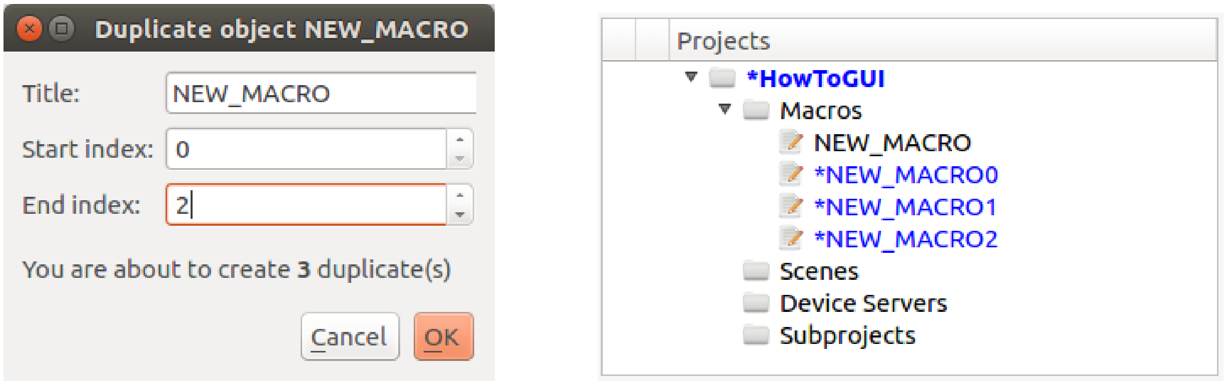
The new name will be reflected on the project upon successfully renaming the macro. Note that such changes are not saved automatically, thus the *unsaved changes* indication.

Duplicating Macros

Creating multiple copies of the macro on the project can be done with the following steps:

1. Right-click the **Macro** object. A context menu will appear.
2. Select **Edit** option. The dialog **Duplicate object** will show up.
3. Supply the following:
   a. **Title** or the new macro name for the duplicate(s) *(must be alphanumeric, dashes, and underscores only)*
   b. **Start index** of the duplicate(s). This value must be less than or equal with the end index.
   c. **End index** of the duplicates(s). This value must be greater than or equal with the start index.

The number of copies that will be produced is the difference of *End index* and *Start index*, which is also indicated in the dialog.. Resulting duplicates will then be listed under the **Macros** folder with names as `new-macro-name` + `index`.



Duplicate object NEW_MACRO

Title: NEW_MACRO

Start index: 0

End index: 2

You are about to create **3** duplicate(s)

Projects

- *HowToGUI
  - Macros
    - NEW_MACRO
    - *NEW_MACRO0
    - *NEW_MACRO1
    - *NEW_MACRO2
  - Scenes
  - Device Servers
  - Subprojects

Fig. 4: Duplicating *NEW_MACRO* with Start Index = 0 and End Index = 2 (left) would result to 3 copies (right).

Note that such changes are not saved automatically, thus the *unsaved changes* indication for the newly added macros.

Deleting Macros

Macros can be deleted from the project with the following steps:

1. Right-click the **Macro** object. A context menu will appear.
2. Select **Delete** option. The user will be asked for confirmation.

The macro will be removed from the **Macros** list upon successfully deleting it. Note that such changes are not saved automatically, thus the *unsaved changes* indication on the **Project**.

Saving Macros to File

Macros can be saved as a Python file (`.py`) on the local machine with the following steps:

1. Right-click the **Macro** object. A context menu will appear.
2. Select **Save to file** option. A file dialog will appear.
3. Locate the folder to save and supply a file name for the macro.

Running Macros

Macros can be run not only on the *Macro Panel* but also in the *Project Panel*. This is possible with the following steps:

1. Right-click the **Macro** object. A context menu will appear.
2. Select **Run** option.

A new macro (device) instance will appear under the *Macro* object upon successfully running the macro. Clicking on it will show its *Configuration* in the *Configuration Panel*. Macros can be instantiated only once, and the running instance must be shut down before instantiating (and accepting changes in the macro code) again.

The macro instance can be shut down from the Configuration Editor (similar with device instances) or by right-clicking the **Macro Instance** object and selecting the **Shutdown** option.

3.3 Devices

Projects enable easier access on devices. This helps bookkeeping of devices for specific operations, such as collecting *FXE LOOP 1* devices in the *FXE_LOOP1 project*.

This can be done by:

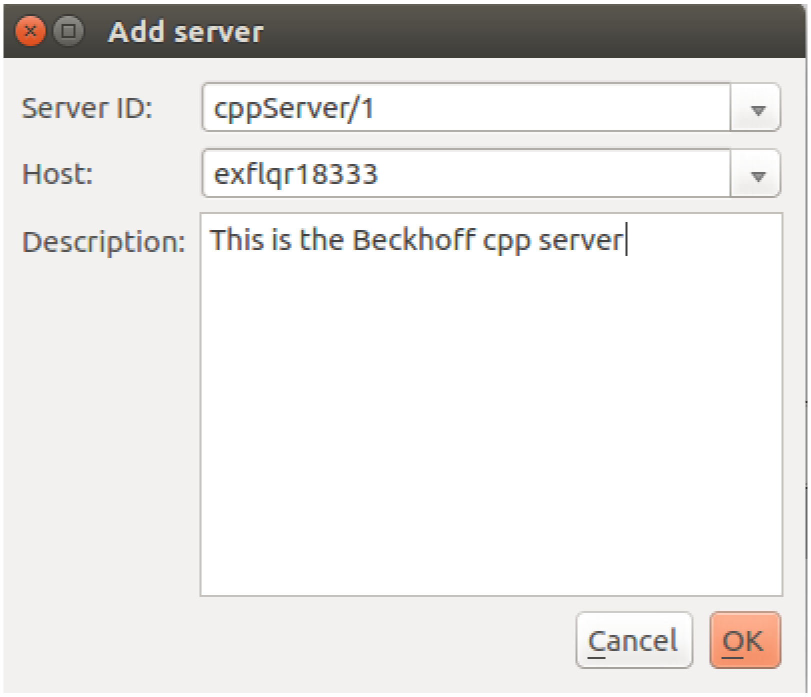
1. Adding a device server
2. Adding a device instance

3.3.1 Adding Device Servers

Device servers can be added in the project with the following steps:

1. Right-click the **Device Servers** folder. A context menu will show up.
2. Select **Add server** option. A dialog will show up. See figure below.
3. In the dialog, select the **Server ID** of the desired server from the list of existing servers.
4. Optionally, select the **Host** of the chosen server from the list of existing hosts. This is useful for servers that are deployed through Karabo.
5. Also optionally, add a **Description** of the server.

Upon successfully adding the device server, it will be added under the **Device Servers** folder. Note that such changes are not saved automatically, thus the *unsaved changes* indication.



Server ID: cppServer/1

Host: exflqr18333

Description: This is the Beckhoff cpp server

Fig. 5: The server configuration dialog.

The device server context menu delivers several options:

1. Edit
2. Delete
3. Shutdown
4. Add device
5. Instantiate all devices
6. Shutdown all devices
7. Delete all devices

Editing a server provides the possibility to change the server name as well as the description in the project. Deleting a server can be sometimes necessary if a project gets restructured. The shutdown option will request the device server and all the devices to shutdown gracefully. Once the server has gone down, it will restart and come online again.

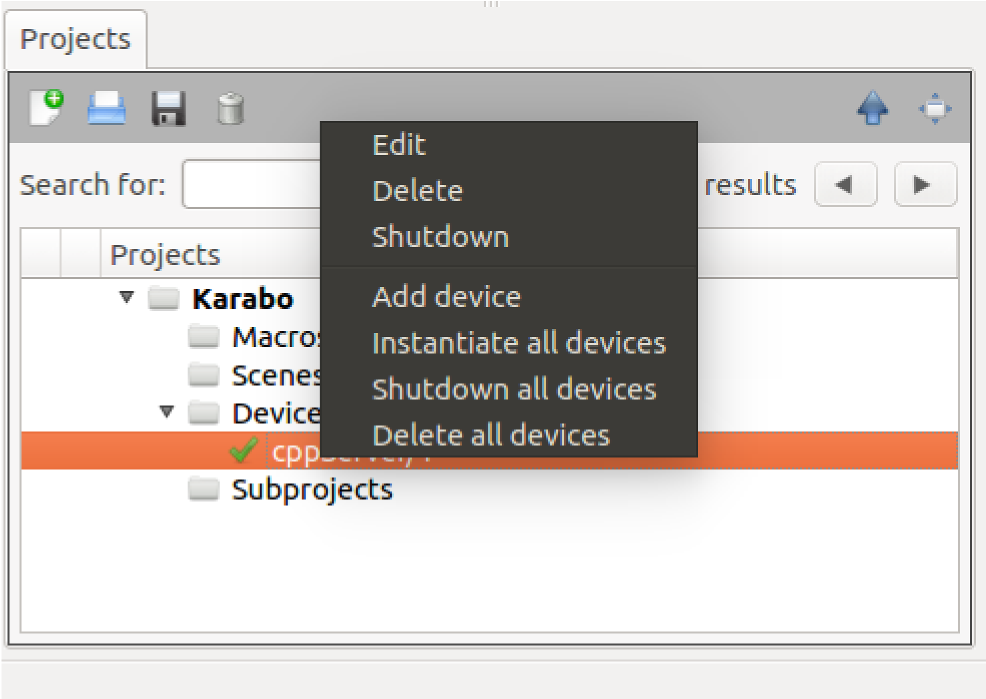


Fig. 6: The device server context menu creation dialog.

It is possible to instantiate and shutdown all devices of this device server. For instantiation, a list of device names is forwarded to the GUI server, who instantiates the devices sequentially, which might take a little bit of time.

In the next chapter the creation and configuration of devices is explained.

3.3.2 Adding Project Devices

Devices can be added on the project with the following steps:

1. Right-click the **Device Server** object
2. Select the **Add device** option from the context menu. A dialog will show up.
3. Supply the following:

   a. **Device ID** or the device name *(must be alphanumeric, dashes, and underscores only)*

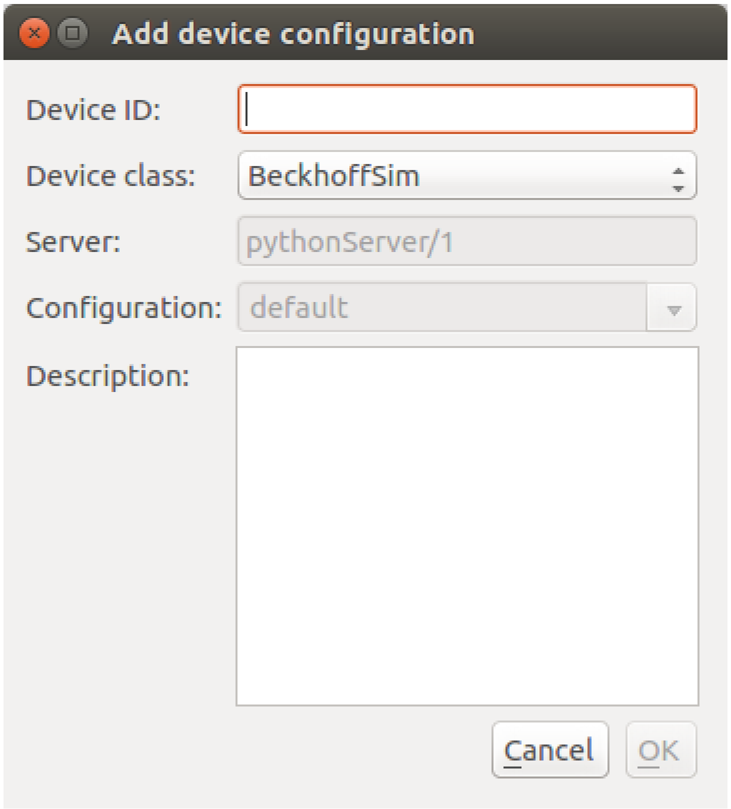
   b. **Device class** of the desired device. The list is the device classes that can be run on the selected server.

   c. **Configuration** name, which is set `default` by default.

   d. Optionally, the **Description** of the device.

Every device instance has a **default** configuration, which is created from the configuration derived from device class schema. This configuration is not supposed to be deleted or renamed in the project.

Upon successfully adding the device, it will appear as a child of the selected device server. If the supplied device name already exists in the system topology, the added device in the project will be the existing device (regardless of selected device server and class). Otherwise, a new device instance will be added instead, and is offline by default. Note that such changes are not saved automatically, thus the unsaved changes indication on the project, device server.



Device ID:

Device class: BeckhoffSim

Server: pythonServer/1

Configuration: default

Description:

Fig. 7: The **Add device configuration** dialog.

3.3.3 Working with Devices

The project device has the following context menu options:

1. Edit
2. Configuration
3. Duplicate
4. Delete
5. Open device macro
6. Open device scene
7. Get Configuration
8. Instantiate
9. Shutdown

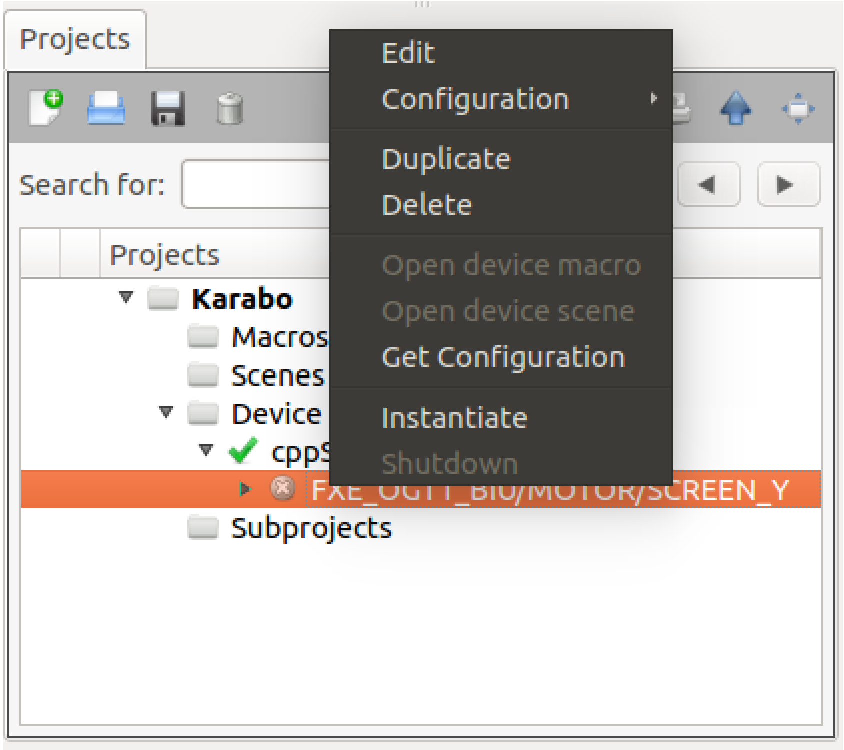


Fig. 8: The device context menu.

Depending on the device status, either *online* or *offline*, there are different possibilities such as starting an offline device or shutting down an online device.

3.3.4 Project Device status

The status of project devices depends on the availability of the class, the online status of server and the device itself.

icon	status	description
⊗	Offline	The device is offline
⊗✖	Offline; No Plugin	The device is offline and the class plugin is missing
⊗✖	Offline; No Server	The device and the server are offline
⊗◆	Online; Incompatible	The device is online but the configured device class is different from the device class that is online
◆	Online	The device is online and did not encounter an error
◆⛔	Online; Error	The device online and has encountered an error

3.4 Device Configuration

Device configurations can be viewed as children of the device. It is convenient to have various configurations for different experiments. Additional device configurations can be added on an offline device with the following steps:

1. Right-click the **Device** object. A context menu will show up.
2. Hover the **Configuration** option and select **Add device configuration** option. A dialog similar with the **Add device** dialog will show up.
3. In the dialog, supply the new **Configuration** name. It should be unique in the device to be added.
5. Optionally, add a **Description** of the new device configuration.

In our example, we have created an additional configuration *nolimits*. The active configuration can be identified by a tick mark and can be swapped by clicking the check boxes next to these.

Note that such changes are not saved automatically, thus the unsaved changes indication on the project, device server, device instance, and device configuration.

Modifications of offline configurations can be done via the **Configurator** in the configuration panel. Every modified device property or attribute will lead again to a modification of the project. It is important to mention that Karabo only stores offline configurations in the project, however, it is possible to retrieve historic configurations from the data loggers. This can be done via by the following steps:

1. Right-click the **Device** object. A context menu will show up.
2. Select **Get Configuration** option. A *Configuration Timepoint* dialog will appear.
3. Supply the **Timepoint** (date-time) of the configuration to be retrieved. For convenience, the user can utilize the calendar on the dropdown menu and/or use the preset time buttons.

Upon successfully requesting for the **configuration from past**, it will appear after a while in the configurator. Depending on whether the device is online or offline, the changes are either highlighted with a blue box and need to be applied or they are merged into the offline configuration.

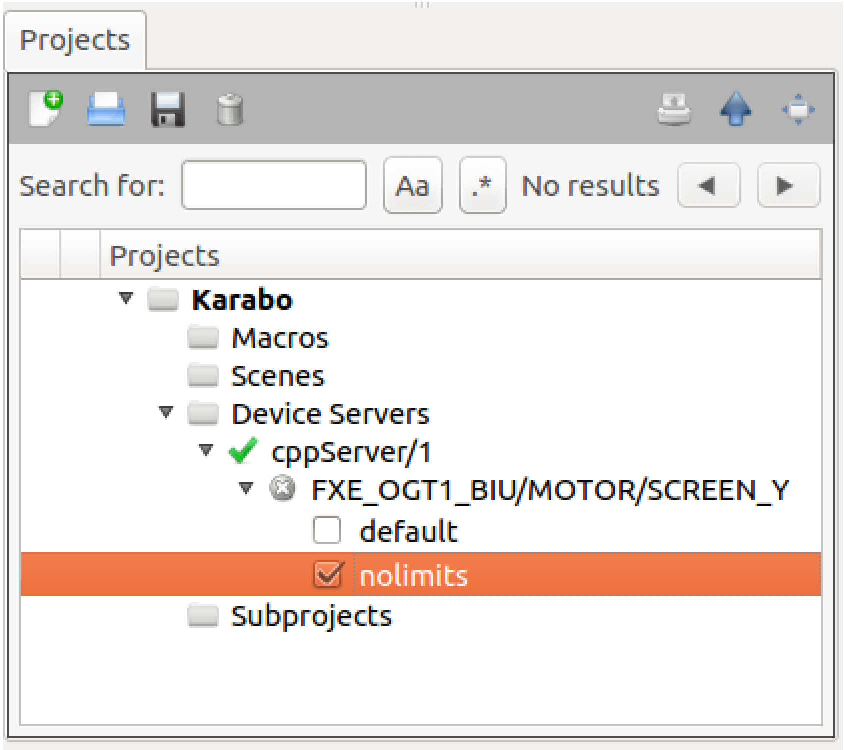
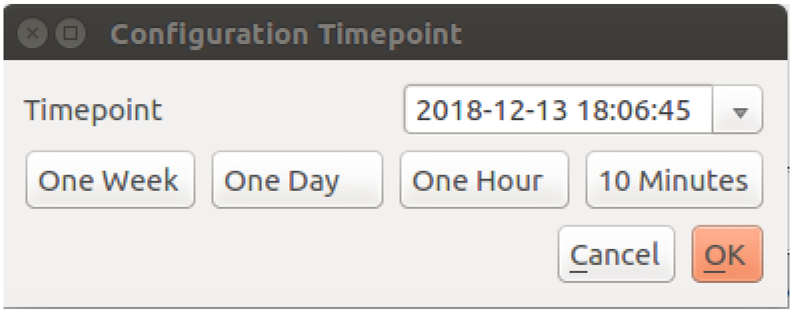


Fig. 9: A device with multiple configurations.



Timepoint 2018-12-13 18:06:45

Fig. 10: The configuration from past dialog

3.5 Scenes

Scenes in Karabo provide a way to customise device views, covering any diagnostic or control elements in a compact and comprehensive view. In the project panel, scenes can be (a) added/created, (b) loaded from local, and (c) loaded from device.

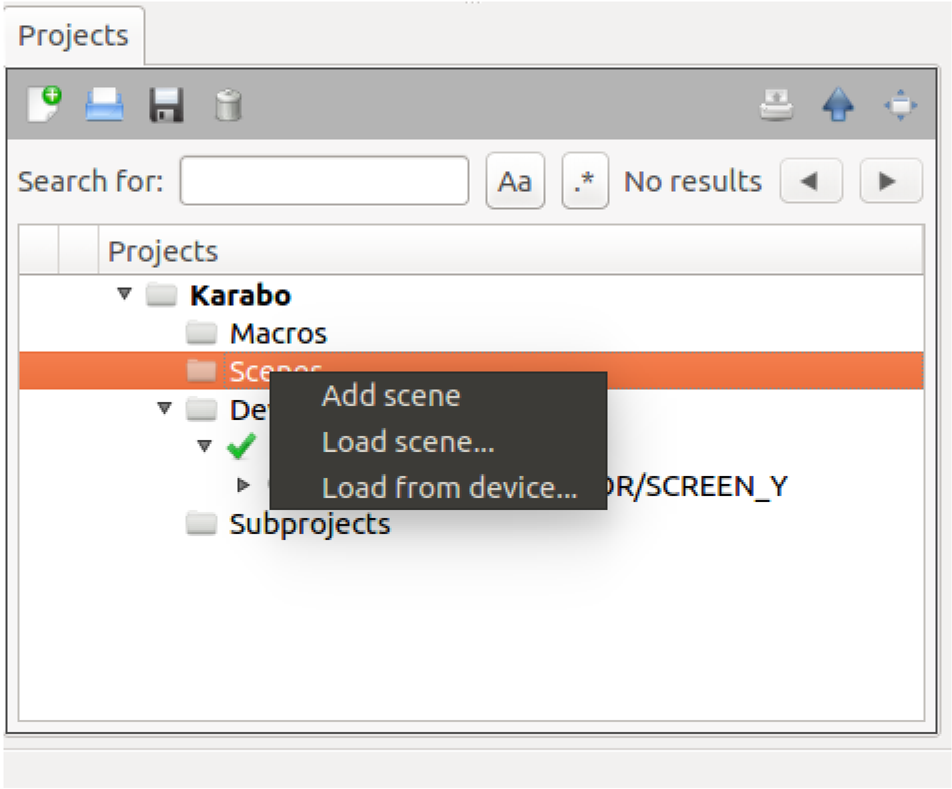


Fig. 11: The scene context menu.

3.5.1 Adding Scenes

A completely new scene can be easily created and added on the project with the following steps:

1. Right-click the **Scenes** folder. A context menu will show up.
2. Select **Add scene** option. The *Add scene* dialog will show up.
3. Supply the dialog with a `scene-name`. Note that the name is limited to alphanumeric characters, dashes, and underscores.

The new scene will now be listed under the **Scenes** folder upon successfully adding it. Double-clicking on it will show a blank scene in the **Scene Panel**, which is located in the upper-middle panel of Karabo GUI. Note that such changes are not saved automatically, thus the unsaved changes indication on the new scene object, scene folder, and project folder.

3.5.2 Loading Scenes from Local Machines

Scenes can be also loaded from local machines as it is an SVG file. This can be done by:

1. Right-click the **Scenes** folder
2. Select **Load from local…** menu. A dialog asking for an SVG file (`*.svg`) will show up.
3. Select the scene file to be loaded in the project.

Upon successfully loading the scene, it will be listed under the **Scenes** folder with the file name as the scenes name. Note that such changes are not saved automatically, thus the *unsaved changes* indication.

3.5.3 Loading Scenes from Devices

Some devices can provide scenes. These can be loaded in the project by the following steps:

1. Right-click the **Scenes** folder
2. Select **Load from device…** menu. A dialog similar to the *Load macro from device* dialog will appear.
3. Select the desired device and scene from the *Device with Capabilities* and *Device Items*, respectively.

3.5.4 Working with Scenes

Opening scenes can be done by double-clicking the **Scene** object. The scene will then be displayed in the *Scene Panel* located at the upper-middle panel of the Karabo GUI.

Note that opening scenes cause unsaved changes, thus the *unsaved changes* indication on the *Scene* object and the *Project* folder.

Additional interaction are also provided in its context menu (right-click).

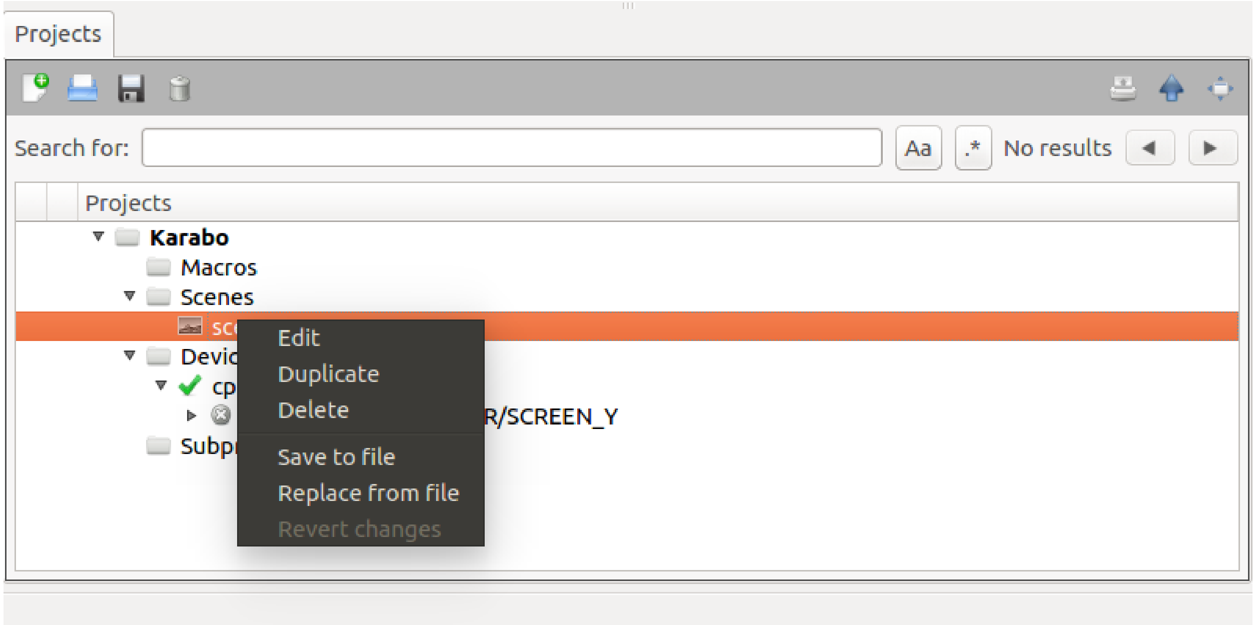


Fig. 12: The scene interaction menu

The scene context menu provides very useful features. It is possible to **revert changes** and **replace from file**.

Renaming Scenes

Renaming scenes can be done with the following steps:

1. Right-click the **Scene** object. A context menu will appear.
2. Select **Edit** option. The dialog **Edit scene** will show up.
3. Supply the new scene **Name**.

The new name will be reflected on the project upon successfully renaming the scene. Note that such changes are not saved automatically, thus the *unsaved changes* indication.

Duplicating Scenes

Creating multiple copies of a scene on the project can be done with the following steps:

1. Right-click the **Scene** object. A context menu will appear.
2. Select **Edit** option. The dialog **Duplicate object** will show up.
3. Supply the following:

    a. **Title** or the new scene name for the duplicate(s) *(must be alphanumeric, dashes, and underscores only)*

    b. **Start index** of the duplicate(s). This value must be less than or equal with the end index.

    c. **End index** of the duplicates(s). This value must be greater than or equal with the start index.

The number of copies that will be produced is the difference of *End index* and *Start index*, which is also indicated in the dialog.. Resulting duplicates will then be listed under the **Scene** folder with names as `new-scene-name` + `index`.

Note that such changes are not saved automatically, thus the *unsaved changes* indication for the newly added scenes.

Deleting Scenes

Macros can be deleted from the project with the following steps:

1. Right-click the **Scene** object. A context menu will appear.
2. Select **Delete** option. The user will be asked for confirmation.

The scene will be removed from the **Scenes** list upon successfully deleting it. Note that such changes are not saved automatically, thus the *unsaved changes* indication on the **Project**.

Saving Scene to File

Macros can be saved as an SVG file (`.svg`) on the local machine with the following steps:

1. Right-click the **Scene** object. A context menu will appear.
2. Select **Save to file** option. A file dialog will appear.
3. Locate the folder to save and supply a file name for the scene.

Replace Scenes from File

Replacing a scene from file is possible when the scene is closed or opened. On this action all scene content from a local file is replaced except the uuid and the simple name. This means that scene links will still work afterwards!

Revert Scenes

Reverting scene changes is translated to replacing the scene with the latest saved scene version in the **local cache**. A condition for this action is that the scene must be closed.

3.6 Subprojects

Projects can link other projects by making them as their *subprojects*. A project can be added under on the **Subprojects** folder by a) adding/creating a completely new project, and b) loading an existing project.

3.6.1 Creating Subprojects

Creating a subproject is very similar with *creating a project*. The main difference is that it will be saved in the the same domain as the current project.

1. Right-click the **Subprojects** folder. A context menu will appear.
2. Select **Add new project…** option. A dialog similar with creating a project will show up.
3. Supply the new project title.

The new project will appear under the **Subprojects** folder upon successfully creating it. Note that such changes are not saved automatically, thus the unsaved changes indication.

3.6.2 Add Existing Projects as Subproject

Linking an existing project as a subproject is very similar with **`loading a project <load_project_>`_**. The main difference is that the lookup is limited to projects of the same domain as the current project.

1. Right-click the **Subprojects** folder. A context menu will appear.
2. Select **Load project…** option. A dialog similar with opening a project will show up.
3. Select the project from the list.

Note: Subprojects are essentially projects that are just linked in another project. Changes that are made and saved in a subproject will always reflect on its links on other projects.

CHAPTER

FOUR

THE CONFIGURATION PANEL

The karabo GUI application offers a dedicated panel for device interaction. By means of this panel, often also referred to as `Configurator`, it is possible to view and alter the device's:

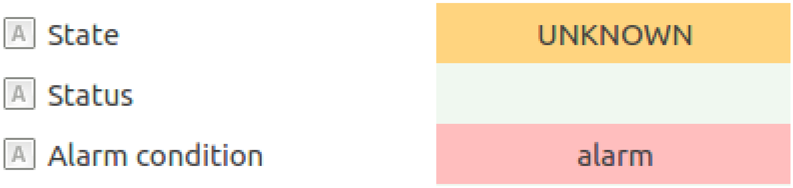
- Properties: ranging from primitive types (bool, int, float, string) to more complex data structures like images, vectors and I/O channels
- Attributes: further specify properties (bounds, units, shape, etc)
- Slots: buttons that when clicked execute some action on the target device (turn on/off, read data, move, etc).

A device can be selected either from one of the Navigation Panel. This can be either the `System Topology`, the `Device Topology` or the Project that contains it. Only in the case when an offline device is selected (device **class**), the operator will be able to modify its default properties. In case of an online device is selected (device **instance**), the current device properties will be shown.

Each property or attribute has an associated icon that helps identifying the type.

Type	Property	Attribute
Boolean	✓	✓A
Floating point	1.1	1.1A
Integer	1	1A
List	☰	☰A
Pair	⊶	⊶A
Schema	⊸	⊸A
String	A	AA
Undefined	□	A

Also, properties like States or that display any type of Alarm are properly colored.

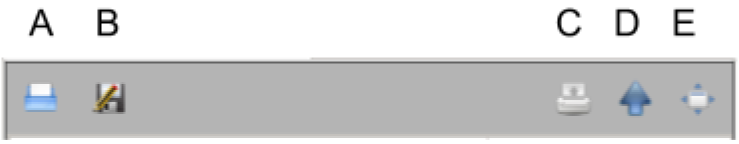


A State	UNKNOWN
A Status	
A Alarm condition	alarm

Note: The user will only be able to access the device properties accordingly to its access level (Admin, Operator, etc).

4.1 The Configuration Editor Toolbar

From the toolbar of the "Configuration Editor" the device configuration can be loaded (A) from an xml file or saved (B) to an xml file, using the dedicated icons:



Besides these actions, the user can Print (C), Undock (D) or Maximize (E) the panel.

4.2 Device Configuration Example

The following image shows, as an example, the configuration panel for the `DataGenerator` **class** using the `Admin` access level. The panel has only two columns:

1. The property or attribute name
2. The current *default* value

Readonly and reconfigurable parameters can be differentiated by color. Readonly parameters have a grey color as they are less important for a default configuration.

In case of an online `DataGenerator` device, as shown in the image below, one additional column appears, which is used to view the device's online value. Additionally, two extra buttons for accepting and declining changes, are appearing.

In this instance we can visualize:

- Read-only properties: State, DeviceID, Bool property, Vector property, True frequency, Output, etc.
- Read/write properties: Update frequency, Speed up
- Slots: Start, Stop and Reset

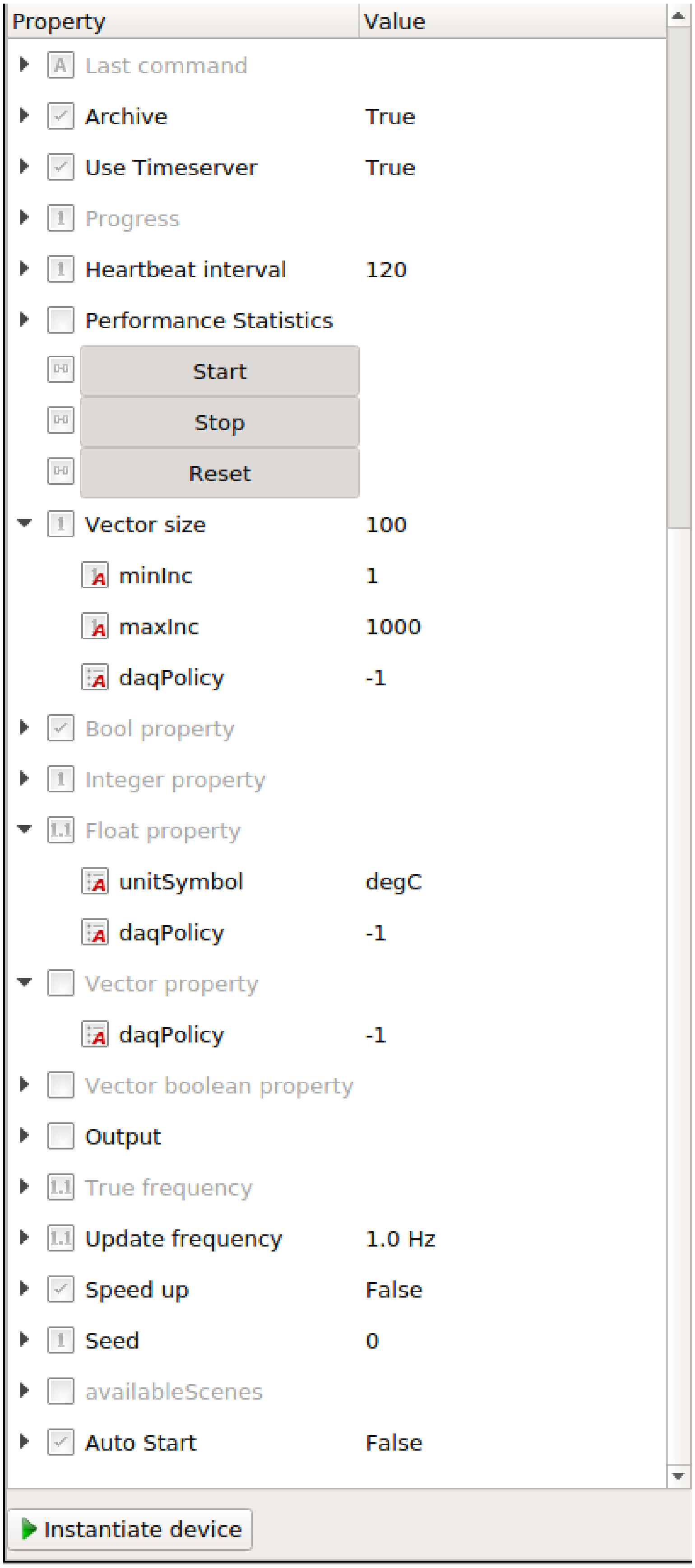
When we modify a property and did not apply the changes to the device, its changes will be highlighted in **blue**, as seen on `Update Frequency` and `Speed Up` fields. It is possible to apply the changes via the **Apply all** button, or rollback the changes to their previous value with the **Decline all** button. Value editing can also be done by immediately pressing **Enter** key to apply the value or or cancel the modification by pressing the **Escape** key.

For every property displaying a value, double-clicking its "Current value on device" (green column) will provide a Trendline Scene showing its evolution over time. For instance, the following image shows the evolution of the Float Property.

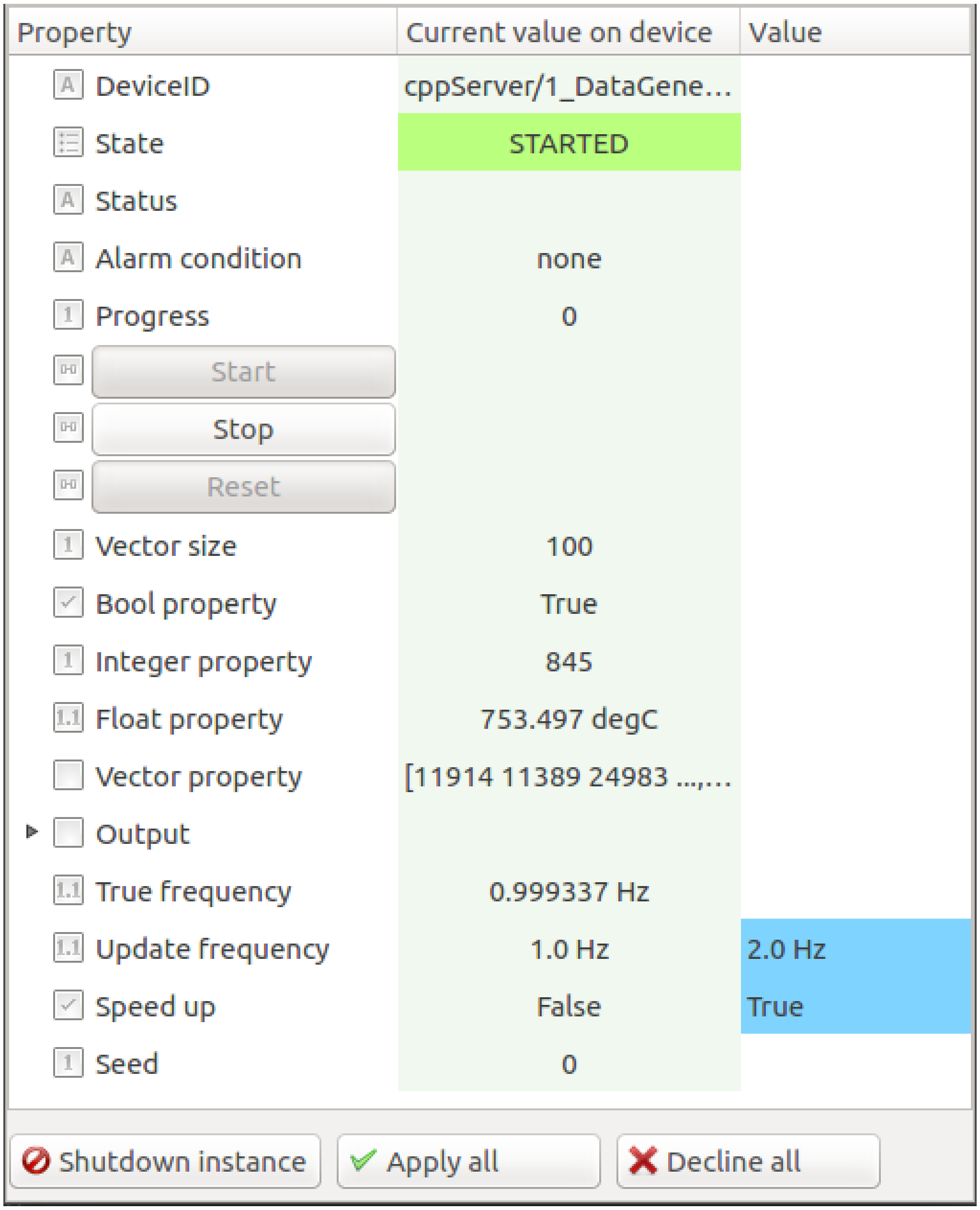
4.2.1 Property Information

To obtain more information about a property, the operator can click on its type icon. For instance, for the **vector property**, it is possible to access a detailed description containing the timestamp, the data type and important attributes such as alias, tags, minimum and maximum values, etc.

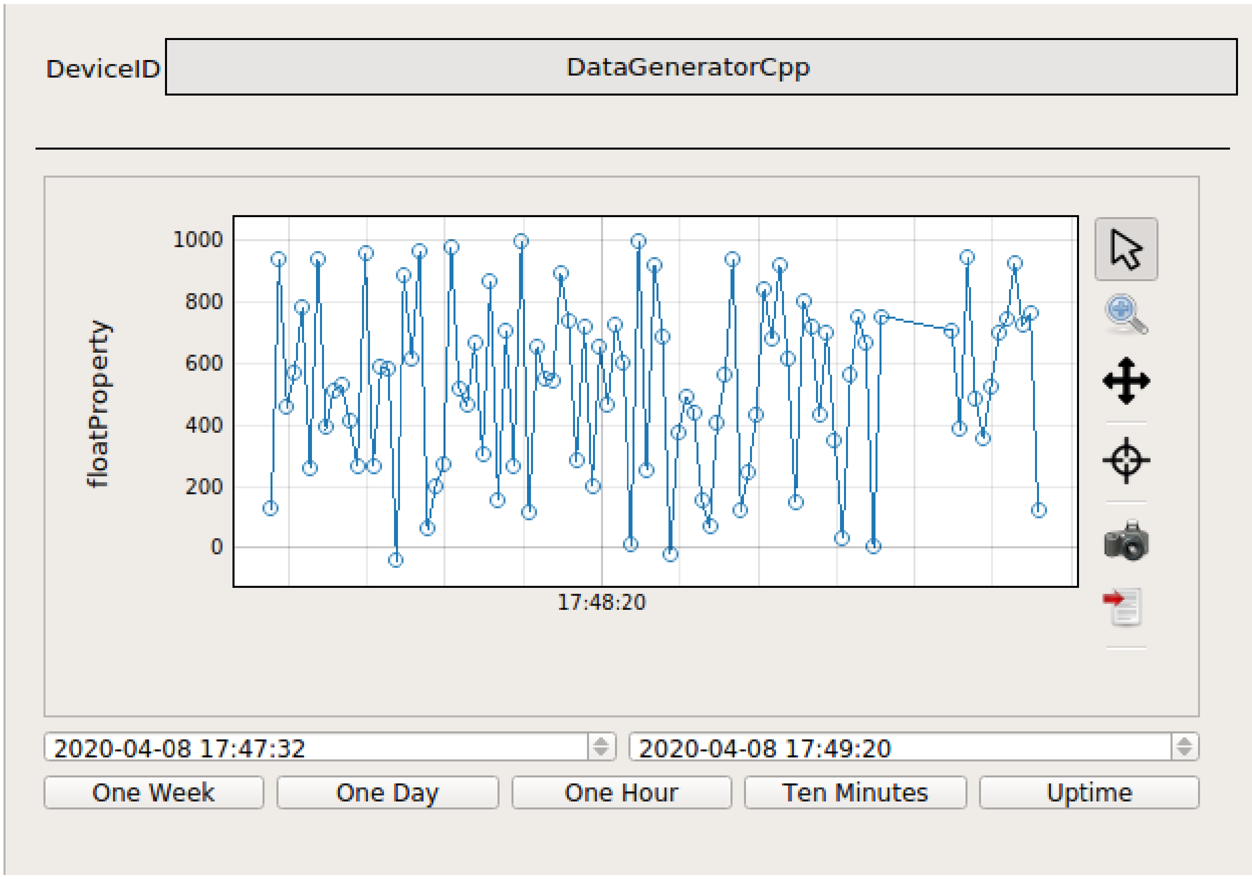
The DataGenerator instance has an *Output* channel where the user can have access to the generated image data. `Image Data` is essentially a noded data container. By clicking on the arrow icon beside the component it is possible to view the encapsulated *properties* that define high and low-level information about the image (encoding, ROI offsets, shape, etc).



Property	Value
Last command	
Archive	True
Use Timeserver	True
Progress	
Heartbeat interval	120
Performance Statistics	
Vector size	100
minInc	1
maxInc	1000
daqPolicy	-1
Bool property	
Integer property	
Float property	
unitSymbol	degC
daqPolicy	-1
Vector property	
daqPolicy	-1
Vector boolean property	
Output	
True frequency	
Update frequency	1.0 Hz
Speed up	False
Seed	0
availableScenes	
Auto Start	False



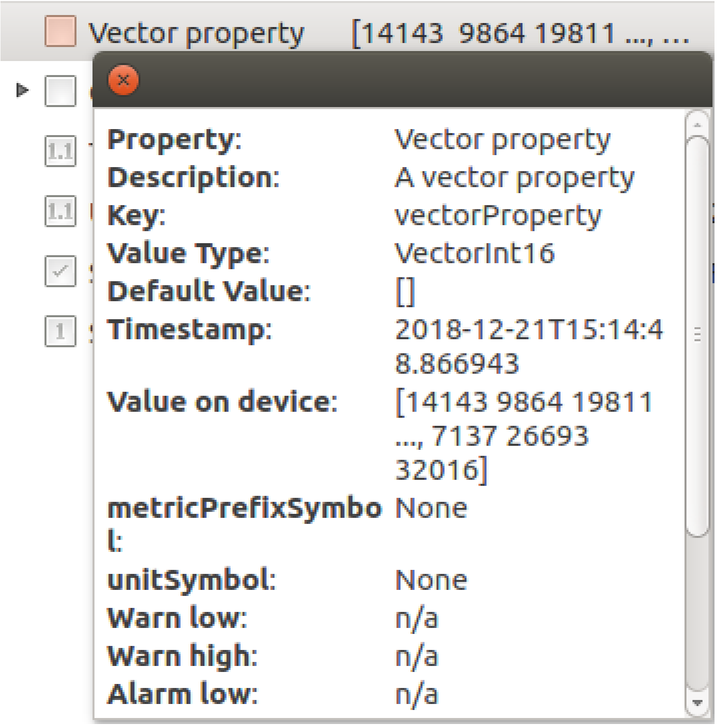
Property	Current value on device	Value
DeviceID	cppServer/1_DataGene…	
State	STARTED	
Status		
Alarm condition	none	
Progress	0	
Start		
Stop		
Reset		
Vector size	100	
Bool property	True	
Integer property	845	
Float property	753.497 degC	
Vector property	[11914 11389 24983 ...,…	
▸ Output		
True frequency	0.999337 Hz	
Update frequency	1.0 Hz	2.0 Hz
Speed up	False	True
Seed	0	



DeviceID: DataGeneratorCpp

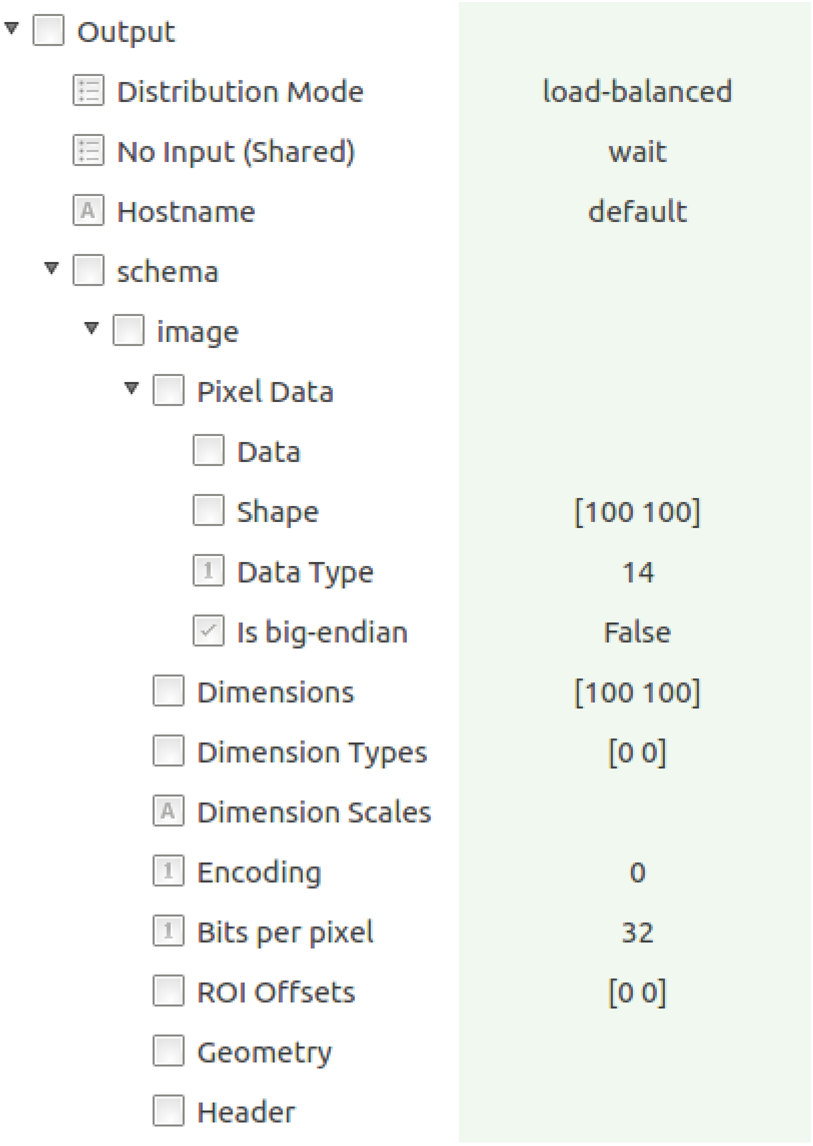
From: 2020-04-08 17:47:32

To: 2020-04-08 17:49:20



Vector property [14143 9864 19811 ..., ...

Property:	Vector property
Description:	A vector property
Key:	vectorProperty
Value Type:	VectorInt16
Default Value:	[]
Timestamp:	2018-12-21T15:14:48.866943
Value on device:	[14143 9864 19811 ..., 7137 26693 32016]
metricPrefixSymbol:	None
unitSymbol:	None
Warn low:	n/a
Warn high:	n/a
Alarm low:	n/a



- Output
  - Distribution Mode: load-balanced
  - No Input (Shared): wait
  - Hostname: default
  - schema
    - image
      - Pixel Data
        - Data
        - Shape: [100 100]
        - Data Type: 14
        - Is big-endian: False
      - Dimensions: [100 100]
      - Dimension Types: [0 0]
      - Dimension Scales
      - Encoding: 0
      - Bits per pixel: 32
      - ROI Offsets: [0 0]
      - Geometry
      - Header

CHAPTER

FIVE

THE SERVICE PANELS

The **service panels** are available in the `Menu Bar` of the karabo GUI. By default, no service panel is visible on startup. The visibility of each panel can be stored in the local machine settings (QSettings) for the karabo GUI.

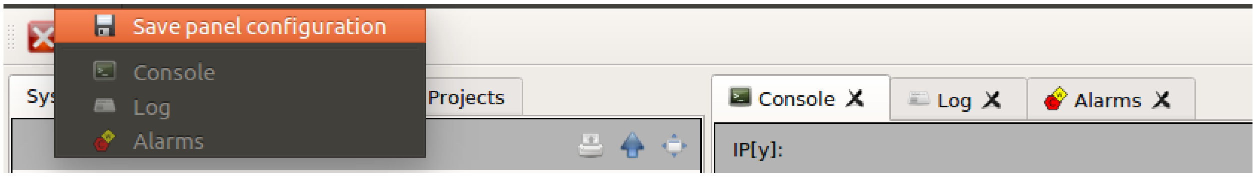


Fig. 1: The Service Panel Overview and Configuration

Activating any service panel will show the service in the `Middle Section` of the karabo GUI. Clicking `Save panel configuration` will store the actual visible configuration.

5.1 The Alarm Panel

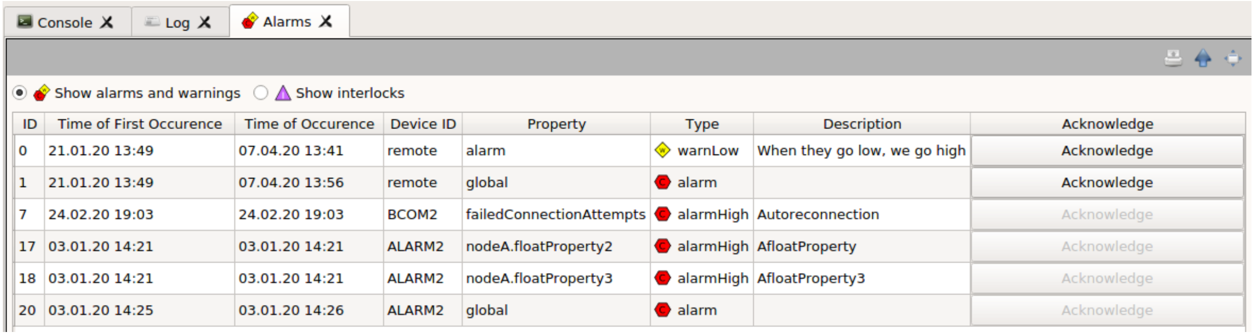
Alarms are tracked by the central alarm service and can be viewed and acknowledged through the `Alarm Panel` located in the middle area of the GUI. It uses the following custom panel.

icon	status	description
	None	The device is working normally.
	Warning	A device property is in a value range where it should be monitored.
	Critical	A device property is in its critical value range.
	Interlock	The device has a triggered interlock condition.

In the first column of the panel the `ID` of the entry in the alarm service is shown, followed by the `first time of occurence` and the `time of occurence`. The time tracking enables the operator to estimate the severity of a long standing *warning* or *alarm* condition. The fourth column shows the `deviceId`, e.g. the device name from where the alarm was triggered from. Afterwards, the property related to the alarm is described, it can be either a normal device property or referred to a `globalAlarmCondition`, presented as *global*. The `Type` of the severity is provided with an icon for quick visualization and identification. If the device author provided additional text information related to this alarm, it will be shown the `description` column.

Alarms can be further of different nature. Next to normal alarm conditions, some alarms may require an active acknowledgement in order to be deregistered from the alarm service. Acknowledgement is only possible when the device is either shutdown or the alarm condition has vanished.

Double-clicking a row in the `Alarm Panel` will look up the `deviceId` in the `Topology Panel`.



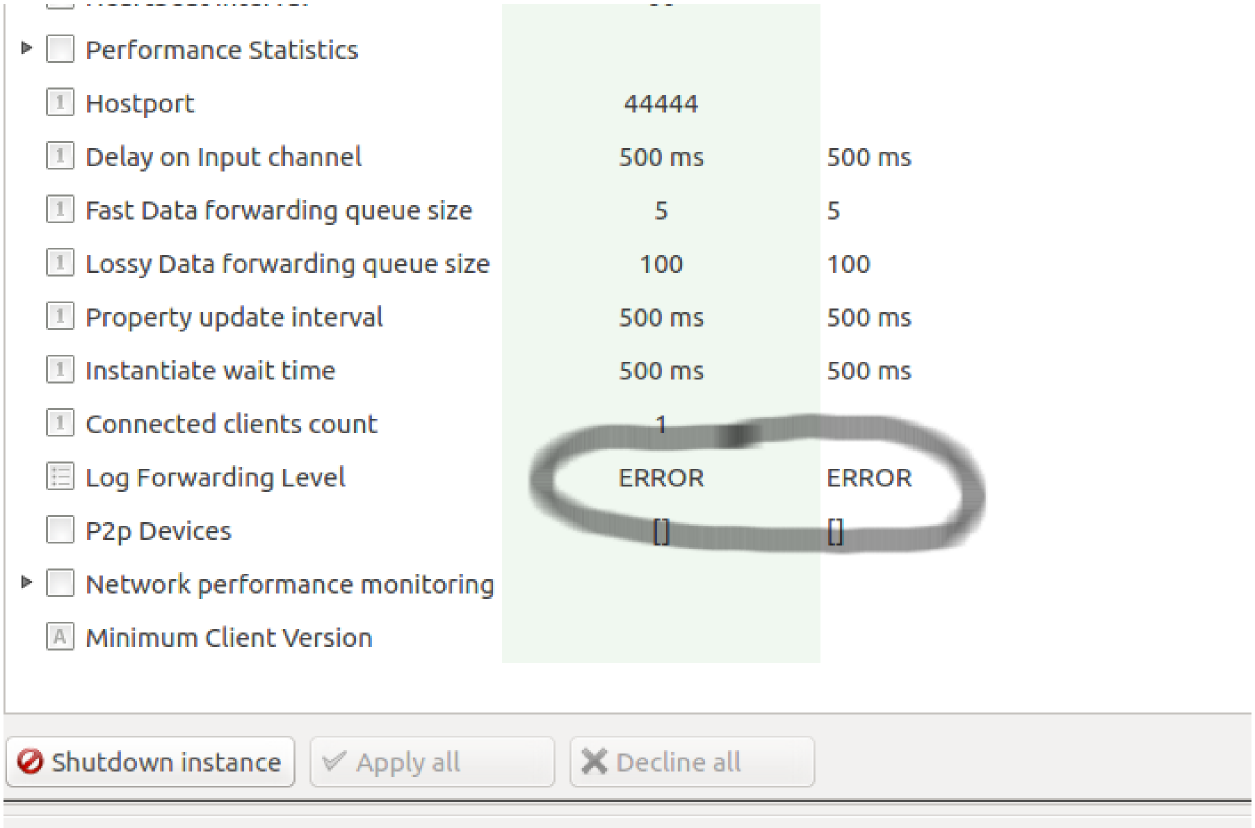
● Show alarms and warnings ○ Show interlocks

ID	Time of First Occurence	Time of Occurence	Device ID	Property	Type	Description	Acknowledge
0	21.01.20 13:49	07.04.20 13:41	remote	alarm	warnLow	When they go low, we go high	Acknowledge
1	21.01.20 13:49	07.04.20 13:56	remote	global	alarm		Acknowledge
7	24.02.20 19:03	24.02.20 19:03	BCOM2	failedConnectionAttempts	alarmHigh	Autoreconnection	Acknowledge
17	03.01.20 14:21	03.01.20 14:21	ALARM2	nodeA.floatProperty2	alarmHigh	AfloatProperty	Acknowledge
18	03.01.20 14:21	03.01.20 14:21	ALARM2	nodeA.floatProperty3	alarmHigh	AfloatProperty3	Acknowledge
20	03.01.20 14:25	03.01.20 14:26	ALARM2	global	alarm		Acknowledge

Fig. 2: The alarm service widget.

5.2 The Logging Panel

The `Logging Panel` is an essential feature of the KaraboGUI. Depending on the filter setting of the **GUI-Server**, log messages are forwarded to the Karabo GUI Client.

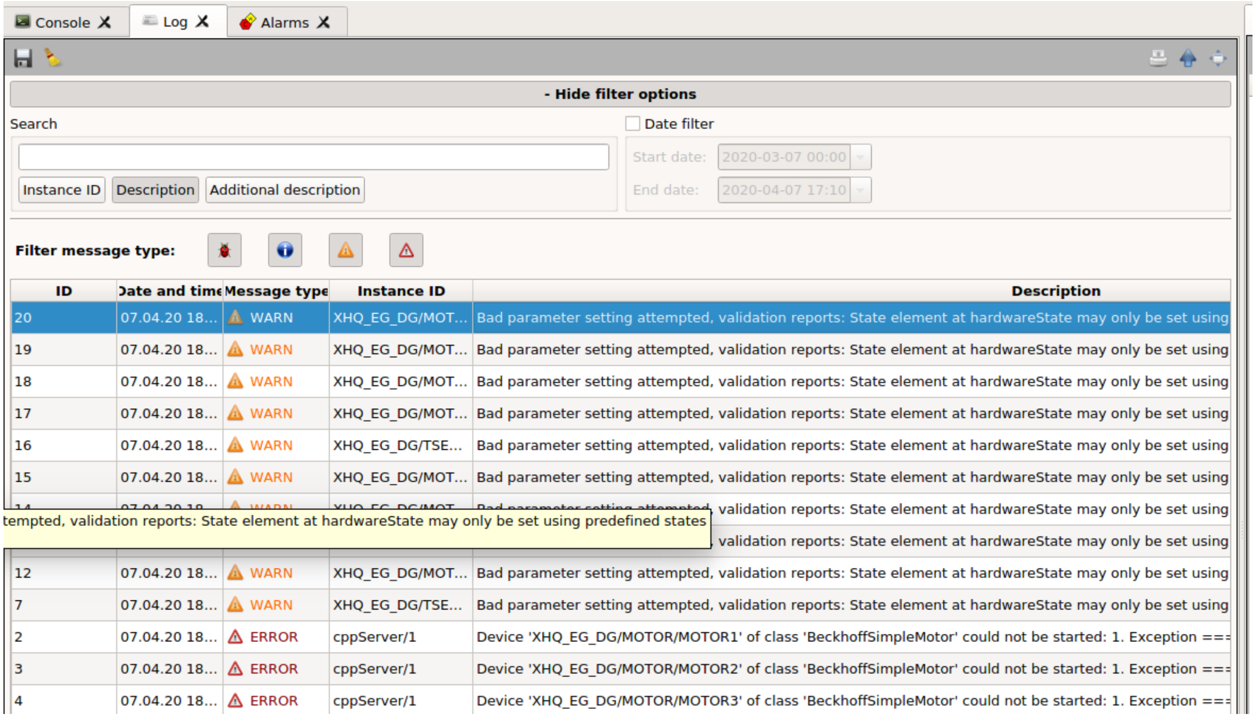


Performance Statistics		
Hostport	44444	
Delay on Input channel	500 ms	500 ms
Fast Data forwarding queue size	5	5
Lossy Data forwarding queue size	100	100
Property update interval	500 ms	500 ms
Instantiate wait time	500 ms	500 ms
Connected clients count	1	
Log Forwarding Level	ERROR	ERROR
P2p Devices	[]	[]
Network performance monitoring		
Minimum Client Version		

Fig. 3: GUI Server configuration

The forwarded log level can be configured on the GUI server as shown in the `(Fig. %s)`. The GUI client will receive every log message from the whole Karabo Topic, which is why the default is set to `INFO`, as logging can result into a denial of service.

Double-clicking a row in the `Logging Panel` will look up the `deviceId` in the `Topology Panel`.



Search

☐ Date filter

Start date: 2020-03-07 00:00

End date: 2020-04-07 17:10

Filter message type:

ID	Date and time	Message type	Instance ID	Description
20	07.04.20 18...	⚠ WARN	XHQ_EG_DG/MOT...	Bad parameter setting attempted, validation reports: State element at hardwareState may only be set using
19	07.04.20 18...	⚠ WARN	XHQ_EG_DG/MOT...	Bad parameter setting attempted, validation reports: State element at hardwareState may only be set using
18	07.04.20 18...	⚠ WARN	XHQ_EG_DG/MOT...	Bad parameter setting attempted, validation reports: State element at hardwareState may only be set using
17	07.04.20 18...	⚠ WARN	XHQ_EG_DG/MOT...	Bad parameter setting attempted, validation reports: State element at hardwareState may only be set using
16	07.04.20 18...	⚠ WARN	XHQ_EG_DG/TSE...	Bad parameter setting attempted, validation reports: State element at hardwareState may only be set using
15	07.04.20 18...	⚠ WARN	XHQ_EG_DG/MOT...	Bad parameter setting attempted, validation reports: State element at hardwareState may only be set using
14	07.04.20 18...	⚠ WARN	XHQ_EG_DG/MOT...	Bad parameter setting attempted, validation reports: State element at hardwareState may only be set using
				, validation reports: State element at hardwareState may only be set using
12	07.04.20 18...	⚠ WARN	XHQ_EG_DG/MOT...	Bad parameter setting attempted, validation reports: State element at hardwareState may only be set using
7	07.04.20 18...	⚠ WARN	XHQ_EG_DG/TSE...	Bad parameter setting attempted, validation reports: State element at hardwareState may only be set using
2	07.04.20 18...	⚠ ERROR	cppServer/1	Device 'XHQ_EG_DG/MOTOR/MOTOR1' of class 'BeckhoffSimpleMotor' could not be started: 1. Exception ===
3	07.04.20 18...	⚠ ERROR	cppServer/1	Device 'XHQ_EG_DG/MOTOR/MOTOR2' of class 'BeckhoffSimpleMotor' could not be started: 1. Exception ===
4	07.04.20 18...	⚠ ERROR	cppServer/1	Device 'XHQ_EG_DG/MOTOR/MOTOR3' of class 'BeckhoffSimpleMotor' could not be started: 1. Exception ===

tempted, validation reports: State element at hardwareState may only be set using predefined states

Fig. 4: Logging Panel

5.3 The Macro Panel

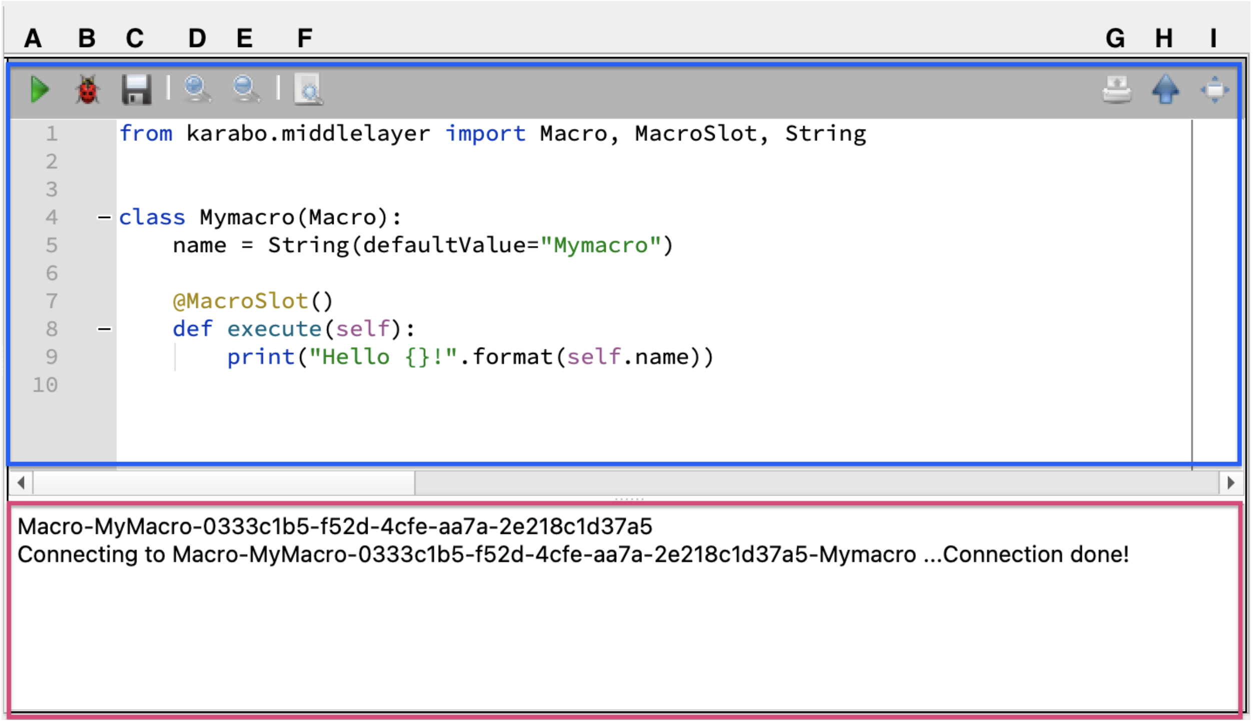
The Macro Panel allows the user to write Python scripts for small, recurring tasks. A macro can be created or loaded from a device via the Project Panel.

Once it has been created or loaded, just double-click it to send it to the Macro Panel, where a view similar to the following will be shown.

This panel consists of a toolbar, which provides shortcuts to:

- A - Create a macro instance.
- B - Create a macro instance in debug mode
- C - Save Macro to a file.
- D - Increase font size.
- E - Decrease font size.
- F - Check the Code quality.
- G - Print the macro panel.
- H - Undock the panel.
- I - Maximize the panel.

The **blue rectangle** is the editor where one can view and edit the macro code. For newly-created macros, a default template is shown. This macro has a property called *Name* and a slot called *execute*. The **red rectangle** is a console that shows information about the macro runtime, such as its connection status or its output.



```
from karabo.middlelayer import Macro, MacroSlot, String


class Mymacro(Macro):
    name = String(defaultValue="Mymacro")

    @MacroSlot()
    def execute(self):
        print("Hello {}!".format(self.name))

```

```
Macro-MyMacro-0333c1b5-f52d-4cfe-aa7a-2e218c1d37a5
Connecting to Macro-MyMacro-0333c1b5-f52d-4cfe-aa7a-2e218c1d37a5-Mymacro ...Connection done!
```

Fig. 5: Macro Panel

5.3.1 Features in Macro Editor

The Macro Editor offers various convenient features, similar to those found in other code editors, that significantly ease the life of coders.

Code quality checker:

The Code Quality Checker tool button (The button at ‘F’ position in the above image) runs *pyflakes* and *pycodestyle* on the code in the editor. Using *pyflakes* the quality checker analyzes the code and identifier various error, highlighting them with a red squiggly underline. Additionally, *pycodestyle* checks the code against the style conventions defined in PEP8, marking any inconsistencies in a blue squiggly underline. A comment below the code-line will also be displayed in case of any error or style inconsistency.

The tool button has three different states.

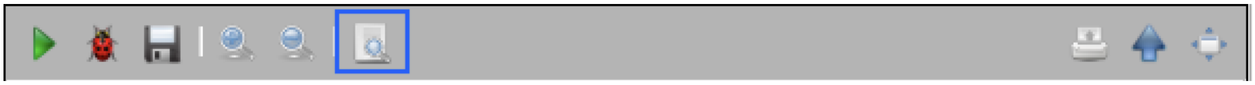


Fig. 6: Code Quality Checker: Before running.

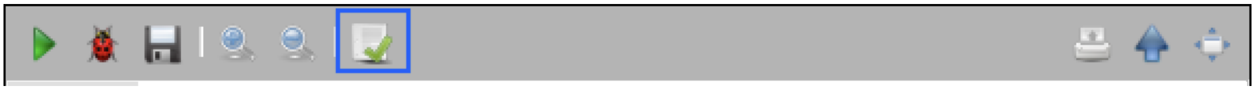


Fig. 7: Code Quality Checker: With no error.

When there is any error or style issue, the tool bar shows an additional button to clear the comment about the error/issue and the underline, from the editor.

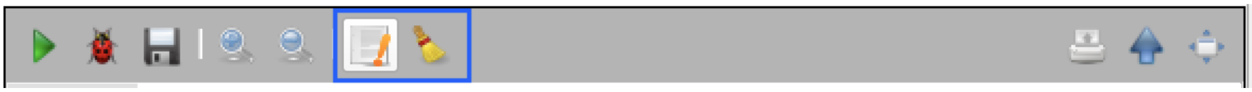


Fig. 8: Code Quality Checker: When any error.

Auto-Suggestion:

The editor suggests the possible completions or options on typing at least three letters. It provides intelligent suggestions of names, functions, classes, keywords from the imported and common namespace like "karabo.middlelayer"

Syntax highlighting:

The Macro editor helps to visually distinguish different element of the code - like keywords, variables, strings, comments, by applying different colors/styles to them.

Find and Replace:

The editor provides a Find or/and Replace functionality through a toolbar. The Ctrl+F keyboard shortcut shows the Find toolbar while Ctrl+R shows the Replace toolbar. The toolbar allows to search for the text with case-sensitive option on or off. The tool highlights all the search hits in the editor and displays the total number of search result in the toolbar.

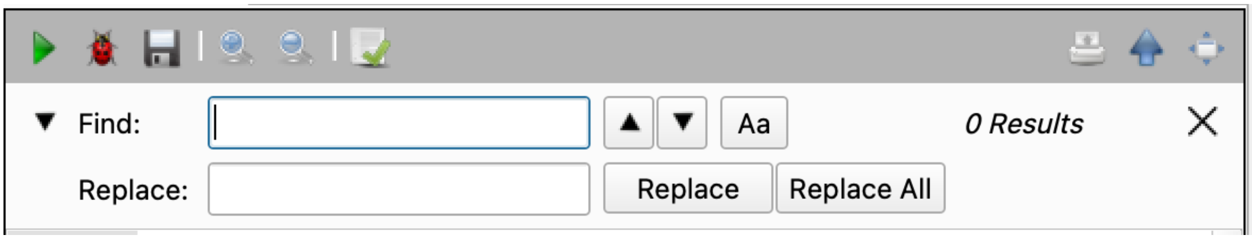


Fig. 9: Find Replace Toolbar

Indentation Guide:

The indentation guide- as vertical line on the left side of the code, represents the level of indentation of each line. This helps to easily identify any inconsistencies or error in the indentation.

Line wrap guide:

The vertical line in the editor indicate when a line reaches the Python standard limit of 79 characters.

Code Fold:

The plus and minus icons on the left side- after the line number- enables to expand and fold a code block.

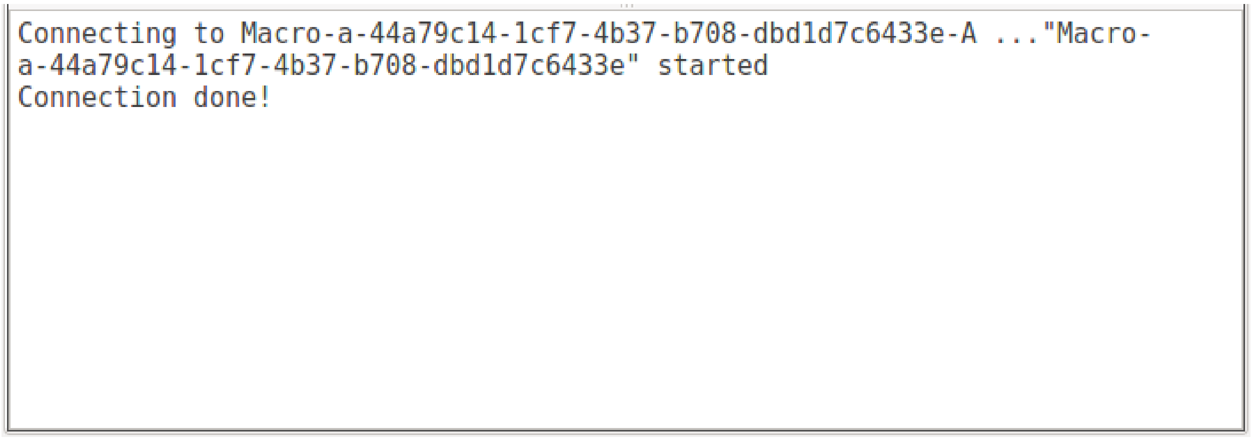
5.3.2 Keyboard shortcuts in Editor

- Ctrl+C : Copy the selection
- Ctrl+X : Cut the selection
- Ctrl+V : Paste from clipboard
- Ctrl+Z : Undo the previous action
- Ctrl+F : Show Find toolbar
- Ctrl+R : Show Replace toolbar
- Ctrl++ : Increase font size
- Ctrl+- : Decrease font size
- Tab : Indent the selected line(s) by four spaces
- Shift+Tab : De-indent the selected line(s) by four spaces

5.3.3 Instantiating a Macro

In order to instantiate a macro, the user can either run it from the Project Panel or use the **A** button from the toolbar. In case there is any error with the writen code, it will be logged in the Log Panel, where the user will have access to its stacktrace, which can help correcting the code.

Once it has been instantiated, information about the connection will be shown in the console window (*red rectangle*).

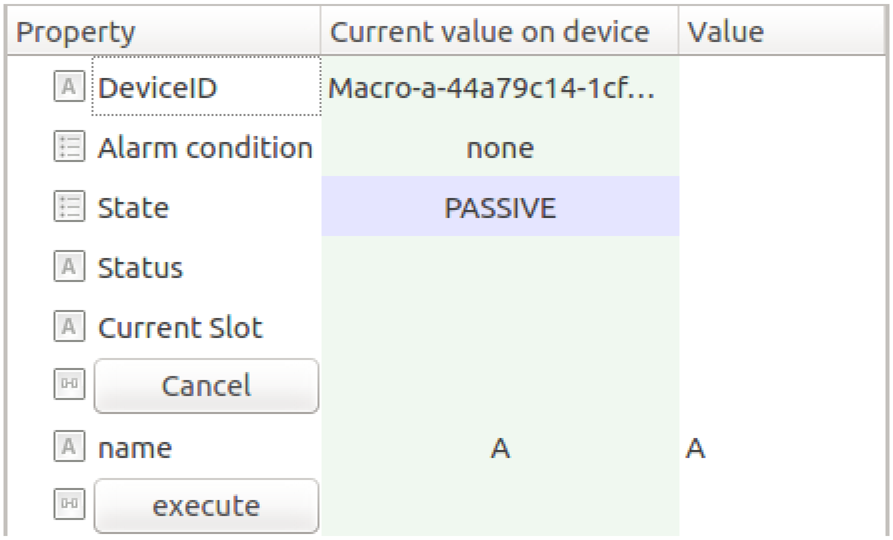


```
Connecting to Macro-a-44a79c14-1cf7-4b37-b708-dbd1d7c6433e-A ..."Macro-
a-44a79c14-1cf7-4b37-b708-dbd1d7c6433e" started
Connection done!
```

Its instance is accessible via the Project Panel where, after selected, will show its configuration in the Configuration Panel:

This panel shows the macro default properties (*State*, *Status*, etc) and also the user-defined properties (only *Name* in this example). Also, the *execute* slot is provided as a button for running the macro. The user can provide as many properties and slots as needed.

In this example, clicking the *execute* slot in the Configuration will print a message containing the *name* property in the console:



Property	Current value on device	Value
DeviceID	Macro-a-44a79c14-1cf...	
Alarm condition	none	
State	PASSIVE	
Status		
Current Slot		
Cancel		
name	A	A
execute		

Note: Ideally the tasks executed in a macro should be small, not having long loops or lasting too long. In order to cancel a macro execution, the user can click on the *Cancel* slot in the Configuration Panel.

5.3.4 Shutting Down a Macro

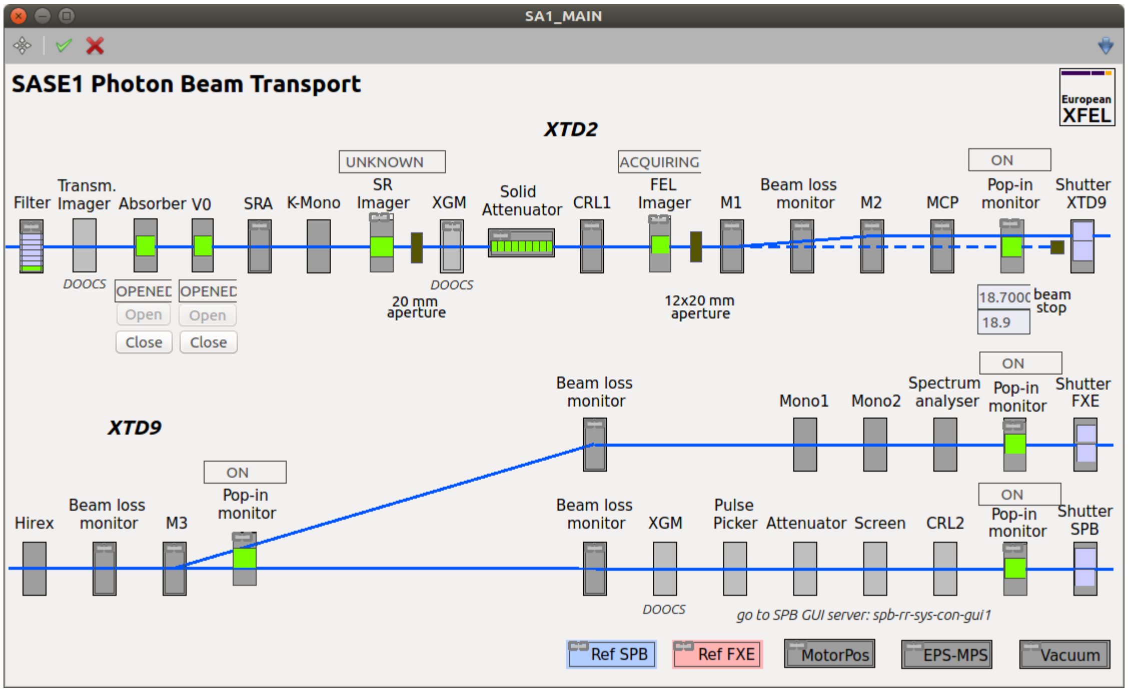
A macro can be shutdown by means of the *Shutdown instance* button in the Configuration Panel or also from the Project Panel.

5.3.5 Rerunning a Macro

To reflect changes made to the macro code, it is necessary to first shutdown the current instance (if any) and instantiate it again from the toolbar or the Project Panel.

THE SCENE PANEL

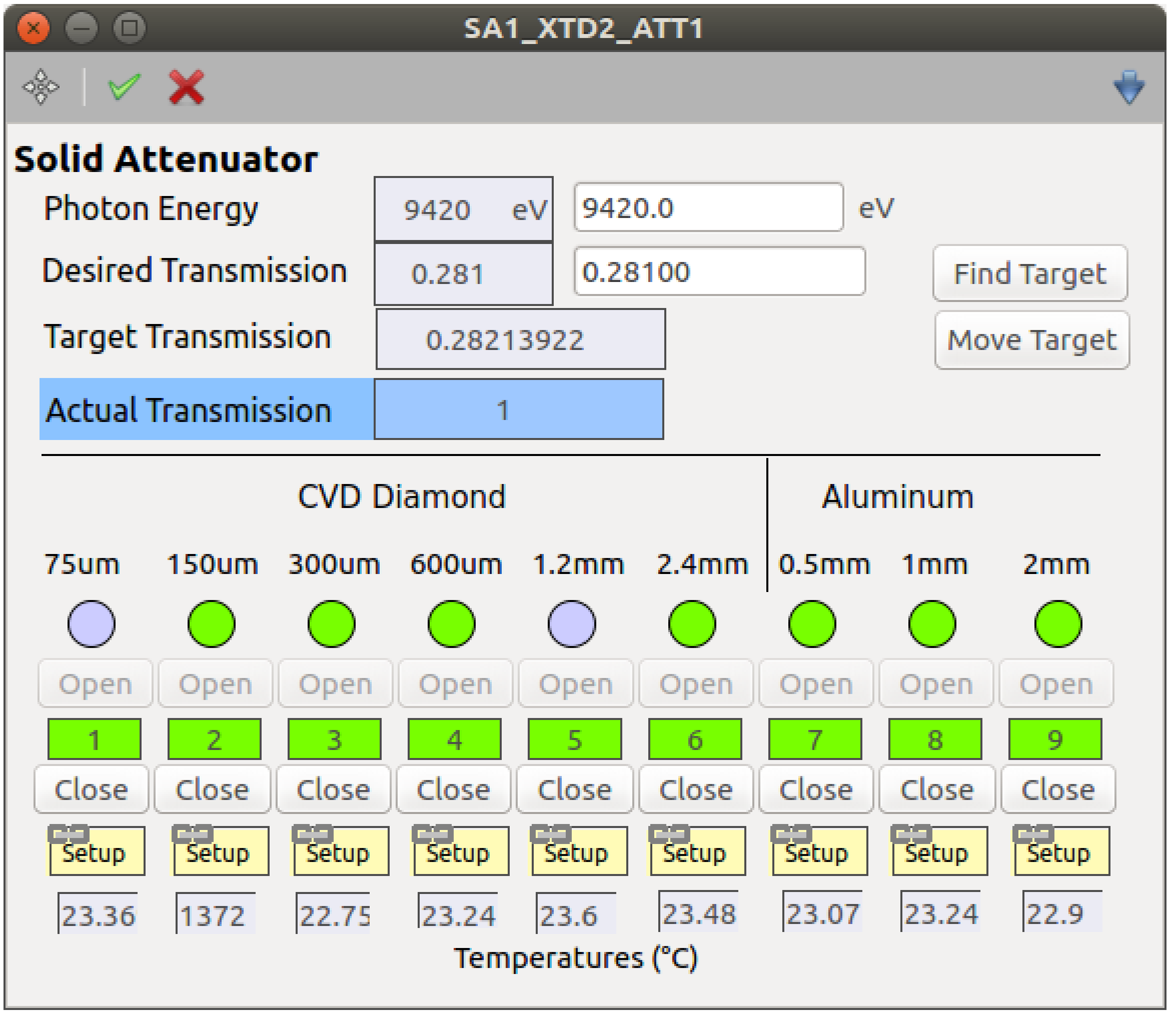
The Scene Panel is Karabo's default container for showing or editing scenes. Scenes in Karabo provide a way to customise device views, covering any diagnostic or control elements in a compact and comprehensive view. The image below shows an example of a master panel in a central scene. Each one of this components will be described in this document. For now, It's relevant to notice that we can represent the relations between devices, access the system state and even provide links to sub-systems with their respective scenes (bottom right).



The following example shows a detailed panel in a central scene. We see how customizable a scene can be, offering display values for gauges, spark lines indicating trends, state and alarm conditions, etc. Note how the state and alarm condition are separated for the gauge **Gauge_Down2**. The bottom buttons are hyper-links to the other detail panels and the master panel.

A Karabo scene is not necessary created by a graphical user interface (GUI), also devices can have scenes where they can be downloaded from. If a scene is derived from a device, it is typically shown in **control-mode** only, meaning that no manipulation of the scene elements can be done.

For editable scenes, a project is also needed. After creating a project, the operator can create a new scene or load a previously created one and is free to start customizing it and saving/replacing it in his project. For this customization, Karabo provides a variety of features that can be used via the **Scene Toolbar** and widgets, the so-called **Karabo Controllers**.



SA1_XTD2_ATT1

Solid Attenuator

Photon Energy 9420 eV 9420.0 eV

Desired Transmission 0.281 0.28100

Target Transmission 0.28213922

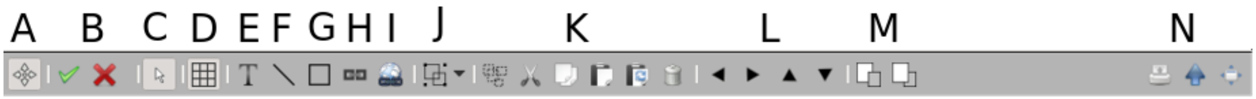
Actual Transmission 1

CVD Diamond						Aluminum		
75um	150um	300um	600um	1.2mm	2.4mm	0.5mm	1mm	2mm
○	●	●	●	○	●	●	●	●
1	2	3	4	5	6	7	8	9
23.36	1372	22.75	23.24	23.6	23.48	23.07	23.24	22.9

Temperatures (°C)

6.1 Scene Toolbar

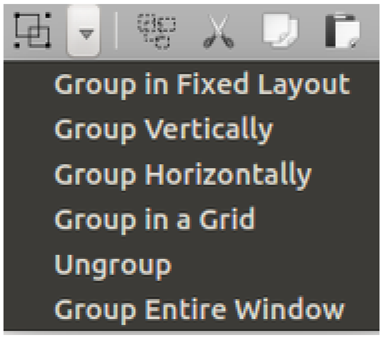
The scene toolbar houses necessary tools for editing the scene.



- A - Enter design mode, where it is possible do add/remove/modify objects
- B - Accept or discard changes
- C - Selection tool
- D - Enable/Disable Grid snapping
- E - Insert text field
- F - Draw line
- G - Draw rectangle
- H - Create *Scene_Link*
- I - Create **Web_Link_**
- J - Layout customization for *Widget_Layouts*
- K - Tools for selecting all items, copying, pasting and deleting
- L - Actions for moving objects. Also accessible using the key strokes (arrow keys)
- M - Bring selected item(s) to front or send to back
- N - Base options: Dock/Undock/Maximize

6.1.1 Widget Layouts

Configuring the correct widget layouts can be tricky. For grouping widgets in a common layout, first it is needed to select the desired widgets using the tool **C** shown above. With this tool, the user is able to either make a rectangle selection (please note that the **whole** widget must be inside the rectangle) or manually select all widgets while holding the **Shift** key. After the selection, if is possible to group the widgets in a layout (tool **I**) in any of the following manners:



- Group in Fixed Layout: the widgets will be grouped as they are;
- Group Vertically: the widgets will be grouped in row major, where each widget will occupy one row of the layout;
- Group Horizontally: the widgets will be grouped in column major, where each widget will ocupy one column of the layout;

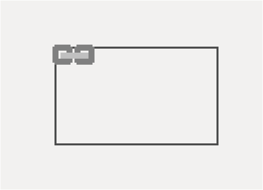
- Group in a Grid: the widgets will be grouped in a grid. The row and column count will be defined more or less based on the current widgets location. That means that the user still need to drag the widget where they are supposed to stay. Note that it is impossible to correctly group the widgets **exactly** as the user meant, but it offers good results in most of the cases;
- Ungroup: ungroup already grouped widgets. You may want to ungroup the default property layouts to provide some finer-grained layouts for your scene;
- Group Entire Window: auxiliar tool for grouping the whole screen as it is.

When dragging a property from a device instance into the scene panel, this property comes along with, maybe, some labels and additional widgets, all grouped in a proper layout. If a more concise grouping is needed, it might be needed to ungroup these layouts and group them again as needed.

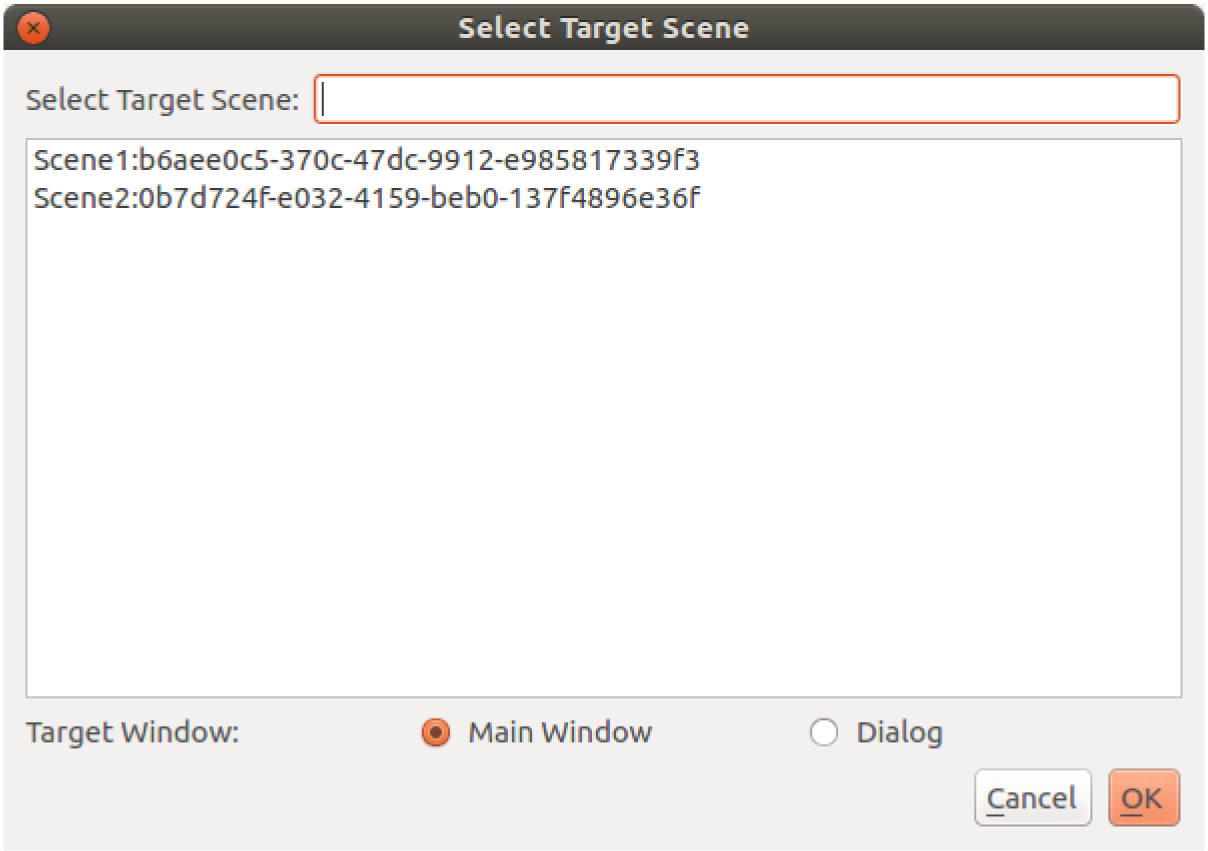
6.1.2 Scenelink - Hyperlinks between Scenes

It's possible to link between scenes using the scene link widget. For this, please go to edit mode of the scene and select the tool *Add scene link to scene* from the toolbar. Afterwards, click on the scene and a dialog will pop up for configuration.

Scenelink - Hyperlink widget:



In this configuration window the target can be selected as well as the link widget should open the scene in the **Main Window**, or in a new pop-up **Dialog**.



Select Target Scene

Select Target Scene:

Scene1:b6aee0c5-370c-47dc-9912-e985817339f3
Scene2:0b7d724f-e032-4159-beb0-137f4896e36f

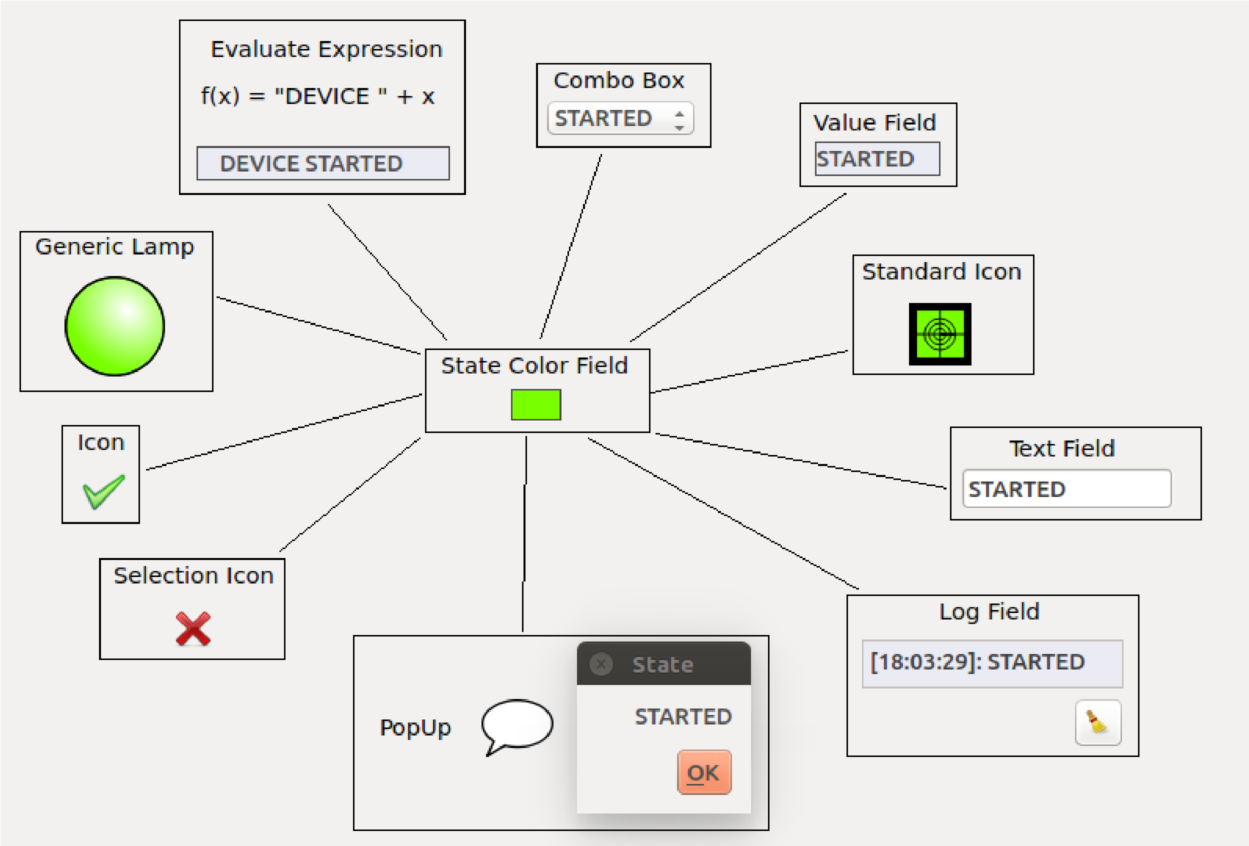
Target Window: ● Main Window ○ Dialog

CHAPTER

SEVEN

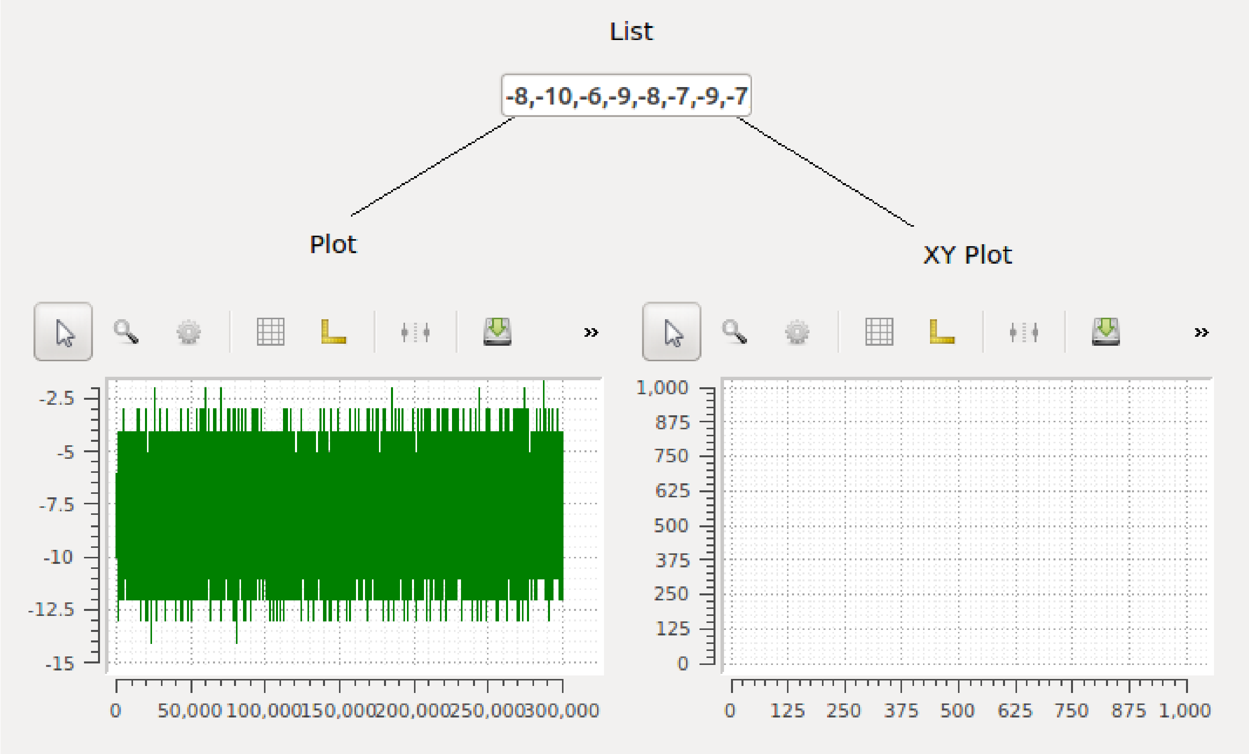
WIDGETS - CONTROLLERS AND CONVERSION

Every property type has a **default** widget. For example, a bool property will be translated to a `Checkbox`. An editable input to a `Text Line Edit`, etc. It is possible to convert from one widget type to another for customization. In order to mutate a controller widget please select the widget and right-click to show the `custom menu`. The picture below shows the possible conversions for a `State` property. The default controller is a `State Color Field` widget.



These conversions offer a vast range of scene customizations.

Other common conversions are from vectors to plots. A vector property (list of values) can be seamlessly converted to a Plot, as shown in the following Figure. As it is known for the XY Plot, one more vector is needed for the plot to be valid.



-8,-10,-6,-9,-8,-7,-9,-7

7.1 Basic Widgets

The Tables below offer a cheatsheet regarding the possible widgets the user can choose for displaying primitive types.

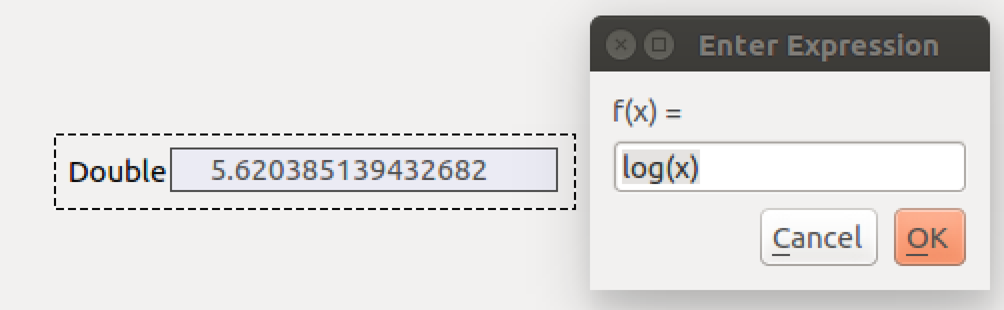
Basic Widgets					
	Integer	Float	Boolean	String	Vector
Evaluate_Expression	R	R		R	
Hexadecimal_	W				
Single_Bit	R				
Value_Field	R	R		R	
SpinBox	W	W			
Toggle_Field			RW		
Switch_Bool			R		
Text-Float-Integer-Field	W	W		W	
List					RW

7.1.1 Evaluate Expression

This widget can be used to manually define the formatting using python string formatting (`"{:.2f}".format(x)`) or to derive another value with a function.

The full namespace of python `builtins` and `numpy` can be used to define the function. The example below uses the *log* function of numpy.

Note: The `Evaluate Expression` widget background is colored according to the alarm and warning attributes of images/widgets/string_change_expression.png



Double 5.620385139432682

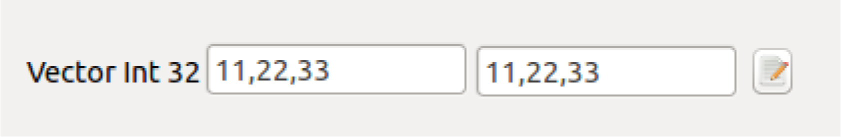
Enter Expression

f(x) = log(x)

the property if the value exceeds a threshold.

7.1.2 List

The `List` widget is used to display vector properties. This widget is used for both displaying and editing vector properties and does not provide any alarm information.



Vector Int 32 11,22,33 11,22,33

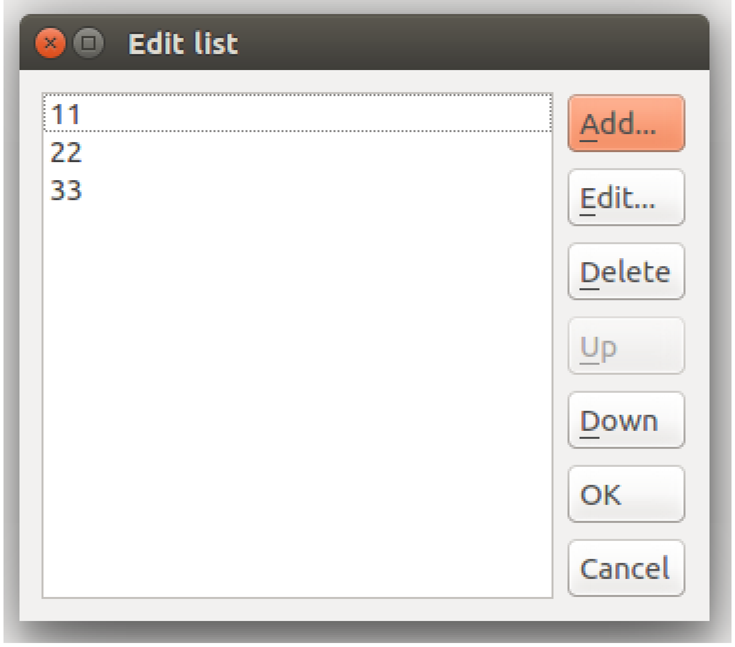
The edit widget provides a `list dialog` which can be opened by clicking on the button to the right.

The `list dialog` offers a comfortable way to modify vectors with a large number of elements.

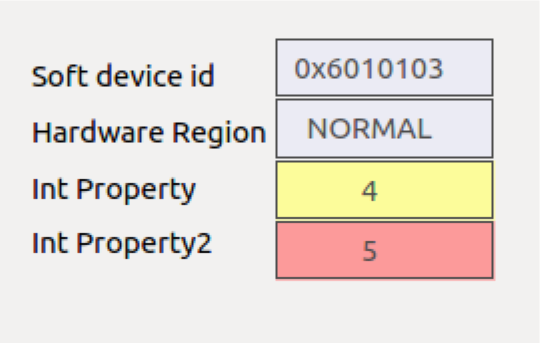
7.1.3 Value Field

The `Value Field` refers to a `default` widget for representing values. It is typically chosen as first widget when a normal readonly property is dragged onto the scene. This widget is able to visualize and work with defined property `displayTypes` of ‘bin’, ‘oct’ and ‘hex’. If the property defines one of the attributes `absoluteError` or `relativeError`, the widget will derive the formatting display. For floating point properties a maximum precision of 8 is provided.

Note: The `Value Field` widget is one of the first choice elements to display alarms and warnings of simple property types. The widget background is colored correspondingly and the value exceeded an alarm value.



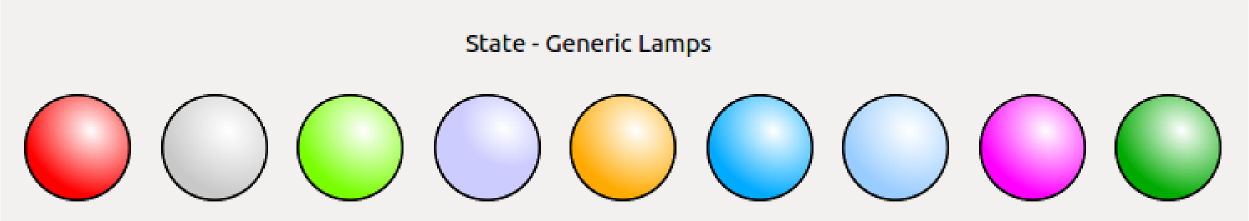
11
22
33



Soft device id	0x6010103
Hardware Region	NORMAL
Int Property	4
Int Property2	5

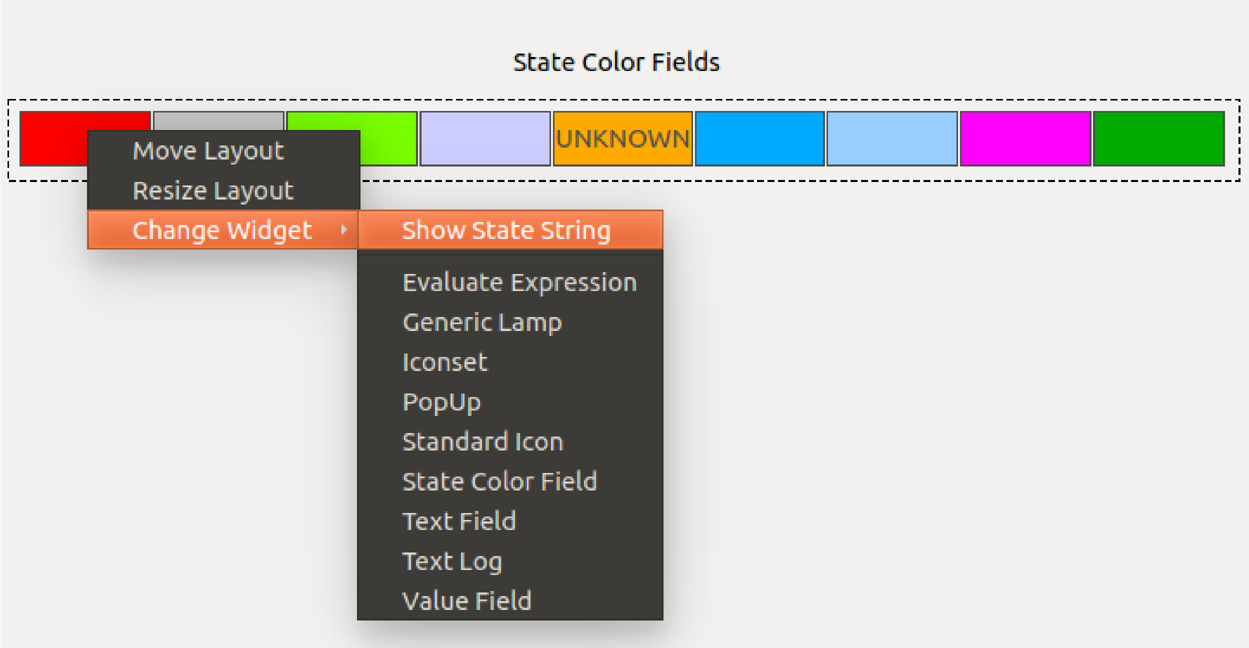
7.1.4 Generic Lamp

This widget is used for string properties which have a `displayType` **State**, the so-called `State Elements`. The property value is displayed with the corresponding state color.



7.1.5 State Color Field

This widget is used for string properties which have a `displayType` **State**, the so-called `State Elements`. In addition, this widget comes with a context menu action to display the state string on the widget!

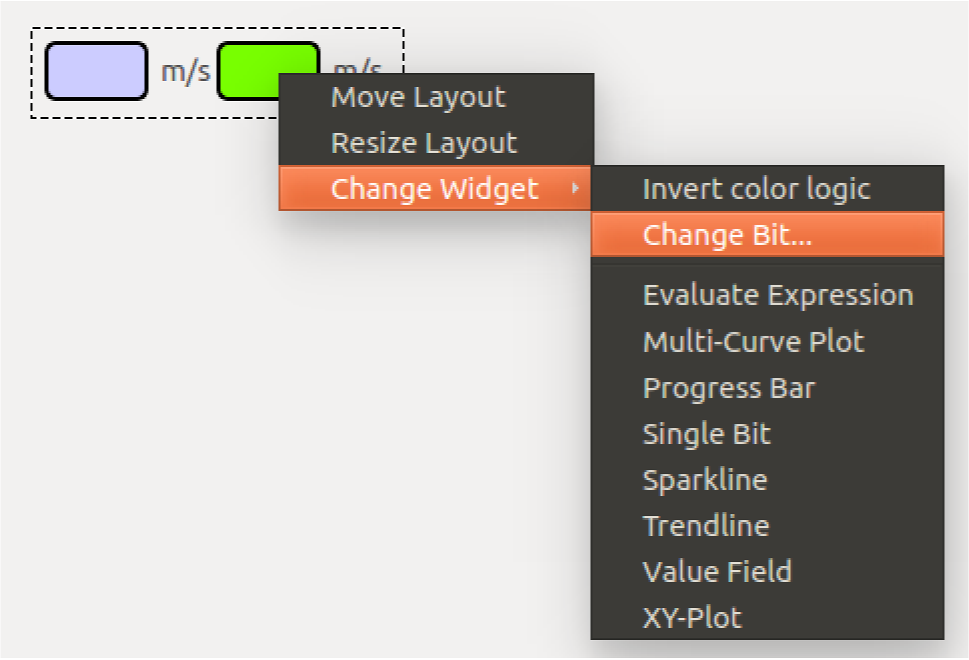


7.1.6 Single Bit

This widget is used to visualize single bits from integer properties. It has a rectangular rounded shape and comes with two context menu options:

- Change the `bit number` to evaluate
- Invert the color logic

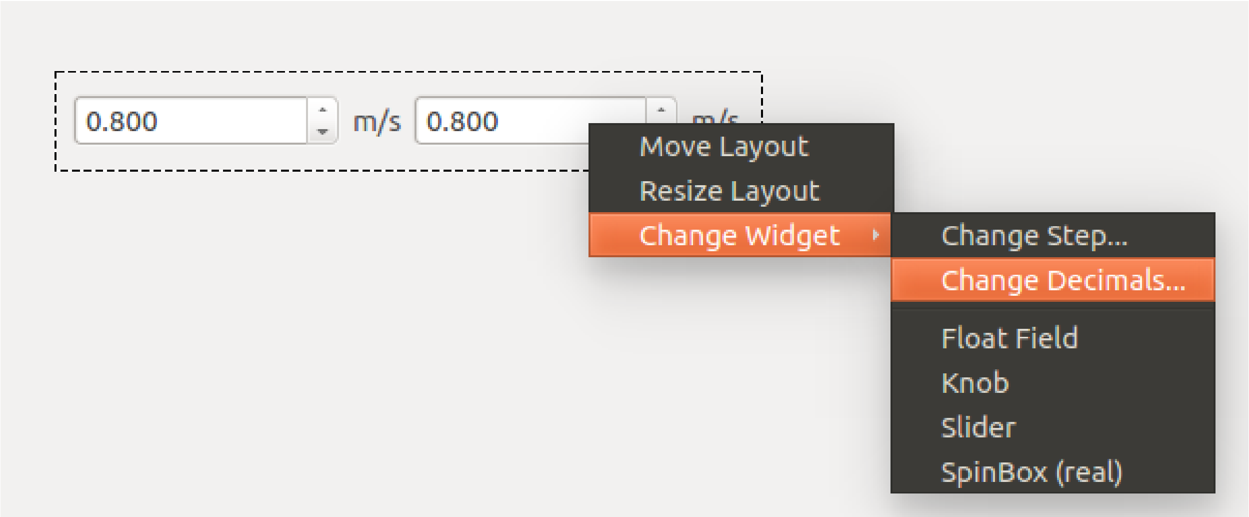
By default the `State.ACTIVE` is used when the Bit is actively set! The `State.PASSIVE` color then illustrates if the Bit is not set.



7.1.7 SpinBox

The spinbox can be referred to as a standard widget in every Qt application. This widget comes with two context menu options

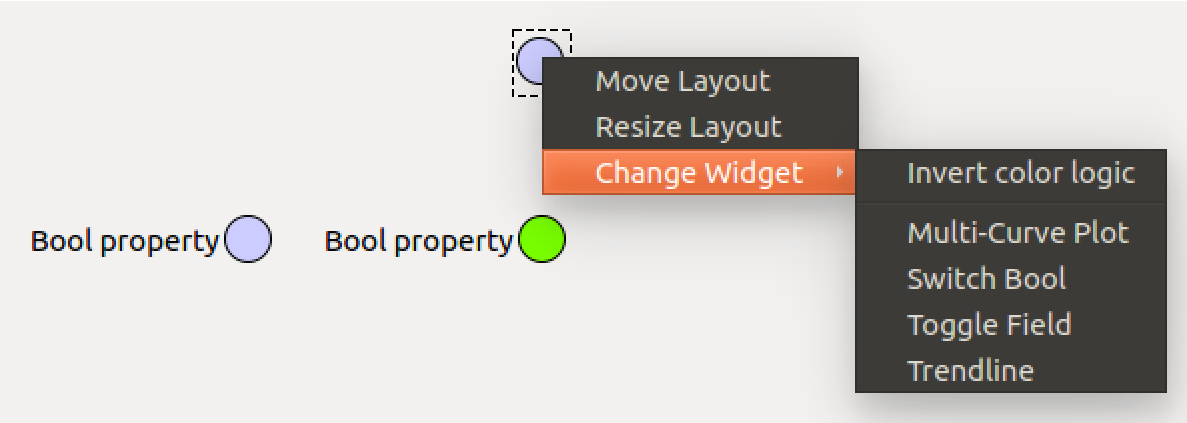
- The number of `decimals` (default: 3)
- The `stepsize` which is can be eventually used for stepping (default: 0.0)



Note: The stepping via a spinbox widget is achieved by using `arrow` key-strokes!

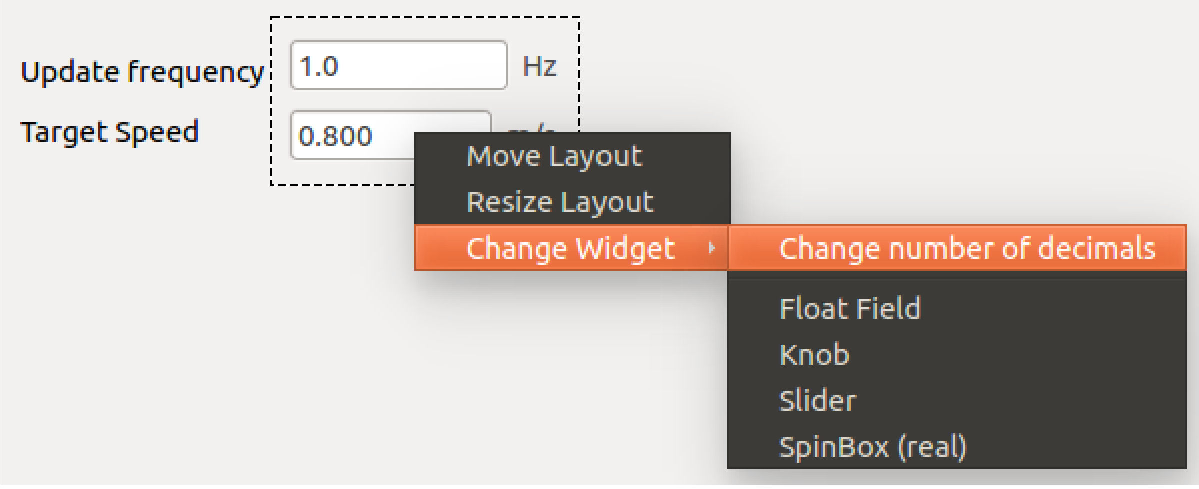
7.1.8 Switch Bool

Shows the operator the state of the bool property with colors in a `circular` shape. By default the `State.PASSIVE` color is used for `False` and the `State.ACTIVE` color belongs to `True`. The logic of the coloring can be *toggled* via the context menu of the widget.



7.1.9 Text-Float-Integer Field

The `field` boxes are building the `default` widgets for **editing** the property values of basic *float*, *integer* and *string* types.



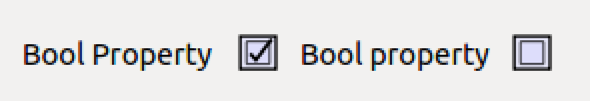
Update frequency 1.0 Hz

Target Speed 0.800

In addition, the `Float Field` has a context menu option to assign the number of *decimals*. The default value (*-1*) disables the decimal formatting. Once, a decimal has been specified, a validator will take care of correct value input.

7.1.10 Toggle Field

As the `default` widget for a `boolean` this widget simply shows the `True` and `False` with a check-mark in a box.



Bool Property ☑ Bool property □

7.1.11 The Table Element

The table-element is applicable for `VECTOR_HASH` properties, which have a 'rowSchema' attribute. The rowSchema is a *Hash*, which defines the column layout, i.e. column count, column data types and column headers.

In case the table description (rowSchema) contains a 'displayedName' field for a property this is used as the column header, otherwise the property key is used.

- For string fields with options supplied the cell is rendered as a drop down menu.
- Boolean fields are rendered as check boxes.

Additional manipulation functionality includes, adding, deleting and duplicating rows (the latter require a cell or row to be selected).

The Table widget supports drag and drop of deviceId's from the navigation panel. Dropping on a string cell will replace the string with the deviceId. Dropping on a non-string cell or on an empty region will add a row in which the first string-type column encountered is pre-filled with the deviceID.

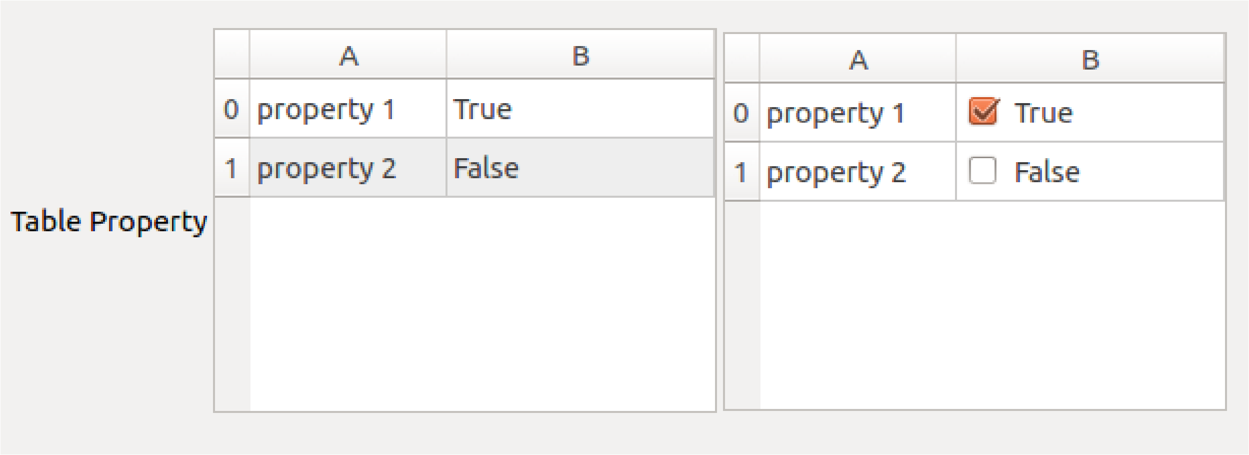


Table Property

	A	B
0	property 1	True
1	property 2	False

	A	B
0	property 1	☑ True
1	property 2	□ False

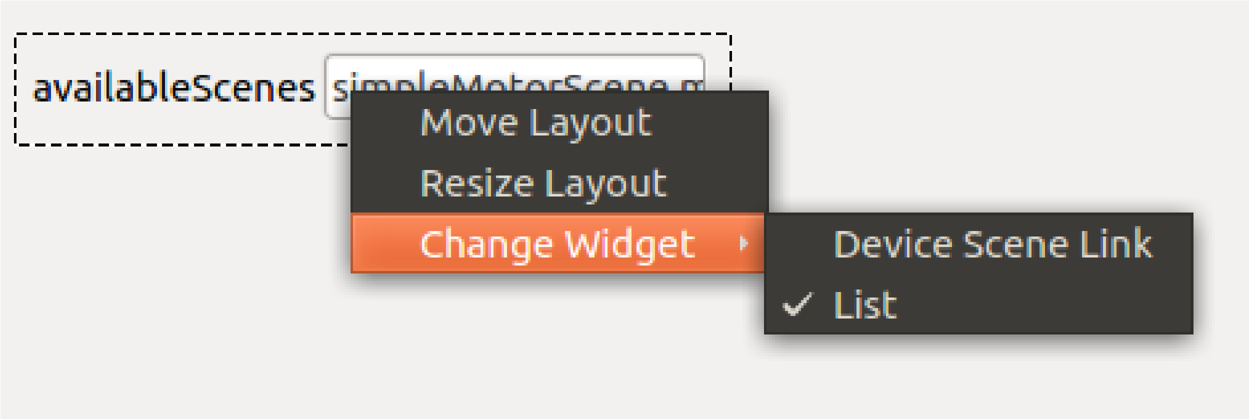
7.2 Functional Widgets

Functional Widgets					
	Integer	Float	Boolean	String	Vector
ComboBox	RW	RW		RW	
Analog_Widget	R	R			
Progress_Bar	R	R			
TextLog				R	
PopUp				R	
Slider	W	W			
Knob_	W	W			
DeviceSceneLink					R

7.2.1 Device Scene link

The `Device Scene Link` is a widget that offer features similar to the one offered by the **Scene_Link_** but differs from it in the sense that the scene offered is not stored by the project but generated dynamically by the device itself.

In order to include a `Device Scene Link` one needs to drag and drop the property `availableScenes` into a scene in edit mode and select the appropriate widget type.

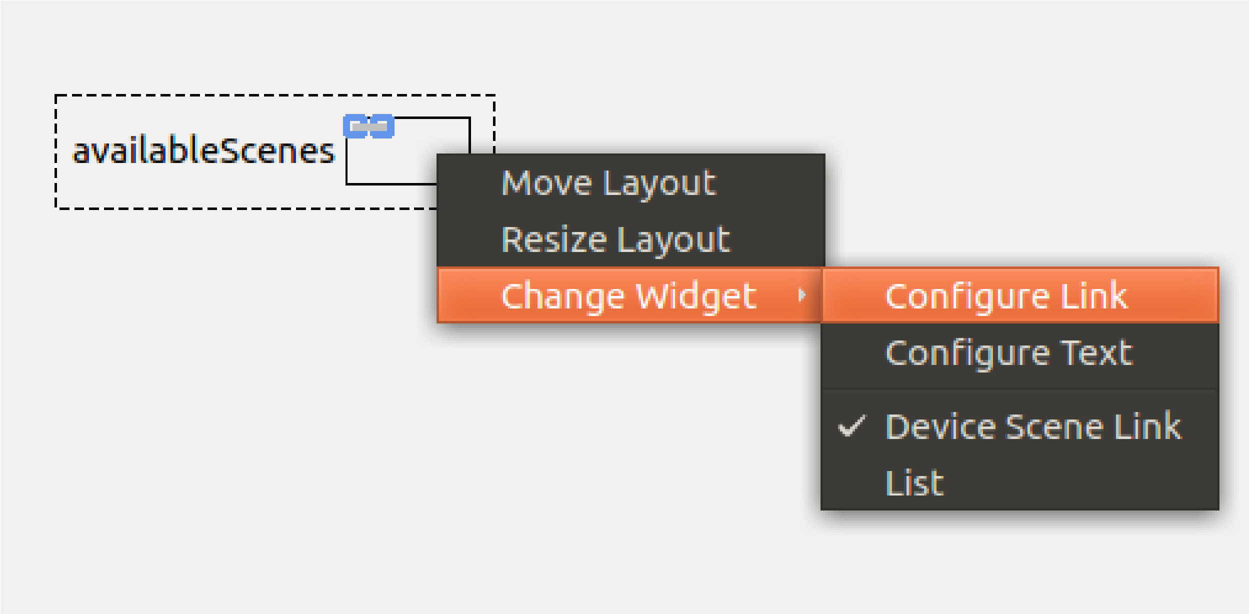


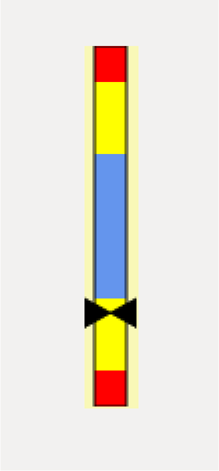
Once the property is on the scene one can adapt which one of the scenes provided by the device, as well as the text and target panel similarly to what is offered by the `Scene Link`.

7.2.2 Analog Widget

The `Analog Widget` is fully designed to display alarms and warnings of a property. It provides an estimate how far the actual value (indicated as black scaler) is within or close to an alarm or warning region.

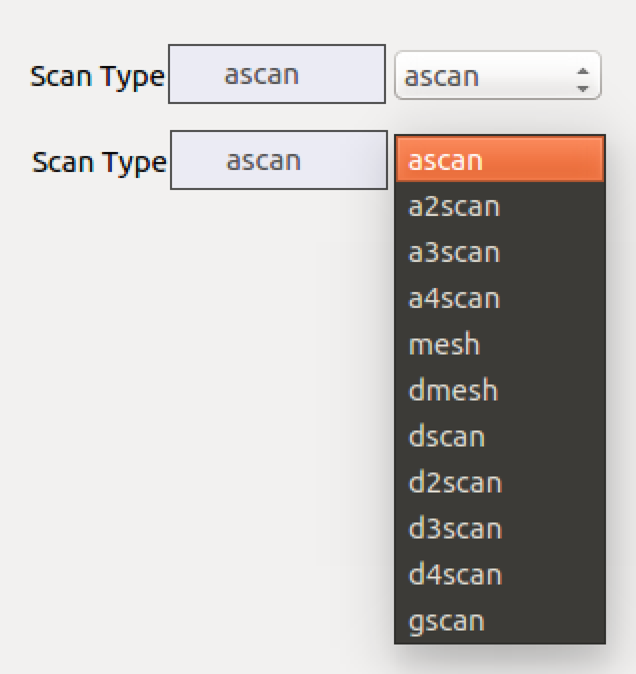
Note: For this widget a set of attribute values for alarms or warnings has to be provided. Hence, either `alarmLow` and `alarmHigh` or `warnLow` and `warnHigh` must be defined on the karabo property.


availableScenes
Move Layout
Resize Layout
Change Widget
Configure Link
Configure Text
✓ Device Scene Link
List



7.2.3 Combo Box

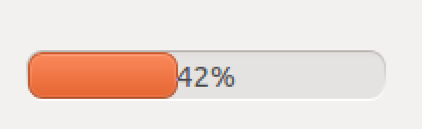
The so-called ComboBox widget is used to edit karabo properties which have a **predefined selection** of values. In the example shown below the Karabacon scantool has a fixed scan type selection which can be then edited.



Note: For this widget the attribute `options` has to be defined on the karabo property with defined values.

7.2.4 Progress Bar

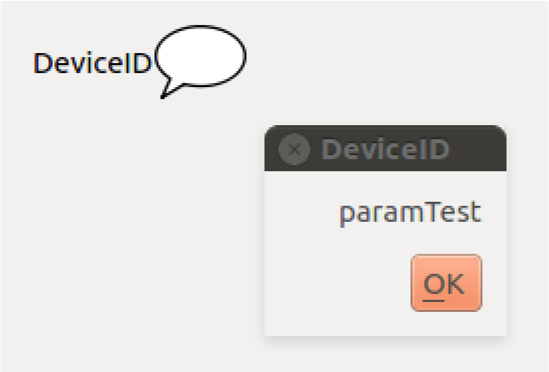
The Progress Bar widget shows to the user the property percentage value taking in consideration its maximum and minimum values.



Note: For this widget attribute values for a mininum value `minInc` or `minExc` and a maximum value `maxInc` or `maxExc` must be defined on the karabo property.

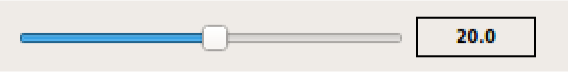
7.2.5 PopUp

If the widget is configured as a **PopUp**, the value of the property will be shown in a pop-up dialog whenever changes occur.



7.2.6 Slider

The widget can be changed to a **Slider** enabling to comfortably slide through the minimum and maximum values of a property. This widget is typically used for gain or exposure settings in cameras.

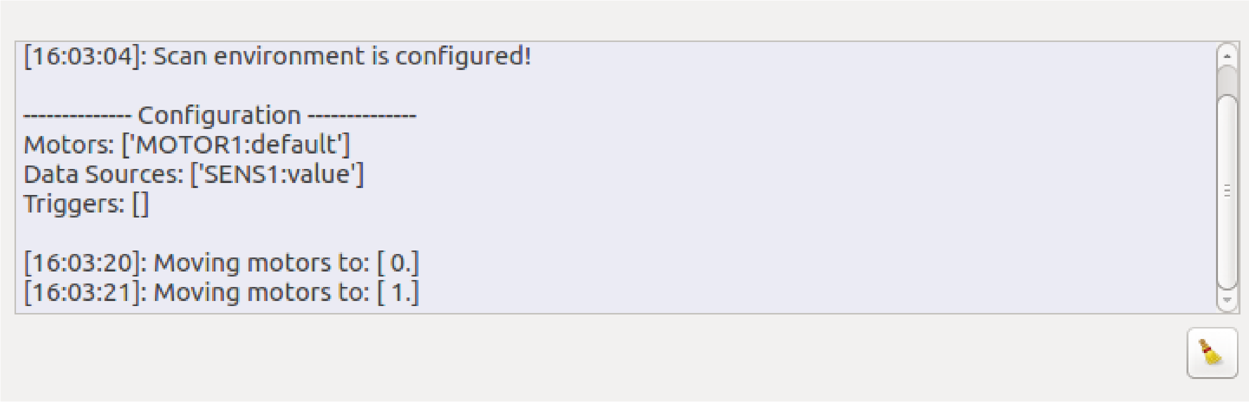


The widget comes with a number widget to show the current edit value. This can be optionally disabled. In addition, the step size of the slider can be configured via the context menu.

Note: For this widget attribute values for a mininum value `minInc` or `minExc` and a maximum value `maxInc` or `maxExc` must be defined on the karabo property.

7.2.7 Text Log

This widget is typically used to display and log the status property of devices. Every value update with a new timestamp will be displayed. It comes together with a clear button and a timestamp for each value modification.

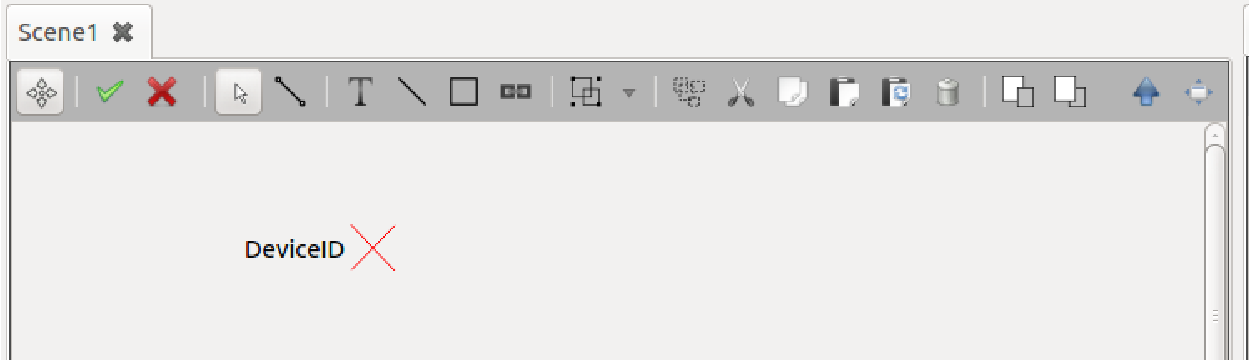


```
[16:03:04]: Scan environment is configured!

------------- Configuration -------------
Motors: ['MOTOR1:default']
Data Sources: ['SENS1:value']
Triggers: []

[16:03:20]: Moving motors to: [ 0.]
[16:03:21]: Moving motors to: [ 1.]
```

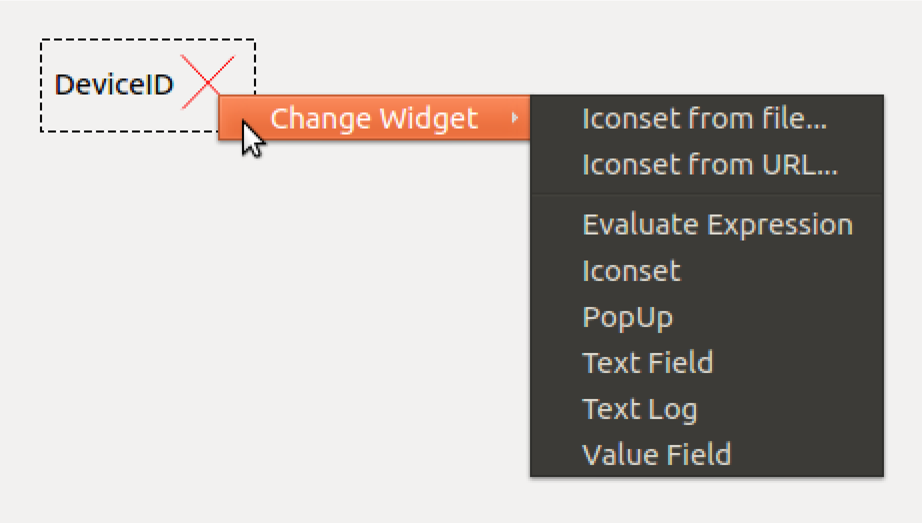
7.3 Icon Widgets

7.3.1 Iconset

This type of widget will just show a icon in the scene.



It's possible to add an icon image using an url or from a file.



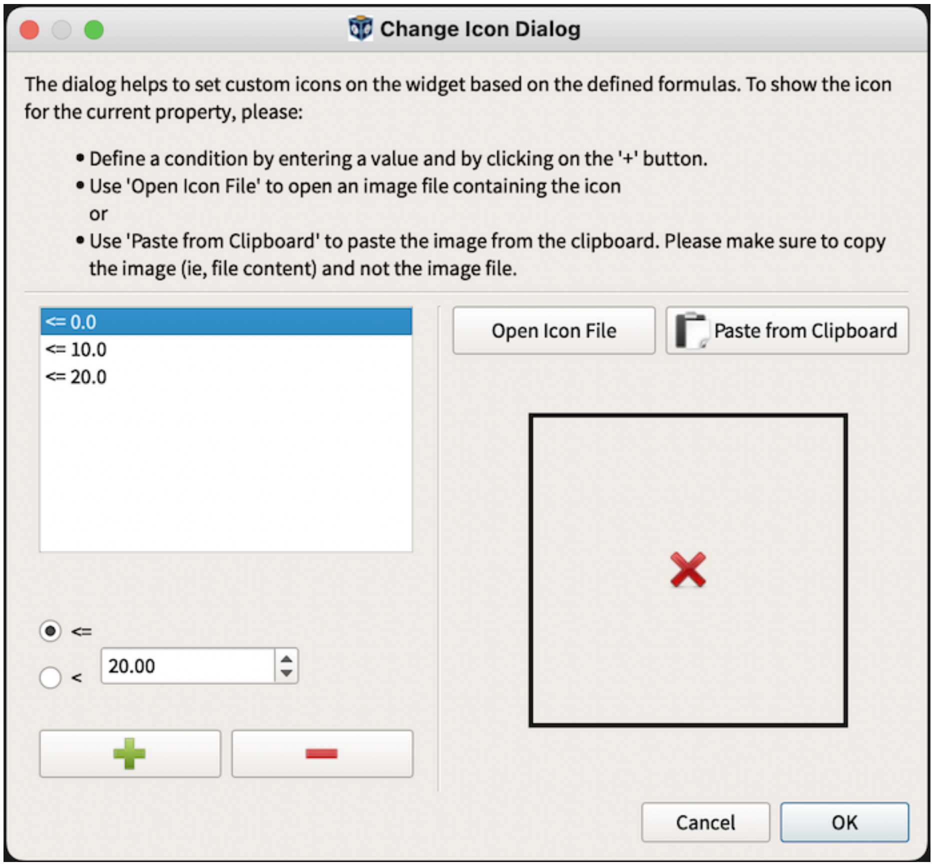
7.3.2 Icons

The 'Value Field' for an integer, a floating or a string property can be converted to an Icon Widget using the right click context menu item *Change Widget* -> *Icons*. The Icon widget can show an icon corresponding to a condition, for the value.

Custom icons can be set on Icon Widget depending on the property values. This can be done using the 'Change Icons' dialog. Use *Icons:Properties*-> *Change Icons…* right click context menu items to open the dialog.

In order to define a condition, enter the value and hit "+" button. For floating properties the dialog provides radio buttons to choose the operator '<' or '<='. For integers the operator '<=' is selected by default and the radio buttons are hidden.

For string properties with options, the dialog shows up with these options listed. For string properties with no predefined options, the dialog provides a widget to enter the value and hit "+" button to add it.



The dialog helps to set custom icons on the widget based on the defined formulas. To show the icon for the current property, please:

- Define a condition by entering a value and by clicking on the '+' button.
- Use 'Open Icon File' to open an image file containing the icon
  or
- Use 'Paste from Clipboard' to paste the image from the clipboard. Please make sure to copy the image (ie, file content) and not the image file.

Select the condition from the list and use

- 'Open Icon File' to open an image file containing the icon.
- 'Paste from Clipboard' to paste the image from the clipboard. Please

make sure to copy the image (ie, file content) and not the image file.

The preview of the imported icon can be seen on the right side of the dialog.

It is further possible to remove a selected condition via "-" button.

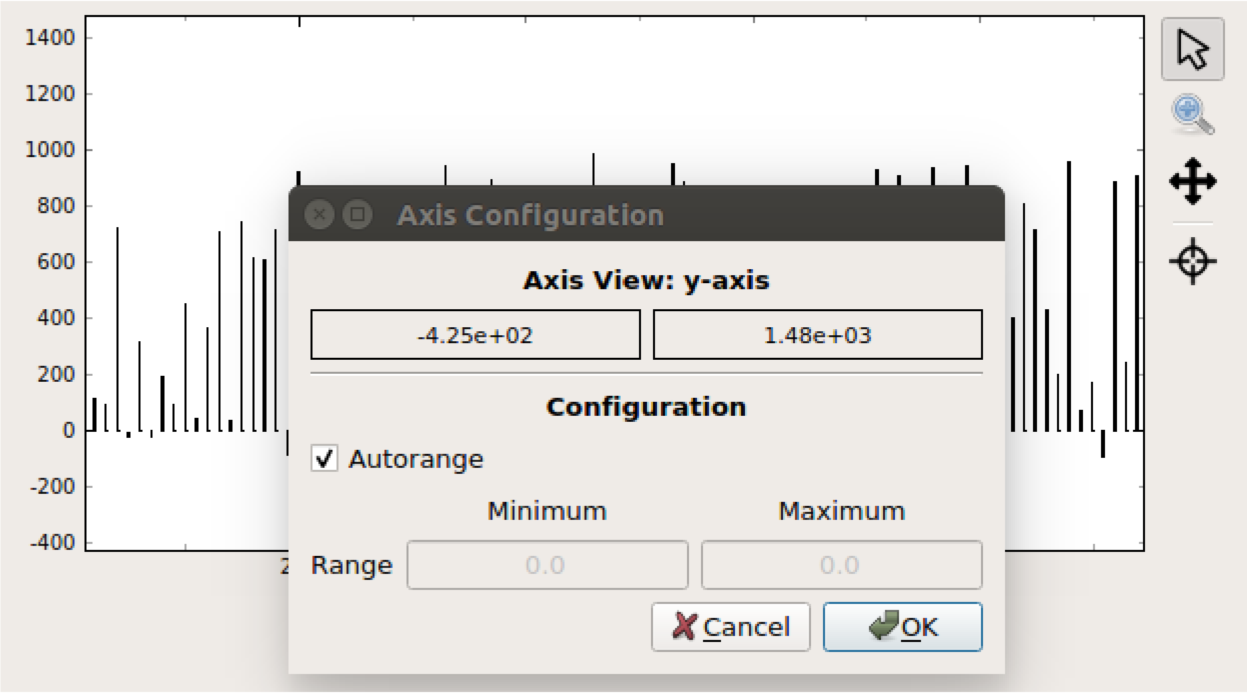
7.4 Plot Framework

Karabo has a builtin plot framwork based on `PyQtGraph`. Typically, a plot comes with natively provided extra features which are shown in the `Context Menu`. The context menu is accessible like in any other widget controller by selecting the controller in the design mode of the scene and via *right-clicking* on the plot widget.

Commonly, axes configurations can be accessed via native *double-clicking* on the plot axes for the `Axis Menu` or a few settings can be adjusted via a *right-click* on the plot itself to trigger the `View Menu`.

The `Axis Menu` is available to set the `Range` of the axis of interest. Either *autorange* or a fixed range can be defined. The `View Menu` offers the possiblity to *autorange* a plot or if available, configure the *linear* or *logarithmic* scale of a plot.

The next sessions will describe all plot widgets available in Karabo.



Axis Configuration

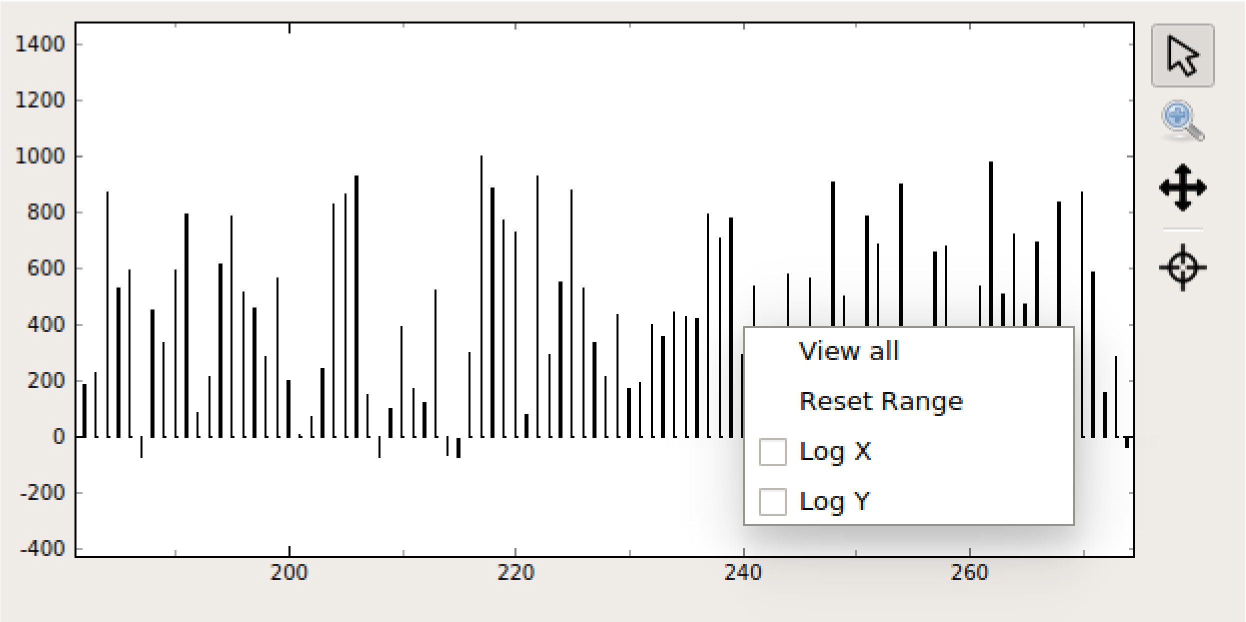
Axis View: y-axis

-4.25e+02 1.48e+03

Configuration

☑ Autorange

	Minimum	Maximum
Range	0.0	0.0



Graph Framework					
	Integer	Float	Boolean	String	Vector
Vector Bar Graph					R
Vector Fill Graph					R
Vector HistoGram Graph					R
Vector XY Graph					R
Vector XY Scatter					R
VectorRoll Graph					R
MultiCurve Graph	R	R			
Sparkline	R	R			
Scatter Graph	R	R			
Trend Graph	R	R	R		
State Graph				R	
Alarm Graph				R	

7.4.1 Trend Graphs

Trend Graph is a widget that shows the evolution of one or more properties over time. The plot itself offers quick access buttons to scale the trend data logs in either:

i. 10 minutes

ii. one hour

iii. one day

iv. one week.

The trend graph may be set to either display the full range of values or a detailed zoomed range. A trendline will provide automatic data reduction after **900** values. In addition, a trend graph can request historic data of up to **500** values from the *Dataloggers*.

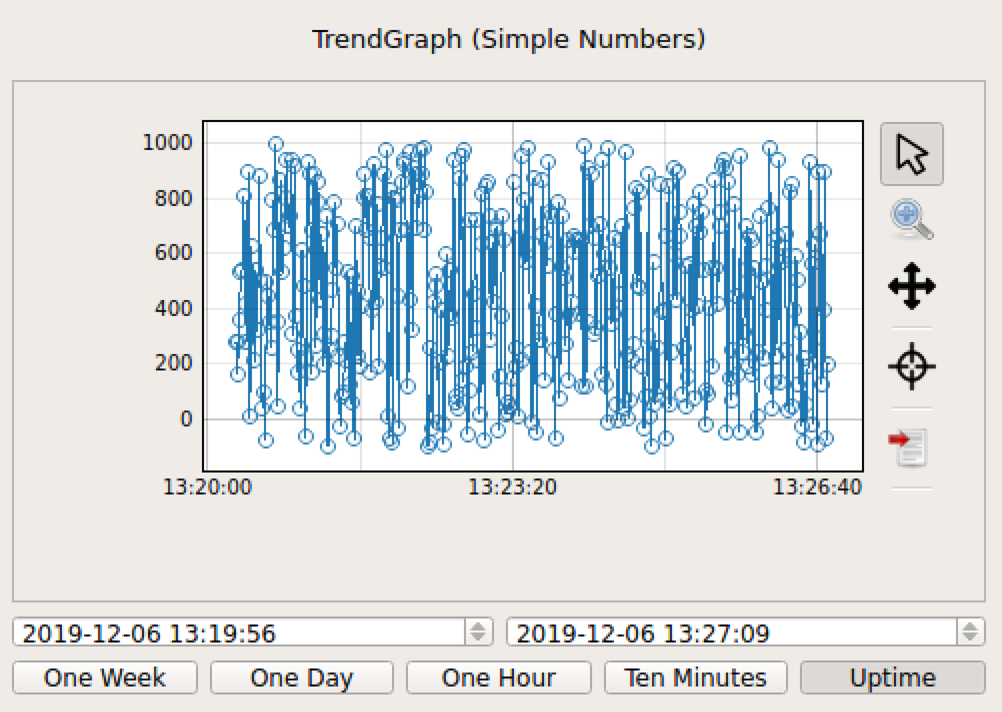
- Data will be requested when there is no data
- Data will be requested when the x-axis is sufficiently changed (buttons)
- Data will be requested when the operator zoomes in and has less data points

The same functionality is available for the `State` and `Alarm` properties of devices.

7.4.2 Multi-Curve Graph

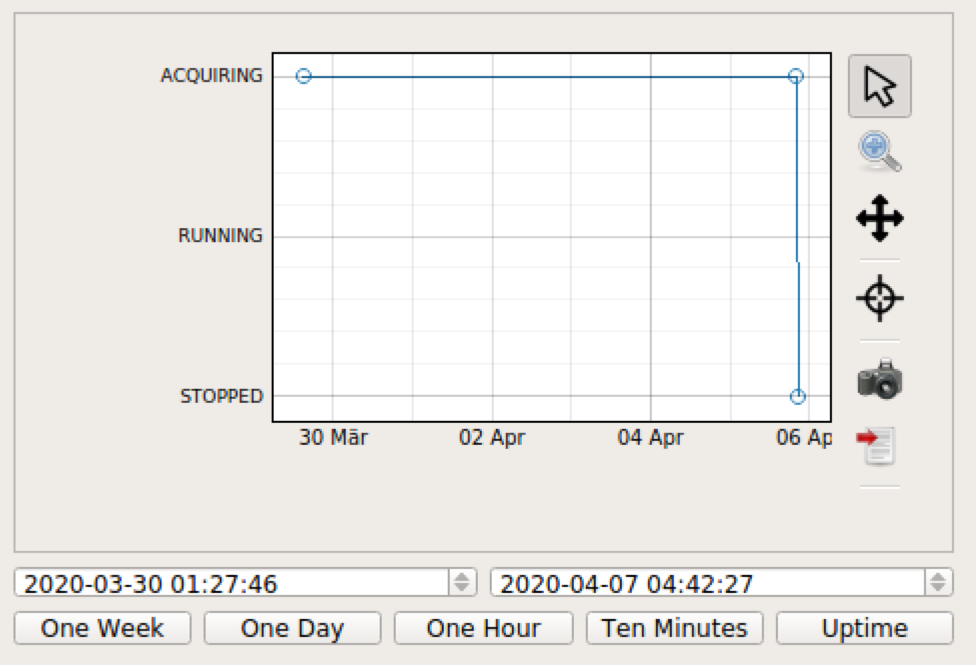
The multi-curve graph is the widget to choose if multiple properties have to be plotted with respect to a reference (x).

The property that is mutated for this controller is referred to as the reference property. Whenever this property is externally updated, the graph will add another measurement point. A typical use case would be a scan of multiple properties with respect to a reference. The toolbar of this widget is equipped with a **reset** button which can be externally armed with a boolean property.



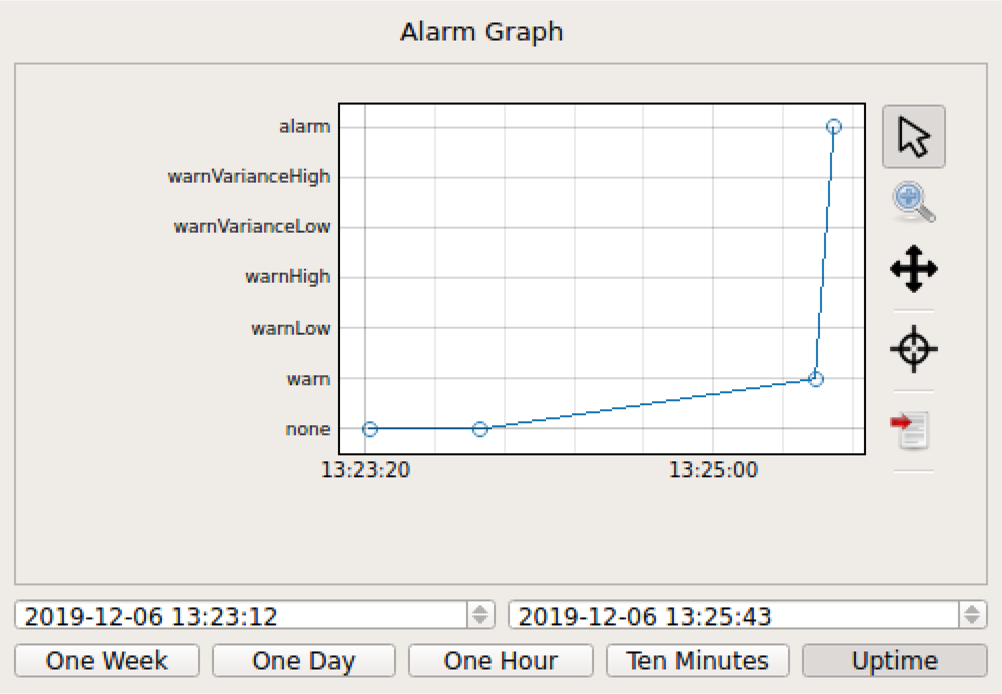
2019-12-06 13:19:56

2019-12-06 13:27:09



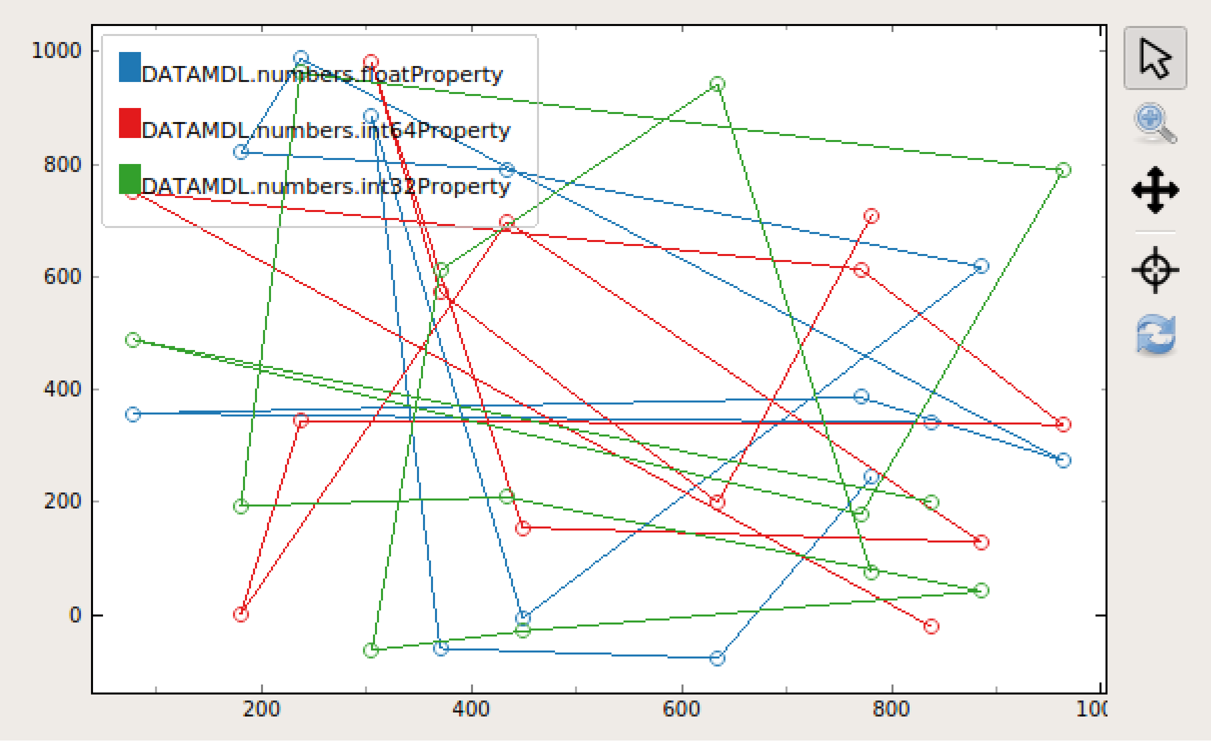
2020-03-30 01:27:46

2020-04-07 04:42:27



2019-12-06 13:23:12

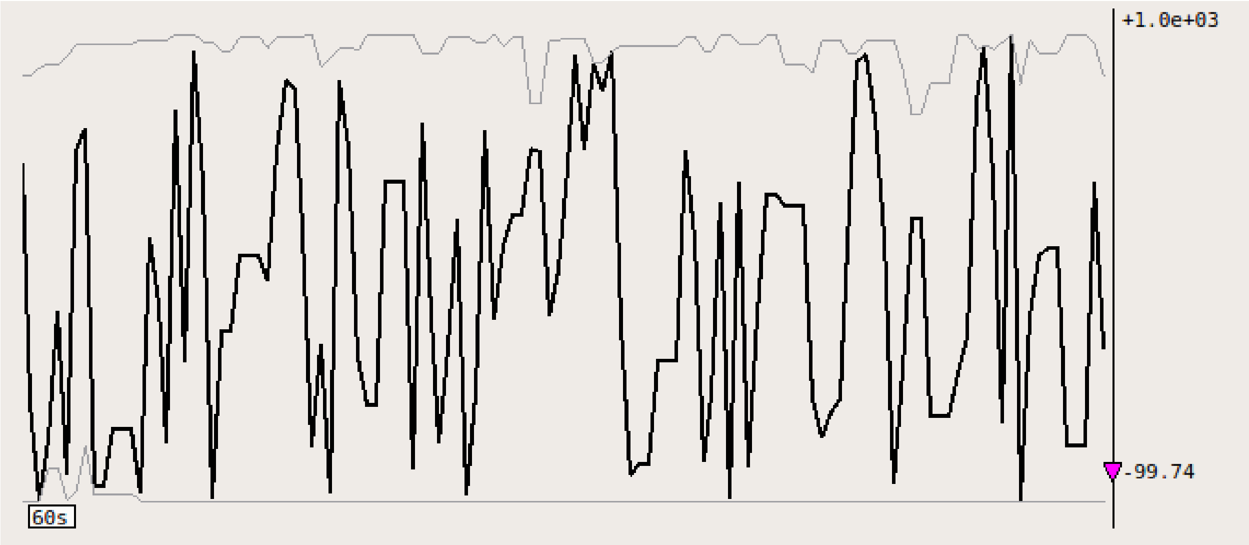
2019-12-06 13:25:43


1000
800
600
400
200
0
200
400
600
800
100
🟦 DATAMDL.numbers.floatProperty
🟥 DATAMDL.numbers.int64Property
🟩 DATAMDL.numbers.int32Property

7.4.3 Sparkline

The sparkline is similar to a trend graph and shows changes of a property over the selected time base.

- The timebases can be chosen to be either 60 seconds, 10 minutes or 60 minutes
- The sparkline maximum number of data points is 120 in an average fashion for all time bases
- The sparkline will request historic data at the beginning
- A change indicator arrow will provide the direction in case of changes larger than 5%
- The minimum and maximum value of the bins are plotted in addition



In this widget it is possible to disable or enable the alarm ranges via the context menu.

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search