Scantool

Release 1.0

Controls

Nov 12, 2024

CONTENTS

Contents:

CHAPTER

ONE

INTRODUCTION

The karabo scantool (Karabacon) is a device that can be controlled via a command line interface (CLI), so-called ScanMacro's or via the graphical user interface (GUI) to perform scans.

For this reason, the scan device is able to carry the whole device environment with aliases in so-called **TableElements**, which should be configured before instantiation and stored in project files. The explanation of the device types are provided in the *General* section.

In addition, recording scans relies on a pre-configured the DAQ environment and the use of a DaqController proxy device. The brief introduction is given in section *How to setup the DAQ*.

1.1 Responsibilities and Limitations

The Karabacon scantool is designed to perform scan-related tasks (motion). This helps to ensure stable operation of the primary functions. These are:

- Controlling motors
- Controlling triggers
- Controlling the DAQ
- Provide basic on-line preview

  Therefore it has known limitations in non-essential tasks, including:

- Online visualization is for now limited to scalar data. For other data types, currently a processor has to be used.
- The scantool does not carry extensive data analysis features. Complex analysis have to go into a separate device.
- The scantool cannot make decisions. It cannot protect you or your equipment against bad configurations. Such functionality can be achieved by limits, interlocks and specialized devices. **However**, it is possible to control the scantool Karabacon via a **macro** and use the **pauseEachStep** feature to use filters and do necessary actions from software.
- The Karabacon is not handling trainId synchronization. This should be done during online/offline data analysis from trainId's.
- High accuracy synchronization to start acquisition of detector devices (triggers) requires dedicated hardware. This is in most cases not available at the moment (Master Trigger).

Karabacon's functionality can be extended with separate devices. We provide multiple interfaces for such extensions.

1.1.1 Troubleshooting

The scan device's status displays error messages to guide through the process of configuring and troubleshooting. Some of the most common messages are related to configuring the scan environment, including:

- Failed to set up the DAQ: General DAQ troubleshooting, probably one of the selected devices was not instantiated, or there are no selected devices.
- Motors not ready for scan: Any selected motor is not instantiated or a motor state is not ACTIVE or ON.
- Data sources not ready for scan: Any selected data source device is not alive (instantiated) or does not have the selected key.
- Trigger sources not ready for scan: Any selected trigger device is not alive (instantiated) or in a wrong state.
- No valid motor axis, sources selected …: The entered axis is not supported by the scan device or the device or property does not exist. The latter can be a typo!
- List of motors and scan points must be of the same length: You need to configure the correct number of motors for your scan.

1.2 Tips and tricks

In principle, Karabacon is capable of ~5 Hz stepping speed, depending on the motor. Yet some of the default settings slow this down to be more in-line with general user requests. These include:

- Reduce pipeline waiting time: In case of pushing speed, the default 1.5 seconds overhead should be reduced. Longer waiting times provide better synchronization for slow processors.
- Averaging normalization: While proper synchronization would require a trainmatcher device, basic synchronization can be achieved via a running average processor.

CHAPTER TWO

GENERAL INFORMATION

2.1 Supported devices

The scantool distinguishes three different device types: **Triggers**, **Data Sources** and **Motors**. These devices either have a standardized interface, or the support of the device is explicitly integrated in an alias mapping within the scantool.

The motor and data source devices can have different axes and sources. An example for a multiple axis system is a slitsystem inheriting five (5) axis - default, x, y, gapx, gapy. A default axis is always provided, in case of the slitsytem the default is the x axis.

The following devices are currently integrated.

2.1.1 Motors

Following device classes as motor interfaces are supported:

Motors	axis name
BeckhoffMC2Base	default
BeckhoffMC2Beckhoff	default
BeckhoffMC2Technosoft	default
BeckhoffMC2Elmo	default
BeckhoffMC2Hexapod	default
BeckhoffSimpleMotor	default
DoocsPhaseshifter	default
CrystalManipulator	default
CrystalManipulatorGroup	default
DoocsOpticalDelay	default
DoocsPhaseshifter	default
DoocsUndulatorEnergy	default
JJAttenuator	default
MonoChromator	default
MonoChromatorGroup	default
MonoChromatorEnergyChanger	default
OpticalDelay	default
PicoMotor	default
PpLaserDelayCopy	default
RobotManipulatorStaubli	default
SlitSystem	default, x, y, gapx, gapy
SoftMono	default
TangoMotor	default
X2TimerML	default

If the device to be scanned is not in the list of supported devic classes then it might be integrated via abstract motor interface.

Abstract Motor Interface

A device might have *interfaces* property that defines interfaces it is compatbile with. If the interface property contains *Motors* then first the *Standard motor with limits* interface with following mandatory properties:

- *actualPosition* (DOUBLE/FLOAT)
- *targetPosition* (DOUBLE/FLOAT)
- *isCWLimit* (BOOL)
- *isCCWLimit* (BOOL)
- *isSWLimitHigh* (BOOL)
- *isSWLimitLow* (BOOL)

and commands

- *move*
- *stop*

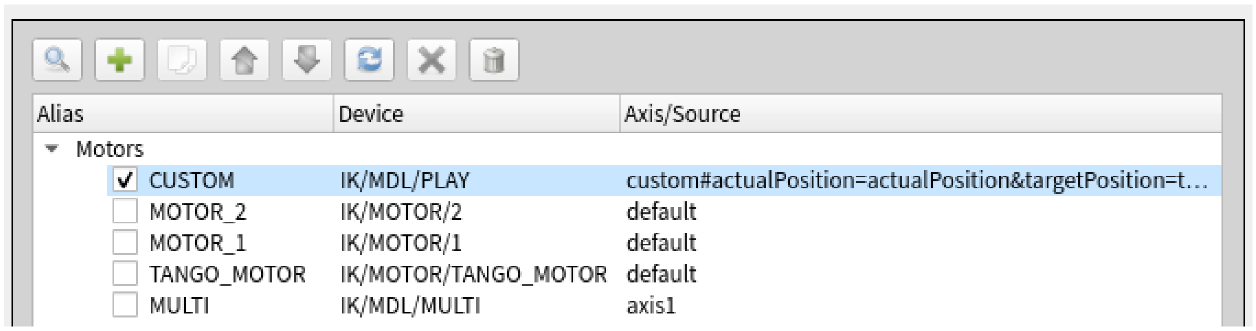
is validated. If one or several properties or commands are missing then the device is validated against the *Standard motor without limits* interface. Such a device should contain the *actualPosition*, and *targetPosition* properties and *move*, *stop* commands.

In general, it is expected that a motor device has the state `State.ON` when it is ready to move and goes to `State.MOVING` when it is moving towards the target position. If a motor is following the target position, it is expected to use `State.ACTIVE` instead of `State.ON`.

Custom Motor Interface

If the requested scan device is not supported with the abstract motor interface, then there is a possibility to use custom a motor interface. The device has to be added as a motor with custom axis definition starting with "custom#":

- Mandatory attributes are `actualPosition` and `targetPosition`: `custom#actualPosition=someReadProperty&targetPo`
- Other attributes `isCWLimit`, `isCCWLimit`, `isSWLimitHigh`, `isSWLimitLow`, `move`, `stop` and `state` are not mandatory.
- Example of custom motor having move and stop slots and soft-limit switches: `custom#actualPosition=actual&targetPosition=target&move=moveSlot&stop=stopSlot&isSWLimitLow=limitLo`



Alias	Device	Axis/Source
Motors		
✓ CUSTOM	IK/MDL/PLAY	custom#actualPosition=actualPosition&targetPosition=t...
MOTOR_2	IK/MOTOR/2	default
MOTOR_1	IK/MOTOR/1	default
TANGO_MOTOR	IK/MOTOR/TANGO_MOTOR	default
MULTI	IK/MDL/MULTI	axis1

Fig. 2.1: Custom motor definition in the device tree.

2.1.2 Data sources

Data sources are considered to represent a single point of data on device level. Any scalar attribute of any karabo device can be added as a data source. Therefore there is no default value anymore. An internal mechanism considers if the data source is a pipeline value or not. If the data source depends on the pipeline processing, the data source is awaiting for a small amount of time (~second) a new value. It is possible to use vector data source. Currently *ascan* and *dscan* in *Single* acquisition mode with one vector data source is supported. Scan is plotted as a mesh scan.

Warning: The "Data Sources" table **does not replace the DAQ**. It should be used only for online preview and alignment. Data should be recorded by the properly configured DAQ.

2.1.3 Triggers

Triggers are a key element of the scantool. They trigger the data generation for the data sources. In the scantool the acquisition of all trigger devices is done in a synchronous way by **software**.

The scantool can only perform basic trigger configuration. Specialized setup must be done manually. The operator must also be familiar with the particular trigger device: some devices can take minutes to configure, others carry significant overhead or synchroniztion issues. The online preview data is collected after the ACQUISITION of all data triggers has completed. Supported devices are:

- LPDMainControl
- GotthardDetector
- JungfrauDetector
- AdqDigitizer

- GreatEyes
- SimulatedTrigger
- AgipdComposite
- PulsePickerTrigger
- DoocsPulseKicker

The trigger devices (device developers) are responsible to provide

- A **start** acquisition slot
- An **acquisition time** property (FLOAT) which configures the trigger

A common state behavior for a trigger device is:

- State.PASSIVE (Derived): Trigger is NOT ready for data acquisition
- State.ACTIVE (Derived): Trigger IS ready for data acquisition
- State.ACQUIRING (Derived): Trigger is acquiring data

Warning: CAMERAS are not supported as trigger sources as they don't tell us when the acquisition is done. Just make sure the camera is ACQUIRING data.

Note: Unfortunately AGIPD cannot be supported at the moment as this device does not provide any information if it is ACQUIRING data or not.

Note: Depending on the type, scan does not always require motors, data sources and triggers.

Scan devices are stored in *deviceEnv.motors*, *deviceEnv.sources* and *deviceEnv.triggers* tables and can be edited via device tree available in the device scene.

2.2 Scans

The scantool provides following scans:

Name	Description
ascan	Linear step scan with up to 4 motors. Uses **absolute** coordinates and stays at last position.
dscan	Linear step scan with up to 4 motors. Uses **relative** coordinates and returns to start.
cscan	N dimensional fly scan based on **absolute** coordinates. These are start and go fly scans with optional velocity setting. If the velocity control is set, motor velocities will be set based on the exposure time. Most motors do not support velocity setting and others have severe restrictions (also in hardware).
tscan	Scan without the motor movement. For defined time acquires data from sources.
mesh	2D scan based on **absolute** coordinates.
dmesh	2D scan based on **relative** coordinates.
custom	Step scan uses absolute coordinates defined in the *scanEnv.customScanPattern*.
external	Linear step scan with up to 4 motors. Uses externaly retrieved scan pattern

- At the moment, scans with up to 4 motors and 6 data sources are available.
- Based on the scan type start and stop positions of motors and the number of scan steps has to be provided.
- After moving motors to the start positions, they are locked by the scantool and unlocked at the end of the scan or when the scan is paused.
- For details on how to use custom scans see: *Custom Scan Patterns*.

2.3 Acquisition Modes

For step scans **acquistionMode** property can be used to adjust the data acquisition mode:

- **Single**: Acquire data once after the acquisition time.
- **Continuous**: Continuous data acquisition during the acquisition time, excluding motor movement.
- **Continuous Averaged**: Average acquired data over the acquisition time.
- **Continuous Extended**: Continuous data acquisition during the acquisition time and motor movements.

Property is available as a combo box in the main scene.

2.4 Device Related Settings

Name	Attribute	Default	Description
Use DAQ	deviceEnv.useDaq	False	Use DAQ
Acquisition Time	deviceEnv.acquisitionTime	0.1 sec	Vector of acquisition times per step
Mottor settling time	deviceEnv.motorSettlingTime	0.0 sec	After motor movement wait this time before further action
Configure Triggers	deviceEnv.configureTriggers	False	True: Triggers will configure the acquisition time
Configure Velocity	deviceEnv.configureVelocity	False	Control motor velocity in cscan based on the acquisition time
Pipeline Wait	deviceEnv.pipelineWait	1.5	Time for the pipeline data sources to wait for a new value
Vector Data range	deviceEnv.vectorDataRange	0	List of one or two integers define slice of vector data

Note:

- Acquisition time is taken into account if the *configureTriggers* is set to True.
- It is a vector of floats and should contain at least one value.
- If a single value is defined then this acquisition time will be used for all steps.
- Number of acquisition time values in the vector should be equal to the number of scan steps +1.
- If the number of values in the vector is less then number of steps + 1,

then missing values are equal to the last value of the vector.

2.5 Scan Environment Settings

Name	Attribute	Default	Description
Comment	scanEnv.comment	“”	Free form text to describe the executed scan.
Bidirectional	scanEnv.bidirectional	False	Enable bidirectional (snake) mesh scan

2.6 Stopping Policy

The XFEL scantool comes with a defined stopping policy

- Absolute scans: The motors stay a final position.
- Relative scans: The motors will move back to start positions.
- Stopping scan: see absolute and relative scan behavior.
- Aborting scan: If a motor experiences an unexpected behavior during scan (e.g .error state), the scan will abort and the motors won’t be moved any further.

Note: A scan can be manually aborted and subsequently any post scan action (move back to start position) is not executed.

CHAPTER

THREE

HOW TO SETUP THE DAQ

The scantool is designed to control the DAQ system via a pre-configured DaqController device. This means it can start and stop a run for a single scan as well as commit the currently selected DAQ configuration groups. Due to collisions, Karabacon will not change the DAQ configuration!

ATTENTION: ALL the groups for your motors and data sources must be selected. Failing to do so means that your data will not be recorded.

The DaqController can control a pre-configures DAQ. In case of DAQ configuration problems please proceed with troubleshooting the DAQ manually.

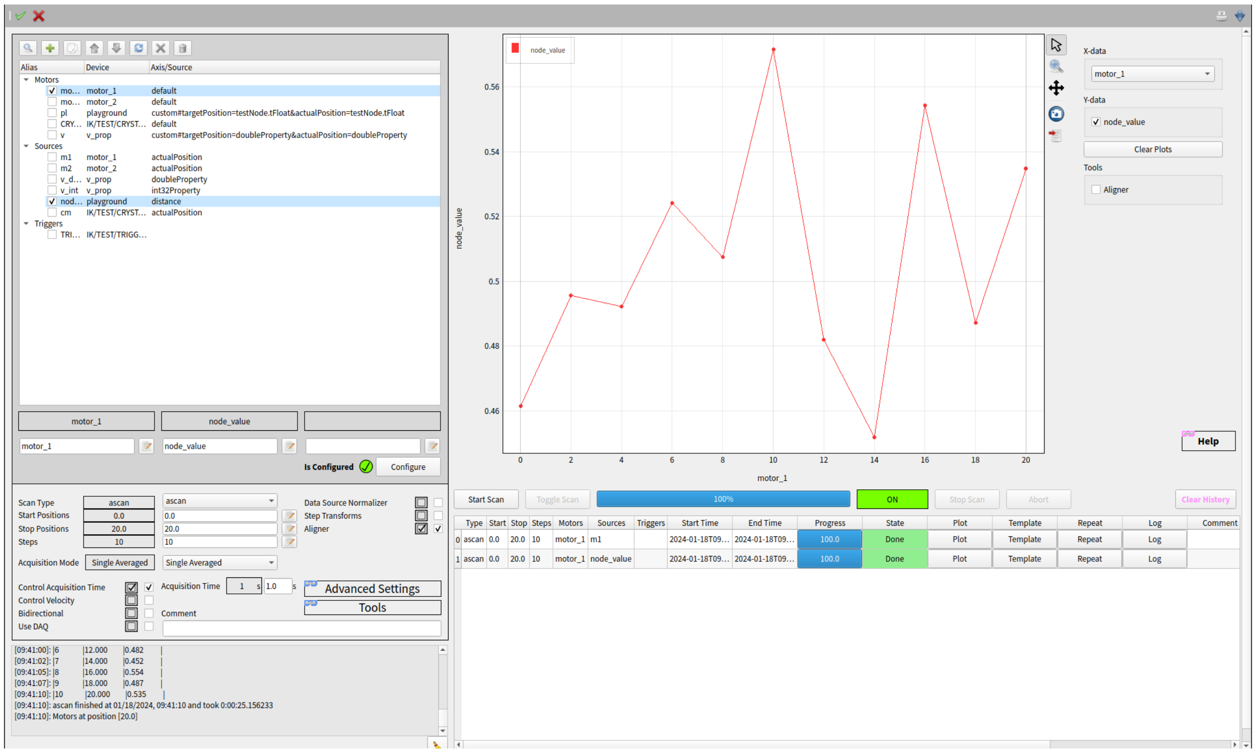
Note: If a DAQ integration is not desired, the **useDAQ** boolean in the karabacon can be set to *False* before instantiation!

Note: The DAQ configuration must be done always **before** the scan configuration if activated. The general system must be in a recordable state.

CHAPTER

FOUR

THE GRAPHICAL USER INTERFACE (GUI)



Alias	Device	Axis/Source
Motors		
✓ mo…	motor_1	default
mo…	motor_2	default
pl	playground	custom#targetPosition=testNode.tFloat&actualPosition=testNode.tFloat
CRY…	IK/TEST/CRYST…	default
v	v_prop	custom#targetPosition=doubleProperty&actualPosition=doubleProperty
Sources		
m1	motor_1	actualPosition
m2	motor_2	actualPosition
v_d…	v_prop	doubleProperty
v_int	v_prop	int32Property
✓ nod…	playground	distance
cm	IK/TEST/CRYST…	actualPosition
Triggers		
TRI…	IK/TEST/TRIGG…	

motor_1 | node_value

motor_1 | node_value

Is Configured ✓

Scan Type	ascan	ascan
Start Positions	0.0	0.0
Stop Positions	20.0	20.0
Steps	10	10
Acquisition Mode	Single Averaged	Single Averaged

Data Source Normalizer ☐ ☐
Step Transforms ☐ ☐
Aligner ☑ ✓

Control Acquisition Time ☑ ✓
Control Velocity ☐ ☐
Bidirectional ☐ ☐
Use DAQ ☐ ☐

Acquisition Time 1 s 1.0 s

Comment

```
[09:41:00]: |6      |12.000    |0.482    |
[09:41:02]: |7      |14.000    |0.452    |
[09:41:05]: |8      |16.000    |0.554    |
[09:41:07]: |9      |18.000    |0.487    |
[09:41:10]: |10     |20.000    |0.535    |
[09:41:10]: ascan finished at 01/18/2024, 09:41:10 and took 0:00:25.156233
[09:41:10]: Motors at position [20.0]
```

	Type	Start	Stop	Steps	Motors	Sources	Triggers	Start Time	End Time	Progress	State	Plot	Template	Repeat	Log	Comment
0	ascan	0.0	20.0	10	motor_1	m1		2024-01-18T09…	2024-01-18T09…	100.0	Done	Plot	Template	Repeat	Log	
1	ascan	0.0	20.0	10	motor_1	node_value		2024-01-18T09…	2024-01-18T09…	100.0	Done	Plot	Template	Repeat	Log	

Fig. 4.1: Main Karabacon scene.

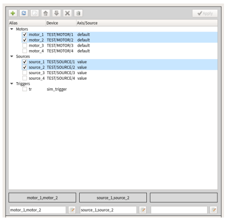
Karabacon scene is devided in several areas:

- Device environment
- Scan settings
- Scan control (start, toggle scan, stop and abort), history table and logging
- Real-time and historical plots

4.1 Device environment

The device tree represents the scan tool device environment. A new device can be added by dragging a device name from the system topology or project view to the tree view. If the device name is dropped on the top level item **Motors**, **Sources** or **Triggers** then a new children entry is added. If it is dropped on the device item then the deviceId is updated.

Note: Device alias has to be unique and can contain letters `a-z, A-Z`, numbers: `0-9`, and special characters: `.-:/_`. White spaces are not allowed. Setting duplicated aliases will set the scantool to error state.



Apply

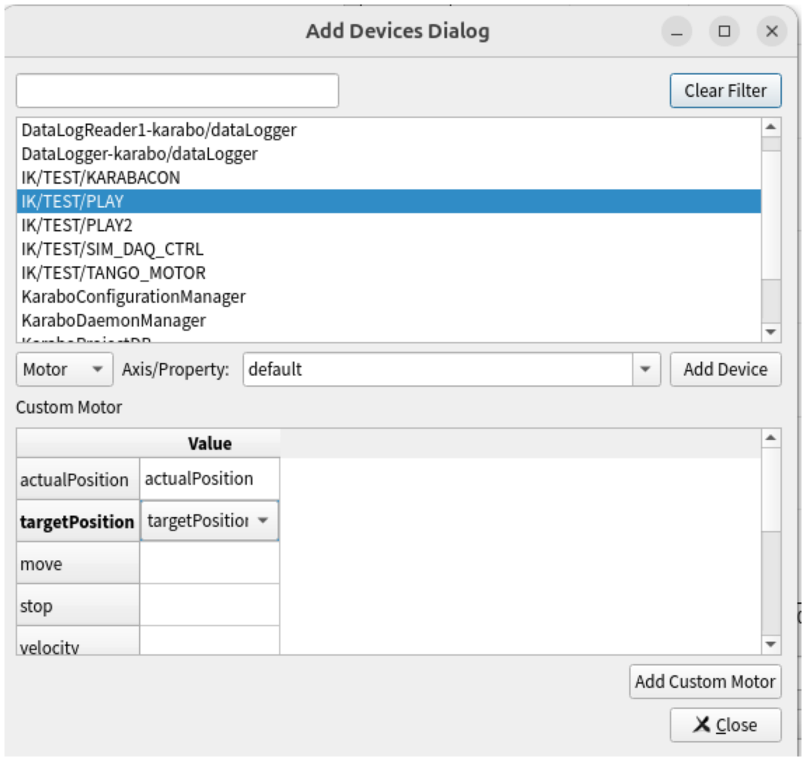
Alias	Device	Axis/Source
Motors		
✓ motor_1	TEST/MOTOR/1	default
✓ motor_2	TEST/MOTOR/2	default
motor_3	TEST/MOTOR/3	default
motor_4	TEST/MOTOR/4	default
Sources		
✓ source_1	TEST/SOURCE/1	value
✓ source_2	TEST/SOURCE/2	value
source_3	TEST/SOURCE/3	value
source_4	TEST/SOURCE/4	value
Triggers		
tr	sim_trigger	

motor_1,motor_2 | source_1,source_2

motor_1,motor_2 | source_1,source_2

Fig. 4.2: Device environment tree and widgets to select device aliases.

After changing the device tree one has to click **Apply** button to approve the changes. Above the tree a toolbar with buttons to add, sort, copy, move and delete devices is available. By clicking on the **Add new devices** a dialog to add new devices is shown. The dialog lists all devices from the topology, allows filtering by text and allows to add devices to the selected device group.



Add Devices Dialog

Clear Filter

DataLogReader1-karabo/dataLogger
DataLogger-karabo/dataLogger
IK/TEST/KARABACON
IK/TEST/PLAY
IK/TEST/PLAY2
IK/TEST/SIM_DAQ_CTRL
IK/TEST/TANGO_MOTOR
KaraboConfigurationManager
KaraboDaemonManager

Motor | Axis/Property: default | Add Device

Custom Motor

	Value
actualPosition	actualPosition
targetPosition	targetPositio
move	
stop	
velocity	

Add Custom Motor

Close

Fig. 4.3: Dialog allowing to add new devices to the device environment.

The dialog contains the following GUI components:

- List with available Karabo devices.

- Filter text field to filter available devices.
- Combo box to select device type and *Axis/Property* field to define the motor axis or source property.
- Table to define new custom motor interface.

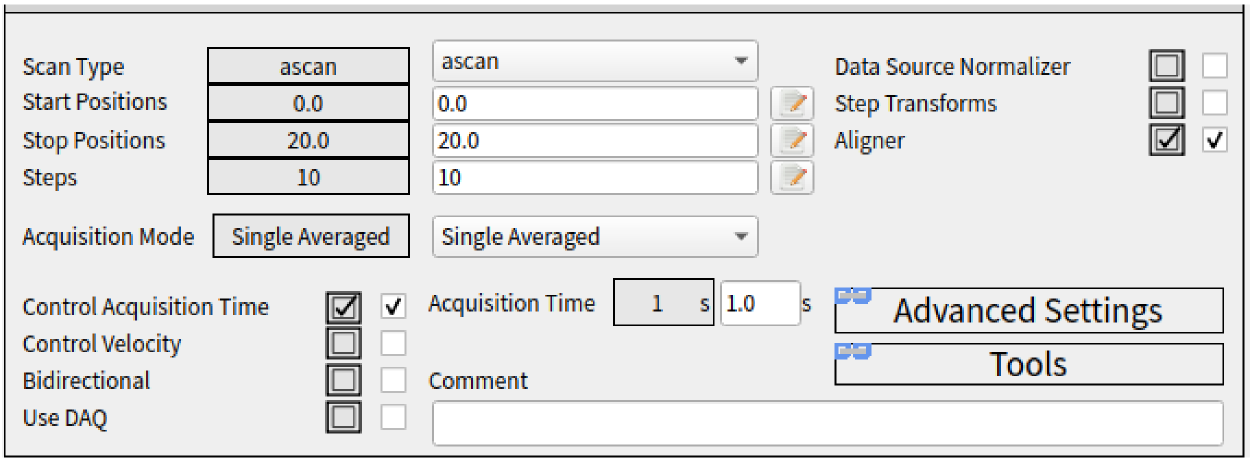
To add a new device, select the device type (*Motor*, *Source*, or *Trigger*), then either double-click the device ID or click the Add Device button. To add a custom motor, select the device ID, complete the necessary table fields, and click the Add Custom Motor button

Tip: For a quick device selection type the requested aliases (comma separated and without whitespaces) in the text fields of the currently selected aliases.

After all devices have been selected the device environment can be configured. Please click the **Configure** button and the karabacon will establish the connection to all devices, check if they are in a proper condition and have been configured properly with a correct axis. You only need to re-configure if you changed the device selection. Please note, that configuring the DAQ might take over a minute!

4.2 Scan settings

If the configuration of the scan environment was successful, the **isConfigured** boolean is set to *True* and the applied configuration is displayed in the status box. If the configuration is not successul, the status box will briefly describe what happened wrong. After successful configuration the scans can be launched with the selected parameters and settings.



Scan Type	ascan	ascan
Start Positions	0.0	0.0
Stop Positions	20.0	20.0
Steps	10	10
Acquisition Mode	Single Averaged	Single Averaged
Control Acquisition Time	✓	✓
Control Velocity	☐	☐
Bidirectional	☐	☐
Use DAQ	☐	☐
Acquisition Time	1 s	1.0 s
Data Source Normalizer	☐	☐
Step Transforms	☐	☐
Aligner	✓	✓

Comment

Fig. 4.4: Scan settings.

Note: The Scan Environment has to be **always configured** when new devices are selected for the scan.

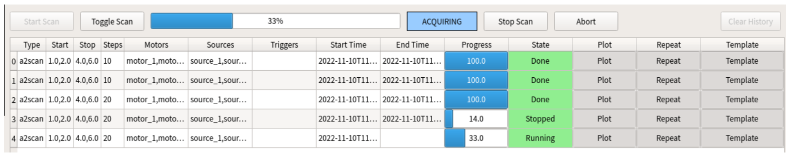
Note: The Scan Environment must **not** be **configured** for scan parameters (start, stop, steps etc.) only.

4.3 Scan control and history

The following scan control buttons are available:

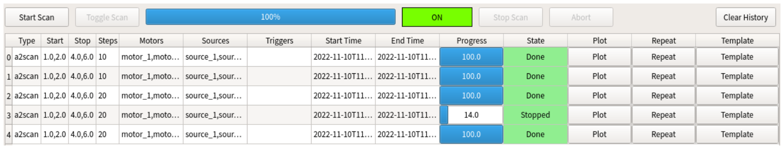
- **Start Scan**: starts the configured scan.
- **Toggle Scan**: pauses or resumes the current scan.
- **Stop Scan**: stops the currently running scan.
- **Abort**: stops the currently running scan. The stop slot of motors and triggers is called.
- **Clear History**: clears the history table.

The history table keeps track of all executed scans. It is possible to plot scan results (**Plot**), repeate a scan (**Repeate**) and configure the scantool based on the historical scan settings (**Template**). During a scan these commands are disabled. The history table has a limited number of rows. You can adjust it by changing **history.cacheSize**.



	Type	Start	Stop	Steps	Motors	Sources	Triggers	Start Time	End Time	Progress	State	Plot	Repeat	Template
0	a2scan	1.0,2.0	4.0,6.0	10	motor_1,moto...	source_1,sour...		2022-11-10T11...	2022-11-10T11...	100.0	Done	Plot	Repeat	Template
1	a2scan	1.0,2.0	4.0,6.0	10	motor_1,moto...	source_1,sour...		2022-11-10T11...	2022-11-10T11...	100.0	Done	Plot	Repeat	Template
2	a2scan	1.0,2.0	4.0,6.0	20	motor_1,moto...	source_1,sour...		2022-11-10T11...	2022-11-10T11...	100.0	Done	Plot	Repeat	Template
3	a2scan	1.0,2.0	4.0,6.0	20	motor_1,moto...	source_1,sour...		2022-11-10T11...	2022-11-10T11...	14.0	Stopped	Plot	Repeat	Template
4	a2scan	1.0,2.0	4.0,6.0	20	motor_1,moto...	source_1,sour...		2022-11-10T11...		33.0	Running	Plot	Repeat	Template

Fig. 4.5: The scan history table with a scan being executed.



	Type	Start	Stop	Steps	Motors	Sources	Triggers	Start Time	End Time	Progress	State	Plot	Repeat	Template
0	a2scan	1.0,2.0	4.0,6.0	10	motor_1,moto...	source_1,sour...		2022-11-10T11...	2022-11-10T11...	100.0	Done	Plot	Repeat	Template
1	a2scan	1.0,2.0	4.0,6.0	10	motor_1,moto...	source_1,sour...		2022-11-10T11...	2022-11-10T11...	100.0	Done	Plot	Repeat	Template
2	a2scan	1.0,2.0	4.0,6.0	20	motor_1,moto...	source_1,sour...		2022-11-10T11...	2022-11-10T11...	100.0	Done	Plot	Repeat	Template
3	a2scan	1.0,2.0	4.0,6.0	20	motor_1,moto...	source_1,sour...		2022-11-10T11...	2022-11-10T11...	14.0	Stopped	Plot	Repeat	Template
4	a2scan	1.0,2.0	4.0,6.0	20	motor_1,moto...	source_1,sour...		2022-11-10T11...	2022-11-10T11...	100.0	Done	Plot	Repeat	Template

Fig. 4.6: Scan history with executed scans.

4.4 Real time and historical plots

The Karabo scan widget is a custom plot widget, for dedicated use with the scantool. Most importantly, it detects 1D and 2D scans and adjusts it's layout accordingly. It also offers basic analysis features, like selecting between multiple datasource c hannel(s) and displayed axis coordinates.

Tip: Double clicking on the plot will move motor(s) to the corresponding position(s).

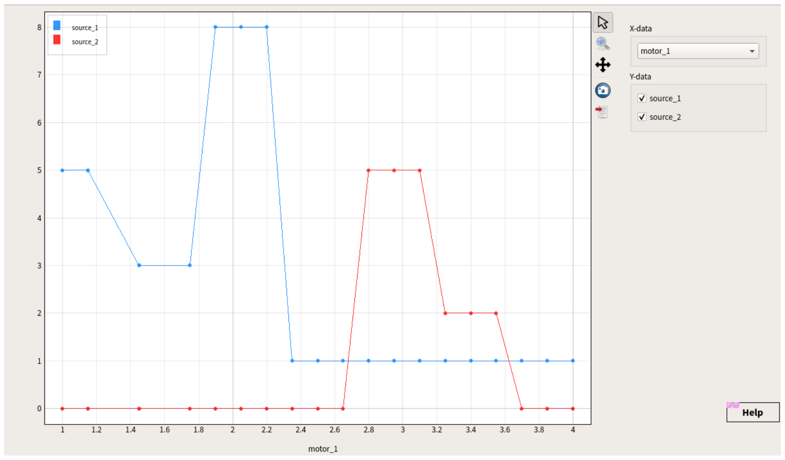


Fig. 4.7: One dimensional plot.

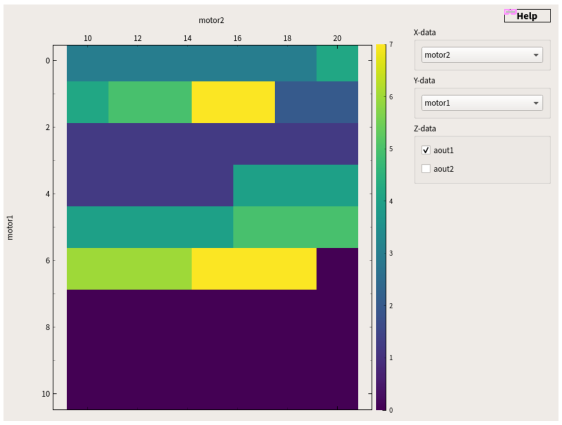
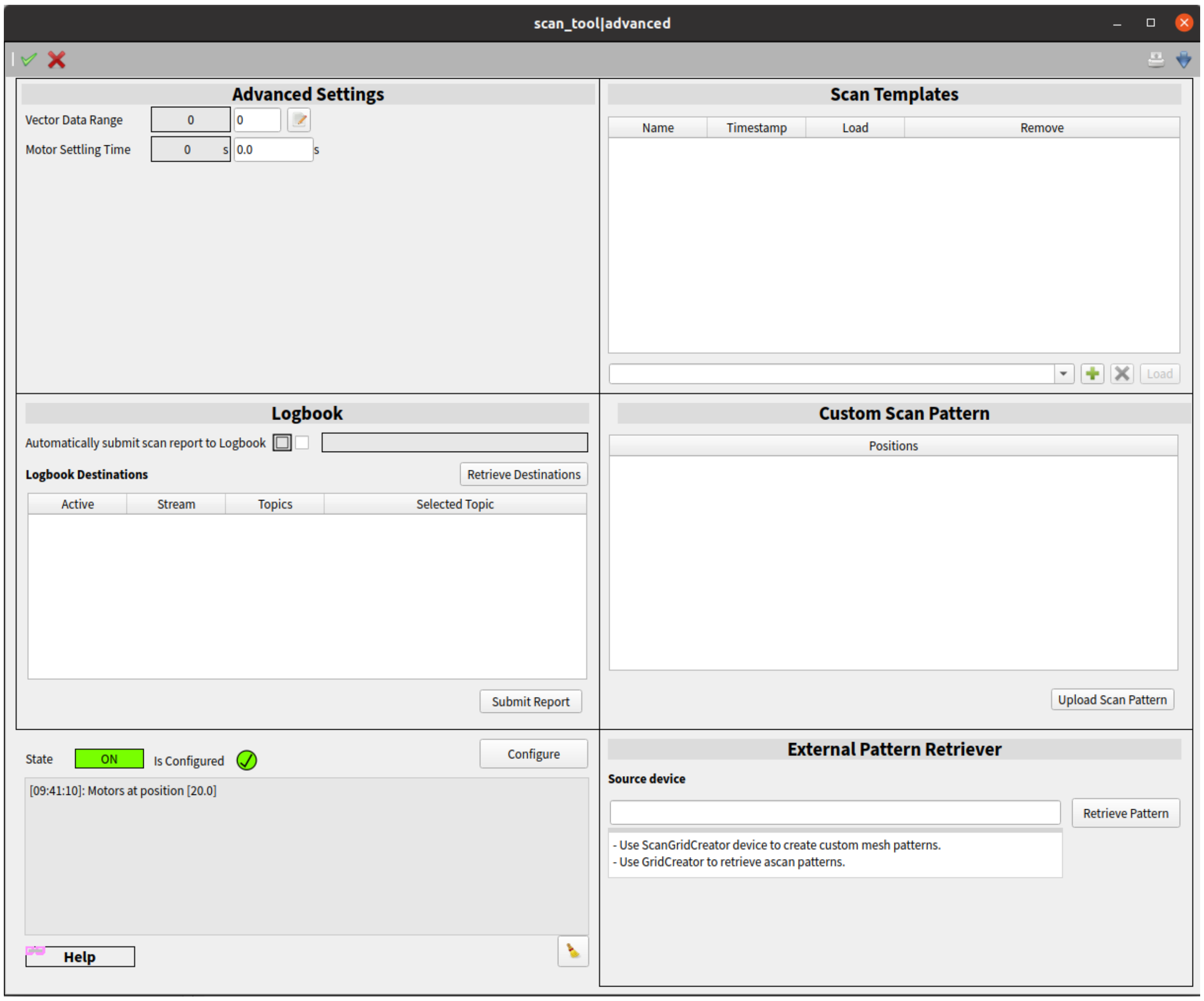


Fig. 4.8: Two dimensional plot.

CHAPTER

FIVE

ADVANCED SETTINGS AND TOOLS



[09:41:10]: Motors at position [20.0]

- Use ScanGridCreator device to create custom mesh patterns.
- Use GridCreator to retrieve ascan patterns.

Fig. 5.1: Fig.: Advanced settings and tools.

5.1 Templates

Templates can be used to store and load named scantool configurations. The template widget contains a drop down menu (combobox) with text completion containing all stored configurations, and buttons to store, remove, and load a template. Each template is associated with a timestamp and loading a template requests a stored scantool configuration at the associated timestamp. Pressing "Add" to an existing template overwrites its timestamp.

5.2 Logbook

Scantool allows sending scan reports to a logbook. In the topics where scantool is deployed a running Karabo *LogBook* device needs to be present. In the Logbook section of the Advanced Setting dialog a table with available logbook destinations is available.

- To retrieve destinations press **Retrieve Destinations**. Adding new destinations manually is not allowed and user may enable or disable a destination or select a topic from available topics.
- **Submit report** button will send a report of the last scan to selected logbook destinations.
- By checking **Automatically submit scan report to Logbook** a report after each scan will be submitted.
- Pressing **Log** button in the scan history table will send a report of a corresponding scan.

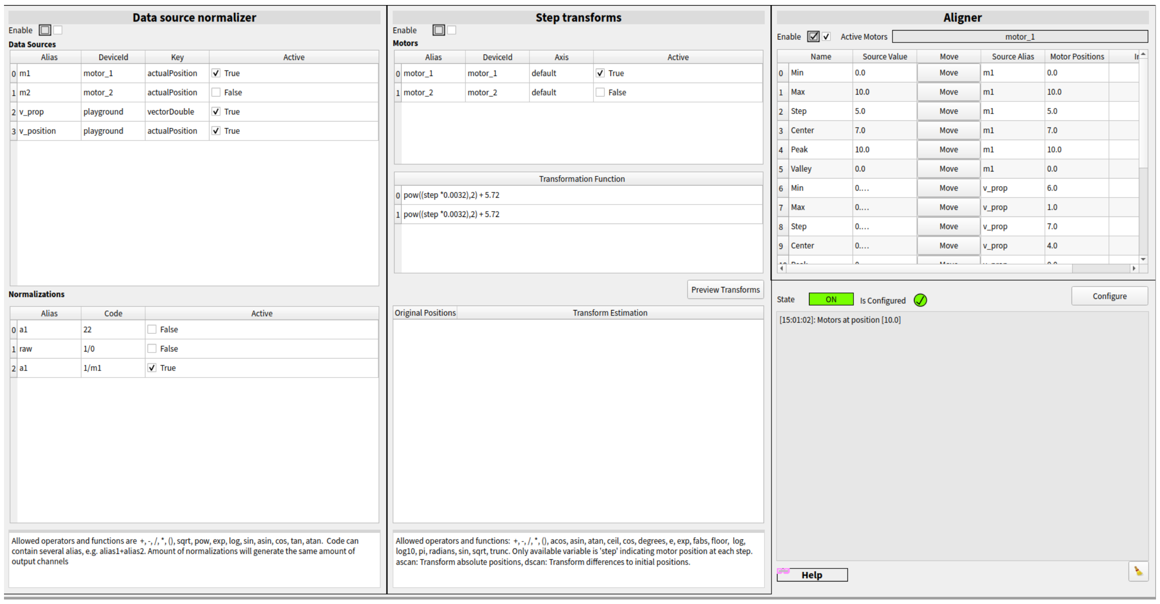
5.3 Custom Scan Patterns

Custom scans are absolute scans based on user defined arbitrary motor positions defined in the *sca-nEnv.customScanPattern* table. Table contains rows with vector double attributes indicating motor positions at each step and can be defined:

- Manually by entering values in the table.
- By uploading a csv or numpy file by pressing *Upload Scan Pattern* button.
- By using a macro. For and example see: *Macro to set custom scan pattern*.

CHAPTER SIX

EXTENSIONS



Data source normalizer

Enable

Data Sources

	Alias	DeviceId	Key	Active
0	m1	motor_1	actualPosition	✓ True
1	m2	motor_2	actualPosition	False
2	v_prop	playground	vectorDouble	✓ True
3	v_position	playground	actualPosition	✓ True

Normalizations

	Alias	Code	Active
0	a1	22	False
1	raw	1/0	False
2	a1	1/m1	✓ True

Allowed operators and functions are +, -, /, *, (), sqrt, pow, exp, log, sin, asin, cos, tan, atan. Code can contain several alias, e.g. alias1+alias2. Amount of normalizations will generate the same amount of output channels

Step transforms

Enable

Motors

	Alias	DeviceId	Axis	Active
0	motor_1	motor_1	default	✓ True
1	motor_2	motor_2	default	False

	Transformation Function
0	pow((step *0.0032),2) + 5.72
1	pow((step *0.0032),2) + 5.72

Original Positions	Transform Estimation

Allowed operators and functions: +, -, /, *, (), acos, asin, atan, ceil, cos, degrees, e, exp, fabs, floor, log, log10, pi, radians, sin, sqrt, trunc. Only available variable is 'step' indicating motor position at each step. ascan: Transform absolute positions, dscan: Transform differences to initial positions.

Aligner

Enable ✓ Active Motors motor_1

	Name	Source Value	Move	Source Alias	Motor Positions
0	Min	0.0	Move	m1	0.0
1	Max	10.0	Move	m1	10.0
2	Step	5.0	Move	m1	5.0
3	Center	7.0	Move	m1	7.0
4	Peak	10.0	Move	m1	10.0
5	Valley	0.0	Move	m1	0.0
6	Min	0....	Move	v_prop	6.0
7	Max	0....	Move	v_prop	1.0
8	Step	0....	Move	v_prop	7.0
9	Center	0....	Move	v_prop	4.0

State ON Is Configured ✓

```
[15:01:02]: Motors at position [10.0]
```

Fig. 6.1: Fig.: Extension and additional tool widget in karabacon scene.

6.1 Data source normalizer

Extension is used to normalize data source values.

How to use the normalization

- Activate necessary data sources.
- Add normalization code.
- Activate necessary normalizations and configure Scantool.
- Start the scan.

Normalization coding guidlines

- Arithmetical operators: `+`, `-`, `*`, `/`, `()` and functions: `sqrt`, `pow`, `exp`, `log`, `sin`, `asin`, `cos`, `tan`, `atan` are allowed.
- Do not use equal sign.

- Several alias are allowed in a single normalization code. For example, `(source_alias1 + source_alias2) / 2` will average source values.
- Nested normalizations are not allowed.

Note:

- The amount of active normalizations will fill the same amount of output channels. For example, 3 active normalizations will fill 3 output channels with normalized values.
- Configure the scan tool after changing the normalization environment.

6.2 Motor step transforms

Transfoms allowes to transform motor step values.

How to use the transforms

- Activate necessary motors.
- Add transforms code and configure the scan tool.
- If necessary preview the estimated positions.
- Start the scan.

Note:

- Configure the scan tool after changing the transforms environment.

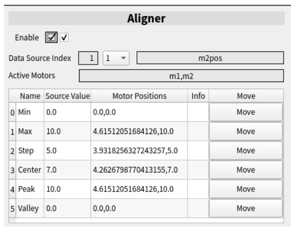
6.3 Aligner

Extension allows to move motors to the positions that corresponds to the data source value at a certain characteristic:

- Minimum value
- Maximum value
- Center of mass
- Peak using L2 metric
- Valley: negative peak
- Step: center of gradients

All characteristics are evaluated at the end of the scan and displayed in the table element.

The table lists all characteristics, including data source value, motor position(s), summary information, and a button to move motors. If a characteristic is not estimated then the move button is disabled and an error message is displayed in the info field. To display aligner results on the plot, right click on the plot and select *Request aligner results*.



Enable ✓

Data Source Index 1 1 m2pos

Active Motors m1,m2

	Name	Source Value	Motor Positions	Info	Move
0	Min	0.0	0.0,0.0		Move
1	Max	10.0	4.61512051684126,10.0		Move
2	Step	5.0	3.9318256327243257,5.0		Move
3	Center	7.0	4.2626798770413155,7.0		Move
4	Peak	10.0	4.61512051684126,10.0		Move
5	Valley	0.0	0.0,0.0		Move

Fig. 6.2: Fig.: Aligner

CHAPTER

SEVEN

THE COMMAND LINE INTERFACE (CLI)

The karabo scan device comes with a dedicated command line interface. Similar to ikarabo an IPython shell is launched. In this python shell a scan client is created and registered on the broker to communicate with the desired scan device.

In order to connect to a scandevice, a *SCANDEVICE* variable has to be defined in the environment of the operating system:

```
export SCANDEVICE=DEVICEID_SCANDEVICE
```

Once the scan device is exported, type **karabacon** and follow the initialization. The scan client is automatically updated during a scan and retrieves the messages from the scanDevice. Hence, the configuration and scan messages in the graphical user interface and the command line are identical.

Note: For any CLI command the brackets don't have to be typed. The karabacon CLI is configured to use the IPython *autocall* option, e.g. to view the data source environment:

```
sources() -> sources
```

In general, the command line interface follows **spec syntax**!

7.1 Configure the scan environment

Before a scan can be launched the scan environment has to be configured. This is required for the scan device to know which triggers and data sources have to be considered (and plotted) in a scan. In addition, this information is provided to the XFEL DAQ system for recording data. The actual configuration can be viewed by:

```
def status():
   """Retrieve the full status of the scan device

   - Current configuration for motors, triggers and data sources
   - State of the scan device
```

In general, the scan device knows the last active configuration and won't perform a new configuration if the settings do not change between scans.

Since the motor environment is most likely the device which change between scans, the motors are configured with the scan function directly. The overview of the motor environment can be viewed by:

```
def motors():
    """Retrieve the full motor environment"""
```

7.1.1 Data sources

Data sources are most likely passive providers of data. The selected sources will be plotted by the scantool. Data sources can be viewed and configured via:

```
def sources():
    """Show the available data sources environment of the scan device
    """

def add_sources(alias):
    """Add desired sources to the scan environment

    :alias: Alias of the sources (string)
    """

def remove_sources(alias):
    """Remove desired sources from the scan environment

    :alias: Alias of the sources (string)
    """
```

Note: The alias for data sources **MUST NOT** be **unique**. Hence multiple data can be configured at once.

7.1.2 Triggers

Triggers are active devices which are responsible for producing data. Triggers can be viewed and selected via:

```
def triggers():
    """Show the available trigger environment
    """

def add_trigger(alias: (str,)):
    """Add a single trigger to the active scan environment

    :alias: Alias of the trigger
    """

def remove_trigger(alias: (str,)):
    """Remove a single trigger from the active scan environment

    :alias: Alias of the trigger
    """
```

Note: The alias for triggers **MUST** be **unique**. The first trigger with found with given alias is taken.

7.2 Launching scans

In general, the motors for a scan are always selected in the scan function. Simply provide a matching alias (string type) of the motor environment. Absolute scans and relative scans are provided and follow the described stopping policy.

The following scan functions are provided in the command line interface

```
def ascan(axis, start, stop, steps, acqtime, wait)
     """Perform an absolute step scan with the scan device

     :axis: alias(es) of the motor axis (str, list or tuple)
     :start: start point(s) of the axis (int, float, list or tuple)
     :stop: stop point(s) of the axis (int, float, list or tuple)
     :steps: number of steps (int)
     :acqtime: acquisition time per step (int, float)
     :wait: Set to True to wait till scan ends

     If the acquisition parameter is 0, the trigger devices
     are not configured.

     NOTE: After the scan, the motors stay a final position.
     """

 def dscan(self, axis, start, stop, steps, acqtime, wait):
     """Perform a relative step scan with the scan device

     :axis: alias(es) of the motor axis (str, list or tuple)
     :start: start point(s) of the axis (int, float, list or tuple)
     :stop: stop point(s) of the axis (int, float, list or tuple)
     :steps: number of steps (int)
     :acqtime: acquisition time per step (int, float)
     :wait: Set to True to wait till scan ends

     If the acquisition parameter is 0, the trigger devices
     are not configured.

     NOTE: After the scan, the motors will move back to start position.
     """

 def cscan(self, axis, start, stop, steps, acqtime, wait):
     """Perform a continuous scan with the scan device

     :axis: alias(es) of the motor axis (str, list or tuple)
     :start: start point(s) of the axis (int, float, list or tuple)
     :stop: stop point(s) of the axis (int, float, list or tuple)
     :acqtime: acquisition time per step (int, float)
     :wait: Set to True to wait till scan ends

     If the acquisition parameter is 0, the trigger devices
     are not configured.

     NOTE: After the scan, the motors will move back to start position.
     """
```

(continues on next page)

(continued from previous page)

```
def tscan(self, acqtime, wait):
    """Perform a time scan with the scan device

    :acqtime: acquisition time
    :wait: Set to True to wait till scan ends
    """


def mesh(self, axis, start, stop, steps, acqtime, bidirectional=False, wait):
    """Perform an absolute mesh step scan with the scan device

    :axis: aliases of the motor axis (list or tuple)
    :start: start points of the axis (list or tuple)
    :stop: stop points of the axis (list or tuple)
    :steps: number of steps (list or tuple)
    :acqtime: acquisition time per step
    :bidirectional: Set to True for a snake scan
    :wait: Set to True to wait till scan ends

    This scan performs a mesh scan - Depending on the setting for
    a *True* bidirectional input, the scan is performed as a snake scan.

    If the acquisition parameter is 0, the trigger devices
    are not configured.

    NOTE: After the scan, the motors stay a final position.
    """


def dmesh(self, axis, start, stop, steps, acqtime, bidirectional=False, wait):
    """Perform a relative dmesh step scan with the scan device

    :axis: aliases of the motor axis (list or tuple)
    :start: start points of the axis (list or tuple)
    :stop: stop points of the axis (list or tuple)
    :steps: number of steps (list or tuple)
    :acqtime: acquisition time per step
    :bidirectional: Set to True for a snake scan
    :wait: Set to True to wait till scan ends

    This scan performs a dmesh scan - Depending on the setting for
    a *True* bidirectional input, the scan is performed as a snake scan.

    If the acquisition parameter is 0, the trigger devices
    are not configured.

    NOTE: After the scan, the motors will move back to start position.
    """
```

7.3 Stopping Scans

Scans can be stopped or aborted. Aborting a scan means, that no motor will move back to start position (valid for relative scans.):

```
def stop():
    """Stop a running scan from the scan device
    """

def abort():
    """Abort a running scan from the scan device
    """
```

CHAPTER

EIGHT

MACROS

Karabo macros allows to extend the scantool functionality. In this chapter various macros to ease the user experience are described.

8.1 Device provided macros

Scantool device, similarly to the device scenes, provides device macros. By right clicking on the device name and selecting **Open device macro** a dialog with available macros is displayed. After choosing a macro name from the *Device Items* list a new macro to the project will be added. This can be foreseen as a macro template and user can adjust it to its needs. Currently following macros are available:

8.1.1 pause_scan_step

Macro allows to perform any available scan and perform actions after each step. Macro sets boolean *pauseEachStep* of *Scan environment* to *True*, starts a scan and in the loop waits till scantool goes into *PAUSE* state. In the pause state user may set attributes or call slots of other karabo devices and resume scan by calling pause slot of scan device.

```
#############################################################################
# Copyright (C) European XFEL GmbH Schenefeld. All rights reserved.
#############################################################################
from karabacon.cli.scan_client import ScanClient
from karabacon.enums import ScanTypes
from karabo.middlelayer import (
    Bool, Double, Overwrite, Slot, State, String, VectorDouble, VectorInt32,
    VectorString, sleep, waitUntil)


class PauseScanStep(ScanClient):
    """This is an example scan to control the scantool
        Use this macro to keep a static scan in your project
    """
    scanDeviceId = Overwrite(
        displayedName="Scantool Device",
        defaultValue="__KARABO_BACON_DEVICE__")

    scanType = String(
        displayedName="Scan Type",
        defaultValue=ScanTypes.ASCAN.value,
        options=[i.value for i in ScanTypes])
```

(continues on next page)

(continued from previous page)

```
    motors = VectorString(
        displayedName="Motors",
        description="Motor aliases to be used during ascan",
        defaultValue=[])

    sources = VectorString(
        displayedName="Sources",
        description="Aliases of the data sources during ascan",
        defaultValue=[])

    triggers = VectorString(
        displayedName="Triggers",
        description="Aliases of the trigger during ascan",
        defaultValue=[])

    startPositions = VectorDouble(
        displayedName="Start positions",
        defaultValue=[0.])

    stopPositions = VectorDouble(
        displayedName="Stop positions",
        defaultValue=[10.])

    steps = VectorInt32(
        displayedName="Steps",
        defaultValue=[10])

    acqtime = Double(
        defaultValue=0.,
        displayedName="Acquisition Time")

    bidirectional = Bool(
        displayedName="Bidirectional",
        defaultValue=False,
        description="Set this value to true for a snake like mesh scan")

    @Slot(displayedName="Start",
        description="Start a scan",
        allowedStates=[State.PASSIVE])
    async def startScan(self):
        """ Execute a scan with the configured properties
            The scan can be cancelled!
        """

        # Connect to the scan device!
        await self.connect()

        if len(self.sources) > 0:
            await self.select_sources(self.sources)
        if len(self.triggers) > 0:
            await self.select_triggers(self.triggers)
```

(continues on next page)

(continued from previous page)

```
        # use self.scan_device to access scantool directly
        self.scan_device.scanEnv.pauseEachStep = True

        # Do some actions before the scan

        # Start a scan
        success, text = await self._start(
            axis=self.motors, start=self.startPositions,
            stop=self.stopPositions, steps=self.steps,
            scantype=self.scanType, acqtime=self.acqtime,
            bidirectional=self.bidirectional)

        # Get the message from the scan device and check if we scan
        print(f"Scan device information: {text}")
        if success:
            # Wait until the scan is done!
            while self._is_scanning:
                await waitUntil(
                    lambda: self.scan_device.state == State.PAUSED)

                # Do something after each step
                print("Do something after each scan step")

                await self.scan_device.pause()

        # Do some actions after the scan

        # Disconnect from the scan device after run!
        self.disconnect()

    @Slot(displayedName="Stop",
        description="Stop scan",
        allowedStates=[State.PASSIVE, State.ACTIVE])
    async def stopAbsoluteScan(self):
        # Call the internal stop method!
        await self.stop()
```

8.2 Other Macros

8.2.1 Macro to set custom scan pattern

```
from karabo.middlelayer import Macro, MacroSlot, Hash, setWait

class SetCustomScanPattern(Macro):

    @MacroSlot()
    def execute(self):
        custom_pattern = []
```

(continues on next page)

(continued from previous page)

```
        # Add a row with two positions
        custom_pattern.append(Hash("positions", [0.0, 0.0]))
        # Add 10 rows
        for row in range(10):
            custom_pattern.append(Hash("positions", [row, row * row]))

        setWait("SCANTOOL/DEVICE/ID", "scanEnv.customScanPattern", custom_pattern)
```

CHAPTER

NINE

KARABACON SCHEDULER

The Karabacon Scheduler is a middlelayer device to schedule and execute queued scans. This device has a scene containing:

- A table to define the scans to be executed.
- Buttons to start and stop the scans in the queue and a button to reset errors.
- Plots.
- A table displaying the queue execution progress.

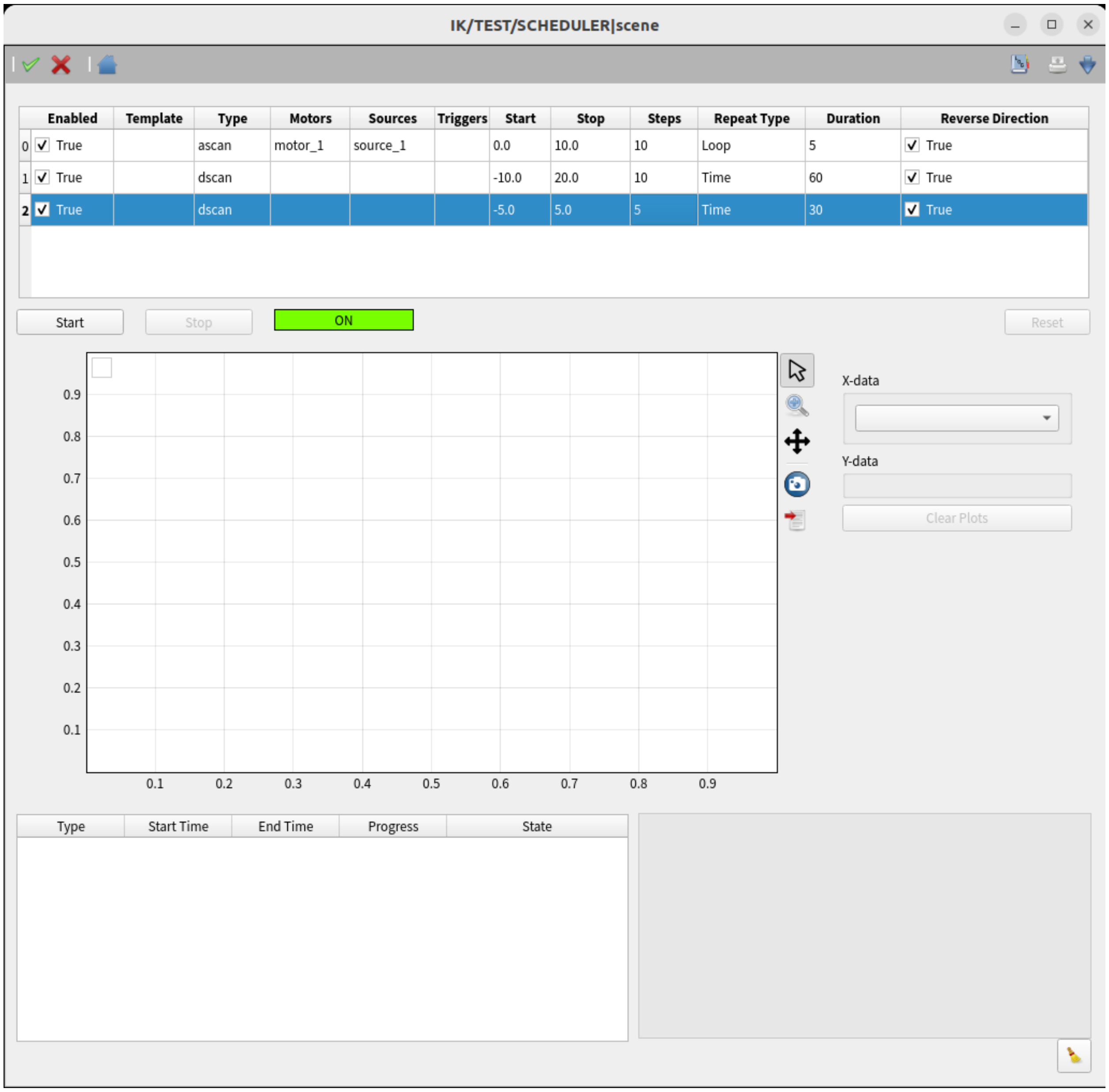
9.1 Scheduling Scans

The following rules are applied to the scheduler table:

- If in the row no scan parameters are provided then the current scantool settings are applied. In other words, the last scan is repeated.
- The attribute *Template* defines a scantool template that will be loaded before the scan.
- The attribute *Repeat Type* together with *Duration* defines how the scans are repeated. *Loop* will repeate scans *n* times where *n* is the *Duration* attribute. *Time* will execute scans for the time in seconds defined as *Duration*.
- Enabling *Reverse Direction* will swap the start and stop positions after each scan.
- If *useDAQ* is True then a DAQ run will be started before the first queue item and stopped after the last queue item.

9.2 Executing Scans

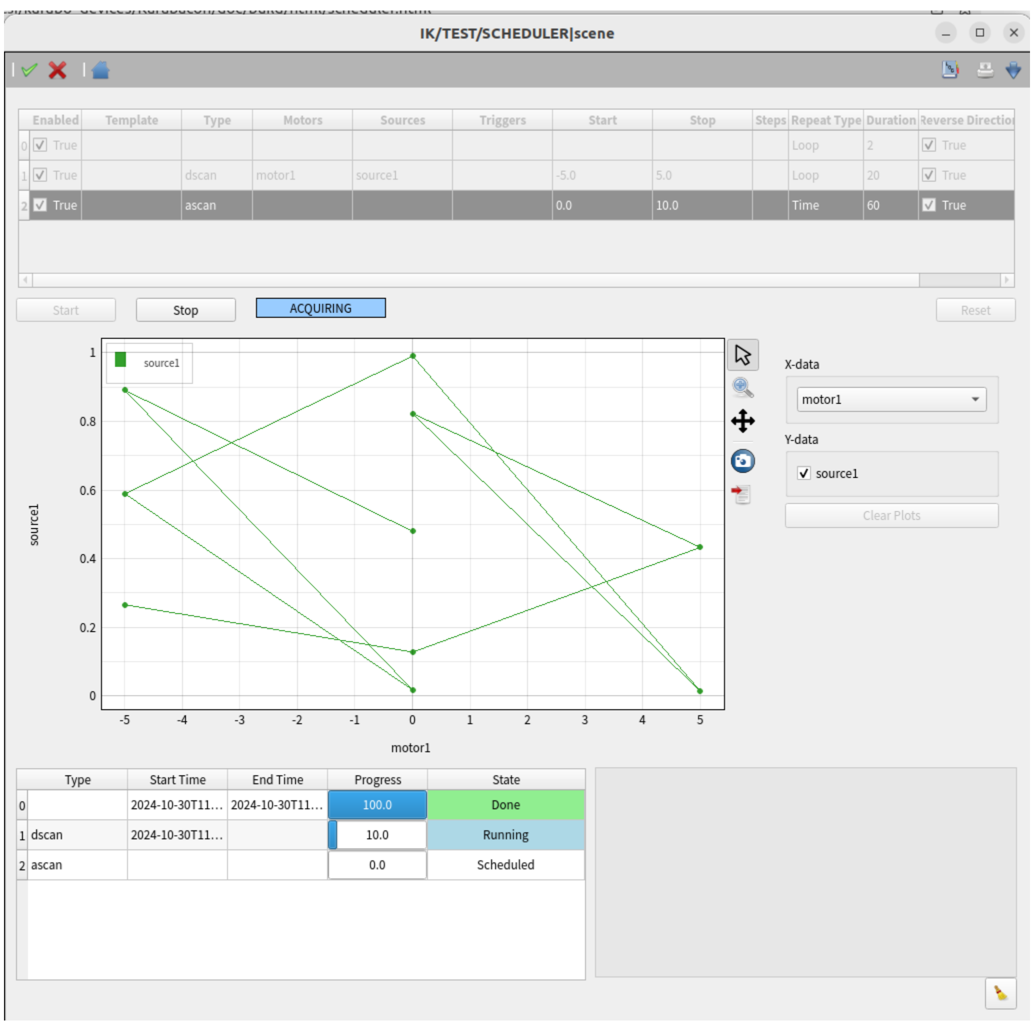
Scans within one scheduled row are joined and plotted as a single scan. If the consecutive scan has the same scan settings then the plot is not cleared and scan results are joined.



	Enabled	Template	Type	Motors	Sources	Triggers	Start	Stop	Steps	Repeat Type	Duration	Reverse Direction
0	✓ True		ascan	motor_1	source_1		0.0	10.0	10	Loop	5	✓ True
1	✓ True		dscan				-10.0	20.0	10	Time	60	✓ True
2	✓ True		dscan				-5.0	5.0	5	Time	30	✓ True

Type	Start Time	End Time	Progress	State

Fig. 9.1: The Karabacon Scheduler scene.



	Enabled	Template	Type	Motors	Sources	Triggers	Start	Stop	Steps	Repeat Type	Duration	Reverse Directio
0	✓ True									Loop	2	✓ True
1	✓ True		dscan	motor1	source1		-5.0	5.0		Loop	20	✓ True
2	✓ True		ascan				0.0	10.0		Time	60	✓ True

	Type	Start Time	End Time	Progress	State
0		2024-10-30T11…	2024-10-30T11…	100.0	Done
1	dscan	2024-10-30T11…		10.0	Running
2	ascan			0.0	Scheduled

Fig. 9.2: Queue execution.

CHAPTER

TEN

CHANGELOG

The Karabacon is continuously updated. The following versions are provided.

3.0.12

- Bugfix: Fixed failing mesh/dmesh scans when acquisition times vector is used.
- Bugfix: Set absolute motor coordinates for dmesh plots. Fixed issue where double-clicking the plot did not move motors to the correct positions.
- Refactoring of extensions.
- Added extension to fit results.
- Added TangoMotor to supported motor interfaces.

3.0.11

- DaqController tag update

3.0.10

- BugFix: Fix KarabaconScheduler synchroniztion with Karabacon

3.0.9

- Added new string property *acquisitionMode* based on an enum to adjust the data acquisition during a scan. Available acquisition modes: *single*: single data point after the end of a step, *continues*: acquire data during the acquisition time. Multiple points during the step, *continues averaged*: acquire data during the acquisition time. Average values and plot single data point. *continues extended*: acquire data during the acquisition time and time when motors are moving.
- Log book integration allows to send scan reports to logbook.
- setup.py replaced with pyproject.toml.
- Fix: Added correct state handling if custom state is defined.
- Fix: Do not populate daq info of a scan history item if the useDAQ is false.
- PicoMotor added to the list of supported motors.
- Karabacon goes open source.
- Added attribute to define state changed timeout for motor environment.
- Fix: After the end of a scan wait till daqController goes to active state.
- New device KarabaconScheduler to schedule scans.
- Fix: Custom coordinate upload failed if the file contains a single row. Now its handled correctly.
- Fix: Pause after each step misbehaved.

- Disable log button of a history item if no logbook destinations are available.
- Fix cli and sync with main scantool device.

3.0.8

- BugFix: Do not move motors to start positions twice.
- Added boolean to allow shuffling of ascan steps
- Use common ci.
- Added state attribute of the custom motor interface.
- Align imports.
- Added sign function to allowed step functions.
- Removed tabulate dependency.
- Added device clones to the supported motor interfaces.
- Unlock motors if the scan is paused.

3.0.7-2.16.2

- DaqController tag update

3.0.6-2.16.2

- Added prefix (scan index in the scan history) to the source ids to allow displaying multiple plots in the gui.
- Added feature to allow uploading custom scan coordinates via uploader gui extension.
- Added comments and selected daq info (proposal num, run num, sample name, experiment) fields to the scan history items.
- Spectrum scan checkbox allows to acquire and plot vector type data.

3.0.5-2.16.2

- Fixed bug in the scan history

3.0.4-2.16.2

- Do not reconfigure DAQ if its in monitoring state

3.0.3-2.16.2

- Fix: if step transform is used then instead of moving to the start position (value entered by the user and not transformed) move to the transformed start position.

3.0.2-2.16.2

- Save scan history and data in xml and json files.
- Lock motors before scan and unlock after the scan.
- Add aligner results to the output schema
- CLI aligned to the latest version
- Raise KaraboError if device alias is not accepted
- New scan type: custom. Ascan with user defined customScanPattern
- 3.0.1-2.16.2
  - Update DaqController to tag 1.5.1-2.16.2
- 3.0.0-2.16.2 (12/01/2023)

  - New Feature: Scan templates. Allows to store scantool settings by name.
  - Refactoring: Extensions moved to the extensions node.
  - Refactoring: motorEnv, dataEnv, triggerEnv moved to devices node. Moved all device related settings to the device node.
  - Refactoring: Replace ascan, a2scan, a3scan, a4scan with ascan, cscan, c2scan, c3scan, c4scan with cscan and dscan, d2scan, d3scan, d4scan with dscan. Scan settings are validated based on the number of motors.
  - Added new targets table item to the Aligner extension to display estimated values and allow to move motors to the corresponding position(s).
  - Bugfix: Move conflicting motors (for example, MultiAxis motor) sequentially.
  - New Feature: double click on the plot move motor(s) to the corresponding position(s).
- 2.4.14-2.13.4 (02/12/2022)
  - New Feature: Added tscan: for defined time acquire data from sources without moving motors
  - Repeating a scan from history sets all settings.
  - Bugfix: Do not plot data points if device is in ERROR state or value is None
- 2.4.13-2.13.4 (10/10/2022)
  - BugFix: fixed scene to be compatible with older Karabo versions.
- 2.4.12-2.13.4 (30/09/2022)
  - Added device selection tree to the build-in scene.
  - Added wait argument to the scan commands in cli.
  - Allow to write activeMotors, activeSources and activeTriggers for fast device selection.
- 2.4.11-2.13.4 (19/08/2022)
  - New Feature: Scan history table element stores information about executed scans. Contains buttons to repeat scan and plot scan results.
  - Added MonoChromator class to supported devices.
  - Improved main scene and added new scene with extensions and additional tools.
  - Added averaging of data source values.
  - BugFix: Fixed failing step scans in case when averageValues is True and no data source is selected.
- 2.4.8-2.13.4 (21/04/2022)
  - Staubli robot manipulator support.
  - Added motorIds and sourceIds to the output schema.
  - On initialization check if assigned device to daq controller has correct device class.
  - Notify user and set isConfigured to false if the scan environment has been changed.
  - Allow to enable/disable DAQ during the runtime.
- 1.6.0-2.6.0 (11/09/2019)
  - Pause instead of aborting scan in case of errors on motors
- 1.5.0-2.6.0 (04/06/2019)
  - Adapted DAQ integration to new DAQ state - PASSIVE

- 1.4.6-2.4.0 (28/05/2019)
  - Added BeamPositionMonitor support
- 1.4.5-2.4.0 (14/05/2019)
  - Added DoocsUndulatorEnergy and Monochromator support
- 1.4.4-2.4.0 (14/05/2019)
  - Added Proxy Processor device
- 1.4.3-2.4.0 (07/05/2019)
  - DoocsOpticalDelay support
- 1.4.1-2.4.0 (07/05/2019)
  - Gridscanner support
- 1.3.19-2.4.0 (26/04/2019)
  - Digitizer Processor support for roi integration and division
- 1.3.15-2.4.0 (29/03/2019)
  - The Karabacon can optionally pause and wait at every scan step
- 1.3.14-2.3.6 (22/03/2019)
  - Fix timestamp of actualStep on stable branch
- 1.3.13-2.3.6 (11/03/2019)
  - update-simulatedTrigger-tag for stable branch
  - Dont stop Karabacon when motor throws an exception !17
- 1.3.12-2.3 (17/12/2018)
  - Fix generic average implementation
- 1.3.11-2.2.5 (24/10/2018)
  - Add Image to spectrum !145
  - Make the runConfigurationGroup property free-form !153
  - Add Support for Pulse Picker Trigger!154
  - Added imageProcessors parameters to environment !155
  - Integration of DoocsPulseKicker device
  - Fix a software limit switch !159
  - genericaverage beamwidth !158
  - genericAverage has nodes in interface !161
  - compatible with new daq version !162
  - Include reconfigurable delay for larger configurations
- 1.3.10-2.2.5 (27/09/2018)
  - Split of development branch with master/stable, which got updated.
  - Full implementation of absolute gscan: CLI and Karabacon !130
  - First macro factory !132

  - Update DAQ documentation !131
  - Macro dscan !133
  - Float to Doubles !134
  - Provide isMoving isAcquiring booleans for daq !135
  - Enable Schema plugin in Karabacon !136
  - Support AGIPD in Karabacon !137
  - Bulkset of tables in Karabacon !138
  - Pipeline wait is configurable !139
  - Pipelining proxies !141
  - Karabacon can send now raw hash data !142
  - Find generically outputchannel !143
  - Cleaning of device proxy, trigger standard addition !144
  - Clean a little bit the Schema proxy !147
  - Add version history !148
  - Allow MultiAxisMotor and Motor interface simultaneously !146
  - Interface datasource implemented !149
  - Mangle the schema in SchemaProxy !150
  - Make the RunConfigurationGroup property freeform !151
  - Timestamp fix in steps !160
- 1.3.9-2.2.4 (22/09/2018)
  - Integration of DoocsPulseKicker device
- 1.3.8-2.2.4 (29/08/2018)
  - Added imageProcessors parameters to environment
- 1.3.6-2.2.4 (10/08/2018)
  - Add Support for Pulse Picker Trigger
- 1.3.5-2.2.4 (06/08/2018)
  - Make the runConfigurationGroup property free-form
- 1.3.4-2.2.4 (03/07/2018)
  - Addition of the ImageToSpectrum device
- 1.3.3-2.2.4 (20/06/2018)
  - Karabacon is made compatible with the new MDL trainId integration
  - The built-in TimeMiXin is removed and implemented in the framework
- 1.3.3-2.2.2 (08/03/2018)
  - Support of doocsUndulator
- 1.3.2-2.2.2 (02/03/2018)
  - Addition of a simulatedTrigger to work with Cameras.

- 1.3.1-2.2.2 (02/03/2018)
  - More DAQ protection
  - Scan configuration is checked, e.g. only done if the devices change. This is an essential performance fix.
  - ikarabo namespace is added to karabacon (connectDevice, setWait …)
- 1.3.0-2.2.0 (28/02/2018)
  - TrainId / Timeserver integration for output channel with TimeMiXin
  - Alignment with the new DAQ Interface, configurable number of aggregators
- 1.2.1-2.2.0 (15/02/2018)
  - Changed Karabacon device class from DeviceClientBase to Device
  - Actively remove Quantity Input in ScanClient in several function
- 1.2.0-2.2.0 (05/02/2018)

  The bacon comes alive! New release with major features and changes!

  Features:
  - Added full **Command Line Interface Support** (CLI) with spec pattern!
  - Validation of input parameters in CLI
  - Plotting from CLI with a scene runner
  - Addition of ScanClient Macro which is later provided with Macro protocol
  - Function added to retrieve descriptors for scalability
  - A single plot scene added to the Karabacon Scan device

Devices:

- Abstract motor axis included considering **KEP28**
- Fixed state machine behavior for Triggers (ACTIVE/PASSIVE/ACQUIRING)
- Triggers must have specific state before start acquiring
- ACTIVE state (follow target) is supported in motors.

Testing:

- Multi-Axis Motor testing with motor mocks
- Data source and Trigger mocks with tests

This release will only work with device versions tagged against Karabo 2.2.0.

- 1.1.3-2.1.18.3 (29/11/2017)
  - Bug fix for limit switches if the motor does not have one.
- 1.1.2-2.1.18.3 (27/11/2017)
  - Refactor base proxy in scan device and remove internal axis from triggers
  - CI Runner addition
  - Implement code style checker
  - Protection against concurrency access to DAQ - Check ACQUIRING state before configuring
- 1.1.1-2.1.18.3 (22/11/2017)

  - New scan option!
  - Implement **dmesh** feature
- 1.1.0-2.1.18.3 (22/11/2017)
  - Major New Feature for accounting triggers
  - Implement **acquisition time** feature
- 1.0.2-2.1.18.3 (16/11/2017)
  - ScanDevice provides deviceId's lookup when failing to connect
  - Support Noded properties (Digitizer)
  - **Limit switch implementation**
- 1.0.1-2.1.18.3 (10/11/2017)
  - The eventloop gets more time to breathe to swallow all the events from the ScanDevice.
  - More process Event calls
- 1.0.0-2.1.18.3 (9/11/2017)
  - First version of scantool providing basic functionalities
  - Scans: ascan (1-4), dscan (1-4), mesh
  - Float plotting
  - scene generation protocol
  - beckhoff motor support
  - greateyes support
  - gotthard support
  - lpd support
  - imageprocessor, analoginput support

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search