MacroExtensions

Release 1.0

Controls

May 03, 2023

CONTENTS

MacroExtensions is a karabo package that allows importing and using various classes, methods and attributes to simplify and improve macros. Extensions are categorized and available via **macro_extensions** namespace.

CHAPTER

ONE

CONTENTS:

1.1 *macro_extensions.motors*

Module contains helper funcions to access karabo devices with a pre-defined motor interface. Main usage of the module is to use unified api to control motors. Currently following Karabo device classes are supported:

- BeckhoffMC2Base
- BeckhoffMC2Beckhoff
- BeckhoffMC2Technosoft
- BeckhoffMC2Elmo
- BeckhoffMC2Hexapod
- BeckhoffSimpleMotor
- DoocsPhaseshifter
- CrystalManipulator
- CrystalManipulatorGroup
- DoocsOpticalDelay
- DoocsPhaseshifter
- DoocsUndulatorEnergy
- JJAttenuator
- MonoChromator
- MonoChromatorGroup
- MonoChromatorEnergyChanger
- PpLaserDelayCopy
- RobotManipulatorStaubli
- SlitSystem
- SoftMono
- X2TimerML

If the requested device class is not within the supported classes then typical motor attributes (*actualPosition*, *targetPosition*, *state*) and slots (*move*, *stop*) are used. If one of the attributes or slots is not available in the device class then an exception is raised.

1.1.1 *getMotor*, *connectMotor*

Methods *getMotor* and *connectMotor* return a *Motor* class instance which inherits karabo middlelayer proxy and has additional attributes:

- *isReady*: identifies if motor is ready for a movement.
- *isInterlocked*: identifies if motor is interlocked (not possible to move before interlock conditions are released).
- *position*: current motor position.
- *targetPosition*: target position.
- *velocity*: velocity.
- *isSWLimitLow*: is motor at the soft low limit.
- *isSWLimitHigh*: is motor at the soft high limit.
- *isCWLimit*: is motor at the clock wise limit switch.
- *isCCWLimit*: is motor at the counter-clockwise limit switch
- *isLimits*: Returns True if at least one of the limits is reached. Otherwise False.

and convenience functions:

- *moveTo*, *moveToWait*: non-blocking and blocking move to target position.
- *moveRelative*, *moveRelativeWait*: non-blocking and blocking move by a relative position.

getMotor returns motor instance for local and temporal usage and *connectMotor* returns motor instance for permanent access. Preferably use *getMotor* method within the *with* statement.

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

from macro_extensions.motors import getMotor

class TestGetMotor(Macro):
    motorName = String(defaultValue="TEST/MOTOR/1")
    target = Double(defaultValue=10)

    @MacroSlot()
    async def move(self):
        with (await getMotor(self.motorName.value)) as motor:

            init_position = motor.position
            # Moves motor to the target position and
            # waits till motor is in on of the ready states
            await motor.moveToWait(self.target.value)

            # Moves motor to the target position. A non blocking call
            await motor.moveTo(init_position)
```

Example of using *connectMotor*:

```
from karabo.middlelayer import (
    Macro, MacroSlot, VectorDouble, VectorString, waitUntil)

from asyncio import gather
```

(continues on next page)

(continued from previous page)

```
from macro_extensions.motors import connectMotor

class TestConnectMotor(Macro):
    motorNames = VectorString(defaultValue=["TEST/MOTOR/1", "TEST/MOTOR/2"])
    targets = VectorDouble(defaultValue=[10, 100])

    @MacroSlot()
    async def move(self):

        motors = [await connectMotor(motorName) for motorName in self.motorNames]
        init_positions = [motor.position for motor in motors]

        # Moving motors sequentially
        for index, motor in enumerate(motors):
            target = self.targets[index]
            print(f"Moving motor {motor.deviceId} to {target}")
            await motor.moveToWait(target)

        # Moving motors simultaneously and wait till all motors are ready
        fut = [motor.moveToWait(init_positions[index]) for index, motor in␣
↪enumerate(motors)]
        await gather(*fut)

        # Moving motors simultaneously and do not wait till motors are ready
        fut = [motor.moveTo(self.targets[index]) for index, motor in enumerate(motors)]
        await gather(*fut)

        # Wait till any of the motor is ready
        await waitUntil(lambda: any(motor.isReady for motor in motors))
        print("Done!")
```

1.1.2 *moveTo*, *moveToWait*, *moveRelative*, *moveRelativeWait*

- *moveTo*, *moveToWait*: non-blocking and blocking move to target position.
- *moveRelative*, *moveRelativeWait*: non-blocking and blocking move by a relative position.

Example:

```
from karabo.middlelayer import  Macro, MacroSlot, VectorDouble, VectorString

from macro_extensions.motors import moveTo, moveToWait, moveRelative, moveRelativeWait

class TestMoveTor(Macro):
    motorName = String(defaultValue="TEST/MOTOR/1")
    targets = Double(defaultValue=10)

    @MacroSlot()
    async def moveToExample(self):
        # Move motor to target position and wait till it is ready
        await moveToWait(self.motorName, target)

```

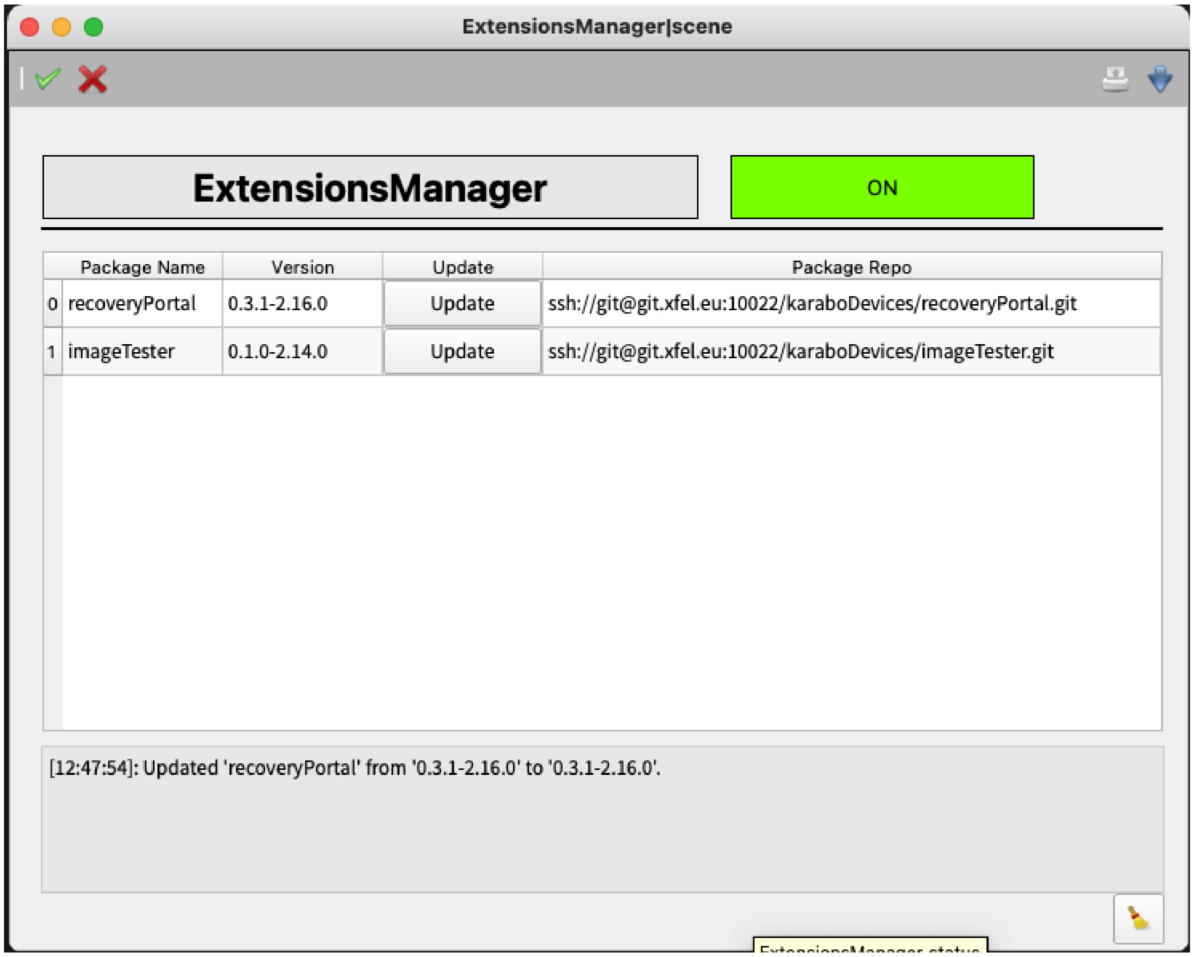
(continues on next page)

(continued from previous page)

```
        # Move motor by a relative position and wait till it is ready
        await moveRelativeWait(self.motorName, -10)
```

1.2 Macro Extensions Manager

Macro extensions manager is a karabo device that allows to upgrade macro extensions on a dedicated macro server. In the topology search for the macro extensions manager device and by double clicking on the device name a buil-in scene will be opened. Scene contains a table with available package names and a button to update selected package.



	Package Name	Version	Update	Package Repo
0	recoveryPortal	0.3.1-2.16.0	Update	ssh://git@git.xfel.eu:10022/karaboDevices/recoveryPortal.git
1	imageTester	0.1.0-2.14.0	Update	ssh://git@git.xfel.eu:10022/karaboDevices/imageTester.git

```
[12:47:54]: Updated 'recoveryPortal' from '0.3.1-2.16.0' to '0.3.1-2.16.0'.
```

CHAPTER

TWO

INDICES AND TABLES

- genindex
- search