# Writing a Simple Procedure in Karabo (How to Macro)

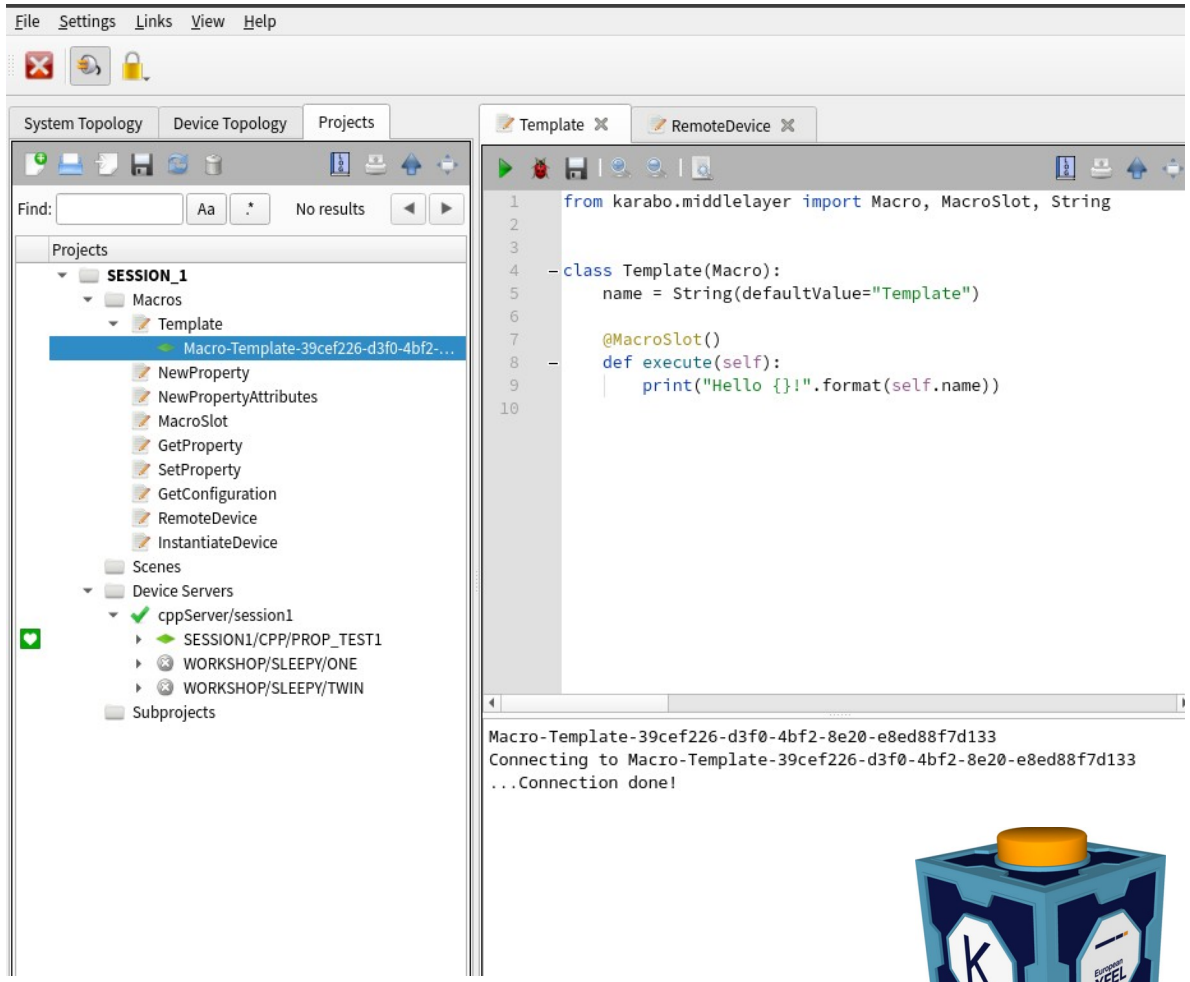
Anna Klimovskaia / Gabriele Giovanetti
European XFEL - Controls Group

NOBUGS2024 - Grenoble, 26.9.2024

# Macro Fundamentals

The Karabo framework provides scripting facilities which enable Instrument scientists to develop so-called ”macros”.

Macros are short python scripts that give the flexibility to write ad-hoc code by Karabo users directly in a karabo GUI.



# Macro Benefits

- Macros can be written quickly by Karabo users
- Macros can be saved as part of project configurations – covers recurring task and calculations
- Macros may be executed from the GUI
- Simplified syntax, wrt. To KArabo middle-layer devices.
- Interaction with karabo devices

# Macro Limitation

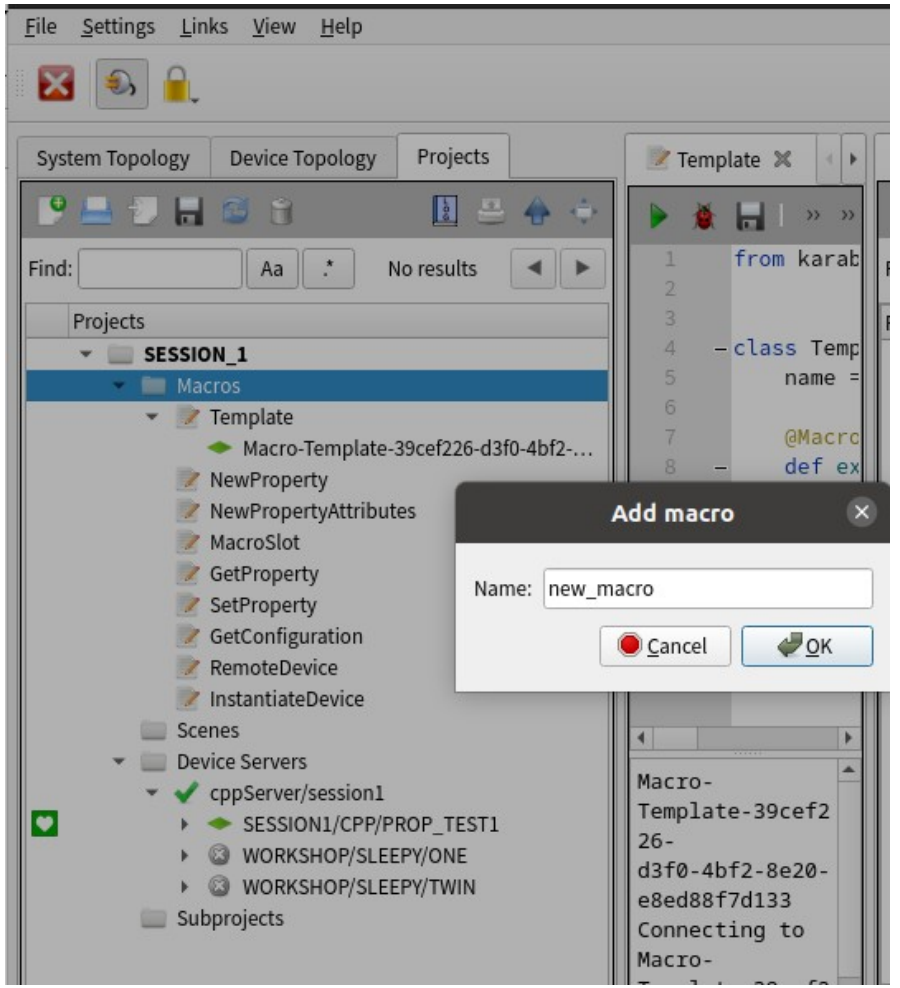
- Macros by concept should be kept simple
- Should not have state machine (Only ACTIVE and PASSIVE automatically set)
- Can be canceled
- Macros utilize the Karabo Middlelayer API interface. However, macros are only suitable for simple use cases
- There is no version control and limited support
- No configuration logging

# Macro Caveats

- A misbehaving macro could screw up all other macros running on the same macro server.
  - Test your macros in the development server when it is available (‘debug‘ button ).
  - Please follow good practices and use advised programming patterns.

# How To Create Macro

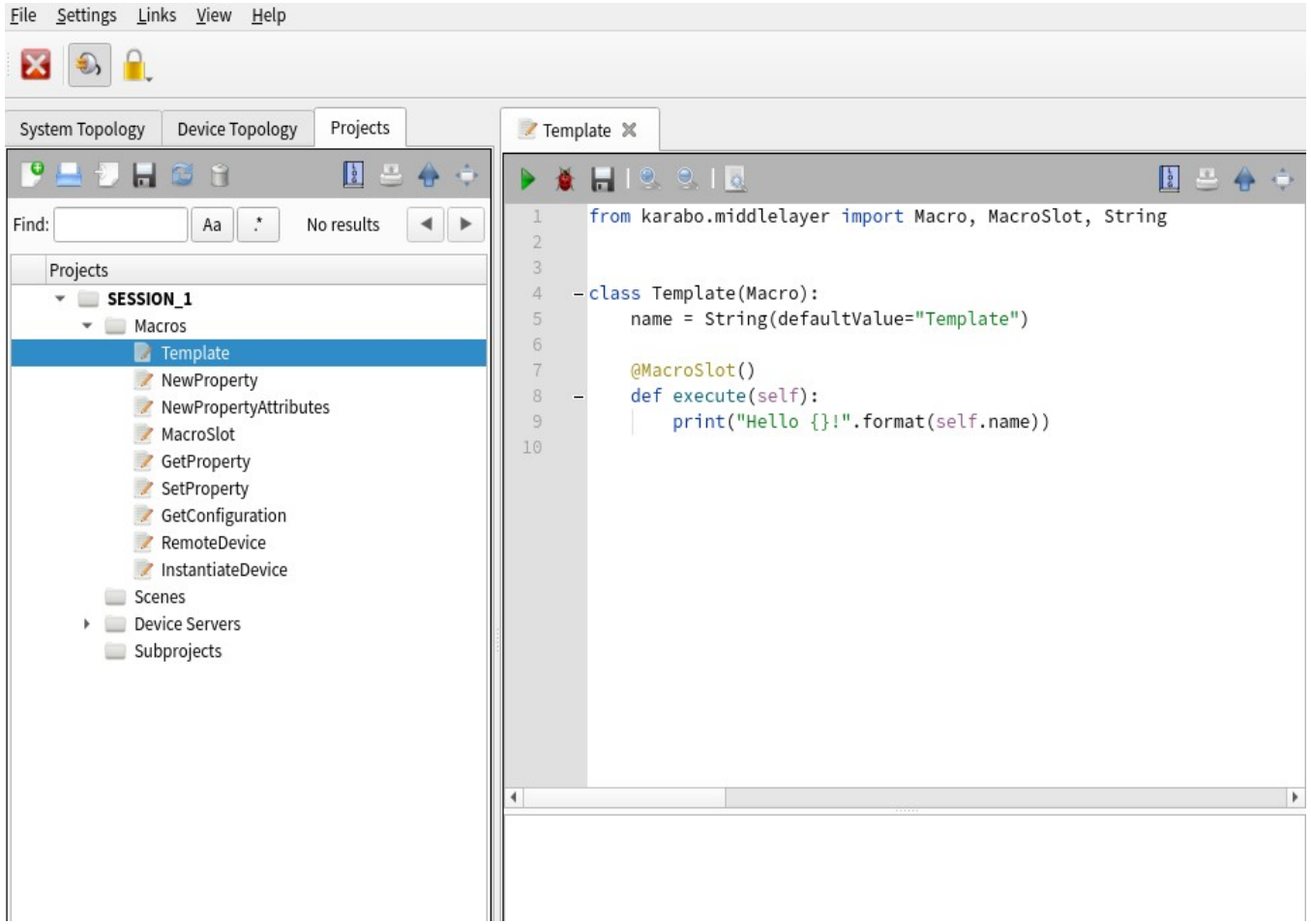
- Connect to karabo (admin, localhost:44444)
- Open project
- Select the macro folder
- Right-click and then click Add macro
- Fill the dialog with a macro-name
- As a result – new macro is added to the project with default content



# Macro Content

- Import from middlelayer api
- A Macro class
- Properties (Karabo descriptors), e.g. String
- MacroSlots to execute assigned functions

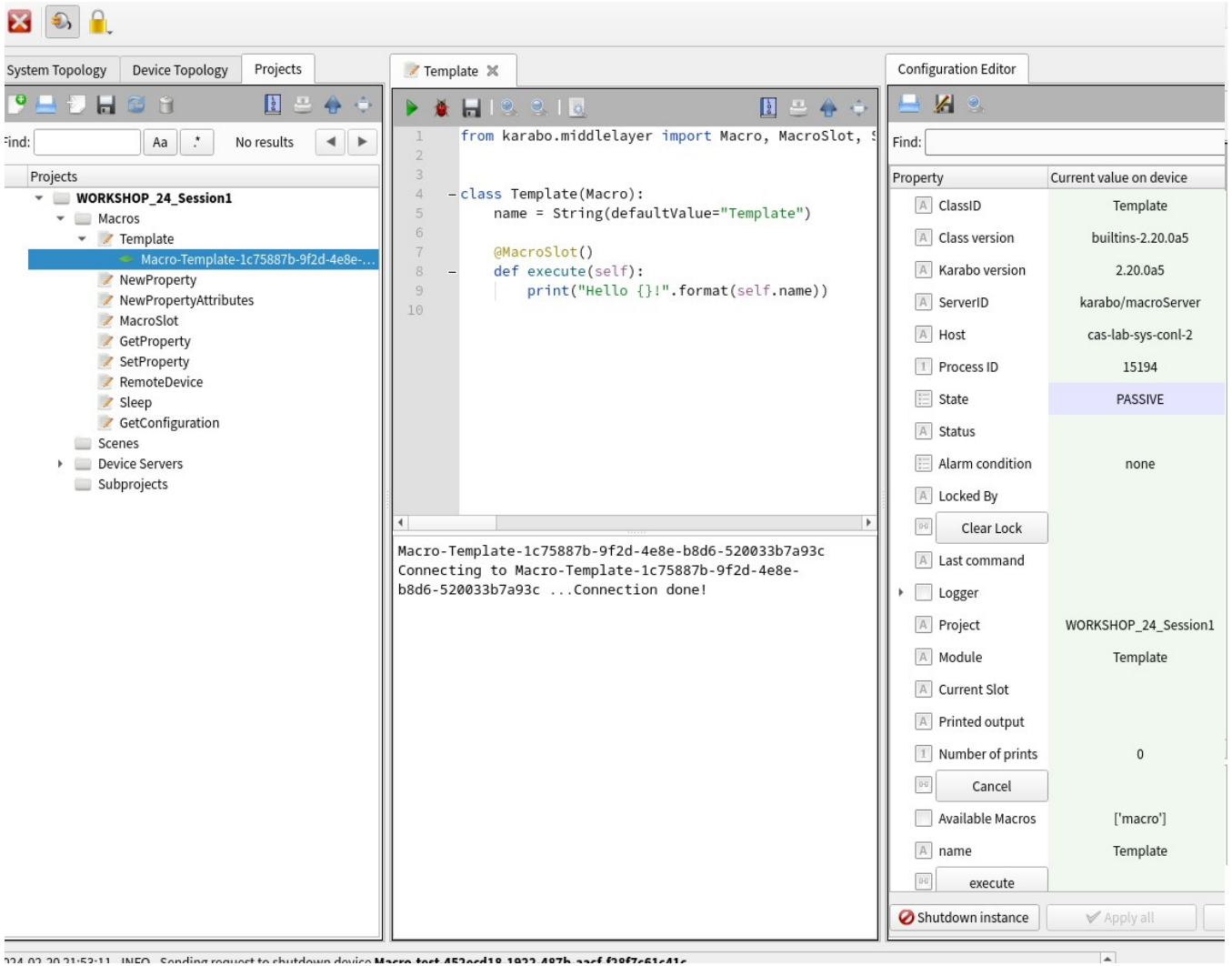
Note: A Macro always comes with a Cancel Slot – no need to define it



# Macro Content

- Import from middlelayer api
- A Macro class
- Properties (Karabo descriptors), e.g. String
- MacroSlots to execute assigned functions

Note: A Macro always comes with a Cancel Slot – no need to define it



# Code Style

- Class Definitions – CamelCase starting with capital letter
- Class Properties - camelCase, whereas non-exposed variables should have_underscores
- Slots are camelCase, methods have_underscores.
- It is preferred to check for conditions to be correct rather than using exceptions
- Use Double and NOT Float

```
class MyMacro(Macro):
    pass

class CallDRC(Macro):
    pass
```

# Code Style

- Class Definitions – CamelCase starting with capital letter
- Class Properties - camelCase, whereas non-exposed variables should have_underscores
- Slots are camelCase, methods have_underscores.
- It is preferred to check for conditions to be correct rather than using exceptions
- Use Double and NOT Float

```
name = String(displayedName="Name")

someOtherString = String(
    displayedName="Other string",
    defaultValue="Hello",
    accessMode=AccessMode.READONLY)

valid_ids = ["44eab", "ff64d"]
```

# Code Style

- Class Definitions – CamelCase starting with capital letter
- Class Properties - camelCase, whereas non-exposed variables should have_underscores
- Slots are camelCase, methods have_underscores.
- It is preferred to check for conditions to be correct rather than using exceptions
- Use Double and NOT Float

```
@MacroSlot(displayedName="Execute")
def executeMe(self):
    await self.some_action()

def some_action(self):
    pass
```

# Code Style

- Class Definitions – CamelCase starting with capital letter
- Class Properties - camelCase, whereas non-exposed variables should have_underscores
- Slots are camelCase, methods have_underscores.
- It is preferred to check for conditions to be correct rather than using exceptions
- Use Double and NOT Float

```
Therefore, the following is discouraged:
def execute_action(self):
try:
    await self.px.move()
except:
    pass

But rather:
def execute_action(self):
    if self.px.state not in {State.ERROR, State.MOVING}:
        await self.px.move()
    else:
        pass
```

# Code Examples

- Template – default macro content
- NewProperty – add new property
- NewPropertyAttributes – property attributes
- MacroSlot – add slot (button)
- GetProperty – get property from another device
- SetProperty – set new value to another device
- GetConfiguration – get full configuration
- RemoteDevice – monitor remote device
- InstantiateDevice – instantiate device with required configuration

# Template

- Import
- Instantiate on macro server
- Instantiate on development macro server
- Slot is a button
- Execution output

```
1   from karabo.middlelayer import Macro, MacroSlot, String
2
3
4   class Template(Macro):
5       name = String(defaultValue="Template")
6
7       @MacroSlot()
8       def execute(self):
9           print("Hello {}!".format(self.name))
10
```

# NewProperty

- Different types
- Properties order

```
1  from karabo.middlelayer import Int32, Double, Macro
2
3
4  class NewProperty(Macro):
5
6      currentVoltage = Double(defaultValue=0.0)
7      steps = Int32()
```

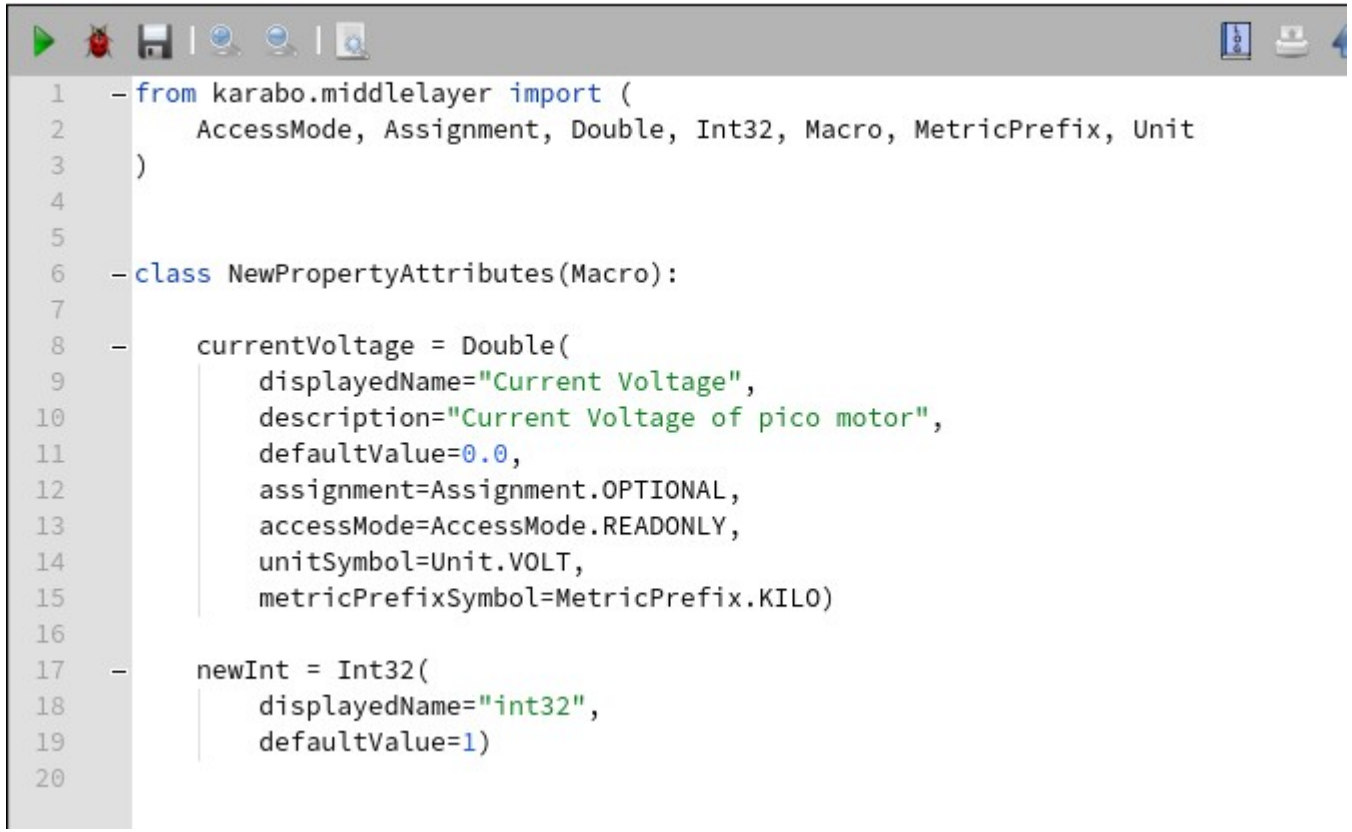
# NewPropertyAttributes

- Import
- List of attributes

```python
from karabo.middlelayer import (
    AccessMode, Assignment, Double, Macro, MetricPrefix, Unit
)


class NewPropertyAttributes(Macro):

    currentVoltage = Double(
        displayedName="Current Voltage",
        description="Current Voltage of pico motor",
        defaultValue=0.0,
        assignment=Assignment.OPTIONAL,
        accessMode=AccessMode.READONLY,
        unitSymbol=Unit.VOLT,
        metricPrefixSymbol=MetricPrefix.KILO)

```

# Task1 - NewPropertyAttributes

- Add new Int32 property
  - Don’t forget to import proper type
  - Change property key
  - Set new default value



```python
from karabo.middlelayer import (
    AccessMode, Assignment, Double, Int32, Macro, MetricPrefix, Unit
)


class NewPropertyAttributes(Macro):

    currentVoltage = Double(
        displayedName="Current Voltage",
        description="Current Voltage of pico motor",
        defaultValue=0.0,
        assignment=Assignment.OPTIONAL,
        accessMode=AccessMode.READONLY,
        unitSymbol=Unit.VOLT,
        metricPrefixSymbol=MetricPrefix.KILO)

    newInt = Int32(
        displayedName="int32",
        defaultValue=1)

```

# MacroSlot

- Import
- Execution output

```
1  from karabo.middlelayer import Macro, MacroSlot, String
2
3
4  class MacroSlot(Macro):
5
6      name = String(defaultValue="Macroslot")
7
8      @MacroSlot()
9      def printName(self):
10         print(f"Hello {self.name}!")
```

# Task2 - MacroSlot

- Calculate new value and set to macro property
  - Update import
  - Change displayed name
  - Add new properties
  - Set new value x2

```python
from karabo.middlelayer import AccessMode, Int8, Macro, MacroSlot, String


class MacroSlot(Macro):

    name = String(defaultValue="Macroslot")

    @MacroSlot()
    def printName(self):
        print(f"Hello {self.name}!")

    newInt = Int8()

    @MacroSlot()
    def miltiplication(self):
        self.calculatedInt = self.newInt*2

    calculatedInt = Int8(accessMode=AccessMode.READONLY)
```

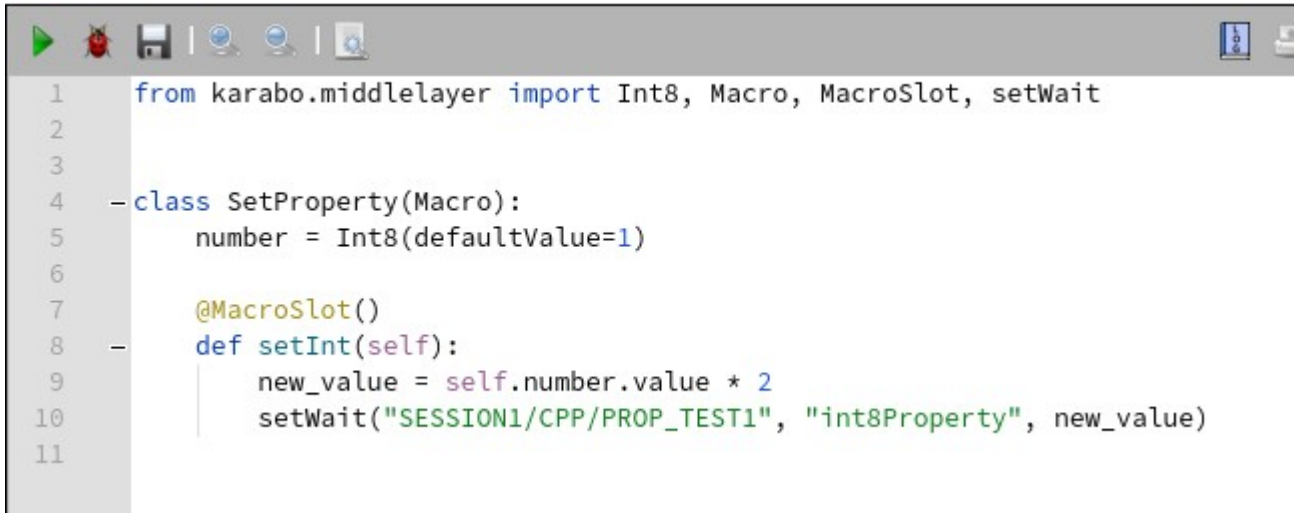
# GetProperty

- Import
- GetProperties input parameters

```
1   from karabo.middlelayer import Macro, MacroSlot, String, getProperties
2
3
4   class GetProperty(Macro):
5
6       @MacroSlot(displayedName="Read Value")
7       def readString(self):
8           self.name = getProperties("SESSION1/CPP/PROP_TEST1",
9                                     ["stringProperty", "int8Property"])["stringProperty"]
10
11      name = String(displayedName="Value on Device")
12
```

# SetProperty

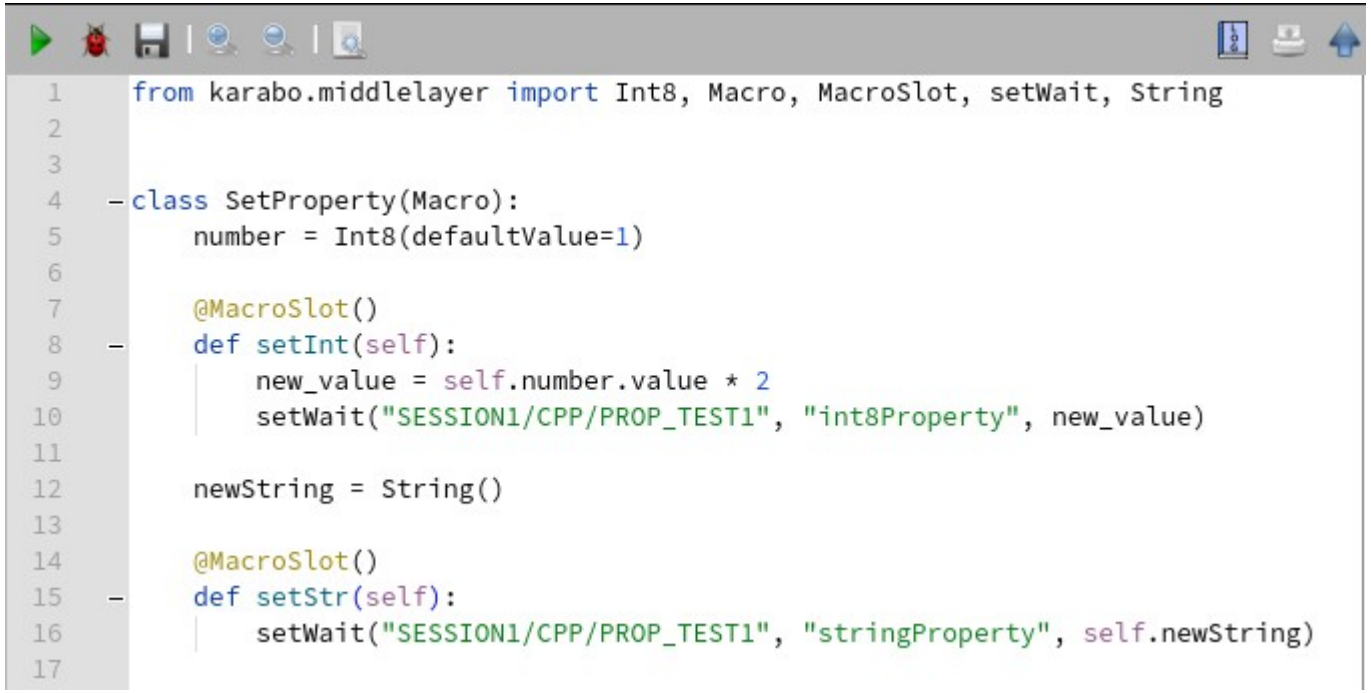
- Import
- Scene Set_Value



```
1   from karabo.middlelayer import Int8, Macro, MacroSlot, setWait
2
3
4   class SetProperty(Macro):
5       number = Int8(defaultValue=1)
6
7       @MacroSlot()
8       def setInt(self):
9           new_value = self.number.value * 2
10          setWait("SESSION1/CPP/PROP_TEST1", "int8Property", new_value)
11
```

# Task3 - SetProperty

- Add new slot to set any other property on remote device
  - Don’t forget to import proper type
  - Change property key
  - Change set property key



```
1  from karabo.middlelayer import Int8, Macro, MacroSlot, setWait, String
2
3
4  class SetProperty(Macro):
5      number = Int8(defaultValue=1)
6
7      @MacroSlot()
8      def setInt(self):
9          new_value = self.number.value * 2
10         setWait("SESSION1/CPP/PROP_TEST1", "int8Property", new_value)
11
12     newString = String()
13
14     @MacroSlot()
15     def setStr(self):
16         setWait("SESSION1/CPP/PROP_TEST1", "stringProperty", self.newString)
17
```

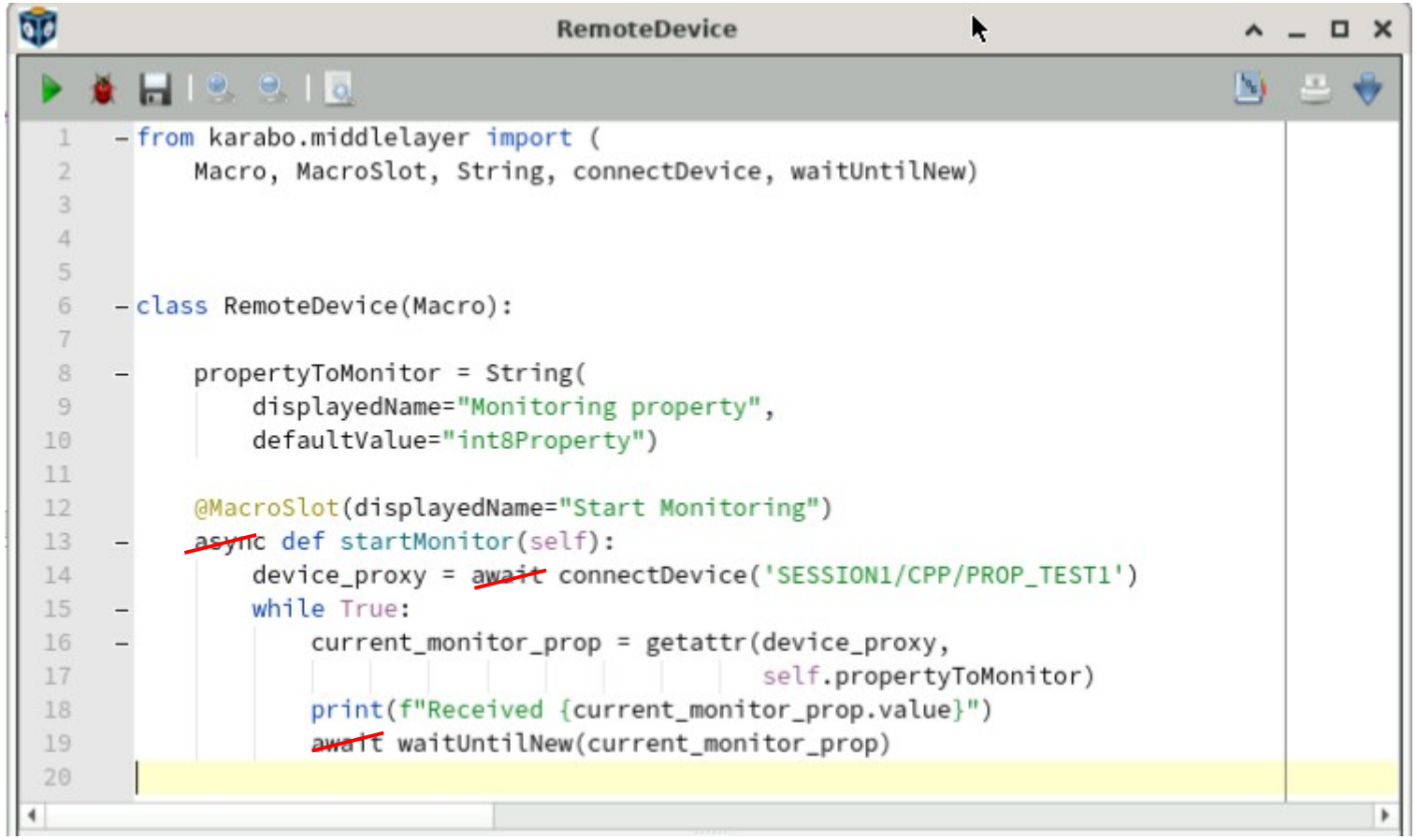
# GetConfiguration

- Import
- getConfiguration input parameters

```
1  from karabo.middlelayer import Int32, Macro, MacroSlot, getConfiguration
2
3
4  class GetConfiguration(Macro):
5
6      @MacroSlot()
7      def execute(self):
8          config = getConfiguration("SESSION1/CPP/PROP_TEST1")
9          print(config.keys())
10         self.testInt = config["int32Property"]
11
12     testInt = Int32()
13
```

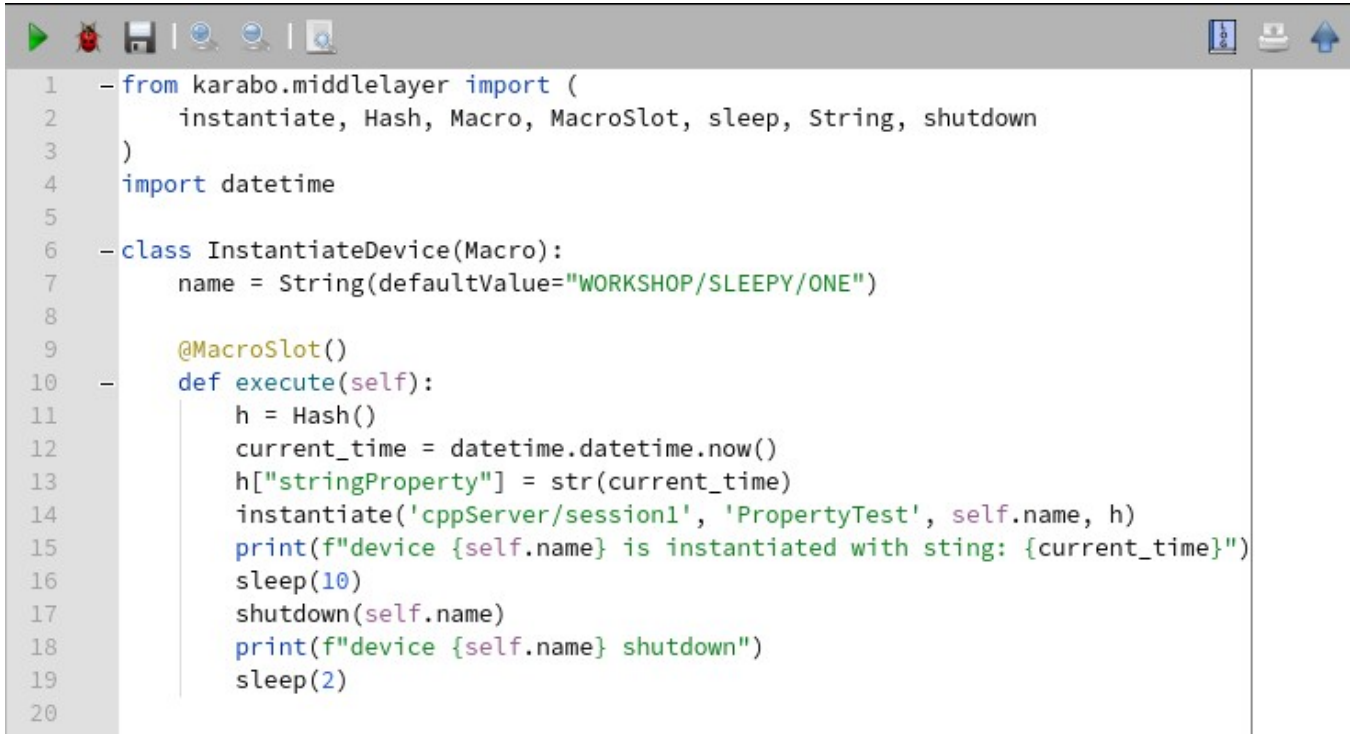
# RemoteDevice

- Import
- RemoveDevice input parameters
- WaitUntilNew
- Change monitoring property
- Note the optional async/await syntax



# InstantiateDevice

- Import
- Sleep
- Set several properties on instantiation



```python
from karabo.middlelayer import (
    instantiate, Hash, Macro, MacroSlot, sleep, String, shutdown
)
import datetime

class InstantiateDevice(Macro):
    name = String(defaultValue="WORKSHOP/SLEEPY/ONE")

    @MacroSlot()
    def execute(self):
        h = Hash()
        current_time = datetime.datetime.now()
        h["stringProperty"] = str(current_time)
        instantiate('cppServer/session1', 'PropertyTest', self.name, h)
        print(f"device {self.name} is instantiated with sting: {current_time}")
        sleep(10)
        shutdown(self.name)
        print(f"device {self.name} shutdown")
        sleep(2)

```

# Task4 - InstantiateDevice

- Add several properties to initial device configuration
  - Import types
  - Add new property to hash

```
    instantiate, Hash, Macro, MacroSlot, sleep, String, shutdown
)
import datetime

class InstantiateDevice(Macro):
    name = String(defaultValue="WORKSHOP/SLEEPY/ONE")
    newInt = Int8(defaultValue=1)

    @MacroSlot()
    def execute(self):
        h = Hash()
        current_time = datetime.datetime.now()
        h["stringProperty"] = str(current_time)
        h["int8Prtoperty"] = self.newInt
        instantiate('cppServer/session1', 'PropertyTest', self.name, h)
        print(f"device {self.name} is instantiated with sting: {current_time}")
        sleep(10)
        shutdown(self.name)
        print(f"device {self.name} shutdown")
        sleep(2)
```