MeaVis Documentation

Release 0.2.2.dev2

Hofheinz' group

Dec 28, 2020

REFERENCE:

MeaVis is a python framework intended to define how **Mea**surements have to be run and a programming interface to **Vis**ualise resulting datasets.

See more details at ReadTheDocs.io.

CHAPTER ONE

BASIC EXAMPLE

1.1 General configuration of intruments

Let's assume two basic drivers as follow:

```python
import vxi11


class AgilentB596X(vxi11.Instrument):
    def __init__(self, host):
        super().__init__(host=host)

        self.channel = None
        self.mode = None

    def conf():
        if self.channel and self.mode:
            self.write(":SOUR{}:FUNC:MODE {}".format(self.channel, self.mode))

    def output(self, value):
        self.conf()
        if self.channel:
            self.write(
                ":OUTP{} {}".format(self.channel, "ON" if value else "OFF")
            )

    def set_channel(self, channel):
        self.channel = channel
        self.write(":SOUR{}:FUNC:SHAP DC".format(self.channel))
        self.conf()

    def set_mode(self, mode):
        self.mode = mode
        self.conf()

    def set_value(self, value):
        self.write(":SOUR{}:{} {}".format(self.channel, self.mode, value))


class KeySight344XX(vxi11.Instrument):
    def __init__(self, host):
        super().__init__(host=host)

        self.write("*CLS")
```

(continues on next page)

(continued from previous page)

```
        self.write("*RST")

        self.write("TRIG:SOUR IMM")

        self.write("CALC:FUNC AVER")
        self.write("CALC:STAT ON")

        self.ACorDC = None
        self.mode = None

    def calc_average(self):
        self.write("*CALC:AVER:AVER")
        return float(self.read())

    def conf(self):
        if self.ACorDC and self.mode:
            self.write("CONF:{}:{}".format(self.mode, self.ACorDC))

    def conf_ACorDC(self, ACorDC):
        self.ACorDC = ACorDC
        self.conf()

    def conf_mode(self, mode):
        self.mode = mode
        self.conf()

    def count(self, value):
        self.write("SAMP:COUN {}".format(value))

    def initiate(self):
        self.write("INIT")

    def opc(self):
        self.write("*OPC")

    def set_aperture(self, value):
        self.write("{}:APER {}".format(self.mode, value))
```

As you see, this driver is very close to a one-to-one correpondance between the SPCI commands and the methods. Of course, methods can be more complexe, however a simplest driver as possible allow more flexiblity.

Then classes specific to MeaVis have to be written to decribe how to use this driver:

- *Drivers*: Front panel instrument.
- *MeaVis classes*: How an experimentalist use the front panel.

```
import drivers

import meavis.tags


@meavis.tags.initialiser("power_source.current_source", mode="CURR")
@meavis.tags.initialiser("power_source.voltage_source", mode="VOLT")
class InitialiserB596X:
    def __init__(self, mode):
        self.mode = mode
```

(continues on next page)

(continued from previous page)

```
    def initialise(self, handler, channel):
        handler_channel = drivers.AgilentB596X(**handler)

        handler_channel.set_channel(channel)
        handler_channel.set_mode(self.mode)
        handler_channel.output(True)

        return handler_channel


@meavis.tags.initialiser("multimeter.ac_current_meter")
@meavis.tags.kwargs(mode="CURR", ACorDC="AC")
@meavis.tags.initialiser("multimeter.ac_volt_meter")
@meavis.tags.kwargs(mode="VOLT", ACorDC="AC")
@meavis.tags.initialiser("multimeter.dc_current_meter")
@meavis.tags.kwargs(mode="CURR", ACorDC="DC")
@meavis.tags.initialiser("multimeter.dc_volt_meter")
@meavis.tags.kwargs(mode="VOLT", ACorDC="DC")
class Initialiser344XX:
    def __init__(self, mode, ACorDC):
        self.ACorDC = ACorDC
        self.mode = mode

    def initialise(self, handler, channel):
        handler_channel = drivers.KeySight344XX(**handler)

        handler_channel.conf_mode(self.mode)
        handler_channel.conf_ACorDC(True)

        return handler_channel


@meavis.tags.parameter("power_source.current_source.current")
@meavis.tags.attributes(unit="A", delay=0.1)
@meavis.tags.parameter("power_source.voltage_source.voltage")
@meavis.tags.attributes(unit="V", delay=0.1)
class SourceValue:
    def __init__(self, data):
        self.data = data

    def apply(self, handler, value):
        handler.set_value(value)


@meavis.tags.parameter("multimeter.~.aperture")
@meavis.tags.attributes(unit="s")
class DMMAperture:
    def __init__(self, data):
        self.data = data

    def apply(self, handler, value):
        handler.set_aperture(value)


@meavis.tags.parameter("multimeter.~.average_count")
class DMMCount:
    def __init__(self, data):
```

(continues on next page)

(continued from previous page)

```
        self.data = data

    def apply(self, handler, value):
        handler.count(value)


@meavis.tags.measurement("multimeter.ac_current_meter|dc_current_meter.current")
@meavis.tags.measurement("multimeter.ac_volt_meter|dc_volt_meter.voltage")
class DMMAverage:
    def trigger(self, handler):
        handler.initiate()

    def wait(self, handler):
        handler.opc()
        handler.calc_average()
```

The elements mapped after *kwargs* will be used to initialise the corresponding instrument when required. For exemple, if a source is used as a voltage source, the statement `handler = meavis_user.InitialiserB596X(mode="VOLT").intialise(/* */)` will be executed.

This file have to be loaded as follow:

```
meavis.instruments.inject(meavis_user._meavis_instruments)
```

Note that it cannot be loaded multiple time, otherwise name collisions will happen.

Up to now the configuration is independant of what we want to measure: it only describes how to use instruments, but not how they are connect or what we want to do.

1.2 Experiment-specific configuration of intruments

First we describe how instruments are wired and for which purpose with a YAML file:

```
junction_bias:
  instrument: power_source
  usage: voltage_source
  kwargs:
    addr: 192.168.0.0
  attributes:
    channel: 1
junction_current:
  instrument: multimeter
  usage: dc_current_meter
  kwargs:
    host: 192.168.0.1
  attributes:
    channel: 1
junction_voltage:
  instrument: multimeter
  usage: dc_volt_meter
  kwargs:
    host: 192.168.0.1
  attributes:
    channel: 2
```

The elements mapped after *kwargs* will be used to construct the corresponding instrument when required. For exemple, the multimeter to measure the junction voltage is constructed with the statement: `handler = meavis_user.ConstructorEthernet(host="192.168.0.1").create()` when required. Morevoer the attribute `channel: 2` is used for the initialisation `handler = meavis_user.InitialiserB596X(/* */).intialise(/* */, channel=2)`.

This file have to be loaded as follow:

```
with open("instances.yaml") as file:
    meavis.instruments.register(yaml.safe_load(file))
```

Note that it cannot be loaded multiple time, otherwise name collisions will happen. After this step, parameters and measurements can be accessed as follow:

```
meavis.parameters.junction_current.aperture([10e-3])
meavis.parameters.junction_current.average_count([100])

meavis.parameters.junction_voltage.aperture([100e-3])
meavis.parameters.junction_voltage.average_count([10])
```

Availbed parameters and measurements are displayed in the log output:

```
INFO --  Register power_source constructor [90e97748d6ea9cbb434602eb177a91c685701667]
↪{host: 192.168.0.0}.
INFO --  Register voltage_source initialiser {mode: VOLT} for junction_bias.
INFO --  Register voltage as parameter named junction_bias.voltage.
INFO --  Register multimeter constructor [1ca226d3ca09e4167fbbfa3bd218a8323d76e12f]
↪{host: 192.168.0.1}.
INFO --  Register dc_current_meter initialiser {mode: CURR, ACorDC: DC} for junction_
↪current.
INFO --  Register aperture as parameter named junction_current.aperture.
INFO --  Register average_count as parameter named junction_current.average_count.
INFO --  Register current as measurement named junction_current.current.
INFO --  Register dc_volt_meter initialiser {mode: VOLT, ACorDC: DC} for junction_
↪voltage.
INFO --  Register aperture as parameter named junction_voltage.aperture.
INFO --  Register average_count as parameter named junction_voltage.average_count.
INFO --  Register voltage as measurement named junction_voltage.voltage.
INFO --  Add completer group for junction_bias : {junction_bias.voltage}.
INFO --  Add completer group for junction_current : {junction_current.average_count,␣
↪junction_current.aperture}.
INFO --  Add completer group for junction_voltage : {junction_voltage.aperture,␣
↪junction_voltage.average_count}.
```

And finally the measurement can be described and processed as follow:

```
measurement_loop = meavis.loop.LoopEngine(
    yaml.safe_load(
        """
parameters:
    - junction_bias.voltage
measurements:
    - junction_current.current
    - junction_voltage.voltage
name: iv_dc_4probes
"""
)).create(
    meavis.parameters.junction_bias.voltage(numpy.linspace(-1e-3, 1e-3, 401)),
```

(continues on next page)

(continued from previous page)

```
    meavis.measurements.junction_current.current(),
    meavis.measurements.junction_voltage.voltage(),
)
measurement_loop.trigger(None)
measurement_loop.wait(None)
```

In the log output, instruments are created and intialised when required:

```
INFO --  Complete parameters with [junction_current.average_count, junction_current.
↪aperture]
INFO --  Complete parameters with [junction_voltage.aperture, junction_voltage.
↪average_count]
INFO --  Create handler of multimeter.constructor␣
↪[1ca226d3ca09e4167fbbfa3bd218a8323d76e12f] with {host: 192.168.0.1}.
INFO --  Initialise channel 2 on handler of multimeter.dc_volt_meter.initialiser with
↪{mode: VOLT, ACorDC: DC}.
INFO --  Set junction_voltage.aperture to 0.1 s.
INFO --  Set junction_voltage.average_count to 10.
INFO --  Initialise channel 1 on handler of multimeter.dc_current_meter.initialiser␣
↪with {mode: CURR, ACorDC: DC}.
INFO --  Set junction_current.aperture to 0.01 s.
INFO --  Set junction_current.average_count to 100.
INFO --  Create handler of power_source.constructor␣
↪[90e97748d6ea9cbb434602eb177a91c685701667] with {host: 192.168.0.0}.
INFO --  Initialise channel 1 on handler of power_source.voltage_source.initialiser␣
↪with {mode: VOLT}.
INFO --  Set junction_bias.voltage to -0.001 V.
INFO --  Trigger junction_current.current, waiting for data.
INFO --  Trigger junction_voltage.voltage, waiting for data.
INFO --  Set junction_bias.voltage to -0.000998 V.
INFO --  Trigger junction_current.current, waiting for data.
INFO --  Trigger junction_voltage.voltage, waiting for data.
```

1.2.1 API

meavis package

Measurement & Visualisation python framework.

Submodules

meavis.completer module

MeaVis parameters completion.

class meavis.completer.**CompleterEngine**(*data*)

Bases: object

Define how parameters of a loop has to be completed.

instances_parameters = {}

__init__(*data*)

Store a data structure as loop pattern.

classmethod clear()

Clear instances set.

classmethod inject_instances(*instances*)

Inject instances in CompleterEngine maps.

complete(*instances=()*)

Complete a loop pattern.

meavis.instruments module

Main loop functions for running MeaVis measurements.

meavis.instruments.**clear**(*module_name*)

Clear injected names by users.

meavis.instruments.**inject**(*instruments*)

Inject instruments.

meavis.instruments.**register**(*instances*)

Inject instances.

meavis.loop module

Main loop functions for running MeaVis measurements.

class meavis.loop.**LoopMeasurement**(*parameters*, *measurements*)

Bases: object

Define a measurement running a loop.

__init__(*parameters*, *measurements*)

Ceate a loop measurement.

trigger(*handler*)

Nothing to trigger.

wait(*handler*)

Run the loop.

class meavis.loop.**LoopEngine**(*data*)

Bases: object

Define how a loop has to be processed.

items_map = {}

default_map = {'measurements': {'handler': None, 'initialiser': None, 'invasive':

__init__(*data*)

Store a data structure as loop pattern.

classmethod clear()

Clear MeaVis item maps.

classmethod inject_items(**items*)

Inject MeaVis items in the LoopEngine.

inject_defaults()

Inject default attributes in LoopEngine maps.

complete()
Complete current data structure.

create(**items*, *completion=True*)
Create a measurement from the pattern.

synchronisers(*state_parameters*, *completion=True*)
Synchronise parameters group from the pattern.

meavis.markup module

Read and write MeaVis markup language.

meavis.markup.**visit_instruments**(*meavis_instruments*, *tag_name*, *meavis_name*)
Visit an instrument hierarchy and return corresponding items.

meavis.measurements module

MeaVis measurements namespace for user-defined injection.

meavis.measurements.**inject**(*cls*, *name*)
Wrap and inject user-defined measurement in this namespace.

meavis.parameters module

MeaVis parameters namespace for user-defined injection.

meavis.parameters.**inject**(*cls*, *name*)
Wrap and inject user-defined parameter in this namespace.

meavis.synchroniser module

Synchronisation of MeaVis parameters.

class meavis.synchroniser.**LoopSynchroniser**(*state_parameters*, *loop_parameters*, *synchronisers*)
Bases: object

Define a looper to pre-synchronise instruments.

__init__(*state_parameters*, *loop_parameters*, *synchronisers*)
Ceate a loop synchroniser.

pre_synchronise(*states*)
Run the loop.

meavis.tags module

Class decorators to tag MeaVis classes.

`meavis.tags.`**`add_metadata`**`(`*metadata_name*, ***kwargs*`)`
Add metadata to a tagged MeaVis class.

`meavis.tags.`**`add_item`**`(`*tag_name*, *meavis_name*, ***kwargs*`)`
Tag a MeaVis class.

`meavis.tags.`**`attributes`**`(`***kwargs*`)`
Add attributes to a tagged MeaVis class.

`meavis.tags.`**`kwargs`**`(`***kwargs*`)`
Add kwargs to a tagged MeaVis class.

`meavis.tags.`**`constructor`**`(`*meavis_name*, ***kwargs*`)`
Tag a class as a MeaVis initialiser.

`meavis.tags.`**`initialiser`**`(`*meavis_name*, ***kwargs*`)`
Tag a class as a MeaVis initialiser.

`meavis.tags.`**`measurement`**`(`*meavis_name*, ***kwargs*`)`
Tag a class as a MeaVis measurement.

`meavis.tags.`**`parameter`**`(`*meavis_name*, ***kwargs*`)`
Tag a class as a MeaVis parameter.

meavis.tasks module

MeaVis tasks for threading.

`meavis.tasks.`**`setup_and_acquire`**`(`*meavis_item*`)`
Set meavis_item handler if required.

`meavis.tasks.`**`settle`**`(`*parameter*, *sample*, *lock_in*, *lock_out*, *delay=True*`)`
Settle a parameter.

`meavis.tasks.`**`trigger_wait`**`(`*measurement*, *lock_in*, *lock_out*, *lock_barrier*`)`
Trigger & wait for a measurement.

1.2.2 ChangeLog

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog and this project adheres to Semantic Versioning.

Unreleased

Added

Change

Deprecated

Removed

Fixed

[0.2.1]

Added

- Basic IV measurement as first example.
- Loop completer for parameters.
- Tags mechanism.

Change

- Mapping inheritance for MeaVis Markup Language – Instrument.
- Constructor and Initialiser tags for MeaVis Markup Language – Instrument.

1.2.3 License

```
                   GNU LESSER GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007

 Copyright (C) 2007 Free Software Foundation, Inc. <https://fsf.org/>
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.


  This version of the GNU Lesser General Public License incorporates
the terms and conditions of version 3 of the GNU General Public
License, supplemented by the additional permissions listed below.

  0. Additional Definitions.

  As used herein, "this License" refers to version 3 of the GNU Lesser
General Public License, and the "GNU GPL" refers to version 3 of the GNU
General Public License.

  "The Library" refers to a covered work governed by this License,
other than an Application or a Combined Work as defined below.

  An "Application" is any work that makes use of an interface provided
by the Library, but which is not otherwise based on the Library.
Defining a subclass of a class defined by the Library is deemed a mode
of using an interface provided by the Library.

  A "Combined Work" is a work produced by combining or linking an
Application with the Library.  The particular version of the Library
with which the Combined Work was made is also called the "Linked
Version".

  The "Minimal Corresponding Source" for a Combined Work means the
```

(continues on next page)

(continued from previous page)

```
Corresponding Source for the Combined Work, excluding any source code
for portions of the Combined Work that, considered in isolation, are
based on the Application, and not on the Linked Version.

  The "Corresponding Application Code" for a Combined Work means the
object code and/or source code for the Application, including any data
and utility programs needed for reproducing the Combined Work from the
Application, but excluding the System Libraries of the Combined Work.

  1. Exception to Section 3 of the GNU GPL.

  You may convey a covered work under sections 3 and 4 of this License
without being bound by section 3 of the GNU GPL.

  2. Conveying Modified Versions.

  If you modify a copy of the Library, and, in your modifications, a
facility refers to a function or data to be supplied by an Application
that uses the facility (other than as an argument passed when the
facility is invoked), then you may convey a copy of the modified
version:

   a) under this License, provided that you make a good faith effort to
   ensure that, in the event an Application does not supply the
   function or data, the facility still operates, and performs
   whatever part of its purpose remains meaningful, or

   b) under the GNU GPL, with none of the additional permissions of
   this License applicable to that copy.

  3. Object Code Incorporating Material from Library Header Files.

  The object code form of an Application may incorporate material from
a header file that is part of the Library.  You may convey such object
code under terms of your choice, provided that, if the incorporated
material is not limited to numerical parameters, data structure
layouts and accessors, or small macros, inline functions and templates
(ten or fewer lines in length), you do both of the following:

   a) Give prominent notice with each copy of the object code that the
   Library is used in it and that the Library and its use are
   covered by this License.

   b) Accompany the object code with a copy of the GNU GPL and this license
   document.

  4. Combined Works.

  You may convey a Combined Work under terms of your choice that,
taken together, effectively do not restrict modification of the
portions of the Library contained in the Combined Work and reverse
engineering for debugging such modifications, if you also do each of
the following:

   a) Give prominent notice with each copy of the Combined Work that
   the Library is used in it and that the Library and its use are
   covered by this License.
```

(continues on next page)

(continued from previous page)

```
   b) Accompany the Combined Work with a copy of the GNU GPL and this license
   document.

   c) For a Combined Work that displays copyright notices during
   execution, include the copyright notice for the Library among
   these notices, as well as a reference directing the user to the
   copies of the GNU GPL and this license document.

   d) Do one of the following:

       0) Convey the Minimal Corresponding Source under the terms of this
       License, and the Corresponding Application Code in a form
       suitable for, and under terms that permit, the user to
       recombine or relink the Application with a modified version of
       the Linked Version to produce a modified Combined Work, in the
       manner specified by section 6 of the GNU GPL for conveying
       Corresponding Source.

       1) Use a suitable shared library mechanism for linking with the
       Library.  A suitable mechanism is one that (a) uses at run time
       a copy of the Library already present on the user's computer
       system, and (b) will operate properly with a modified version
       of the Library that is interface-compatible with the Linked
       Version.

   e) Provide Installation Information, but only if you would otherwise
   be required to provide such information under section 6 of the
   GNU GPL, and only to the extent that such information is
   necessary to install and execute a modified version of the
   Combined Work produced by recombining or relinking the
   Application with a modified version of the Linked Version. (If
   you use option 4d0, the Installation Information must accompany
   the Minimal Corresponding Source and Corresponding Application
   Code. If you use option 4d1, you must provide the Installation
   Information in the manner specified by section 6 of the GNU GPL
   for conveying Corresponding Source.)

  5. Combined Libraries.

  You may place library facilities that are a work based on the
Library side by side in a single library together with other library
facilities that are not Applications and are not covered by this
License, and convey such a combined library under terms of your
choice, if you do both of the following:

   a) Accompany the combined library with a copy of the same work based
   on the Library, uncombined with any other library facilities,
   conveyed under the terms of this License.

   b) Give prominent notice with the combined library that part of it
   is a work based on the Library, and explaining where to find the
   accompanying uncombined form of the same work.

  6. Revised Versions of the GNU Lesser General Public License.

  The Free Software Foundation may publish revised and/or new versions
```

(continues on next page)

(continued from previous page)

```
of the GNU Lesser General Public License from time to time. Such new
versions will be similar in spirit to the present version, but may
differ in detail to address new problems or concerns.

  Each version is given a distinguishing version number. If the
Library as you received it specifies that a certain numbered version
of the GNU Lesser General Public License "or any later version"
applies to it, you have the option of following the terms and
conditions either of that published version or of any later version
published by the Free Software Foundation. If the Library as you
received it does not specify a version number of the GNU Lesser
General Public License, you may choose any version of the GNU Lesser
General Public License ever published by the Free Software Foundation.

  If the Library as you received it specifies that a proxy can decide
whether future versions of the GNU Lesser General Public License shall
apply, that proxy's public statement of acceptance of any version is
permanent authorization for you to choose that version for the
Library.
```

PYTHON MODULE INDEX

INDEX