ruediPy documentaion

Matthias Brennwald

Version June 6, 2016

Abstract

ruediPy is a collection of Python programs for instrument control and data acquisition using RUEDI instruments(?). ruediPy also includes some GNU Octave (or Matlab) tools to load, process, and manipulate RUEDI data acquired with ruediPy Python classes.

ruediPy is distributed as free software under the GNU General Public License (see LICENSE.txt).

This document describes the ruediPy software only. The RUEDI instrument is described in a separate document(?).

1 Overview

ruediPy is a collection of Python programs for instrument control and data acquisition using RUEDI instruments. ruediPy also includes some GNU Octave (or Matlab) tools to load, process, and manipulate RUEDI data acquired with ruediPy Python classes. The RUEDI instrument itself is described in a separate document(?).

The Python classes for instrument control and data acquisition are designed to reflect the different hardware units of a RUEDI instrument, such as the mass spectrometer, selector valve, or probes for total gas pressure or temperature. These classes, combined with additional helper classes (e.g., for data file handling), allow writing simple Python scripts that perform user-defined procedures for a specific analysis task.

The GNU Octave tools (m-files) are designed to work hand-in-hand with the data files produced by the data acquisition parts of the Python classes. ★[1]

ruediPy is developed on Linux and Mac OS X systems, but should also work on any other system that run Python and GNU Octave.

[1]TO DO: expand this: load raw data, process / calibrate data, etc.

2 Obtaining and installing `ruediPy`

`ruediPy` can be downloaded from `http://brennmat.github.io/ruediPy` either as a compressed archive file, or using Subversion or Git version control systems. `ruediPy` can be installed to just about any directory on the computer that is used for instrument control – but the user home directory (`~/ruediPy`) may seem like a sensible choice, and that's what is assumed throughout the examples shown in this manual.

3 Python classes

The Python classes are used to control the various hardware units of the RUEDI instruments, to acquire measurement data, and to write these data to well-formatted and structured data files.

Currently, the following classes are implemented:

- `rgams_SRS.py`: control and data acquisition from the SRS mass spectrometer
- `selectorvalve_VICI.py`: control of the VICI inlet valve
- `pressuresensor_WIKA.py`: control and data acquisition from the WIKA pressure sensor
- `datafile.py`: data file handling
- `misc.py`: helper functions

The Python class files are located at `~/ruediPy/python/classes/`. To make sure Python knows where to find the `ruediPy` Python classes, set your `PYTHONPATH` environment variable accordingly.[2]

These classes are continuously expanded and new classes are added to `ruediPy` as required by new needs or developments of the RUEDI instruments. The various methods / functions included are documented in the class files. Due to the ongoing development of the code, it seems futile to keep an up-to-date copy of the methods / functions documentation in this manual. Please refer to the detailed documentation in the class files directly.

[2]A convenient method to achieve this on Linux or similar UNIXy systems is to put the following line to the `.profile` file: `export PYTHONPATH=~/ruediPy/python`

3.1 Python classes reference

3.1.1 Class `selectorvalve_VICI`

`ruediPy/python/classes/selectorvalve_VICI.py`

No class description available.

Method `getpos`

```
pos = selectorvalve_VICI.getpos()

Get valve position

INPUT:
(none)

OUTPUT:
pos:  valve postion (integer)
```

Method `label`

```
label = selectorvalve_VICI.label()

Return label / name of the SELECTORVALVE object

INPUT:
(none)

OUTPUT:
label:  label / name (string)
```

Method `setpos`

```
selectorvalve_VICI.setpos(val,f)

Set valve position
```

```
INPUT:
val:  new valve position (integer)
f:  datafile object for writing data (see datafile.py).  If f = 'nofile',
data is not written to any data file.

OUTPUT:
(none)
```

Method `warning` No method description available.

3.1.2 Class `pressuresensor_WIKA`

`ruediPy/python/classes/pressuresensor_WIKA.py`

No class description available.

Method `label`

```
label = pressuresensor_WIKA.label()

Return label / name of the PRESSURESENSOR object

INPUT:
(none)

OUTPUT:
label:  label / name (string)
```

Method `pressure`

```
press,unit = pressuresensor_WIKA.pressure(f)

Read out current pressure value (in hPa).
```

```
INPUT:
f:  file object for writing data (see datafile.py).  If f = 'nofile',
data is not written to any data file.

OUTPUT:
press:  pressure value in hPa (float)
```

Method `serial_checksum`

```
cs = pressuresensor_WIKA.serial_checksum( cmd )

Return checksum used for serial port communication with WIKA pressure
sensor.

INPUT:
cmd:  serial-port command string without checksum

OUTPUT:
cs:  checksum byte
```

Method `warning`

```
pressuresensor_WIKA.warning(msg)

Issue warning about issues related to operation of pressure sensor.

INPUT:
msg:  warning message (string)

OUTPUT:
(none)
```

3.1.3 Class `datafile`

`ruediPy/python/classes/datafile.py`

ruediPy class for handling of data files.

Method `basepath`

```
pat = datafile.basepath()

Return the base path where datafiles are stored

INPUT:
(none)

OUTPUT:
pat:  datafile base path (string)
```

Method `close`

```
datafile.close()

Close the currently open data file (if any)

INPUT:
(none)

OUTPUT:
(none)
```

Method `fid`

```
f = datafile.fid()

Return the file ID / object of the current file

INPUT:
```

```
(none)

OUTPUT:
f:  datafile object
```

Method `label`

```
lab = datafile.label()

Return label / name of the DATAFILE object

INPUT:
(none)

OUTPUT:
lab:  label / name (string)
```

Method `name`

```
n = datafile.name()

Return the name the current file (or empty string if not datafile has
been created)

INPUT:
(none)

OUTPUT:
n:  ile name (string)
```

Method `next`

```
datafile.next()
```

Close then current data file (if it’s still open) and start a new file.

INPUT:
typ (optional): string that will be appended to the file name. This may be useful to indicate ’type’ of mesurement data, e.g. typ = ’SAMPLE’, typ = ’S’, typ = ’BLANK’, typ = ’B’, typ = ’CAL’, typ = ’C’, etc.). The string can be anything. If omitted or typ = ’’, nothing will be appended to the file name

OUTPUT:
(none)

Method `warning`
`datafile.warning(msg)`

Warn about issues related to DATAFILE object

INPUT:
msg: warning message (string)

OUTPUT:
(none)

Method `writeComment`
`datafile.writeComment(caller,cmt)`

Write COMMENT line to the data file.

INPUT:
caller: label / name of the calling object (string)
cmt: comment string

OUTPUT:
(none)

Method `writePeak`

```
datafile.writePeak(caller,mz,intensity,unit,det,gate,timestmp)

Write PEAK data line to the data file.

INPUT:
caller:  type of calling object, i.e.  the "data origin" (string)
label:  name/label of the calling object (string)
mz:  mz value (integer)
intensity:  peak intensity value (float)
unit:  unit of peak intensity value (string)
det:  detector (string), e.g., det='F' for Faraday or det='M' for multiplier
gate:  gate time (float)
timestmp:  timestamp of the peak measurement (see misc.nowUNIX)

OUTPUT:
(none)
```

Method `writeScan`

```
datafile.writeScan(caller,mz,intensity,unit,det,gate,timestmp)

Write PEAK data line to the data file.

INPUT:
caller:  type of calling object, i.e.  the "data origin" (string)
label:  name/label of the calling object (string)
mz:  mz values (floats)
intensity:  intensity values (floats)
unit:  unit of intensity values (string)
det:  detector (string), e.g., det='F' for Faraday or det='M' for multiplier
gate:  gate time (float)
timestmp:  timestamp of the peak measurement (see misc.nowUNIX)
```

```
OUTPUT:
(none)
```

Method `writeValvePos`

```
datafile.writeValvePos(caller,position,timestmp)

Write multi-port valve position data line to the data file.

INPUT:
caller:  type of calling object, i.e.  the "data origin" (string)
label:  name/label of the calling object (string)
position:  valve position (integer)
timestmp:  timestamp of the peak measurement (see misc.nowUNIX)

OUTPUT:
(none)
```

Method `writeZero`

```
datafile.writeZero(caller,mz,mz_offset,intensity,unit,det,gate,timestmp)

Write ZERO data line to the data file.

INPUT:
caller:  type of calling object, i.e.  the "data origin" (string)
label:  name/label of the calling object (string)
mz:  mz value (integer)
mz_offset:  mz offset value (integer, positive offset corresponds to
higher mz value)
intensity:  zero intensity value (float)
unit:  unit of peak intensity value (string)
det:  detector (string), e.g., det='F' for Faraday or det='M' for multiplier
gate:  gate time (float)
timestmp:  timestamp of the peak measurement (see misc.nowUNIX)
```

```
OUTPUT:
(none)
```

Method `write_analysis_type`

```
datafile.write_analysis_type( caller , typ , timestmp )

Write ANALYSIS TYPE info line to the data file.

INPUT:
caller:  type of calling object, i.e.  the "data origin" (string)
typ:  analysis type (string / char)
timestmp:  timestamp of the peak measurement (see misc.nowUNIX)

OUTPUT:
(none)
```

Method `write_pressure`

```
datafile.write_pressure(caller,label,value,unit,timestmp)

Write PRESSURE data line to the data file.

INPUT:
caller:  type of calling object, i.e.  the "data origin" (string)
label:  name/label of the calling object (string)
value:  pressure value (float)
unit:  unit of peak intensity value (string)
timestmp:  timestamp of the peak measurement (see misc.nowUNIX)

OUTPUT:
(none)
```

Method `writeln`

```
datafile.writeln(caller,identifier,data,timestmp)

Write a text line to the data file (format:  TIMESTAMP CALLER[LABEL]
IDENTIFIER: DATA). CALLER, LABEL, and IDENTIFIER should not contain spaces
or similar white space (will be removed before writing to file).  If
LABEL == '' or LABEL == CALLER, the [LABEL] part is omitted.

INPUT:
caller:  type of calling object, i.e.  the "data origin" (string)
label:  name/label of the calling object (string)
identifier:  data type identifier (string)
data:  data / info string
timestmp:  timestamp of the data in unix time (see misc.nowUNIX)

OUTPUT:
(none)
```

3.1.4 Class `misc`

`ruediPy/python/classes/misc.py`

No class description available.

Method `now_UNIX`

```
dt = misc.now_UNIX()

Return date/time as UNIX time / epoch (seconds after Jan 01 1970 UTC)

INPUT:
(none)

OUTPUT:
dt:  date-time (UNIX / epoch time)
```

Method `now_string`

```
dt = misc.now_string()

Return string with current date and time

INPUT:
(none)

OUTPUT:
dt:  date-time (string) in YYYY-MM-DD hh:mm:ss format
```

Method `warnmessage`

```
misc.warnmessage(caller,msg)

Print a warning message

INPUT:
caller:  caller label / name of the calling object (string)
msg:  warning message

OUTPUT:
(none)
```

4 GNU Octave tools

★[3]

5 Examples

★[4]

[3]TO DO: add content
[4]TO DO: add content

References