labdrivers Documentation

Release 0.1

Mason Lab

Sep 11, 2018

Contents

labdrivers is a Python module containing a collection of drivers for common research lab instruments.

It contains a suite of instrument-specific drivers which can be used to interface measurement hardware with Python code, along with a set of Jupyter notebooks demonstrating example use cases. The drivers within the project are intended to be used 'manually', either in python scripts or in Jupyter notebooks.

Labdrivers is not a measurement framework with a GUI; if that's what you're looking for then you might want to check out one of the projects listed at https://github.com/pyinstruments/pyinstruments.

CHAPTER 1

Organization of the project

Drivers within the project are organized by instrument brand. Each instrument manufacturer has its own submodule under the `labdrivers` namespace, and each individual instrument driver is in a file named after the class which implements the driver.

Currently labdrivers contains drivers for the following instruments:

```
labdrivers/
|-- keithley
    `-- keithley2400.py
|-- lakeshore
    `-- ls332.py
|-- ni
    `-- nidaq.py
|-- oxford
    |-- itc503.py
    |-- ips120.py
    |-- mercuryips.py
    `-- triton200.py
|-- quantumdesign
    `-- qdinstrument.py
|-- srs
    `-- sr830.py
```

So for example, to load the driver for a Keithley 2400 SMU:

```
from labdrivers.keithley import keithley2400
```

CHAPTER 2

Documentation Table of Contents

2.1 Installation

Before attempting to use the drivers for the Quantum Design instruments, please be sure that you are using a version of Python which is 3.6.x and below. Additionally, please be sure that .NET 4.0+ is installed, as that allows for access to the classes within the compiled C# code.

There are a couple ways to install labdrivers.

2.1.1 Installing to the system Python Folder

To install labdrivers to the default Python installation you can use *pip*:

```
> pip install labdrivers
```

Note that this requires *git*, which can be found here: https://git-scm.com/.

If you are running an outdated version of *labdrivers*, please upgrade:

```
> pip install --upgrade labdrivers
```

If you prefer to install manually, download the labdrivers package from here: https://github.com/masonlab/labdrivers, then copy the *labdrivers* folder to:

```
(system python path)/lib/site-packages/
```

2.2 Using the drivers

For examples of how to use the various instrument drivers please see the example Jupyter notebooks included in the github repository (link).

2.3 Driver APIs

2.3.1 labdrivers.keithley

class labdrivers.keithley.keithley2400.**Keithley2400**(*gpib_addr=23*)

abort_cycle()
Aborts the source or measure cycle, bringing the SourceMeter back into an idle state.

buffer_memory_status()
Check buffer memory status.

clear_status()
Clears all event registers and Error Queue.

clear_trace()
Clear the buffer.

current_compliance
Sets or gets the current compliance level in Amperes.

disable_buffer()
Disables the buffer.

expected_current_reading
Gets or sets the expected current reading from the device under test.

expected_ohms_reading
Gets or sets the expected range of a resistance reading from the device under test.

expected_voltage_reading
Gets or sets the expected voltage reading from the device under test.

fill_buffer()
Fill buffer and stop.

four_wire_sensing
Gets the status of or sets four-wire sensing.

Expected booleans for setting: True, False.

gpib_addr
Returns the GPIB address of the Keithley 2400 Sourcemeter.

identify()
Returns manufacturer, model number, serial number, and firmware revision levels.

initiate_cycle()
Initiates source or measure cycle, taking the SourceMeter out of an idle state.

measure_type
The type of measurement the Keithley 2400 SourceMeter will make.

Expected strings for setting: 'voltage', 'current', 'resistance'

num_readings_in_buffer
Gets the number of readings that are stored in the buffer.

output
Gets or sets the source output of the Keithley 2400.

Expected input: boolean

Returns boolean

output_off_mode
Gets or sets the output mode when the output is off.

Expected input strings: 'himp', 'normal', 'zero', 'guard'

Returns description of the output's off mode

ramp_to_setpoint(*setpoint: float*, *step: float*, *wait: float*)

ramp_to_zero()

read(**measurements*)
Reads data from the Keithley 2400. Equivalent to the command :INIT; :FETCH?

Multiple string arguments may be used. For example:

```
keithley.read('voltage', 'current')
keithley.read('time')
```

The first line returns a list in the form [voltage, current] and the second line returns a list in the form [time].

Note: The returned lists contains the values in the order that you requested.

Parameters ***measurements** (*str*) – Any number of arguments that are from: 'voltage', 'current', 'resistance', 'time'

Return list measure_list A list of the arithmetic means in the order of the given arguments

Return list measure_stdev_list A list of the standard deviations (if more than 1 measurement) in the order of the given arguments

read_trace()
Read contents of buffer.

reset_to_defaults()
Resets to defaults of Sourcemeter.

resistance_ohms_mode
Gets or sets the resistance mode.

Expected strings for setting: 'manual', 'auto'

send_bus_trigger()
Sends a bus trigger to SourceMeter.

source_mode
Gets or sets the mode of the source.

Expected strings for setting: 'fixed', 'sweep', 'list'

source_type
Gets or sets the source type of the Keithley 2400 SourceMeter.

Expected strings for setting: 'voltage', 'current'

source_value
Get or set the numeric value of the source chosen from Keithley2400.source_type.

sweep_center
To be implemented.

sweep_direction
To be implemented.

sweep_end
To be implemented.

sweep_points
To be implemented.

sweep_ranging
To be implemented.

sweep_scale
To be implemented.

sweep_span
To be implemented.

sweep_start
To be implemented.

trace_delay
The amount of time the SourceMeter waits after the trigger to perform Device Action.

trace_feed_source(*value*)
Sets the source of the trace feed.

Expected strings: 'sense', 'calculate1', 'calculate2'

trace_points
Gets or sets the size of the buffer

Expected integer value range: 1 <= n <= 2500

trigger
Gets or sets the type of trigger to be used.

Expected strings for setting: 'immediate', 'tlink', 'timer', 'manual', 'bus', 'nst', 'pst', 'bst' (see source code for other possibilities)

trigger_count
Gets or sets the number of triggers

Expected integer value range: 1 <= n <= 2500

voltage_compliance
Gets or sets the voltage compliance.

Expected range of floats: 200e-6 <= x <= 210

within_current_compliance()
Queries if the measured current is within the set compliance.

Returns boolean

within_voltage_compliance()
Queries if the measured voltage is within the set compliance.

Returns boolean

2.3.2 labdrivers.lakeshore

class labdrivers.lakeshore.ls332.**Ls332**(*gpib_addr=12*)

CHANNELS = ('A', 'B')

channel
Gets or sets the current channel of the LS332.

Setting expects the strings: 'A', 'B'

Returns The channel 'A' or 'B'

reset()
Resets device to power-up settings.

temperature
Reads the temperature of the currently-set channel.

temperature_setpoint
Reads the temperature set point of the currently-set channel.

2.3.3 labdrivers.oxford

class labdrivers.oxford.itc503.**Itc503**(*gpib_addr=24*)
Module to connect to an ITC 503.

Modes supported: GPIB

Parameters **gpib_addr** – GPIB address of the ITC 503

getValue(*variable=0*)
Read the variable defined by the index.

The possible inputs are:

```
0: SET TEMPERATURE
1: SENSOR 1 TEMPERATURE
2: SENSOR 2 TEMPERATURE
3: SENSOR 3 TEMPERATURE
4: TEMPERATURE ERROR
5: HEATER O/P (as %)
6: HEATER O/P (as V)
7: GAS FLOW O/P (a.u.)
8: PROPORTIONAL BAND
9: INTEGRAL ACTION TIME
10: DERIVATIVE ACTION TIME
```

Parameters **(int)** (*variable*) – Index of variable to read.

setAutoControl(*auto_manual=0*)
Sets automatic control for heater/gas(needle valve).

Value:Status map 0: heater manual, gas manual 1: heater auto , gas manual 2: heater manual, gas auto 3: heater auto , gas auto

Parameters **auto_manual** – Index for gas/manual.

setControl(*unlocked=1*, *remote=1*)
Set the LOCAL / REMOTE control state of the ITC 503

Parameters

- **(int)** (*remote*) – 0 to lock, 1 to unlock
- **(int)** – 0 for local, 1 for remote

Returns None

setDerivative(*derivative=0*)
Sets the derivative action time.

Parameters **derivative** – Derivative action time. Ranges from 0 to 273 minutes.

setGasOutput(*gas_output=0*)
Sets the gas (needle valve) output level.

Parameters **gas_output** – Sets the percent of the maximum gas output in units of 0.1%. Min: 0. Max: 999.

setHeaterOutput(*heater_output=0*)
Sets the heater output level.

Parameters **heater_output** – Sets the percent of the maximum heater output in units of 0.1%. Min: 0. Max: 999.

setHeaterSensor(*sensor=1*)
Selects the heater sensor.

Parameters **sensor** – Should be 1, 2, or 3, corresponding to the heater on the front panel.

setIntegral(*integral=0*)
Sets the integral action time.

Parameters **integral** – Integral action time, in steps of 0.1 minute. Ranges from 0 to 140 minutes.

setProportional(*prop=0*)
Sets the proportional band.

Parameters **(float)** (*prop*) – Proportional band, in steps of 0.0001K.

setSweeps(*sweep_parameters*)
Sets the parameters for all sweeps.

This fills up a dictionary with all the possible steps in a sweep. If a step number is not found in the sweep_parameters dictionary, then it will create the sweep step with all parameters set to 0.

Parameters **sweep_parameters** – A dictionary whose keys are the step numbers (keys: 1-16). The value of each key is a dictionary whose keys are the parameters in the sweep table (see _setSweepStep).

setTemperature(*temperature=0.01*)
Change the temperature set point.

Parameters **(float)** (*temperature*) – Temperature set point in Kelvin (default: 0.010)

class labdrivers.oxford.ips120.**Ips120**(*GPIBaddr*)

readField()
Read the current magnetic field in Tesla

Returns current magnetic field in Tesla

Return type field(float)

readFieldSetpoint()
Read the current set point for the magnetic field in Tesla

Returns current set point for the magnetic field in Tesla

Return type setpoint(float)

readFieldSweepRate()
Read the current magnetic field sweep rate in Tesla/min

Returns current magnetic field sweep rate in Tesla/min

Return type sweep_rate(float)

setActivity(*state=1*)
Set the field activation method

0 - Hold 1 - To Set Point 2 - To Zero 3 - Clamp (clamp the power supply output)

Parameters **state** (*int*) – the field activation method

setControl(*state=3*)
Set the LOCAL / REMOTE control state of the IPS 120-10

0 - Local & Locked (default state) 1 - Remote & Locked 2 - Local & Unlocked 3 - Remote & Locked

Parameters **state** (*int*) – the state in which to place the IPS 120-10

setDisplay(*display*)
Set the display to show amps or tesla

Parameters **display** (*str*) – One of ['amps','tesla']

setFieldSetpoint(*field*)
Set the magnetic field set point, in Tesla

Parameters **field** (*float*) – the magnetic field set point, in Tesla

setFieldSweepRate(*rate*)
Set the magnetic field sweep rate, in Tesla/min

Parameters **rate** (*float*) – the magnetic field sweep rate, in Tesla/min

setHeater(*state=1*)
Set the switch heater activation state

0 - Heater Off (close switch) 1 - Heater On if PSU=Magnet (open switch) 2 - Heater On, no checks (open switch)

Parameters **state** (*int*) – the switch heater activation state

waitForField(*timeout=600*, *error_margin=0.01*)
Wait for the field to reach the set point

Parameters

- **timeout** (*int*) – maximum time to wait, in seconds
- **error_margin** (*float*) – how close the field needs to be to the set point, in tesla

Returns whether the field set point was reached

Return type (bool)

class labdrivers.oxford.mercuryips.**MercuryIps**(*mode='ip'*, *resource_name=None*, *ip_address=None*, *port=7020*, *timeout=10.0*, *bytes_to_read=2048*)

class Magnet(*axis*, *mode='ip'*, *resource_name=None*, *ip_address=None*, *port=7020*, *timeout=10.0*, *bytes_to_read=2048*)
Constructor for a magnet along a certain axis.

Parameters

- **axis** (*string*) – The axis for the magnet, given by ['GRPX'|'GRPY'|'GRPZ']
- **mode** (*string*) – Connection, given by ['ip'|'visa']
- **resource_name** (*string*) – VISA resource name of the MercuryIPS
- **ip_address** (*string*) – IP address of the MercuryIPS
- **port** (*integer*) – Port number of the Mercury iPS
- **timeout** (*float*) – Time to wait for a response from the MercuryIPS before throwing an error.
- **bytes_to_read** (*integer*) – Amount of information to read from a response

QUERY_AND_RECEIVE = {'ip': <function MercuryIps.Magnet.query_ip at 0x7f9249f17ae8>

clamp()
Puts a magnet in a CLAMP state.

clamped()
Queries if magnet is in a CLAMP state.

current_ramp_rate
The ramp rate of the current for a magnet in Amperes per minute.

current_setpoint
The set point of the current for a magnet in Amperes.

static extract_value(*response*, *noun*, *unit*)
Finds the value that is contained within the response to a previously sent query.

Parameters
- **response** (*string*) – The response from a query.
- **noun** – The part of the query that refers to the NOUN (refer to MercuryIPS documentation).
- **unit** – The measurement unit (e.g. K for Kelvin, T for Tesla).

Returns float The value of the response, but without units.

field_ramp_rate
The magnetic field ramp rate in Tesla per minute along the magnet axis.

field_setpoint
The magnetic field set point in Tesla

hold()
Puts a magnet in a HOLD state.

This action does one of the following: 1) Stops a ramp 2) Allows the field and current to ramp

holding()
Queries if magnet is in a HOLD state.

magnetic_field
Gets the magnetic field.

query_ip(*command*)
Sends a query to the MercuryIPS via ethernet.

Parameters **command** (*string*) – The command, which should be in the NOUN + VERB format

Returns str The MercuryIPS response

query_visa(*command*)
Sends a query to the MercuryIPS via VISA.

Parameters **command** (*string*) – The command, which should be in the NOUN + VERB format

Returns str The MercuryIPS response

ramp_to_setpoint()
Ramps a magnet to the setpoint.

ramp_to_zero()
Ramps a magnet from its current magnetic field to zero field.

ramping()
Queries if magnet is ramping.

circle_sweep(*field_radius*, *number_points*)

class labdrivers.oxford.triton200.**Triton200**(*ip_address*, *port_number=33576*, *timeout=10000*, *bytes_to_read=2048*)

Create an instance of the Triton200 class.

Supported modes: IP

Parameters

- **ip_address** (*str*) – The IP address of the Triton 200.
- **port_number** (*int*) – The associated port number of the Triton 200 (default: 33576)
- **timeout** (*int*) – How long to wait for a response (default: 10000)
- **bytes_to_read** (*int*) – How many bytes to accept from the response (default: 2048)

controlled_ramp_off()
Stops a temperature sweep for the current temperature channel.

controlled_ramp_on()
Starts a temperature sweep for the current temperature channel.

static extract_value(*response*, *noun*, *unit*)

query_and_receive(*command*)
Queries the Oxford Triton 200 with the given command.

Parameters **command** – Specifies a read/write of a property.

temperature
The temperature reading from the current temperature channel.

temperature_channel
returns str – The temperature channel, either the cernox (5) or the RuO2 (6)

temperature_setpoint

turbo_off()
Turns off a turbo pump.

WARNING: Do not use this unless you know what you are doing.

turbo_on()
Turns on a turbo pump.

WARNING: Do not use this unless you know what you are doing.

update_heater()
Associates the heater with the current temperature channel and changes the heater current to preset values given the temperature set point.

2.3.4 labdrivers.quantumdesign

class labdrivers.quantumdesign.qdinstrument.**QdInstrument**(*instrument_type*, *ip_address*, *remote=True*)

A class to interface with Quantum Design instruments.

This class is a thin wrapper around the C# QuantumDesign.QDInstrument.QDInstrumentBase class provided in the QDInstrument.dll file.

There is now support for using a Quantum Design DynaCool, PPMS, SVSM, and VersaLab. The MPMS class requires testing, but should work. For some reason, the MPMS enum was hard-coded in as the number 4121982 casted as a QDInstrumentBase.QDInstrumentType enum.

getField()

Returns the Magnetic field in Gauss.

Parameters are from: GetField(ref double Field, ref QDInstrumentBase.FieldStatus Status)

Returns Field in Gauss.

getPosition()

Retrieves the position of the rotator.

GetPosition(string Axis, ref double Position, ref QDInstrumentBase.PositionStatus Status)

"Horizontal Rotator" seems to be the name that one should pass to GetPosition, as observed in the WaitConditionReached function.

getTemperature()

Returns the instrument temperature in Kelvin.

Parameters are from: SetTemperature(double Temperature, double Rate, QDInstrumentBase.TemperatureApproach Approach)

setField(*field*, *rate=200*)

Ramps the instrument magnetic field to the set point.

Parameters are from: SetField(double Field, double Rate, QDInstrumentBase.FieldApproach Approach, QDInstrumentBase.FieldMode Mode)

Parameters

- **field** – Set point of the applied magnetic field in Gauss.
- **rate** – Ramp rate of the applied magnetic field in Gauss/sec.

Returns None

setPosition(*position*, *speed*)

Ramps the instrument position to the set point.

Parameters are from: SetPosition(string Axis, double Position, double Speed, QDInstrumentBase.PositionMode Mode)

Parameters

- **position** – Position on the rotator to move to.
- **speed** – Rate of change of position on the rotator.

setTemperature(*temp*, *rate=10*)

Ramps the instrument temperature to the set point.

Parameters are from: GetTemperature(ref double Temperature, ref QDInstrumentBase.TemperatureStatus Status)

Parameters

- **temp** – Desired temperature in Kelvin
- **rate** – Temperature ramp rate in Kelvin/min.

Returns None

waitForField(*delay=5*, *timeout=600*)
Prevents other processes from executing while the QD instrument magnetic field is settling down.

Parameters

- **delay** – Length of time to wait after wait condition achieved in seconds.
- **timeout** – Length of time to wait to achieve wait condition in seconds.

Returns 0 when complete.

waitForPosition(*delay*, *timeout*)
Prevents other processes from executing while the QD instrument rotator position is settling down.

Parameters

- **delay** – Length of time to wait after wait condition achieved in seconds.
- **timeout** – Length of time to wait to achieve wait condition in seconds.

Returns 0 when complete.

waitForTemperature(*delay=5*, *timeout=600*)
Prevents other processes from executing while the QD instrument temperature is settling down.

Parameters

- **delay** – Length of time to wait after wait condition achieved in seconds.
- **timeout** – Length of time to wait to achieve wait condition in seconds.

Returns 0 when complete.

class labdrivers.quantumdesign.qdinstrument.**Dynacool**(*ip_address*)
QdInstrument subclass that connects to the Quantum Design PPMS DynaCool.

class labdrivers.quantumdesign.qdinstrument.**Ppms**(*ip_address*)
QdInstrument subclass that connects to the Quantum Design PPMS.

class labdrivers.quantumdesign.qdinstrument.**Svsm**(*ip_address*)
QdInstrument subclass that connects to the Quantum Design SVSM.

class labdrivers.quantumdesign.qdinstrument.**VersaLab**(*ip_address*)
QdInstrument subclass that connects to the Quantum Design VersaLab.

class labdrivers.quantumdesign.qdinstrument.**Mpms**(*ip_address*)
QdInstrument subclass that connects to the Quantum Design MPMS.

2.3.5 labdrivers.srs

class labdrivers.srs.sr830.**Sr830**(*gpib_addr*)
Interface to a Stanford Research Systems 830 lock in amplifier.

amplitude
The amplitude of the voltage output.

auto_gain()
Mimics pressing the Auto Gain button. Does nothing if the time constant is more than 1 second.

auto_offset(*parameter*)
Automatically offsets the given voltage parameter.

Parameters **parameter** – A string from ['x'|'y'|'r'], case insensitive.

auto_phase()
Mimics pressing the Auto Phase button.

auto_reserve()
Mimics pressing the Auto Reserve button.

data_sample_rate
Data sample rate, which can be 62.5 mHz, 512 Hz, or Trigger.

Expected strings: 62.5, 62.5 mhz, 62.5mhz, mhz, 0, 512, 512hz, 512 hz, hz, 13, trig, trigger, 14.

data_scan_mode
Data scan mode, which is either a 1-shot or a loop.

Expected strings: 1-shot, 1 shot, 1shot, loop.

frequency
The frequency of the output signal.

get_display(*channel*)
Get the display configuration of the amplifier.

Display options are: (for channel 1) (for channel 2)

0: X 0: Y 1: R 1: Theta 2: X Noise 2: Y Noise 3: Aux in 1 3: Aux in 3 4: Aux in 2 4: Aux in 4

Parameters **channel** (*int*) – which channel to return the configuration for

Returns the parameter being displayed by the amplifier

Return type int

input
**The input on the SR830 machine. Possible values* – 0* – A 1: A-B 2: I (1 MOhm) 3: I (100 MOhm)

input_shield_grounding
Tells whether the shield is floating or grounded.

low_pass_filter_slope
The low pass filter slope in units of dB/octave. The choices are – i slope(dB/oct) — ————-

0 6 1 12 2 18 3 24

multiple_output(**values*)
Queries the SR830 for multiple output. See below for possibilities.

Possible parameters are: 1: X 2: Y 3: R 4: Theta 5: Aux in 1 6: Aux in 2 7: Aux in 3 8: Aux in 4 9: Reference frequency 10: CH1 display 11: CH2 display

Parameters **values** – A variable number of arguments to obtain output

Returns

pause_scan()
Pauses a scan.

phase
The phase of the output relative to the input.

reserve
The reserve mode of the SR830.

reset_scan()
Resets a scan and releases all stored data.

sensitivity
Voltage/current sensitivity for inputs.

set_display(*channel*, *display*, *ratio=0*)
Set the display of the amplifier.

Display options are: (for channel 1) (for channel 2)

0: X 0: Y 1: R 1: Theta 2: X Noise 2: Y Noise 3: Aux in 1 3: Aux in 3 4: Aux in 2 4: Aux in 4

Ratio options are (i.e. divide output by): 0: none 0: none 1: Aux in 1 1: Aux in 3 2: Aux in 2 2: Aux in 4

Parameters

- **channel** (*int*) – which channel to modify (1 or 2)
- **display** (*int*) – what to display
- **ratio** (*int, optional*) – display the output as a ratio

single_output(*value*)
Get the current value of a single parameter. Possible parameter values are:

1: X 2: Y 3: R 4: Theta

Returns the value of the specified parameter

Return type float

start_scan()
Starts or continues a scan.

sync_filter
The state of the sync filter (< 200 Hz).

time_constant
The time constant of the SR830.

trigger()
Sends a software trigger.

trigger_starts_scan
Determines if a Trigger starts scan mode.

2.3.6 labdrivers.ni

class labdrivers.ni.nidaq.**Nidaq**(*device='Dev1'*)
Class to interface with a National Instruments DAQ

Built on top of PyDAQmx.

Reference: NI-DAQmx C Reference Help

output_current(*channel*, *output=0*)
Outputs a current from the NI DAQ.

Parameters

- **channel** – The name of the channel for output.
- **output** – The value of the output in Amps

Returns None

output_voltage(*channel*, *output=0*)
Outputs a voltage from the NI DAQ.

Parameters

- **channel** – The name of the channel for output.
- **output** – The value of the output in Volts

Returns None

read_current(*channel*, *minVal=-10.0*, *maxVal=10.0*)
Read the current input level of *channel*

Using narrow bounds (with minVal and maxVal) will improve accuracy since the same digital resolution will be applied to a smaller analog range.

Parameters

- **channel** (*str*) – the name of the channel to use
- **minVal** (*float*) – the minimum value you expect to measure
- **maxVal** (*float*) – the maximum value you expect to measure

read_voltage(*channel*, *minVal=-10.0*, *maxVal=10.0*)
Read the voltage input level of *channel*

Using narrow bounds (with minVal and maxVal) will improve accuracy since the same digital resolution will be applied to a smaller analog range.

Parameters

- **channel** (*str*) – the name of the channel to use
- **minVal** (*float*) – the minimum value you expect to measure
- **maxVal** (*float*) – the maximum value you expect to measure

reset()
Reset the device. Equivalent to <R click> -> Reset in NI MAX

2.4 Contributing new drivers

Each driver is implemented as a class named after the corresponding instrument. The driver classes are intended to be used directly and usable by anyone who decides to use these for experiment.

2.4.1 Coding conventions

The modules in the *labdrivers* package follow the following conventions:

- Class names are all CamelCase, e.g.:

```
Keithley2400
Sr830
Dynacool
```

- Method names are in lower case (there are exceptions, like in the Quantum Design instruments classes), e.g.:

```
measure_single_point
auto_gain
set_output_voltage
```

Depending on the future of the use of the *labdrivers* package, the coding conventions may be revised to maintain consistency.

2.4.2 Driver design principles

- Minimize the amount of 'internal state' maintained by the driver

The only 'internal state' that an instance should keep are properties like an IP address or a GPIB resource name. Most classes use the property decorator, but they are used only to make a query and return the response directly to the user.

E.g. do this:

```
def getOutput(self):
    return self.instrument.query("OUTPUT?")
```

instead of this:

```
def setOutput(self, out_type, level):
    self.instrument.write("OUTPUT {out_type} {level}".format(out_type, level))
    self.output = (out_type, level)

def getOutput():
    return self.output
```

- Use property decorators when possible to avoid writing getter and setter methods
- Use function names that are intuitive and minimize the amount of input required.

E.g. do this:

```
def output_current(self, current):
    thing.output('current', current)

def output_voltage(self, voltage):
    thing.output('voltage', voltage)
```

It might be more "efficient" to just allow for two inputs, but generally it would be less confusing if there were only one input. This is a change from older versions of *labdrivers* (0.8.x and below?).

2.4.3 Documenting drivers

Each method in the driver should be documented using a the reStructuredText format.

Example:

```
"""
First line of documentation.

:param thing1: Description of thing1 parameter
:param thing2: Description of thing2 parameter
:returns: Description of the returned data
:raises SomeError: Description of when SomeError shows up
"""
```

Index

S

T

U

V

W