RPyMostat-sensor Documentation

Release 0.1.0

Jason Antman

Dec 24, 2017

Contents

Master: 

Develop: 

CHAPTER 1

Contents

1.1 Sensor Support

1.1.1 Built-In Sensor Support

- *OWFS* (Dallas Semi 1-Wire Sensors via OneWire FileSystem (OWFS))

1.1.2 Adding Hardware Support

Adding hardware support is relatively straightforward:

1. Follow the instructions on installing for development (below).
2. Add a new class under `sensors/` that implements *BaseSensor* and any other methods you require. See the existing sensor classes as examples.
3. Ensure full test coverage for the class.
4. Add a new `rpymostat.sensors` entrypoint to `setup.py` that points to your new class.
5. Open a pull request for your changes.

`rpymostat-sensor` uses Setuptools entrypoints setuptools entrypoints for dynamic discovery of sensor classes. While it's preferred that new sensors be merged into this repository, it's possible to implement them as standalone packages as long as they have the required entrypoints.

1.2 rpymostat_sensor

1.2.1 rpymostat_sensor package

1.2.1.1 Subpackages

rpymostat_sensor.sensors package

Submodules

rpymostat_sensor.sensors.base module

class rpymostat_sensor.sensors.base.**BaseSensor**
Bases: object

Base class for the interface that all hardware Sensor classes must implement. Any class that implements this interface will be usable to discover and read sensors. Note that classes implementing this must also have a matching entrypoint in order to be discovered.

If the class constructor takes any arguments, they must be documented in Sphinx format in the docstring of the __init__ method.

_abc_cache = <_weakrefset.WeakSet object>

_abc_negative_cache = <_weakrefset.WeakSet object>

_abc_negative_cache_version = 31

_abc_registry = <_weakrefset.WeakSet object>

_description = 'Unknown'

get_description()
Return the sensor class's _description attribute.

Returns Sensor class's description

Return type str

read()
Read all present temperature sensors. Returns a dict of sensor unique IDs (keys) to dicts of sensor information.

Return dict format:

```
{
    'unique_id_1': {
        'type': 'sensor_type_string',
        'value': 1.234,
        'alias': 'str',
        'extra': ''
    },
    ...
}
```

Each dict key is a globally-unique sensor ID. Each value is a dict with the following keys:

- type: (str) sensor type
- **value: (float) current temperature in degress Celsius, or None if** there is an error reading it.

- alias: (str) a human-readable alias/name for the sensor, if present
- extra: (str) any extra information about the sensor

Returns dict of sensor values and information.

Return type dict

sensors_present()
Discover all matching sensors on the system. Return True if sensors were discovered, False otherwise. The class should cache information on the discovered sensors in order to read them later.

Returns whether or not matching sensors are present

Return type bool

rpymostat_sensor.sensors.dummy module

class rpymostat_sensor.sensors.dummy.**DummySensor**(*host_id*)
Bases: *rpymostat_sensor.sensors.base.BaseSensor*

Dummy sensor class that returns random temperatures.

_abc_cache = <_weakrefset.WeakSet object>

_abc_negative_cache = <_weakrefset.WeakSet object>

_abc_negative_cache_version = 31

_abc_registry = <_weakrefset.WeakSet object>

read()
Returns a dict, where the value is a pseudo-random float in the range of 18 to 26.75 (inclusive) incremented by .25.

Return dict format:

```
{
    '<self.host_id>_dummy1': {
        'type': 'dummy',
        'value': <value>,
        'alias': 'dummy'
    }
}
```

Returns dict of sensor values and information.

Return type dict

sensors_present()
Discover a single dummy temperature sensor.

Returns True because it's always here

Return type bool

rpymostat_sensor.sensors.owfs module

class rpymostat_sensor.sensors.owfs.**OWFS**(*owfs_path=None*)

Bases: *rpymostat_sensor.sensors.base.BaseSensor*

Sensor class to read OWFS sensors. Currently only tested with DS18S20.

_abc_cache = <_weakrefset.WeakSet object>

_abc_negative_cache = <_weakrefset.WeakSet object>

_abc_negative_cache_version = 31

_abc_registry = <_weakrefset.WeakSet object>

_description = 'Dallas Semi 1-Wire Sensors via OneWire FileSystem (OWFS)'

_discover_owfs()

If `owfs_path` is not specified for `OWFS.__init__`, attempt to find an OWFS mounted at some of the common paths. If one is found, return the path to it. If not, return None.

Returns path to OWFS mountpoint or None

_find_sensors()

Find all OWFS temperature sensors present. Return a list of dicts of information about them. Dicts have the format:

Return dict format:

```
{
    'temp_path': 'absolute path to read temperature from',
    'alias': 'sensor alias, if set',
    'address': 'sensor address',
    'type': 'sensor type'
}
```

The only *required* key in the dict is `temp_path`.

Returns list of dicts describing present temperature sensors.

Return type dict

_get_temp_scale(*owfs_path*)

Read and return the temperature_scale setting in use by OWFS mounted at owfs_path.

Parameters **owfs_path** (*str*) – OWFS mountpoint

Returns temperature scale in use ('C', 'F', 'K', or 'R')

Return type str

_read_owfs_file(*sensor_dir*, *fname*)

Read the contents of a file from OWFS; return None if the file does not exist, or the strip()'ed contents if it does. Really just a helper for cleaner unit testing.

Parameters

- **sensor_dir** (*str*) – `self.owfs_path` subdir for the sensor
- **fname** (*str*) – file name/path under `sensor_dir`

Returns stripped content str or None

owfs_paths = ['/run/owfs', '/owfs', '/mnt/owfs', '/var/owfs', '/1wire', '/var/1wire',

read()
Read all present temperature sensors.

Returns a dict of sensor unique IDs (keys) to dicts of sensor information.

Return dict format:

```
{
    'unique_id_1': {
        'type': 'sensor_type_string',
        'value': 1.234,
        'alias': 'str',
        'extra': ''
    },
    ...
}
```

Each dict key is a globally-unique sensor ID. Each value is a dict with the following keys:

- type: (str) sensor type
- value: (float) current temperature in degress Celsius, or None if there is an error reading it.
- alias: (str) a human-readable alias/name for the sensor, if present
- extra: (str) any extra information about the sensor

Returns dict of sensor values and information.

Return type dict

sensor_dir_re = <_sre.SRE_Pattern object>

sensors_present()
Determine whethere there are OWFS temperature sensors present or not.

Returns True because it's always here

Return type bool

1.2.1.2 Submodules

rpymostat_sensor.runner module

rpymostat_sensor.sensor_daemon module

rpymostat_sensor.version module

1.3 Changelog

1.3.1 x.y.z (YYYY-MM-DD)

- something

1.4 Development

To install for development:

1. Fork the RPyMostat-sensor repository on GitHub
2. Create a new branch off of master in your fork.

```
$ git clone git@github.com:YOURNAME/RPyMostat-sensor.git
$ cd RPyMostat-sensor
$ virtualenv . && source bin/activate
$ pip install -r requirements_dev.txt
$ python setup.py develop
```

The git clone you're now in will probably be checked out to a specific commit, so you may want to `git checkout BRANCHNAME`.

1.4.1 Guidelines

- pep8 compliant with some exceptions (see pytest.ini)
- 100% test coverage with pytest (with valid tests)

1.4.2 Testing

Testing is done via pytest, driven by tox.

- testing is as simple as:
  - `pip install tox`
  - `tox`
- If you want to see code coverage: `tox -e cov`
  - this produces two coverage reports - a summary on STDOUT and a full report in the `htmlcov/` directory
- If you want to pass additional arguments to pytest, add them to the tox command line after "–". i.e., for verbose pytext output on py27 tests: `tox -e py27 -- -v`

1.4.3 Release Checklist

1. Open an issue for the release; cut a branch off master for that issue.
2. Confirm that there are CHANGES.rst entries for all major changes.
3. Ensure that Travis tests passing in all environments.
4. Ensure that test coverage is no less than the last release (ideally, 100%).
5. Increment the version number in RPyMostat-sensor/version.py and add version and release date to CHANGES.rst, then push to GitHub.
6. Confirm that README.rst renders correctly on GitHub.
7. Upload package to testpypi, confirm that README.rst renders correctly.
   - Make sure your ~/.pypirc file is correct
   - `python setup.py register -r https://testpypi.python.org/pypi`

- `python setup.py sdist upload -r https://testpypi.python.org/pypi`
- Check that the README renders at https://testpypi.python.org/pypi/RPyMostat-sensor

8. Create a pull request for the release to be merge into master. Upon successful Travis build, merge it.
9. Tag the release in Git, push tag to GitHub:
   - tag the release. for now the message is quite simple: `git tag -a vX.Y.Z -m 'X.Y.Z released YYYY-MM-DD'`
   - push the tag to GitHub: `git push origin vX.Y.Z`
11. Upload package to live pypi:
    - `python setup.py sdist upload`
10. make sure any GH issues fixed in the release were closed.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

2.1 License

RPyMostat-sensor is licensed under the GNU Affero General Public License, version 3 or later.

Python Module Index

r

Index

Symbols

B

D

G

O

R

S