
pywatlow

Release 0.1.4

Apr 15, 2022

Contents

1	Overview	1
1.1	Installation	1
1.2	Documentation	1
1.3	Development	2
2	Installation	3
3	Usage	5
3.1	Command Line Usage	5
3.2	Module Usage	6
3.3	Reading Other Parameters	6
3.4	Setting Other Parameters	7
3.5	Error Handling	7
4	Reference	9
4.1	Watlow	9
4.2	Watlow Messaging Structure	10
5	Contributing	17
5.1	Bug reports	17
5.2	Documentation improvements	17
5.3	Feature requests and feedback	17
5.4	Development	18
6	Authors	19
7	Changelog	21
7.1	0.1.4 (2022-04-15)	21
7.2	0.1.3 (2021-07-25)	21
7.3	0.1.2 (2021-01-13)	21
7.4	0.1.1 (2020-08-15)	21
7.5	0.1.0 (2020-08-08)	21
8	Indices and tables	23
	Python Module Index	25
	Index	27

CHAPTER 1

Overview

docs	
tests	
package	

A Python 3 driver for the Watlow EZ-Zone PM temperature controller standard bus protocol. This driver allows the user to access each of the parameters outlined in the Watlow [user manual](#) with just a few functions.

- Free software: GNU Lesser General Public License v3 (LGPLv3)

1.1 Installation

```
pip install pywatlow
```

You can also install the in-development version with:

```
pip install https://github.com/BrendanSweeny/pywatlow/archive/master.zip
```

1.2 Documentation

<https://pywatlow.readthedocs.io/>

1.3 Development

To run the all tests run:

```
tox
```

Note, to combine the coverage data from all the tox environments run:

Windows	<pre>set PYTEST_ADDOPTS=--cov-append tox</pre>
Other	<pre>PYTEST_ADDOPTS=--cov-append tox</pre>

CHAPTER 2

Installation

At the command line:

```
pip install pywatlow
```


3.1 Command Line Usage

Format:

```
pywatlow [-h] [-r PORT ADDR PARAM | -s PORT ADDR TEMP]
```

Read the current temperature in degrees Celsius. This is equivalent to calling `Watlow(port='COM5',address=1).read()`:

```
# Read the current temperature (parameter 4001) of the Watlow controller
# at address 1 (Watlow controller's default) on serial port COM5:
>>> pywatlow -r COM5 1 4001
{'address': 1, 'param': 4001, 'data': 50.5, 'error': None}
```

Read the setpoint temperature in degrees Celsius. This is equivalent to calling `Watlow(port='COM5',address=1).readSetpoint()`:

```
# Read the setpoint (parameter 7001) of the Watlow controller
# at address 1 on serial port COM5:
>>> pywatlow -r COM5 1 7001
{'address': 1, 'data': 60.0, 'error': None}
```

Change the setpoint temperature (degrees Celsius). This is equivalent to calling `Watlow(port='COM5',address=1).write(60)`:

```
# Change the setpoint of the Watlow controller
# at address 1 on serial port COM5:
>>> pywatlow -w COM5 1 60
{'address': 1, 'param': 7001, 'data': 60.0, 'error': None}
```

At this time, CLI usage only supports reading the current temperature and the current setpoint.

3.2 Module Usage

To use pywatlow in a project:

```
from pywatlow.watlow import Watlow
watlow = Watlow(port='COM5', address=1)
print(watlow.read())
print(watlow.readSetpoint())
print(watlow.write(55))

##### Returns #####
{'address': 1, 'param': 4001, 'data': 59.5, 'error': None}
{'address': 1, 'param': 7001, 'data': 60.0, 'error': None}
{'address': 1, 'param': 7001, 'data': 55.0, 'error': None}
```

Using multiple temperature controllers on a single USB to RS485 converter:

```
from pywatlow.watlow import Watlow
import serial

ser = serial.Serial()
ser.port = 'COM5'
ser.baudrate = 38400 # Default baudrate for Watlow controllers
ser.timeout = 0.5
ser.open()

watlow_one = Watlow(serial=ser, address=1)
watlow_two = Watlow(serial=ser, address=2)
print(watlow_one.read())
print(watlow_two.read())

##### Returns #####
{'address': 1, 'param': 4001, 'data': 50.5, 'error': None}
{'address': 2, 'param': 4001, 'data': 60.0, 'error': None}
```

3.3 Reading Other Parameters

See the Watlow [user manual](#) for more information about the different parameter IDs and their functions.

The messaging structure for Watlow temperature controllers is set up to return data as integers or floats depending on the nature of the data. Often, ints are used to represent a state, such as in parameter ID 8003, the control loop heat algorithm.

`data_type` is not optional. Passing the incorrect type to `data_type` may result in an error (e.g. `watlow.readParam(7001, int)`). To see which data type each parameter expects, see the [Watlow user manual](#).

In some cases, specifying the wrong data type (e.g. `watlow.readParam(8003, float)`) will result in the instance of `Watlow` reading characters from a separate queued message if multiple controllers are being used on the same USB/RS485 port without any async handling.

Here, a returned value of 71 for parameter 8003 corresponds to the PID algorithm. We can read the state of 8003 like so:

```
from pywatlow.watlow import Watlow
watlow = Watlow(port='COM5', address=1)
```

(continues on next page)

(continued from previous page)

```

# Read the current heat algorithm
print(watlow.readParam(8003, int))

# Errors:
# Here the incorrect data type is given
print(watlow.readParam(7001, int))

##### Returns #####
{'address': 1, 'param': 8003, 'data': 71, 'error': None} # 71 --> PID algorithm

```

If the user wishes to avoid parameters altogether, basic functionality (read temp/setpoint and write temperature) can be achieved with wrapper functions `read()`, `readSetpoint()`, and `write()`, described above. Other parameters can be set at the controllers using the physical buttons and hardware menu.

Note: All methods of the Watlow object that read/write parameters use an instance of 1 unless specified (e.g. `watlow.readParam(7001, int, instance='01')`)

3.4 Setting Other Parameters

`watlow.writeParam()` is used to write to specific Watlow parameters. The message structure required for the set request depends on the data type (int or float). pywatlow will build the message based on this data type, which can be specified by passing the type class (either `int` or `float`) to the `data_type` argument.

If instead of a PID algorithm we would like something relatively simple like an “on-off” algorithm, we can set the value of parameter 8003 to 64:

```

from pywatlow.watlow import Watlow
watlow = Watlow(port='COM5', address=1)

print(watlow.readParam(8003, int))
print(watlow.writeParam(8003, 64, int))
print(watlow.writeParam(8003, 71, int))

# Errors:
# Here the incorrect data type is given
print(watlow.writeParam(8003, 71, float))

##### Returns #####
{'address': 1, 'param': 8003, 'data': 71, 'error': None} # 71 --> PID algorithm
{'address': 1, 'param': 8003, 'data': 64, 'error': None} # 64 --> on/off algorithm
{'address': 1, 'param': 8003, 'data': 71, 'error': None} # Back to 71, PID
# Error resulting from specifying the wrong data type:
{'address': 1, 'param': None, 'data': None, 'error': Exception('Received a message_
↳that could not be parsed from address 1')}

```

3.5 Error Handling

Errors are passed through using the ‘error’ key of the returned dictionary. Here there is no temperature controller at address 2:

```
print(watlow_one.read())
print(watlow_two.read())

##### Returns #####
{'address': 1, 'param': 4001, 'data': 55.0, 'error': None}
{'address': 2, 'param': None, 'data': None, 'error': Exception('Exception: No_
↳response at address 2')}
```

4.1 Watlow

class `watlow.Watlow` (*serial=None, port=None, timeout=0.5, address=1*)

Object representing a Watlow PID temperature controller. This class facilitates the generation and parsing of BACnet TP/MS messages to and from Watlow temperature controllers.

- **serial**: serial object (see pySerial's `serial.Serial` class) or *None*
- **port** (str): string representing the serial port or *None*
- **timeout** (float): Read timeout value in seconds
- **address** (int): Watlow controller address (found in the setup menu). Acceptable values are 1 through 16.

timeout and *port* are not necessary if a serial object was already passed with those arguments. The baudrate for Watlow temperature controllers is 38400 and hardcoded.

read (*instance='01'*)

Reads the current temperature. This is a wrapper around `readParam()` and is equivalent to `readParam(4001, float, instance)`.

Returns a dict containing the response data, parameter ID, and address. * **instance**: a two digit string corresponding to the channel to read (e.g. '01', '05')

readParam (*param, data_type, instance='01'*)

Takes a parameter and writes data to the watlow controller at object's internal address. Using this function requires knowing the data type for the parameter (int or float). See the [Watlow user manual](#) for individual parameters and the Usage section of these docs.

- **param**: a four digit integer corresponding to a Watlow parameter (e.g. 4001, 7001)
- **data_type**: the Python type representing the data value type (i.e. *int* or *float*)
- **instance**: a two digit string corresponding to the channel to read (e.g. '01', '05')

data_type is used to determine how many characters to read following the controller's response. If *int* is passed when the data type should be *float*, it will not read the entire message and will throw an error. If

float is passed when it should be *int*, it will timeout, possibly reading correctly. If multiple instances of *Watlow()* are using the same serial port for different controllers it will read too many characters. It is best to be completely sure which data type is being used by each parameter (*int* or *float*).

Returns a dict containing the response data, parameter ID, and address.

readSetpoint (*instance='01'*)

Reads the current setpoint. This is a wrapper around *readParam()* and is equivalent to *readParam(7001, float, instance)*.

Returns a dict containing the response data, parameter ID, and address. * **instance**: a two digit string corresponding to the channel to read (e.g. '01', '05')

write (*value, instance='01'*)

Changes the watlow temperature setpoint. Takes a value (in degrees F by default), builds request, writes to watlow, receives and returns response object.

- **value**: an int or float representing the new target setpoint in degrees F by default
- **instance**: a two digit string corresponding to the channel to set (e.g. '01', '05')

This is a wrapper around *writeParam()* and is equivalent to *writeParam(7001, value, float, instance)*.

Returns a dict containing the response data, parameter ID, and address.

writeParam (*param, value, data_type, instance='01'*)

Changes the value of the passed watlow parameter ID. Using this function requires knowing the data type for the parameter (int or float). See the Watlow [user manual](#) for individual parameters and the Usage section of these docs.

- **value**: an int or float representing the new target setpoint in degrees F by default
- **data_type**: the Python type representing the data value type (i.e. *int* or *float*)
- **instance**: a two digit string corresponding to the channel to read (e.g. '01', '05')

data_type is used to determine how the BACnet TP/MS message will be constructed and how many serial characters to read following the controller's response.

Returns a dict containing the response data, parameter ID, and address.

4.2 Watlow Messaging Structure

The following is an incomplete understanding of the messaging structure that the Watlow temperature controllers use. Enough of the structure is understood for the driver to be functional, but the function of many of the bytes is unknown to the author(s).

General Message Structure:

- BACnet MS/TP protocol
- First two bytes of any message make up the preamble, and are always: 55 FF
- Byte 3 seems to define the type of message (read/response)
- Byte 4 of the request defines the zone (internal Watlow address)
- The zone appears in byte 5 of the response
- Byte 7 appears to define the type of request
- Byte 8 is the header check byte (more info at reverseengineering.stackexchange.com)

- Immediately following the parameter bytes is the instance. I have yet to encounter a situation where this is not 01
- The final two bytes of any message are the data check bytes (more info at reverseengineering.stackexchange.com)

How to read the example message tables:

- Addr.: The address used in the request and response
- Param: The parameter being read/set
- Process Val.: The value represented in the “Data” column of the response/request
- The message begins with the column labeled “Preamble” and continues to the right
- Sometimes the byte number is not the same in the response as the request (e.g. Zone), hence the “-”

4.2.1 Read Requests

- Read requests seem to be 16 bytes long no matter the data type
- Byte 7 is 06 for read requests
- Bytes 12 and 13 represent the parameter to be read
- Byte 14 is the instance, generally 01

Where the response is a float:

- Request is 16 bytes long
- Response is 21 bytes long
- Response types appear to be defined by bytes 7
- Note: The process value for 4001 is ~2500 in these examples because no probe is connected

Table 1: Example Messages

Addr.	Param	Process Val	Preamble	Req/Res	Req/Res	Zone	??	??	??	Chk	??	Param	Instance	??	Data	Data Chk
1	4001	-	55 FF	05	-	10	00	00	06	E8	01 03 01	04 01	01	-	-	E3 99
		2529.6	55 FF	06	00	10	-	00	0B	88	02 03 01	04 01	01	08	45 1E 3C D4	A7 28
2	4001	-	55 FF	05	-	11	00	00	06	61	01 03 01	04 01	01	-	-	E3 99
		2528.7	55 FF	06	00	11	-	00	0B	10	02 03 01	04 01	01	08	45 1E 0C 06	9A 6B
1	4012	-	55 FF	05	-	10	00	00	06	E8	01 03 01	04 0C	01	-	-	9B 29
		0.0	55 FF	06	00	10	-	00	0B	88	02 03 01	04 0C	01	08	00 00 00 00	2D 64
2	4012	-	55 FF	05	-	11	00	00	06	61	01 03 01	04 0C	01	-	-	9B 29
		0.0	55 FF	06	00	11	-	00	0B	10	02 03 01	04 0C	01	08	00 00 00 00	2D 64
1	7001	-	55 FF	05	-	10	00	00	06	E8	01 03 01	07 01	01	-	-	87 76
		392.0	55 FF	06	00	10	-	00	0B	88	02 03 01	07 01	01	08	43 C4 00 00	33 9A

Where the response is an integer:

- Request is 16 bytes long
- Response is 20 bytes long

Table 2: Example Messages

Addr	Param	Process Val	Preamble	Req/Res	Zone	??	??	??	Chk	??	Param	Instance	??	Data	Data Chk	
1	8003	-	55 FF	05	-	10	00	00	06	E8	01 03 01	08 03	01	-	-	F0 0F
		71	55 FF	06	00	10	-	00	0A	76	02 03 01	08 03	01	0F 01	00 47	C5 6B
2	8003	-	55 FF	05	-	11	00	00	06	61	01 03 01	08 03	01	-	-	F0 0F
		71	55 FF	06	00	11	-	00	0A	EE	02 03 01	08 03	01	0F 01	00 47	C5 6B
1	4037	-	55 FF	05	-	10	00	00	06	E8	01 03 01	04 25	01	-	-	B0 DD
		1449	55 FF	06	00	10	-	00	0A	76	02 03 01	04 25	01	0F 01	05 A9	0D 37
2	4037	-	55 FF	05	-	11	00	00	06	61	01 03 01	04 25	01	-	-	B0 DD
		1449	55 FF	06	00	11	-	00	0A	EE	02 03 01	04 25	01	0F 01	05 A9	0D 37

4.2.2 Set Requests

- Bytes 11 and 12 represent the parameter to be set
- Byte 13 is the instance, generally 01

Where the value/response is a float

- Request is 20 bytes long
- Response is 20 bytes long
- Byte 7 is 0A when the process value is a float
- When the process value is a float, the byte preceding the data (14) is 08

Table 3: Example Messages

Addr.	Param	Pro- cess Val	Pream- ble	Req/Res	Zone	??	??	??	Chk	??	Param	In- stance	??	Data	Data Chk	
1	7001	392	55 FF	5	-	10	0	0	0A	EC	01 04	07 01	1	8	43 C4 00 00	EB 77
		392	55 FF	6	0	10	-	0	0A	76	02 04	07 01	1	8	43 C4 00 00	82 03
2	7001	392	55 FF	5	-	11	0	0	0A	65	01 04	07 01	1	8	43 C4 00 00	EB 77
		392	55 FF	6	0	11	-	0	0A	EE	02 04	07 01	1	8	43 C4 00 00	82 03

Where the value/response is an integer:

- Request is 19 bytes long
- Response is 19 bytes long
- Byte 7 is 09 when the process value is an integer
- When the process value is an integer, the two bytes preceding the data (14, 15) are 0F 01

Table 4: Example Messages

Addr.	Param	Pro- cess Val	Pream- ble	Req/Res	Zone	??	??	Msg Type?	Chk	??	Param	In- stance	??	Data	Data Chk	
1	8003	71	55 FF	05	-	10	03	00	09	46	01 04	08 03	01	0F 01	00 47	8f ED
		71	55 FF	06	03	10	-	00	09	EF	02 04	08 03	01	0F 01	00 47	88 3B

Errors

Currently, it is unclear exactly what these responses mean, or what the structure of an error message is, but the following are possibly errors:

This is likely an access denied error response received when trying to write a read only parameter (4001, 100 degrees, address 2):

- *55 FF 06 00 11 00 02 17 02 80 FF B8*

Other possible errors that have been received:

- *55 FF 06 00 10 00 02 8F 02 85 52 EF*

- *55 FF 06 00 10 00 02 8F 02 86 C9 DD*
- *55 FF 06 00 10 00 02 8F 02 83 64 8A*
- *55 FF 06 00 10 00 02 8F 02 80 FF B8*
- *55 FF 06 00 10 00 05 73 02 05 08 03 00 02 5B*
- *55 FF 06 00 10 00 05 73 02 05 01 08 00 B4 23*

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

5.1 Bug reports

When [reporting a bug](#) please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.2 Documentation improvements

pywatlow could always use more documentation, whether as part of the official pywatlow docs, in docstrings, or even on the web in blog posts, articles, and such.

5.3 Feature requests and feedback

The best way to send feedback is to file an issue at <https://github.com/BrendanSweeny/pywatlow/issues>.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that code contributions are welcome :)

5.4 Development

To set up *pywatlow* for local development:

1. Fork *pywatlow* (look for the “Fork” button).
2. Clone your fork locally:

```
git clone git@github.com:BrendanSweeny/pywatlow.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

4. When you’re done making changes run all the checks and docs builder with *tox* one command:

```
tox
```

5. Commit your changes and push your branch to GitHub:

```
git add .  
git commit -m "Your detailed description of your changes."  
git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

5.4.1 Pull Request Guidelines

If you need some code review or feedback while you’re developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run *tox*)¹.
2. Update documentation when there’s new API, functionality etc.
3. Add a note to *CHANGELOG.rst* about the changes.
4. Add yourself to *AUTHORS.rst*.

5.4.2 Tips

To run a subset of tests:

```
tox -e envname -- pytest -k test_myfeature
```

To run all the test environments in *parallel* (you need to *pip install detox*):

```
detox
```

¹ If you don’t have all the necessary python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.
It will be slower though ...

CHAPTER 6

Authors

- Brendan Sweeny - brendansweeny.com
- Joe Rickwalder - jrickwal@gmail.com

7.1 0.1.4 (2022-04-15)

- Fixed message generation/parsing for addresses 11 through 16

7.2 0.1.3 (2021-07-25)

- Fixed support for multiple instances where the parameter data type is an integer

7.3 0.1.2 (2021-01-13)

- added support for multiple instances, like in the RM series controllers with multiple channels

7.4 0.1.1 (2020-08-15)

- Initial PyPI release
- Various tox, badge, and metadata fixes
- Removed unused temperature unit conversion functions
- Updated Python version requirements
- Minor documentation clean up

7.5 0.1.0 (2020-08-08)

- Added basic functionality

- Reading/writing of Watlow parameters
- Request message generation
- Calculation of check bytes
- Response message parsing

CHAPTER 8

Indices and tables

- `genindex`
- `modindex`
- `search`

W

watlow, 9

R

`read()` (*watlow.Watlow method*), 9
`readParam()` (*watlow.Watlow method*), 9
`readSetpoint()` (*watlow.Watlow method*), 10

W

`Watlow` (*class in watlow*), 9
`watlow` (*module*), 9
`write()` (*watlow.Watlow method*), 10
`writeParam()` (*watlow.Watlow method*), 10