
Dataplicity Lomond Documentation

Release 0.1.7

Will McGugan

May 26, 2017

Contents:

1	Sponsor	3
2	Authors	5
3	Guide	7
3.1	Introduction	7
3.2	Installing	7
3.3	Basic Usage	7
3.4	Events	8
3.5	Closing the Websocket	8
3.6	Pings and Pongs	9
3.7	Polling	9
3.8	WebSockets and Threading	9
3.9	Persistent Connections	10
4	Code Documentation	11
4.1	Exceptions	11
4.2	Events	12
4.3	Persist	13
4.4	Response	13
4.5	WebSocket	14
5	Indices and tables	17
	Python Module Index	19

Lomond is a Pythonic Websockets client library designed for reliability and ease of use.

CHAPTER 1

Sponsor

Lomond was sponsored by [Dataplicity](#) – Control your Raspberry Pi from anywhere!

CHAPTER 2

Authors

- Will McGugan (willmcgugan+lomond@gmail.com)
- Mateusz Mikołajczyk

Introduction

Lomond is a websocket client library designed to make adding websocket support to your application as tranquil as the [Scottish Loch](#) it was named after.

Installing

You can install Lomond with `pip` as follows:

```
pip install lomond
```

Or to upgrade to the most recent version:

```
pip install lomond --upgrade
```

Alternatively, if you would like to install from source, check out [the code from Github](#).

You may wish to install *wsaccel*, which is a C module containing optimizations for some websocket operations. Lomond will use it if available:

```
pip install wsaccel
```

Basic Usage

To connect to a websocket server, first construct a *WebSocket* object, with a *ws://* or *wss://* URL. Here is an example:

```
from lomond.websocket import WebSocket
ws = WebSocket('wss://echo.websocket.org')
```

No socket connection is made by a freshly constructed `WebSocket` object. To connect and interact with a websocket server, iterate over the `WebSocket` instance, which will yield a number of `Event` objects. Here's an example:

```
for event in ws:
    print(event)
```

Here is an example of the output you might get from the above code:

```
Connecting(url='wss://echo.websocket.org')
Connected(url='wss://echo.websocket.org')
Ready(<response HTTP/1.1 101 Web Socket Protocol Handshake>, protocol=None,
↪extensions=set([]))
```

The `Ready` event indicates a successful connection to a websocket server. You may now use the `send_text()` and `send_binary()` methods to send data to the server.

When you receive data from the server, a `Text` or `Binary` event will be generated.

Iterating over the `WebSocket` instance in this way calls `connect()` with default parameters, i.e. it is equivalent to the following:

```
for event in ws.connect():
    print(event)
```

You may want to call `connect` manually to change the default behavior.

Events

Events inform your application when data is received from the server or when the websocket state changes.

All events are derived from `Event` and will contain at least 2 attributes; `received_time` is the epoch time the event was received, and `name` is the name of the event. Some events have additional attributes with more information. See the [Events](#) for details.

When handling events, you can either check the type with `isinstance` or by looking at the `name` attribute.

For example, the following two lines are equivalent:

```
if isinstance(event, events.Ready):
```

or:

```
if event.name == "ready":
```

Note: The `isinstance` method is possibly uglier, but has the advantage that you are less likely to introduce a bug with a typo in the event name.

If an event is generated that you aren't familiar with, then you should simply ignore it. This is important for backwards compatibility; future versions of Lomond may introduce new event types.

Closing the WebSocket

To close a websocket, call the `close()` method to initiate a *websocket close handshake*. You may call this method from within the websocket loop, or from another thread.

When a WebSocket wishes to close, it sends a close packet to the server. The server will respond by sending a close packet of its own. Only when this echoed close packet is received will the WebSocket close the underlying socket. This allows both ends of the connection to finish what they are doing, without losing data.

Note: When you call the `close()` method, you will no longer be able to send data, but you may still receive packets from the server until the close has completed.

When the websocket has been closed, you will receive a `Closed` event, followed by a `Disconnected` event, and the event loop will exit.

Pings and Pongs

Both the server and client may send ‘ping’ packets, which should be responded to with a ‘pong’ packet. This allows both ends of the connection to know if the other end is really listening.

By default, Lomond will send pings packets every 30 seconds. If you wish to change this rate or disable ping packets entirely, you may use the `connect()` method.

Here’s how you would disable pings:

```
websocket = WebSocket('wss://ws.example.org')
for event in WebSocket.connect(ping_rate=0):
    on_event(event)
```

Lomond will also automatically respond to ping requests. Since this is a requirement of the websocket specification, you probably don’t want to change this behaviour. But it may be disabled with the `auto_pong` flag in `connect()`.

Regardless of whether `auto_pong` is enabled, a `Pong` event will be generated when Lomond receives a ping packet. If `auto_pong` is disabled, you should manually call `send_pong()` in response to a ping, or the server may disconnect you.

Polling

Lomond checks for automatic pings and performs other housekeeping tasks at a regular intervals. This *polling* is exposed as `Poll` events. Your application can use these events to do any processing that needs to be invoked at regular intervals.

The default poll rate of 5 seconds is granular enough for Lomond’s polling needs, while having negligible impact on CPU. If your application needs to process at a faster rate, you may set the `poll` parameter of `connect()`.

Note: If your application needs to be more realtime than polling once a second, you should probably use threads in tandem with the event loop.

WebSockets and Threading

WebSocket objects are *thread safe*, but Lomond does not need to launch any threads to run a websocket. For many applications, responding to data and poll events is all you will need. However, if your application needs to do more than communicate with a websocket server, you may want to run a websocket in a thread of its own.

Persistent Connections

Lomond supports a simple mechanism for persistent connections. Essentially, you can tell Lomond to continually retry a websocket connection if it is dropped for any reason. This allows an application to maintain a websocket connection even if there are any outages in connectivity.

To run a persistent connection, wrap a websocket with `persist()`. Here is an example:

```
from lomond.persist import persist
websocket = WebSocket('wss://ws.example.org')
for event in persist(websocket):
    # handle event
```

You will receive events as normal with the above loop.

If the connection is dropped for any reason, you will receive `Disconnected` as usual, followed by `Connecting` when Lomond retries the connection. Lomond will keep retrying the connection until it is successful, and a `Ready` event is generated.

The `persist()` function implements *exponential backoff*. If the websocket object fails to connect, it will wait for a random period between zero seconds and an upper limit. Every time the connection fails, it will double the upper limit until it connects, or a maximum delay is reached.

The exponential backoff prevents a client from hammering a server that may already be overloaded. It also prevents the client from being stuck in a cpu intensive spin loop.

Exceptions

Lomond takes great care not to leak any socket or system related exceptions. During normal usage of *WebSocket* objects you can expect only the following exception hierarchy to be thrown.

exception `lomond.errors.CriticalProtocolError (msg, *args, **kwargs)`

Critical protocol error. An egregious error in the protocol resulting in an immediate disconnect.

exception `lomond.errors.FrameBuildError (msg, *args, **kwargs)`

Raised when trying to build an invalid websocket frame.

exception `lomond.errors.HandshakeError (msg, *args, **kwargs)`

Raised when the server doesn't respond correctly to the websocket handshake.

exception `lomond.errors.PayloadTooLarge (msg, *args, **kwargs)`

The payload length field is too large.

WebSocket messages have a maximum payload of 2^{63} bytes. In practice it may be impossible to generate such a packet for real, but its feasible a corrupt packet header could make it appear that such a packet was being sent.

exception `lomond.errors.ProtocolError (msg, *args, **kwargs)`

Raised in response to a protocol violation.

exception `lomond.errors.TransportFail (msg, *args, **kwargs)`

The transport (socket) failed when sending.

Likely indicating connectivity issues.

exception `lomond.errors.WebSocketClosed (msg, *args, **kwargs)`

Raised when attempting to send over a closed websocket.

exception `lomond.errors.WebSocketClosing (msg, *args, **kwargs)`

Raised when attempting to send over a closing websocket.

exception `lomond.errors.WebSocketError (msg, *args, **kwargs)`

Base exception.

exception `lomond.errors.WebSocketUnavailable` (*msg, *args, **kwargs*)
The websocket can not be used.

Events

class `lomond.events.BackOff` (*delay*)

Generated when a persistent connection has to wait before re- attempting a connection.

Parameters **delay** (*float*) – The delay (in seconds) before Lomond will re- attempt to connect.

class `lomond.events.Binary` (*data*)

Generated when Lomond receives a binary message from the server.

Parameters **data** (*bytes*) – The binary payload.

class `lomond.events.Closed` (*code, reason*)

Generated when the websocket was closed. The websocket may no longer send packets after this event has been received.

Parameters

- **code** – The closed code returned from the server.
- **reason** (*str*) – An optional description why the websocket was closed, as returned from the server.

class `lomond.events.ConnectFail` (*reason*)

Generate when Lomond was unable to connect to a Websocket server.

Parameters **reason** (*str*) – A short description of the reason for the failure.

class `lomond.events.Connected` (*url*)

Generated when Lomond has connected to a server but not yet negotiated the websocket upgrade.

class `lomond.events.Connecting` (*url*)

Generated prior to establishing a websocket connection to a server.

Parameters **url** – The websocket URL the websocket is connecting to.

class `lomond.events.Disconnected` (*reason=u'closed', graceful=False*)

Generated when a websocket connection has been dropped.

Parameters

- **reason** (*str*) – A description of why the websocket was closed.
- **graceful** (*bool*) – Flag indicating if the connection was dropped gracefully (*True*), or disconnected due to a socket failure (*False*).

class `lomond.events.Event`

Base class for a websocket 'event'.

class `lomond.events.Ping` (*data*)

Generated when Lomond received a ping packet from the server.

Parameters **data** (*bytes*) – Ping payload data.

class `lomond.events.Poll`

A generated poll event.

class `lomond.events.Pong` (*data*)

Generated when Lomond receives a pong packet from the server.

Parameters **data** (*bytes*) – The pong payload data.

class `lomond.events.Ready` (*response, protocol, extensions*)

Generated when Lomond has connected to the server, and successfully negotiated the websocket upgrade.

Parameters

- **response** – A *Response* object.
- **protocol** (*str*) – A websocket protocol or *None* if no protocol was supplied.
- **extensions** (*set*) – A set of negotiated websocket extensions. Currently Lomond does not support any extensions, so this will be an empty set.

class `lomond.events.Rejected` (*response, reason*)

Server rejected WS connection.

class `lomond.events.Text` (*text*)

Generated when Lomond receives a text message from the server.

Parameters **text** (*str*) – The text payload.

class `lomond.events.UnknownMessage` (*message*)

An application message was received, with an unknown opcode.

Persist

Maintains a persistent websocket connection.

`lomond.persist.persist` (*websocket, poll=5, min_wait=5, max_wait=30, ping_rate=30, exit_event=None*)

Run a websocket, with a retry mechanism and exponential back-off.

Parameters

- **websocket** – A *WebSocket* instance.
- **poll** (*float*) – The websocket poll rate, in seconds.
- **min_wait** (*float*) – The minimum time to wait between reconnect attempts (seconds).
- **max_wait** (*float*) – The maximum time to wait between reconnect attempts (seconds).
- **ping_rate** (*float*) – Delay between pings (seconds), or 0 for no auto ping.
- **exit_event** – A threading event object, which can be used to exit the persist loop if it is set. Set to *None* to use an internal event object.

`lomond.persist.random` () → x in the interval [0, 1).

Response

A simple abstraction for an HTTP response.

A response object is supplied in the *Ready* event.

class `lomond.response.Response` (*header_data*)

A HTTP response.

Parameters **header_data** (*bytes*) – Raw response.

get (*name*, *default=None*)

Get a header.

Parameters

- **name** (*bytes*) – Name of the header to retrieve.
- **default** – Default value if header is not present.

Return type *bytes*

get_list (*name*)

Extract a list from a header.

Parameters **name** (*bytes*) – Name of the header to retrieve.

Return type *list*

Returns A list of strings in the header.

WebSocket

Abstract websocket functionality.

class `lomond.websocket.WebSocket` (*url*, *protocols=None*, *agent=None*)

IO independent websocket functionality.

Parameters **url** (*str*) – A websocket URL, must have a *ws://* or *wss://* protocol.

Params list protocols A list of supported protocols (defaults to no protocols).

Params str agent A user agent string to be sent in the header. The default uses the value *USER_AGENT* defined in `lomond.constants`.

build_request ()

Get the websocket request (in bytes).

This method is called from the session, and should not be invoked explicitly.

close (*code=None*, *reason=None*)

Close the websocket.

Parameters

- **code** (*int*) – A closing code, which should probably be one of the enumerations in `lomond.status.Status` or a valid value as specified in <https://tools.ietf.org/html/rfc6455#section-7.4>
- **reason** (*str*) – A short descriptive reason why the websocket is closing. This value is intended for the remote end to help in debugging.

Note: Closing the websocket won't exit the main loop immediately; it will put the websocket in to a *closing* state while it waits for the server to echo back a close packet. No data may be sent by the application when the websocket is closing.

connect (*session_class=<class 'lomond.session.WebsocketSession'>*, *poll=5.0*, *ping_rate=30.0*, *auto_pong=True*)

Connect the websocket to a session.

Parameters

- **session_class** – An object to manage the *session*. This object is an extension mechanism that will allow the WebSocket to be *driven* by different back-ends. For now, treat it as an implementation detail and leave it as the default.
- **poll** (*float*) – Rate (in seconds) that poll events should be generated.
- **ping_rate** (*float*) – Rate that ping packets should be sent. Set to 0 to disable auto pings.
- **auto_pong** (*bool*) – Enable (default) automatic response to ping events.

Returns An iterable of `Event` instances.

feed (*data*)

Feed with data from the socket, and yield any events.

Parameters **data** (*bytes*) – data received over a socket.

is_closed

Flag that indicates if the websocket is closed.

is_closing

Boolean that indicates if the websocket is in a closing state. No further messages may be sent when a websocket is closing.

is_secure

Boolean that indicates if the websocket is over ssl (i.e. the wss protocol).

on_disconnect ()

Called on disconnect.

on_response (*response*)

Called when the HTTP response has been received.

reset ()

Reset the state.

send_binary (*data*)

Send a binary frame.

Parameters **data** (*bytes*) – Binary data to send.

Raises **TypeError** – If data is not bytes.

send_ping (*data*='')

Send a ping packet.

Parameters **data** (*bytes*) – Data to send in the ping message (must be <= 125 bytes).

Raises

- **TypeError** – If *data* is not bytes.
- **ValueError** – If *data* is > 125 bytes.

send_pong (*data*)

Send a pong packet.

Parameters **data** (*bytes*) – Data to send in the ping message (must be <= 125 bytes).

A *pong* may be sent in response to a ping, or unsolicited to keep the connection alive.

Raises

- **TypeError** – If *data* is not bytes.
- **ValueError** – If *data* is > 125 bytes.

send_text (*text*)

Send a text frame.

Parameters **text** (*str*) – Text to send.

Raises **TypeError** – If data is not str (or unicode on Py2).

CHAPTER 5

Indices and tables

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

I

- `lomond.errors`, [11](#)
- `lomond.events`, [12](#)
- `lomond.persist`, [13](#)
- `lomond.response`, [13](#)
- `lomond.websocket`, [14](#)

B

BackOff (class in lomond.events), 12
Binary (class in lomond.events), 12
build_request() (lomond.websocket.WebSocket method), 14

C

close() (lomond.websocket.WebSocket method), 14
Closed (class in lomond.events), 12
connect() (lomond.websocket.WebSocket method), 14
Connected (class in lomond.events), 12
ConnectFail (class in lomond.events), 12
Connecting (class in lomond.events), 12
CriticalProtocolError, 11

D

Disconnected (class in lomond.events), 12

E

Event (class in lomond.events), 12

F

feed() (lomond.websocket.WebSocket method), 15
FrameBuildError, 11

G

get() (lomond.response.Response method), 13
get_list() (lomond.response.Response method), 14

H

HandshakeError, 11

I

is_closed (lomond.websocket.WebSocket attribute), 15
is_closing (lomond.websocket.WebSocket attribute), 15
is_secure (lomond.websocket.WebSocket attribute), 15

L

lomond.errors (module), 11

lomond.events (module), 12
lomond.persist (module), 13
lomond.response (module), 13
lomond.websocket (module), 14

O

on_disconnect() (lomond.websocket.WebSocket method), 15
on_response() (lomond.websocket.WebSocket method), 15

P

PayloadTooLarge, 11
persist() (in module lomond.persist), 13
Ping (class in lomond.events), 12
Poll (class in lomond.events), 12
Pong (class in lomond.events), 12
ProtocolError, 11

R

random() (in module lomond.persist), 13
Ready (class in lomond.events), 13
Rejected (class in lomond.events), 13
reset() (lomond.websocket.WebSocket method), 15
Response (class in lomond.response), 13

S

send_binary() (lomond.websocket.WebSocket method), 15
send_ping() (lomond.websocket.WebSocket method), 15
send_pong() (lomond.websocket.WebSocket method), 15
send_text() (lomond.websocket.WebSocket method), 15

T

Text (class in lomond.events), 13
TransportFail, 11

U

UnknownMessage (class in lomond.events), 13

W

`WebSocket` (class in `lomond.websocket`), [14](#)

`WebSocketClosed`, [11](#)

`WebSocketClosing`, [11](#)

`WebSocketError`, [11](#)

`WebSocketUnavailable`, [11](#)