

---

# **Tornadose Documentation**

***Release 0.2.1***

**Michael V. DePalatis**

October 18, 2015



<b>1</b>	<b>Installation</b>	<b>3</b>
<b>2</b>	<b>Usage</b>	<b>5</b>
<b>3</b>	<b>Contributing</b>	<b>7</b>
<b>4</b>	<b>See also</b>	<b>9</b>
<b>5</b>	<b>License</b>	<b>11</b>
<b>6</b>	<b>Contents</b>	<b>13</b>
6.1	Data storage and publishing . . . . .	13
6.2	Request handlers . . . . .	14
6.3	Changelog . . . . .	14



An implementation of the publish/subscribe pattern for the [Tornado](#) web server.



---

# Installation

---

Tornadose is on PyPI:

```
$ pip install tornadose
```

This will grab the latest official release. Alternatively, or for development, you can clone the repository and install it manually:

```
$ git clone https://github.com/mivade/tornadose.git
$ cd tornadose
$ python setup.py install
```





---

## Usage

---

A simple example of using server-sent events (a.k.a. EventSource):

```
import random
from tornado.ioloop import IOLoop, PeriodicCallback
from tornado.web import Application
from tornadose.handlers import EventSource
from tornadose.stores import DataStore

store = DataStore()

app = Application(
    [(r'/', EventSource, {'store': store})],
    debug=True)
app.listen(9000)

loop = IOLoop.instance()
PeriodicCallback(lambda: store.set_data(random.random()), 1000).start()
loop.start()
```

To monitor the stream with `curl`:

```
$ curl http://localhost:9000
```

or with `HTTPIe`:

```
$ http -S get localhost:9000
```

Additional demos can be found in the `demos` directory.



---

## Contributing

---

Contributions, complaints, criticisms, and whatever else are welcome. The source code and issue tracker can be found on [GitHub](#).



---

### See also

---

Some other implementations of server-sent events with Tornado include:

- [tornado-sse](#)
- [tornado-eventsource](#)



---

### License

---

Tornadose is freely available under the terms of the MIT license. See `LICENSE` for details.





---

## Contents

---

### 6.1 Data storage and publishing

In order to publish data to listeners, Tornadoose utilizes a data store concept somewhat reminiscent of that used in [Flux](#). In short, subscribers start listening to a data store and are notified when there are updates.

**class** `tornadoose.stores.BaseStore (*args, **kwargs)`

Base class for all data store types.

At a minimum, derived classes should implement `submit` and `publish` methods.

**deregister** (*subscriber*)

Stop publishing to a subscriber.

**initialize** (*\*args, \*\*kwargs*)

Hook for doing custom initialization. Child classes should implement this method instead of overwriting `__init__`.

**publish** ()

Push messages to all listeners. This method must be implemented by child classes. A recommended way to implement this method is as a looping coroutine which yields until new data is available via the `submit ()` method.

**register** (*subscriber*)

Register a new subscriber. This method should be invoked by listeners to start receiving messages.

**submit** (*message*)

Add a new message to be pushed to subscribers. This method must be implemented by child classes.

This method exists to store new data. To actually publish the data, implement the `publish` method.

**class** `tornadoose.stores.DataStore (*args, **kwargs)`

Generic object for producing data to feed to clients.

To use this, simply instantiate and update the `data` property whenever new data is available. When creating a new `EventSource` handler, specify the `DataStore` instance so that the `EventSource` can listen for updates.

**class** `tornadoose.stores.QueueStore (*args, **kwargs)`

Publish data via queues.

This class is meant to be used in cases where subscribers should not miss any data. Compared to the `DataStore` class, new messages to be broadcast to clients are put in a queue to be processed in order.

`QueueStore` will work with any `tornado.web.RequestHandler` subclasses which implement a `submit` method. It is recommended that a custom subscription handler's `submit ()` method also utilize a

queue to avoid losing data. The subscriber must also register/deregister itself with the `QueueStore` via the `QueueStore.register()` and `QueueStore.deregister()` methods.

A `QueueStore`-compatible request handler is included in `tornadoose.handlers.WebSocketSubscriber`.

## 6.2 Request handlers

**class** `tornadoose.handlers.BaseHandler` (*application, request, \*\*kwargs*)  
Bases: `tornado.web.RequestHandler`

Base handler for subscribers. To be compatible with data stores defined in `tornadoose.stores`, custom handlers should inherit this class and implement the `submit()` and `publish()` methods.

**class** `tornadoose.handlers.EventSource` (*application, request, \*\*kwargs*)  
Bases: `tornadoose.handlers.BaseHandler`

Handler for server-sent events a.k.a. EventSource.

The `EventSource` interface has a few advantages over websockets:

- It is a normal HTTP connection and so can be more easily monitored than websockets using tools like `curl` or `HTTPie`.
- Browsers generally automatically try to reestablish a lost connection.
- The publish/subscribe pattern is better suited to some applications than the full duplex model of websockets.

**initialize** (*store, period=None*)

If `period` is given, publishers will sleep for approximately the given time in order to throttle data speeds.

**publish** (*\*args, \*\*kwargs*)  
Pushes data to a listener.

**class** `tornadoose.handlers.WebSocketSubscriber` (*application, request, \*\*kwargs*)  
Bases: `tornadoose.handlers.BaseHandler`, `tornado.websocket.WebSocketHandler`

A WebSocket-based subscription handler to be used with `tornadoose.stores.QueueStore`.

**open** (*\*args, \*\*kwargs*)  
Register with the publisher.

**publish** (*\*args, \*\*kwargs*)  
Push a new message to the client. The data will be available as a JSON object with the key `data`.

## 6.3 Changelog

### 6.3.1 Version 0.2.1

2015-10-17

- Subscription handlers automatically get registered with stores. This simplifies creating custom handlers.

### 6.3.2 Version 0.2.0

2015-10-11

- Reworks stores and handlers (backwards incompatible!).
- Adds a new queue-based `QueueStore` store.
- Implements a websocket-based subscriber to supplement `EventSource`.
- Begins to add unit testing.

### 6.3.3 Version 0.1.2

*2015-09-20*

- Defines an `EventSource` request handler and a `DataStore` object for using server-sent events with Tornado.



## B

BaseHandler (class in `tornadose.handlers`), [14](#)

BaseStore (class in `tornadose.stores`), [13](#)

## D

DataStore (class in `tornadose.stores`), [13](#)

deregister() (`tornadose.stores.BaseStore` method), [13](#)

## E

EventSource (class in `tornadose.handlers`), [14](#)

## I

initialize() (`tornadose.handlers.EventSource` method), [14](#)

initialize() (`tornadose.stores.BaseStore` method), [13](#)

## O

open() (`tornadose.handlers.WebSocketSubscriber` method), [14](#)

## P

publish() (`tornadose.handlers.EventSource` method), [14](#)

publish() (`tornadose.handlers.WebSocketSubscriber` method), [14](#)

publish() (`tornadose.stores.BaseStore` method), [13](#)

## Q

QueueStore (class in `tornadose.stores`), [13](#)

## R

register() (`tornadose.stores.BaseStore` method), [13](#)

## S

submit() (`tornadose.stores.BaseStore` method), [13](#)

## W

WebSocketSubscriber (class in `tornadose.handlers`), [14](#)