Terminado Documentation

Release 0.6

Thomas Kluyver

Apr 21, 2017

Contents

Contents:

CHAPTER 1

Using the TermSocket handler

`terminado.TermSocket` is the main API in Terminado. It is a subclass of `tornado.web.WebSocketHandler`, used to communicate between a pseudoterminal and term.js. You add it to your web application as a handler like any other:

```
app = tornado.web.Application([
        # ... other handlers ...
        (r"/websocket", terminado.TermSocket,
            {'term_manager': terminado.SingleTermManager(shell_command=['bash'])}),
    ], **kwargs)
```

Now, a page in your application can connect to `ws://<host>/websocket`. Using `terminado/_static/terminado.js`, you can do this using:

```
make_terminal(target_html_element, {rows:25, cols:80}, "ws://<host>/websocket");
```

Warning: `TermSocket` does not authenticate the connection at all, and using it with a program like `bash` means that anyone who can connect to it can run commands on your server. It is up to you to integrate the handler with whatever authentication system your application uses. For instance, in IPython, we subclass it like this:

```
class TermSocket(terminado.TermSocket, IPythonHandler):
    def get(self, *args, **kwargs):
        if not self.get_current_user():
            raise web.HTTPError(403)
        return super(TermSocket, self).get(*args, **kwargs)
```

Terminal managers

The terminal manager control the behaviour when you connect and disconnect websockets. Terminado offers three options:

class terminado.**SingleTermManager**(***kwargs*)
All connections to the websocket share a common terminal.

class terminado.**UniqueTermManager**(*max_terminals=None*, ***kwargs*)
Give each websocket a unique terminal to use.

class terminado.**NamedTermManager**(*max_terminals=None*, ***kwargs*)
Share terminals between websockets connected to the same endpoint.

You can also define your own behaviours, by subclassing any of these, or the base class. The important methods are described here:

class terminado.**TermManagerBase**(*shell_command*, *server_url=''*, *term_settings={}*, *extra_env=None*, *ioloop=None*)
Base class for a terminal manager.

get_terminal(*url_component=None*)
Override in a subclass to give a terminal to a new websocket connection

The TermSocket handler works with zero or one URL components (capturing groups in the URL spec regex). If it receives one, it is passed as the url_component parameter; otherwise, this is None.

new_terminal(***kwargs*)
Make a new terminal, return a PtyWithClients instance.

start_reading(*ptywclients*)
Connect a terminal to the tornado event loop to read data from it.

client_disconnected(*websocket*)
Override this to e.g. kill terminals on client disconnection.

This may still be subject to change as we work out the best API.

In the example above, the terminal manager was only attached to the websocket handler. If you want to access it from other handlers, for instance to list running terminals, attach the instance to your application, for instance in the settings dictionary.

CHAPTER 2

Using the Tornado UI Module

Terminado provides a Tornado UI Module. Once you have the websocket handler set up (see *Using the TermSocket handler*), add the module to your application:

```
from terminado import uimodule
# ...

app = tornado.web.Application(...
    ui_modules = {'Terminal': uimodule.Terminal},
)
```

Now, when you want a terminal in your application, you can put this in the template:

```
{% module Terminal("/websocket", rows=30, cols=90) %}
```

This will create a div, and include the necessary Javascript code to set up a terminal in that div and connect it to a websocket on `ws://<host>/websocket`.

If not specified, rows and cols default to 25 and 80, respectively.

For now, this assumes that term.js is available at `/xstatic/termjs/term.js`, and terminado.js at `/static/terminado.js`. To serve them from different locations, subclass `terminado.uimodule.Terminal`, overriding `javascript_files()`.

This is a Tornado websocket backend for the term.js Javascript terminal emulator library.

It evolved out of pyxterm, which was part of GraphTerm (as lineterm.py), v0.57.0 (2014-07-18), and ultimately derived from the public-domain Ajaxterm code, v0.11 (2008-11-13) (also on Github as part of QWeb).

Modules:

- `terminado.management`: controls launching virtual terminals, connecting them to Tornado's event loop, and closing them down.
- `terminado.websocket`: Provides a websocket handler for communicating with a terminal.
- `terminado.uimodule`: Provides a `Terminal` Tornado UI Module.

JS:

- `terminado/_static/terminado.js`: A lightweight wrapper to set up a term.js terminal with a websocket.

Usage example:

```
import os.path
import tornado.web
import tornado.ioloop
# This demo requires tornado_xstatic and XStatic-term.js
import tornado_xstatic

import terminado
STATIC_DIR = os.path.join(os.path.dirname(terminado.__file__), "_static")

class TerminalPageHandler(tornado.web.RequestHandler):
    def get(self):
        return self.render("termpage.html", static=self.static_url,
                           xstatic=self.application.settings['xstatic_url'],
                           ws_url_path="/websocket")

if __name__ == '__main__':
    term_manager = terminado.SingleTermManager(shell_command=['bash'])
    handlers = [
                (r"/websocket", terminado.TermSocket,
                     {'term_manager': term_manager}),
                (r"/", TerminalPageHandler),
                (r"/xstatic/(.*)", tornado_xstatic.XStaticFileHandler,
                     {'allowed_modules': ['termjs']})
               ]
    app = tornado.web.Application(handlers, static_path=STATIC_DIR,
                      xstatic_url = tornado_xstatic.url_maker('/xstatic/'))
    # Serve at http://localhost:8765/ N.B. Leaving out 'localhost' here will
    # work, but it will listen on the public network interface as well.
    # Given what terminado does, that would be rather a security hole.
    app.listen(8765, 'localhost')
    try:
        tornado.ioloop.IOLoop.instance().start()
    finally:
        term_manager.shutdown()
```

See the demos directory for more examples. This is a simplified version of the `single.py` demo.

Run the unit tests with:

```
$ nosetests
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

Index