tcpy Documentation

Release 0.0.1

Patrick Brodie

May 16, 2014

Contents:

CHAPTER 1

Overview

tcpy is a lightweight framework for asynchronous TCP Servers and associated clients.

Read, star, or contribute to the source code on Github

1.1 Installation

tcpy is hosted on PyPI. Easiest installation is with pip:

```
pip install tcpy
```

Alternatively, you can download a tarball of the source here.

Note: **tcpy** versions do not yet exist for Python 3 or for Windows.

CHAPTER 2

TCPY Tutorial

tcpy makes it extremely simple to make TCP Servers and associated clients in Python.

Associate a command to a TCPHandler, define its execute() method and tcpy has you up and running:

```
# Server
from tcpy import TCPServer, TCPHandler

# Our handler class must inherit from TCPHandler
class AdditionHandler(TCPHandler):
    def __init__(self, x, y):
        # Capture parameters as members of the class
        self.x = x
        self.y = y

    def execute(self):
        # success() will provide a well-formed success response
        return self.success(solution=self.x + self.y)

# Instantiate the server at default localhost:7272
server = TCPServer()
server.commands = {
    # Associate a command to our handler
    'add': AdditionHandler
}

if __name__ == "__main__":
    # Start listening for requests!
    server.listen()
```

On the client side, just execute() one of the server's commands:

```
# Client
from tcpy import TCPClient

print TCPClient().execute(cmd="add", x=1, y=2)
```

Which outputs: `{'solution': 3, 'success': True}`.

CHAPTER 3

TCPServer Objects

The `TCPServer` class handles accepting requests and queuing tasks for worker threads to complete.

3.1 TCPServer([host, port, commands, threads, poll_intv])

Initializes an instance of the `TCPServer` class.

- `host`: the hostname where the server will live. Defaults to locahost.
- `port`: the port on which the server will listen. Defaults to 7272.
- `commands`: dictionary mapping command strings to handler classes.
- `threads`: number of worker threads the server will spawn to execute tasks. Defaults to 4.
- `poll_intv`: the period of time a worker will sleep before polling the request queue for work.

3.2 listen()

Tells a `TCPServer` object to begin listening for requests. TCPY will log the host and port where it is listening to stdout.

3.3 The TCPServer.commands Dictionary

The `commands` dictionary of a `TCPServer` object is how the server knows which commands to execute. It maps command names (strings) to handler classes.

For example:

```
from tcpy import TCPServer
from foo import FooHandler

server = TCPServer()
server.commands = {
    'foo': FooHandler    # maps the command 'foo' onto the FooHandler class
}
server.listen()
```

Defining commands this way allows clients to execute specific commands similar to a remote procedure call. A `TCPClient` may call `execute` on a given command, and the `TCPServer` will instantiate the appropriate handler class to serve the client's request.

CHAPTER 4

TCPHandler Objects

In **tcpy**, commands are associated to handlers. A client can ask the server to execute a command, and the server will invoke the handler whose responsibility is to carry out that command. The `TCPHandler` class is the base building block for implementing handlers, which compose a `TCPServer`'s functionality.

All **tcpy** handlers should inherit from this class and define their behavior in an `execute()` method.

Associating a string command to a handler class within the `TCPServer`'s `command` dictionary will give the server the ability to execute the handler.

4.1 __init__(**params)

All parameters passed by a client with a request will be forwarded into the appropriate handler's `__init__()` method. They should be captured here as members of the handler class.

Note: In many cases a client's connection to the server will need to be maintained to communicate back and forth. Calling `super(MyHandler, self).__init__()` when initializing a handler will give the handler access to the connection to the client.

4.2 execute()

Defines the behavior of a given handler. Called on a worker thread when the command associated with a given handler is requested by a client.

Note: Must be implemented by subclasses of the `TCPHandler` class.

4.3 success([**kwargs])

Provides a wrapper for well-formed success responses. Returns a dictionary of the form:

```
{
    'success': True,
    ...       # kwargs
}
```

4.4 error(message[, **kwargs])

Provides a wrapper for well-formed error responses. Returns a dictionary of the form:

```
{
    'error': True,
    'message': message,
    ...    # kwargs
}
```

4.5 send(data)

Sends the given `data` (in dictionary form) to a client without closing the connection.

Note: A handler must call its parent's `__init__()` method in order to use the connection.

4.6 recv()

Receives data from a connected client and returns it in dictionary form.

Note: A handler must call its parent's `__init__()` method in order to use the connection.

CHAPTER 5

TCPClient Objects

The `TCPClient` class provides a concise interface for connecting and speaking to a `TCPServer` instance.

5.1 TCPClient([host, port])

Instantiates a `TCPClient` object.

- `host`: The host the target server is listening on. Defaults to localhost.
- `port`: The port the target server is listening on. Defaults to 7272.

5.2 connect()

Connects to the server.

Note: Available in v0.0.5 or higher. For prior versions, use `self.conn.connect()`

5.3 send(data)

Sends the given data to the server.

5.4 recv()

Receives data from the server and returns it.

5.5 disconnect()

Closes a connection to the server.

Note: Available in v0.0.5 or higher. For prior versions, use `self.conn.finish()`

5.6 execute(cmd[, **params])

Calls the server to execute the given command and returns the result.