FatBotSlim Documentation

Release 0.2

Mathieu D. (MatToufoutu)

April 12, 2015

Contents

CHAPTER 1

Introduction

1.1 About FatBotSlim

Author Mathieu D. (MatToufoutu)

Contact mattoufootu[at]gmail.com

Homepage https://github.com/mattoufoutu/fatbotslim

Documentation http://fatbotslim.rtfd.org

Issue tracker https://github.com/mattoufoutu/fatbotslim/issues

Git clone URI https://github.com/mattoufoutu/fatbotslim.git

FatBotSlim is an asynchronous IRC bot library written in Python using the `gevent` library.

A FatBotSlim bot consists of a bot object to which handlers are registered to make it react to events. When triggered, the handlers' methods are called asynchronously, therefore you don't have to worry about time-consuming operations that could block your code execution.

1.2 Installation

This document describes the various ways to install FatBotSlim.

1.2.1 Using pip

From GitHub

Just run:

```
pip install git+git://github.com/mattoufoutu/fatbotslim.git
```

From PyPI

TODO

1.2.2 Manually

From GitHub

1. Clone the repository:

```
git clone git://github.com/mattoufoutu/fatbotslim.git
```

2. Install the package:

```
cd fatbotslim
python setup.py install
```

From PyPI

TODO

CHAPTER 2

Usage

2.1 Tutorials

2.1.1 Writing a simple bot

In this tutorial, we'll take the code from the README.md and explain what happens, it is a bot that answers *Hello <username>!* when someone says *!hello* in a public message.

The code

```
1  from fatbotslim.cli import make_bot, main
2  from fatbotslim.handlers import CommandHandler, EVT_PUBLIC
3
4
5  class HelloCommand(CommandHandler):
6      triggers = {
7          u'hello': [EVT_PUBLIC],
8      }
9
10     def hello(self, msg):
11         self.irc.msg(msg.dst, u"Hello {0}!".format(msg.src.name))
12
13
14 bot = make_bot()
15 bot.add_handler(HelloCommand)
16 main(bot)
```

Explanations

Imports:

- Line 1:

  We import 2 functions, `fatbotslim.cli.make_bot()` and `fatbotslim.cli.main()`, both are useful when dealing with a bot that will connect to only one server.

  The `fatbotslim.cli.make_bot()` function takes care of gathering the command line options, which are used to create a `fatbotslim.irc.IRC` instance, and then returns it.

The `fatbotslim.cli.main()` function is there to spawn the bot in a greenlet (see gevent's documentation for details about this) and run the main loop for you.

- Line 2:

  Here we import the `fatbotslim.handlers.CommandHandler` that will be used to make the bot react to the *!hello* command and the `EVT_PUBLIC` variable, which is used to define

The handler:

- Lines 5-7:

  The `triggers` attribute is a `dict` used to define to which commands and on which type of events the bot should react. It consists of command names mapped to a tuple listing events. If we wanted the bot to react to *!hello* both in public and private messages, the value would have been *('public', 'private')*.

- Lines 9-10:

  This is the method that will be called when the *!hello* command will be triggered, it must have the same name as defined in `triggers` and takes two arguments, the first is the instance of the `fatbotslim.irc.bot.Message` that triggered the command, and the second is an instance of `fatbotslim.irc.bot.IRC` which is used to communicate back with the server.

Starting the bot:

- Line 12:

  Nothing to explain here, the `fatbotslim.cli.make_bot()` function is already described in the imports part.

- Line 13:

  The previously created handler is instanciated and added to the bot's handlers.

- Line 14:

  The bot is passed to the `fatbotslim.cli.main()` function, which will take care of

2.1.2 Writing a multi-server bot

In this tutorial, we'll take the code from the "simple bot" tutorial, and make it run on multiple servers.

The code

```
1  from fatbotslim.irc.bot import IRC, run_bots
2  from fatbotslim.handlers import CommandHandler, EVT_PUBLIC
3
4
5  class HelloCommand(CommandHandler):
6      triggers = {
7          u'hello': [EVT_PUBLIC],
8      }
9
10     def hello(self, msg):
11         self.irc.msg(msg.dst, u"Hello {0}!".format(msg.src.name))
```

```
12
13
14  servers = [
15      {
16          'server': 'irc.rizon.net',
17          'port': 6697,
18          'ssl': True,
19          'nick': 'fatbotslim',
20          'realname': 'fatbotslim',
21          'channels': ['#testbot']
22      },
23      {
24          'server': 'irc.freenode.net',
25          'port': 6697,
26          'ssl': True,
27          'nick': 'fatbotslim',
28          'realname': 'fatbotslim',
29          'channels': ['#testbot']
30      }
31  ]
32
33  bots = []
34  for server in servers:
35      bot = IRC(server)
36      bot.add_handler(HelloCommand)
37      bots.append(bot)
38
39  run_bots(bots)
```

Explanations

Imports

- Line 1:

  The `fatbotslim.irc.bot.IRC` class will be used to instanciate and configure our bots. And the `fatbotslim.irc.bot.run_bots()` will allow us to run all our bots in parallel.

Configuration

- Lines 12-29:

  Instead of having our settings passed via the command line, we create a `dict` for each server our bot will connect to.

Running the bots

- Lines 31-35:

  Here we create a `list` to hold our `fatbotslim.irc.bot.IRC` instances. We loop over each server configuration, and use them in our bots instanciations.

- Line 37:

The `fatbotslim.irc.bot.run_bots()` function takes a `list` of bots as argument, and launches each one's main loop in a `greenlet`.

2.2 Creating custom handlers

Note: Internally, all processed data are unicode strings. If you need to decode strings, you can use the provided `fatbotslim.irc.u()` function.

2.2.1 Basic handlers

Basic handlers simply react to IRC events by calling methods mapped to them, they are subclasses of `fatbotslim.handlers.BaseHandler`.

Events are mapped to methods names in the handler's `commands` attribute, which is a `dict` where the key is an IRC event as defined in the `fatbotslim.irc.codes` module and the value is the name of the method to call. The mapped methods take just one attribute, the `fatbotslim.irc.bot.Message` object which triggered the event. To communicate back to the server, use the handler's `irc` (which is a `fatbotslim.bot.IRC` instance).

A good example of a custom handler is FatBotSlim's integrated `fatbotslim.handlers.CTCPHandler`:

```
class CTCPHandler(BaseHandler):
    """
    Reacts to CTCP events (VERSION,SOURCE,TIME,PING). (enabled by default)
    """
    commands = {
        CTCP_VERSION: 'version',
        CTCP_SOURCE: 'source',
        CTCP_TIME: 'time',
        CTCP_PING: 'ping'
    }

    def version(self, msg):
        self.irc.ctcp_reply(
            u'VERSION', msg.src.name,
            u'{0}:{1}:{2}'.format(NAME, VERSION, platform.system())
        )

    def source(self, msg):
        self.irc.ctcp_reply(
            u'SOURCE', msg.src.name,
            u'https://github.com/mattoufoutu/fatbotslim'
        )
        self.irc.ctcp_reply(u'SOURCE', msg.src.name)

    def time(self, msg):
        now = datetime.now().strftime(u'%a %b %d %I:%M:%S%p %Y %Z').strip()
        self.irc.ctcp_reply(u'TIME', msg.src.name, now)

    def ping(self, msg):
        self.irc.ctcp_reply(u'PING', msg.src.name, u' '.join(msg.args))
```

Another, simpler, basic handler is the integrated `fatbotslim.handlers.PingHandler`, this one simply answers to server's PINGs:

```
class PingHandler(BaseHandler):
    """
    Answers to PINGs sent by the server. (enabled by default)
    """
    commands = {
        PING: 'ping'
    }

    def ping(self, msg):
        self.irc.cmd(u'PONG', u' '.join(msg.args))
```

2.2.2 Command handlers

The `fatbotslim.handlers.CommandHandler` is a special kind of handler that reacts only to `PRIVMSG` and `NOTICE` messages, they are used to implement `!foo`-like commands to your bot.

The prefix character is defined by the handler's `trigger_char` attribute, and defaults to `!`.

Commands are defined in the handler's `triggers` attribute, a dict that maps method names to events they should react to. Possible events are `EVT_PUBLIC`, `EVT_PRIVATE`, and `EVT_NOTICE`. The methods take just 1 argument, the first is a `fatbotslim.irc.bot.Message` object, and the second is a `fatbotslim.irc.bot.IRC` object used to send messages back to the server.

For example, the message `!foo bar` would call the handler's `foo()` method.

Here is a command handler that says hello when it receives `!hello` in public:

```
from fatbotslim.handlers import CommandHandler, EVT_PUBLIC

class HelloCommand(CommandHandler):
    triggers = {
        u'hello': [EVT_PUBLIC],
    }

    def hello(self, msg):
        self.irc.msg(msg.dst, u"Hello, {0}!".format(msg.src.name))
```

If you wanted the handler to answer also to private messages, you would simply have to add 'private' to the "hello" event list and set the answer destination accordingly:

```
from fatbotslim.handlers import CommandHandler, EVT_PUBLIC, EVT_PRIVATE

class HelloCommand(CommandHandler):
    triggers = {
        u'hello': [EVT_PUBLIC, EVT_PRIVATE],
    }

    def hello(self, msg):
        dst = msg.src.name if (msg.dst == irc.nick) else msg.dst
        self.irc.msg(dst, u"Hello {0}!".format(msg.src.name))
```

2.3 Utilities

2.3.1 Commands Help

If you want to have help messages for your handlers' commands automatically generated, *fatbotslim* provides a convenience handler.

The `fatbotslim.handlers.HelpHandler` provides two kind of help messages, one that simply lists available commands when *!help* is called (assuming *!* is your current *trigger_char*), and one that displays the command's docstring when *help [command]* is called.

To use this feature, simply document your commands' docstrings (try to keep a consistent format across all your docstrings), and add the handler to your bot.

```
from fatbotslim.cli import make_bot, main
from fatbotslim.handlers import CommandHandler, HelpHandler, EVT_PUBLIC

class HelloCommand(CommandHandler):
    triggers = {
        u'hello': [EVT_PUBLIC],
    }

    def hello(self, msg):
        """hello - says hello"""
        self.irc.msg(msg.dst, u"Hello {0}!".format(msg.src.name))

    def say(self, msg):
        """say <message> - simply repeats the message"""
        self.irc.msg(msg.dst, ' '.join(msg.args[1:]))

bot = make_bot()
bot.add_handler(HelpHandler)
bot.add_handler(HelloCommand)
main(bot)
```

Given the previous code, if one called *!help*, the bot would answer:

```
Available commands: hello, say
```

and if one called *!help say*, the bot would answer:

```
say <message> - simply repeats the message
```

2.3.2 Colored Messages

To send colored messages, you can use the `fatbotslim.irc.colors.ColorMessage` class. It mimics strings behaviour, and thus allows to call string methods on it.

```
from fatbotslim.irc.colors import ColorMessage
message = ColorMessage('sOmE rAnDoM tExT', color='blue')
message = ColorMessage('sOmE rAnDoM tExT', color='blue', bold=True)
message = ColorMessage('sOmE rAnDoM tExT', color='blue', underline=True)
message = ColorMessage('sOmE rAnDoM tExT', color='blue', highlight=True)
upper_message = message.upper()
title_message = message.title()
```

Available colors are:

- blue
- brown
- dark_green
- magenta
- purple
- dark_grey
- light_grey
- yellow
- black
- teal
- cyan
- olive
- green
- white
- dark_blue
- red

If no color is specified when instanciating `fatbotslim.irc.colors.ColorMessage`, the value defaults to "black".

2.3.3 Rights Management

FatBotSlim provides a built-in handler (`fatbotslim.handlers.RightsHandler`) to manage who should be allowed to run specific commands. It allows to define which users can run which commands, and on which event(s) type(s).

This special handler is automatically enabled, and is accessible through the `fatbotslim.irc.bot.IRC.rights` attribute. It can be permanently disabled using the `fatbotslim.irc.bot.IRC.enable_rights()` method, and can be re-enabled using the `fatbotslim.irc.bot.IRC.disable_rights()` method.

Settings permissions

Adding a new permission is done using the `fatbotslim.irc.bot.IRC.rights.set_restriction()` method.

For example, to restrict usage of the *hello* command to a user named *LeetUser* in public messages, the following code should be used (assuming *bot* is the `fatbotslim.irc.bot.IRC` instance:

```
bot.rights.set_permission('hello', 'LeetUser', [EVT_PUBLIC])
```

Once this is done, only *LeetUser* will be allowed to use the *hello* command, and only in public messages.

Global rights can also be set using * as the username. In the following example, *LeetUser* would be allowed to use the *hello* command in private messages only, and all the other users would be allowed to use it in public messages and notices only.

```
bot.rights.set_restriction('hello', 'LeetUser', [EVT_PRIVATE])
bot.rights.set_restriction('hello', '*', [EVT_PUBLIC, EVT_NOTICE])
```

Removing permissions

Removing a permission is done using the `fatbotslim.irc.bot.IRC.rights.del_restriction()` method.

The following code snippet would remove the previously set permission for *LeetUser*.

```
bot.rights.del_restriction('hello', 'LeetUser', [EVT_PRIVATE])
```

Only given event(s) type(s) are removed from the permission, so, if *LeetUser* was previously allowed to use the *hello* command in public messages too, it would still have the right to.

CHAPTER 3

API Reference

3.1 fatbotslim.irc

3.1.1 fatbotslim.irc.bot

This module contains IRC protocol related stuff.

class `fatbotslim.irc.bot.`**`IRC`**(*settings*)

The main IRC bot class.

The only expected argument is the bot's configuration, it should be a `dict` with at least the following keys defined:

- server: the ircd's host (`str`)
- port: the ircd's port (`int`)
- ssl: connect to the server using SSL (`bool`)
- channels: the channels to join upon connection (`list`)
- nick: the bot's nickname (`str`)
- realname: the bot's real name (`str`)

Parameters **settings** (*dict*) – bot configuration.

`add_handler`(*handler*, *args=None*, *kwargs=None*)

Registers a new handler.

Parameters

- **handler** (`fatbotslim.handlers.BaseHandler`) – handler to register.
- **args** (*list*) – positional arguments to pass to the handler's constructor.
- **kwargs** (*dict*) – keyword arguments to pass to the handler's constructor.

`cmd`(*command*, *args*, *prefix=None*)

Sends a command to the server.

Parameters

- **command** (*unicode*) – IRC code to send.
- **args** (*basestring*) – arguments to pass with the command.
- **prefix** (*str or None*) – optional prefix to prepend to the command.

ctcp_reply(*command*, *dst*, *message=None*)
Sends a reply to a CTCP request.

Parameters

- **command** (*str*) – CTCP command to use.
- **dst** (*str*) – sender of the initial request.
- **message** (*str*) – data to attach to the reply.

disable_rights()
Disables rights management provided by `fatbotslim.handlers.RightsHandler`.

disconnect()
Disconnects the bot from the server.

enable_rights()
Enables rights management provided by `fatbotslim.handlers.RightsHandler`.

join(*channel*)
Make the bot join a channel.

Parameters **channel** (*str*) – new channel to join.

msg(*target*, *msg*)
Sends a message to an user or channel.

Parameters

- **target** (*str*) – user or channel to send to.
- **msg** (*str*) – message to send.

notice(*target*, *msg*)
Sends a NOTICE to an user or channel.

Parameters

- **target** (*str*) – user or channel to send to.
- **msg** (*basestring*) – message to send.

classmethod randomize_nick(*base*, *suffix_length=3*)
Generates a pseudo-random nickname.

Parameters

- **base** (*unicode*) – prefix to use for the generated nickname.
- **suffix_length** (*int*) – amount of digits to append to *base*

Returns generated nickname.

Return type unicode

run()
Connects the bot and starts the event loop.

set_nick(*nick*)
Changes the bot's nickname.

Parameters **nick** (*unicode*) – new nickname to use

class fatbotslim.irc.bot.**Message**(*data*)
Holds informations about a line received from the server.

Parameters **data** (*unicode*) – line received from the server.

classmethod **parse**(*data*)
Extracts message informations from *data*.

Parameters **data** (*unicode*) – received line.

Returns extracted informations (source, destination, command, args).

Return type tuple(Source, str, str, list)

Raise `fatbotslim.irc.NullMessage` if *data* is empty.

exception `fatbotslim.irc.bot.`**NullMessage**
Raised when an empty line is received from the server.

class `fatbotslim.irc.bot.`**Source**(*prefix*)
Holds informations about a message sender.

Parameters **prefix** (*unicode*) – prefix with format `<servername>|<nick>['!'<user>]['@'<host>]`.

classmethod **parse**(*prefix*)
Extracts informations from *prefix*.

Parameters **prefix** (*unicode*) – prefix with format `<servername>|<nick>['!'<user>]['@'<host>]`.

Returns extracted informations (nickname or host, mode, username, host).

Return type tuple(str, str, str, str)

`fatbotslim.irc.bot.`**run_bots**(*bots*)
Run many bots in parallel.

Parameters **bots** (*list*) – IRC bots to run.

3.1.2 fatbotslim.irc.tcp

This module contains the low-level networking stuff.

class `fatbotslim.irc.tcp.`**SSL**(*host*, *port*, *timeout=300*)
SSL wrapper for a `fatbotslim.irc.tcp.TCP` connection.

Parameters

- **host** (*str*) – server's hostname
- **port** (*int*) – server's port
- **timeout** (*int*) – maximum time a request/response should last.

class `fatbotslim.irc.tcp.`**TCP**(*host*, *port*, *timeout=300*)
A TCP connection.

Parameters

- **host** (*str*) – server's hostname
- **port** (*int*) – server's port
- **timeout** (*int*) – maximum time a request/response should last.

connect()
Connects the socket and spawns the send/receive loops.

disconnect()
Closes the socket.

3.1.3 fatbotslim.irc.codes

This module lists the known IRC codes as defined in the **RFC 1459**.

—

It also defines some custom codes:

RPL_CONNECTED (`001`): Connection established with the server.

RPL_SERVERVERSION (`002`): Server's name/version.

RPL_SERVERCREATED (`003`): Server's creation date.

RPL_SERVERMODES (`004`): Modes supported by the server.

RPL_ISUPPORT (`005`): Protocol extensions supported by the server.

CTCP_VERSION, **CTCP_PING**, **CTCP_TIME**, **CTCP_SOURCE**: self-explanatory.

PING, **PRIVMSG**, **NOTICE**, **JOIN**, **PART**: self-explanatory.

—

Codes are also grouped by type to make matching them easier:

ERRORS, **RESPONSES**, **RESERVED**, **CTCP**, **OTHERS**

—

There are also 2 special code sets:

ALL_CODES: Matches any code if listed in the groups previously mentionned.

UNKNOWN_CODE: Matches any code not listed in `ALL_CODES`.

3.1.4 fatbotslim.irc.colors

This module provides message colors capabilities.

class `fatbotslim.irc.colors.`**`ColorMessage`**(*content*, *color='black'*, *bold=False*, *underline=False*, *highlight=False*)

Allows to create colorized strings. Created objects behave like real strings, allowing to call *str* methods.

Parameters

- **content** (*unicode*) – message to colorize.
- **color** (*str*) – one of `fatbotslim.irc.colors.ColorMessage._colors`.
- **bold** (*bool*) – if the string has to be in bold.
- **underline** (*bool*) – if the string has to be underlined.
- **highlight** (*bool*) – if the string foreground and background has to be switched.

static `colorize`(*string*, *color='black'*, *bold=False*, *underline=False*, *highlight=False*)

Parameters

- **string** (*unicode*) – message to colorize.
- **color** (*str*) – one of `fatbotslim.irc.colors.ColorMessage._colors`.
- **bold** (*bool*) – if the string has to be in bold.
- **underline** (*bool*) – if the string has to be underlined.

- **highlight** (*bool*) – if the string foreground and background has to be switched.

fatbotslim.irc.**u**(*s*, *errors='ignore'*)
Automatically detects given string's encoding and returns its unicode form. Decoding errors are handled according to the *errors* argument, see *unicode()* documentation for more details.

Parameters

- **s** (*str*) – string to decode.
- **errors** (*str*) – decoding error handling behaviour.

Returns decoded string

Return type unicode

3.2 fatbotslim.cli

This module contains utilities to run a bot from the command line.

fatbotslim.cli.**main**(*bot*)
Entry point for the command line launcher.

Parameters **bot** (fatbotslim.irc.bot.IRC) – the IRC bot to run

fatbotslim.cli.**make_bot**()
Creates a new bot instance ready to be launched.

fatbotslim.cli.**make_parser**()
Creates an argument parser configured with options to run a bot from the command line.

Returns configured argument parser

Return type argparse.ArgumentParser

3.3 fatbotslim.handlers

This module contains a collection of handlers to react to basic IRC events and allow creation of custom handlers.

class fatbotslim.handlers.**BaseHandler**(*irc*)
The base of every handler.

A handler should at least have a commands attribute of type dict which maps IRC codes (as defined in fatbotslim.irc.codes) to methods.

Mapped methods take 1 argument, the fatbotslim.irc.bot.Message object that triggered the event.

class fatbotslim.handlers.**CommandHandler**(*irc*)
The CommandHandler is a special kind of handler that eases the creation of bots that react to prefixed commands (like !command). It only reacts to PRIVMSG and NOTICE messages.

The prefix character is defined by the handler's trigger_char attribute, and defaults to !.

Commands are defined in the handler's triggers attribute, a dict that maps method names to events to which they should react. Possible events are EVT_PUBLIC, EVT_PRIVATE, and EVT_NOTICE. The methods should take 1 argument, which is the fatbotslim.irc.bot.Message object that triggered the event.

For example, the message !foo bar would call the handler's foo() method.

Here is a command handler that says hello when it receives !hello in public:

```
class HelloCommand(CommandHandler):
    triggers = {
        'hello': [EVT_PUBLIC],
    }
    def hello(self, msg):
        self.irc.msg(msg.dst, "Hello, {0}!".format(msg.src.name))
```

class fatbotslim.handlers.**PingHandler**(*irc*)
Answers to PINGs sent by the server. (enabled by default)

class fatbotslim.handlers.**CTCPHandler**(*irc*)
Reacts to CTCP events (VERSION,SOURCE,TIME,PING). (enabled by default)

class fatbotslim.handlers.**UnknownCodeHandler**(*irc*)
Logs messages for which the IRC code is unknown. (enabled by default)

class fatbotslim.handlers.**HelpHandler**(*irc*)
Provides automatic help messages for fatbotslim.handlers.CommandHandler commands.

help(*msg*)
help [command] - displays available commands, or help message for given command

class fatbotslim.handlers.**RightsHandler**(*irc*)
Provides rights management for fatbotslim.handlers.CommandHandler commands.

del_restriction(*command*, *user*, *event_types*)
Removes restriction for given *command*.

Parameters

- **command** (*str*) – command on which the restriction should be removed.
- **user** (*str*) – username for which restriction should be removed.
- **event_types** (*list*) – types of events that should be removed from restriction.

handle_rights(*msg*)
Catch-all command that is called whenever a restricted command is triggered.

Parameters **msg** (fatbotslim.irc.Message) – message that triggered the command.

set_restriction(*command*, *user*, *event_types*)
Adds restriction for given *command*.

Parameters

- **command** (*str*) – command on which the restriction should be set.
- **user** (*str*) – username for which the restriction applies.
- **event_types** (*list*) – types of events for which the command is allowed.

3.4 fatbotslim.log

This module contains everything useful to enable logging.

class fatbotslim.log.**ColorFormatter**(*fmt=None*, *datefmt=None*)
A logging formatter that displays the loglevel with colors and the logger name in bold.

Initialize the formatter with specified format strings.

Initialize the formatter either with the specified format string, or a default as described above. Allow for specialized date formatting with the optional datefmt argument (if omitted, you get the ISO8601 format).

format(*record*)
Overrides the default `logging.Formatter.format()` to add colors to the `record`'s `levelname` and `name` attributes.

`fatbotslim.log.`**create_logger**(*name*, *level='INFO'*)
Creates a new ready-to-use logger.

Parameters

- **name** (*str*) – new logger's name
- **level** (`str` or `int`) – default logging level.

Returns new logger.

Return type `logging.Logger`

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*

f

T

U