irc3 Documentation

Release 1.1.5

Gael Pasgrimaud

Jan 18, 2020

Contents

1 Installation **3**

2 Quick start **5**

3 Documentation **9**

3.1 irc3.dec decorators . . . 9
3.2 irc3.utils Utils . . . 10
3.3 irc3.rfc RFC1459 . . . 12
3.4 irc3.dcc DCC . . . 44
3.5 Reloadable plugins . . . 45
3.6 irc3.plugins.asynchronious Asynchronious events . . . 46
3.7 irc3.plugins.autocommand Autocommand plugin . . . 48
3.8 irc3.plugins.autojoins Auto join plugin . . . 49
3.9 irc3.plugins.casefold casefolding plugin . . . 49
3.10 irc3.plugins.command Command plugin . . . 49
3.11 irc3.plugins.core Core plugin . . . 53
3.12 irc3.plugins.cron Cron plugin . . . 53
3.13 irc3.plugins.ctcp CTCP replies . . . 54
3.14 irc3.plugins.dcc DCC Chat plugin . . . 54
3.15 irc3.plugins.feeds Feeds plugin . . . 55
3.16 irc3.plugins.fifo Fifo plugin . . . 57
3.17 irc3.plugins.human Human plugin . . . 57
3.18 irc3.plugins.log Log plugin . . . 57
3.19 irc3.plugins.logger Channel logger plugin . . . 58
3.20 irc3.plugins.pager Paginate large output . . . 58
3.21 irc3.plugins.quakenet QuakeNet authorization . . . 59
3.22 irc3.plugins.sasl SASL authentification . . . 59
3.23 irc3.plugins.search Search plugin . . . 60
3.24 irc3.plugins.shell_command Shell commands . . . 60
3.25 irc3.plugins.slack Slack plugin . . . 60
3.26 irc3.plugins.social Social networking . . . 61
3.27 irc3.plugins.storage Storage plugin . . . 62
3.28 irc3.plugins.uptime Uptime plugin . . . 64
3.29 irc3.plugins.userlist User list plugin . . . 64
3.30 irc3.plugins.web Web plugin . . . 65
3.31 Contribute . . . 65

4 Indices and tables 67

Python Module Index 69

Index 71

A pluggable irc client library based on python's asyncio.

Requires python 3.5+

Python 2 is no longer supported, but if you don't have a choice you can use an older version:

```
$ pip install "irc3<0.9"
```

Source: https://github.com/gawel/irc3/

Docs: https://irc3.readthedocs.io/

Irc: irc://irc.freenode.net/irc3 (www)

I've spent hours writing this software, with love. Please consider tipping if you like it:

BTC: 1PruQAwByDndFZ7vTeJhyWefAghaZx9RZg

ETH: 0xb6418036d8E06c60C4D91c17d72Df6e1e5b15CE6

LTC: LY6CdZcDbxnBX9GFBJ45TqVj8NykBBqsmT

CHAPTER 1

Installation

Using pip:

```
$ pip install irc3
```

CHAPTER 2

Quick start

irc3 provides a basic template to help you to quickly test a bot. Here is how to create a bot named `mybot`.

Create a new directory and cd to it:

```
$ mkdir mybot
$ cd mybot
```

Then use the template:

```
$ python -m irc3.template mybot
```

This will create an almost ready to use `config.ini` file and a simple plugin named `mybot_plugin.py` that says «Hi» when the bot or someone else joins a channel and includes an `echo` command.

Here is what the config file will looks like:

```
[bot]
nick = mybot
username = mybot

host = localhost
port = 6667

# uncomment this if you want ssl support
# ssl = true
# uncomment this if you don't want to check the certificate
# ssl_verify = CERT_NONE

# uncomment this if you want to use sasl authentication
# sasl_username = mybot
# sasl_password = yourpassword

includes =
    irc3.plugins.command
#    irc3.plugins.uptime
```

(continues on next page)

(continued from previous page)

```
#    irc3.plugins.ctcp
    mybot_plugin

# the bot will join #mybot_channel
# ${#} is replaced by the # char
autojoins =
    ${#}mybot_channel

# Autojoin delay, disabled by default
# float or int value
# autojoin_delay = 3.1

# The maximum amount of lines irc3 sends at once.
# Default to 4, set to 0 to disable
# flood_burst = 10

# The number of lines per $flood_rate_delay seconds irc3 sends after reaching
# the $flood_burst limit.
# Default to 1
# flood_rate = 2

# The bot will send $flood_rate messages per $flood_rate_delay seconds
# Default to 1
# flood_rate_delay = 5

[irc3.plugins.command]
# command plugin configuration

# set command char
cmd = !

# set guard policy
guard = irc3.plugins.command.mask_based_policy

[irc3.plugins.command.masks]
# this section is used by the guard to secure the bot's command
# change your nickname and uncomment the line below
# mynick!*@* = all_permissions
* = view
```

And here is the plugin:

```
# -*- coding: utf-8 -*-
from irc3.plugins.command import command
import irc3


@irc3.plugin
class Plugin:

    def __init__(self, bot):
        self.bot = bot

    @irc3.event(irc3.rfc.JOIN)
    def say_hi(self, mask, channel, **kw):
        """Say hi when someone join a channel"""
        if mask.nick != self.bot.nick:
```

(continues on next page)

(continued from previous page)

```
            self.bot.privmsg(channel, 'Hi %s!' % mask.nick)
        else:
            self.bot.privmsg(channel, 'Hi!')

    @command(permission='view')
    def echo(self, mask, target, args):
        """Echo

            %%echo <message>...
        """
        yield ' '.join(args['<message>'])
```

Have a look at those file and edit the config file for your needs. You may have to edit:

- the autojoin channel
- your irc mask in the `irc3.plugins.command.mask` section

Once you're done with editing, run:

```
$ irc3 config.ini
```

Check the help of the `irc3` command.

```
$ irc3 -h
```

If you're enjoying it, you can check for more detailed docs below. And some more examples here: https://github.com/gawel/irc3/tree/master/examples

CHAPTER 3

Documentation

3.1 irc3.dec decorators

3.1.1 plugin

irc3.dec.**plugin**(*wrapped*)
register a class as plugin

3.1.2 bot event

class irc3.dec.**event**(*regexp*, *callback=None*, *iotype='in'*, *venusian_category='irc3.rfc1459'*)
register a method or function an irc event callback:

```
>>> @event('^:\S+ 353 [^&#]+(?P<channel>\S+) :(?P<nicknames>.*)')
... def on_names(bot, channel=None, nicknames=None):
...     '''this will catch nickname when you enter a channel'''
...     print(channel, nicknames.split(':'))
```

The callback can be either a function or a plugin method

If you specify the *iotype* parameter to *"out"* then the event will be triggered when the regexp match something **sent** by the bot.

For example this event will repeat private messages sent by the bot to the *#irc3* channel:

```
>>> @event(r'PRIVMSG (?P<target>[^#]+) :(?P<data>.*)', iotype='out')
... def msg3(bot, target=None, data=None):
...     bot.privmsg('#irc3',
...                 '<{0}> {1}: {2}'.format(bot.nick, target, data))
```

3.1.3 bot extend

irc3.dec.**extend**(*func*)
Allow to extend a bot:

Create a module with some useful routine:

```
# -*- coding: utf-8 -*-
import irc3


@irc3.extend
def my_usefull_function(bot, *args):
    return 'my_usefull_function(*%s)' % (args,)


@irc3.plugin
class MyPlugin(object):

    def __init__(self, bot):
        self.bot = bot

    @irc3.extend
    def my_usefull_method(self, *args):
        return 'my_usefull_method(*%s)' % (args,)
```

Now you can use those routine in your bot:

```
>>> bot = IrcBot()
>>> bot.include('myextends')
>>> print(bot.my_usefull_function(1))
my_usefull_function(*(1,))
>>> print(bot.my_usefull_method(2))
my_usefull_method(*(2,))
```

3.2 irc3.utils Utils

class irc3.utils.**IrcString**
Argument wrapper

hostname
return host name:

```
>>> print(IrcString('foo!user@host').hostname)
host
>>> IrcString('#foo').hostname is None
True
>>> IrcString('irc.freenode.net').hostname is None
True
```

is_channel
return True if the string is a channel:

```
>>> IrcString('#channel').is_channel
True
```

(continues on next page)

(continued from previous page)

```
>>> IrcString('&channel').is_channel
True
```

is_nick
return True if the string is a nickname:

```
>>> IrcString('foo').is_nick
True
```

is_server
return True if the string is a server:

```
>>> IrcString('irc.freenode.net').is_server
True
```

lnick
return nick name in lowercase:

```
>>> print(IrcString('Foo').lnick)
foo
```

nick
return nick name:

```
>>> print(IrcString('foo').nick)
foo
>>> print(IrcString('foo!user@host').nick)
foo
>>> IrcString('#foo').nick is None
True
>>> IrcString('irc.freenode.net').nick is None
True
```

tagdict
return a dict converted from this string interpreted as a tag-string

```
>>> from pprint import pprint
>>> dict_ = IrcString('aaa=bbb;ccc;example.com/ddd=eee').tagdict
>>> pprint({str(k): str(v) for k, v in dict_.items()})
{'aaa': 'bbb', 'ccc': 'None', 'example.com/ddd': 'eee'}
```

username
return user name:

```
>>> print(IrcString('foo!user@host').username)
user
>>> IrcString('#foo').username is None
True
>>> IrcString('irc.freenode.net').username is None
True
```

irc3.utils.**as_list**(*value*)
clever string spliting:

```
>>> print(as_list('value'))
['value']
```

(continues on next page)

(continued from previous page)

```
>>> print(as_list('v1 v2'))
['v1', 'v2']
>>> print(as_list(None))
[]
>>> print(as_list(['v1']))
['v1']
```

irc3.utils.**as_channel**(*value*)
Always return a channel name:

```
>>> print(as_channel('chan'))
#chan
>>> print(as_channel('#chan'))
#chan
>>> print(as_channel('&chan'))
&chan
```

irc3.utils.**split_message**(*message*, *max_length*)
Split long messages

class irc3.utils.**Logger**(*name*, *level=0*)
Replace the default logger to add a set_irc_targets() method

set_irc_targets(*bot*, **targets*)
Add a irc Handler using bot and log to targets (can be nicks or channels:

```
>>> log = logging.getLogger('irc.mymodule')
>>> log.set_irc_targets(bot, '#chan', 'admin')
```

class irc3.utils.**Config**
Simple dict wrapper:

```
>>> c = Config(dict(a=True))
>>> c.a
True
```

irc3.utils.**parse_config**(*main_section*, **filenames*)
parse config files

irc3.utils.**extract_config**(*config*, *prefix*)
return all keys with the same prefix without the prefix

irc3.utils.**maybedotted**(*name*)
Resolve dotted names:

```
>>> maybedotted('irc3.config')
<module 'irc3.config' from '...'>
>>> maybedotted('irc3.utils.IrcString')
<class 'irc3.utils.IrcString'>
```

3.3 irc3.rfc RFC1459

3.3.1 Replies (REPL)

259 - RPL_ADMINEMAIL

Format `:{srv} 259 {nick} :{admin_info}`

Match `^:(?P<srv>\S+) 259 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_ADMINEMAIL)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

257 - RPL_ADMINLOC1

Format `:{srv} 257 {nick} :{admin_info}`

Match `^:(?P<srv>\S+) 257 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_ADMINLOC1)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

258 - RPL_ADMINLOC2

Format `:{srv} 258 {nick} :{admin_info}`

Match `^:(?P<srv>\S+) 258 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_ADMINLOC2)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

256 - RPL_ADMINME

Format `:{srv} 256 {nick} {server} :Administrative info`

Match `^:(?P<srv>\S+) 256 (?P<me>\S+) (?P<server>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_ADMINME)
def myevent(bot, srv=None, me=None, server=None, data=None):
    # do something
```

301 - RPL_AWAY

Format `:{srv} 301 {nick} {nick} :{away_message}`

Match `^:(?P<srv>\S+) 301 (?P<me>\S+) (?P<nick>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_AWAY)
def myevent(bot, srv=None, me=None, nick=None, data=None):
    # do something
```

367 - RPL_BANLIST

Format :`{srv} 367 {nick} {channel} {banid}`

Match ^:`(?P<srv>\S+) 367 (?P<me>\S+) (?P<channel>\S+) (?P<banid>\S+)`

Example:

```
@irc3.event(rfc.RPL_BANLIST)
def myevent(bot, srv=None, me=None, channel=None, banid=None):
    # do something
```

324 - RPL_CHANNELMODEIS

Format :`{srv} 324 {nick} {channel} {mode} {mode_params}`

Match ^:`(?P<srv>\S+) 324 (?P<me>\S+) (?P<channel>\S+) (?P<mode>\S+) (?P<mode_params>\S+)`

Example:

```
@irc3.event(rfc.RPL_CHANNELMODEIS)
def myevent(bot, srv=None, me=None, channel=None, mode=None, mode_params=None):
    # do something
```

368 - RPL_ENDOFBANLIST

Format :`{srv} 368 {nick} {channel} :End of channel ban list`

Match ^:`(?P<srv>\S+) 368 (?P<me>\S+) (?P<channel>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_ENDOFBANLIST)
def myevent(bot, srv=None, me=None, channel=None, data=None):
    # do something
```

374 - RPL_ENDOFINFO

Format :`{srv} 374 {nick} :End of /INFO list`

Match ^:`(?P<srv>\S+) 374 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_ENDOFINFO)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

365 - RPL_ENDOFLINKS

Format : `{srv} 365 {nick} {mask} :End of /LINKS list`

Match `^:(?P<srv>\S+) 365 (?P<me>\S+) (?P<mask>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_ENDOFLINKS)
def myevent(bot, srv=None, me=None, mask=None, data=None):
    # do something
```

376 - RPL_ENDOFMOTD

Format : `{srv} 376 {nick} :End of /MOTD command`

Match `^:(?P<srv>\S+) 376 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_ENDOFMOTD)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

366 - RPL_ENDOFNAMES

Format : `{srv} 366 {nick} {channel} :End of /NAMES list`

Match `^:(?P<srv>\S+) 366 (?P<me>\S+) (?P<channel>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_ENDOFNAMES)
def myevent(bot, srv=None, me=None, channel=None, data=None):
    # do something
```

219 - RPL_ENDOFSTATS

Format : `{srv} 219 {nick} {stats_letter} :End of /STATS report`

Match `^:(?P<srv>\S+) 219 (?P<me>\S+) (?P<stats_letter>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_ENDOFSTATS)
def myevent(bot, srv=None, me=None, stats_letter=None, data=None):
    # do something
```

394 - RPL_ENDOFUSERS

Format : `{srv} 394 {nick} :End of users`

Match `^:(?P<srv>\S+) 394 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_ENDOFUSERS)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

315 - RPL_ENDOFWHO

Format : `{srv} 315 {nick} {nick} :End of /WHO list`

Match ^: `(?P<srv>\S+) 315 (?P<me>\S+) (?P<nick>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_ENDOFWHO)
def myevent(bot, srv=None, me=None, nick=None, data=None):
    # do something
```

318 - RPL_ENDOFWHOIS

Format : `{srv} 318 {nick} {nick} :End of /WHOIS list`

Match ^: `(?P<srv>\S+) 318 (?P<me>\S+) (?P<nick>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_ENDOFWHOIS)
def myevent(bot, srv=None, me=None, nick=None, data=None):
    # do something
```

369 - RPL_ENDOFWHOWAS

Format : `{srv} 369 {nick} {nick} :End of WHOWAS`

Match ^: `(?P<srv>\S+) 369 (?P<me>\S+) (?P<nick>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_ENDOFWHOWAS)
def myevent(bot, srv=None, me=None, nick=None, data=None):
    # do something
```

371 - RPL_INFO

Format : `{srv} 371 {nick} :{string}`

Match ^: `(?P<srv>\S+) 371 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_INFO)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

341 - RPL_INVITING

Format `:{srv} 341 {nick} {channel} {nick}`

Match `^:(?P<srv>\S+) 341 (?P<me>\S+) (?P<channel>\S+) (?P<nick>\S+)`

Example:

```
@irc3.event(rfc.RPL_INVITING)
def myevent(bot, srv=None, me=None, channel=None, nick=None):
    # do something
```

303 - RPL_ISON

Format `:{srv} 303 {nick} :{nicknames}`

Match `^:(?P<srv>\S+) 303 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_ISON)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

364 - RPL_LINKS

Format `:{srv} 364 {nick} {mask} {server} :{hopcount} {server_info}`

Match `^:(?P<srv>\S+) 364 (?P<me>\S+) (?P<mask>\S+) (?P<server>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_LINKS)
def myevent(bot, srv=None, me=None, mask=None, server=None, data=None):
    # do something
```

322 - RPL_LIST

Format `:{srv} 322 {nick} {channel} {visible} :{topic}`

Match `^:(?P<srv>\S+) 322 (?P<me>\S+) (?P<channel>\S+) (?P<visible>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_LIST)
def myevent(bot, srv=None, me=None, channel=None, visible=None, data=None):
    # do something
```

323 - RPL_LISTEND

Format `:{srv} 323 {nick} :End of /LIST`

Match `^:(?P<srv>\S+) 323 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_LISTEND)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

321 - RPL_LISTSTART

Format `:{srv} 321 {nick} Channel :Users Name`

Match `^:(?P<srv>\S+) 321 (?P<me>\S+) Channel :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_LISTSTART)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

254 - RPL_LUSERCHANNELS

Format `:{srv} 254 {nick} {x} :channels formed`

Match `^:(?P<srv>\S+) 254 (?P<me>\S+) (?P<x>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_LUSERCHANNELS)
def myevent(bot, srv=None, me=None, x=None, data=None):
    # do something
```

251 - RPL_LUSERCLIENT

Format `:{srv} 251 {nick} :There are {x} users and {y} invisible on {z} servers`

Match `^:(?P<srv>\S+) 251 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_LUSERCLIENT)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

255 - RPL_LUSERME

Format `:{srv} 255 {nick} :I have {x} clients and {y}`

Match `^:(?P<srv>\S+) 255 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_LUSERME)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

252 - RPL_LUSEROP

Format `:{srv} 252 {nick} {x} :operator(s) online`

Match `^:(?P<srv>\S+) 252 (?P<me>\S+) (?P<x>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_LUSEROP)
def myevent(bot, srv=None, me=None, x=None, data=None):
    # do something
```

253 - RPL_LUSERUNKNOWN

Format `:{srv} 253 {nick} {x} :unknown connection(s)`

Match `^:(?P<srv>\S+) 253 (?P<me>\S+) (?P<x>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_LUSERUNKNOWN)
def myevent(bot, srv=None, me=None, x=None, data=None):
    # do something
```

372 - RPL_MOTD

Format `:{srv} 372 {nick} :- {text}`

Match `^:(?P<srv>\S+) 372 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_MOTD)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

375 - RPL_MOTDSTART

Format `:{srv} 375 {nick} :- {server} Message of the day -`

Match `^:(?P<srv>\S+) 375 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_MOTDSTART)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

353 - RPL_NAMREPLY

Format `:{srv} 353 {nick} {m} {channel} :{nicknames}`

Match `^:(?P<srv>\S+) 353 (?P<me>\S+) (?P<m>\S+) (?P<channel>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_NAMREPLY)
def myevent(bot, srv=None, me=None, m=None, channel=None, data=None):
    # do something
```

331 - RPL_NOTOPIC

Format : `{srv} 331 {nick} {channel} :No topic is set`

Match ^: `(?P<srv>\S+) 331 (?P<me>\S+) (?P<channel>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_NOTOPIC)
def myevent(bot, srv=None, me=None, channel=None, data=None):
    # do something
```

395 - RPL_NOUSERS

Format : `{srv} 395 {nick} :Nobody logged in`

Match ^: `(?P<srv>\S+) 395 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_NOUSERS)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

306 - RPL_NOWAWAY

Format : `{srv} 306 {nick} :You have been marked as being away`

Match ^: `(?P<srv>\S+) 306 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_NOWAWAY)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

382 - RPL_REHASHING

Format : `{srv} 382 {nick} {config_file} :Rehashing`

Match ^: `(?P<srv>\S+) 382 (?P<me>\S+) (?P<config_file>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_REHASHING)
def myevent(bot, srv=None, me=None, config_file=None, data=None):
    # do something
```

213 - RPL_STATSCLINE

Format : `{srv} 213 {nick} C {host} * {nick} {port} {class}`

Match `^:(?P<srv>\S+) 213 (?P<me>\S+) C (?P<host>\S+) . (?P<nick>\S+) (?P<port>\S+) (?P<class>\S+)`

Example:

```
@irc3.event(rfc.RPL_STATSCLINE)
def myevent(bot, srv=None, me=None, host=None, nick=None, port=None, class=None):
    # do something
```

212 - RPL_STATSCOMMANDS

Format : `{srv} 212 {nick} {cmd} {count}`

Match `^:(?P<srv>\S+) 212 (?P<me>\S+) (?P<cmd>\S+) (?P<count>\S+)`

Example:

```
@irc3.event(rfc.RPL_STATSCOMMANDS)
def myevent(bot, srv=None, me=None, cmd=None, count=None):
    # do something
```

244 - RPL_STATSHLINE

Format : `{srv} 244 {nick} H {hostmask} * {servername}`

Match `^:(?P<srv>\S+) 244 (?P<me>\S+) H (?P<hostmask>\S+) . (?P<servername>\S+)`

Example:

```
@irc3.event(rfc.RPL_STATSHLINE)
def myevent(bot, srv=None, me=None, hostmask=None, servername=None):
    # do something
```

215 - RPL_STATSILINE

Format : `{srv} 215 {nick} I {host} * {host1} {port} {class}`

Match `^:(?P<srv>\S+) 215 (?P<me>\S+) I (?P<host>\S+) . (?P<host1>\S+) (?P<port>\S+) (?P<class>\S+)`

Example:

```
@irc3.event(rfc.RPL_STATSILINE)
def myevent(bot, srv=None, me=None, host=None, host1=None, port=None, class=None):
    # do something
```

216 - RPL_STATSKLINE

Format : `{srv} 216 {nick} K {host} * {username} {port} {class}`

Match `^:(?P<srv>\S+) 216 (?P<me>\S+) K (?P<host>\S+) . (?P<username>\S+) (?P<port>\S+) (?P<class>\S+)`

Example:

```
@irc3.event(rfc.RPL_STATSKLINE)
def myevent(bot, srv=None, me=None, host=None, username=None, port=None, class=None):
    # do something
```

211 - RPL_STATSLINKINFO

Format `:{srv} 211 {nick} :{linkname} {sendq} {sent_messages} {received_bytes} {time_open}`

Match `^:(?P<srv>\S+) 211 (?P<me>\S+) (?P<linkname>\S+) (?P<sendq>\S+) (?P<sent_messages>\S+) (?P<received_bytes>\S+) (?P<time_open>\S+)`

Example:

```
@irc3.event(rfc.RPL_STATSLINKINFO)
def myevent(bot, srv=None, me=None, linkname=None, sendq=None, sent_messages=None,␣
↪received_bytes=None, time_open=None):
    # do something
```

241 - RPL_STATSLLINE

Format `:{srv} 241 {nick} L {hostmask} * {servername} {maxdepth}`

Match `^:(?P<srv>\S+) 241 (?P<me>\S+) L (?P<hostmask>\S+) . (?P<servername>\S+) (?P<maxdepth>\S+)`

Example:

```
@irc3.event(rfc.RPL_STATSLLINE)
def myevent(bot, srv=None, me=None, hostmask=None, servername=None, maxdepth=None):
    # do something
```

214 - RPL_STATSNLINE

Format `:{srv} 214 {nick} N {host} * {nick} {port} {class}`

Match `^:(?P<srv>\S+) 214 (?P<me>\S+) N (?P<host>\S+) . (?P<nick>\S+) (?P<port>\S+) (?P<class>\S+)`

Example:

```
@irc3.event(rfc.RPL_STATSNLINE)
def myevent(bot, srv=None, me=None, host=None, nick=None, port=None, class=None):
    # do something
```

243 - RPL_STATSOLINE

Format `:{srv} 243 {nick} O {hostmask} * {nick}`

Match `^:(?P<srv>\S+) 243 (?P<me>\S+) O (?P<hostmask>\S+) . (?P<nick>\S+)`

Example:

```
@irc3.event(rfc.RPL_STATSOLINE)
def myevent(bot, srv=None, me=None, hostmask=None, nick=None):
    # do something
```

242 - RPL_STATSUPTIME

Format : `{srv} 242 {nick} :Server Up{days}days {hours}`

Match ^: `(?P<srv>\S+) 242 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_STATSUPTIME)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

218 - RPL_STATSYLINE

Format : `{srv} 218 {nick} frequency> {max_sendq}`

Match ^: `(?P<srv>\S+) 218 (?P<me>\S+) frequency> (?P<max_sendq>\S+)`

Example:

```
@irc3.event(rfc.RPL_STATSYLINE)
def myevent(bot, srv=None, me=None, max_sendq=None):
    # do something
```

342 - RPL_SUMMONING

Format : `{srv} 342 {nick} {nick} :Summoning user to IRC`

Match ^: `(?P<srv>\S+) 342 (?P<me>\S+) (?P<nick>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_SUMMONING)
def myevent(bot, srv=None, me=None, nick=None, data=None):
    # do something
```

391 - RPL_TIME

Format : `{srv} 391 {nick} {server} :{string_showing_server's_local_time}`

Match ^: `(?P<srv>\S+) 391 (?P<me>\S+) (?P<server>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_TIME)
def myevent(bot, srv=None, me=None, server=None, data=None):
    # do something
```

332 - RPL_TOPIC

Format `:{srv} 332 {nick} {channel} :{topic}`

Match `^:(?P<srv>\S+) 332 (?P<me>\S+) (?P<channel>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_TOPIC)
def myevent(bot, srv=None, me=None, channel=None, data=None):
    # do something
```

201 - RPL_TRACECONNECTING

Format `:{srv} 201 {nick} Try. {class} {server}`

Match `^:(?P<srv>\S+) 201 (?P<me>\S+) Try. (?P<class>\S+) (?P<server>\S+)`

Example:

```
@irc3.event(rfc.RPL_TRACECONNECTING)
def myevent(bot, srv=None, me=None, class=None, server=None):
    # do something
```

202 - RPL_TRACEHANDSHAKE

Format `:{srv} 202 {nick} H.S. {class} {server}`

Match `^:(?P<srv>\S+) 202 (?P<me>\S+) H.S. (?P<class>\S+) (?P<server>\S+)`

Example:

```
@irc3.event(rfc.RPL_TRACEHANDSHAKE)
def myevent(bot, srv=None, me=None, class=None, server=None):
    # do something
```

200 - RPL_TRACELINK

Format `:{srv} 200 {nick} {next_server}`

Match `^:(?P<srv>\S+) 200 (?P<me>\S+) (?P<next_server>\S+)`

Example:

```
@irc3.event(rfc.RPL_TRACELINK)
def myevent(bot, srv=None, me=None, next_server=None):
    # do something
```

261 - RPL_TRACELOG

Format `:{srv} 261 {nick} File {logfile} {debug_level}`

Match `^:(?P<srv>\S+) 261 (?P<me>\S+) File (?P<logfile>\S+) (?P<debug_level>\S+)`

Example:

```
@irc3.event(rfc.RPL_TRACELOG)
def myevent(bot, srv=None, me=None, logfile=None, debug_level=None):
    # do something
```

208 - RPL_TRACENEWTYPE

Format : `{srv} 208 {nick} {newtype} 0 {client}`

Match ^: `(?P<srv>\S+) 208 (?P<me>\S+) (?P<newtype>\S+) 0 (?P<client>\S+)`

Example:

```
@irc3.event(rfc.RPL_TRACENEWTYPE)
def myevent(bot, srv=None, me=None, newtype=None, client=None):
    # do something
```

204 - RPL_TRACEOPERATOR

Format : `{srv} 204 {nick} Oper {class} {nick}`

Match ^: `(?P<srv>\S+) 204 (?P<me>\S+) Oper (?P<class>\S+) (?P<nick>\S+)`

Example:

```
@irc3.event(rfc.RPL_TRACEOPERATOR)
def myevent(bot, srv=None, me=None, class=None, nick=None):
    # do something
```

206 - RPL_TRACESERVER

Format : `{srv} 206 {nick} {mask}`

Match ^: `(?P<srv>\S+) 206 (?P<me>\S+) (?P<mask>\S+)`

Example:

```
@irc3.event(rfc.RPL_TRACESERVER)
def myevent(bot, srv=None, me=None, mask=None):
    # do something
```

203 - RPL_TRACEUNKNOWN

Format : `{srv} 203 {nick} ???? {class} [{clientip}]`

Match ^: `(?P<srv>\S+) 203 (?P<me>\S+) \S+ (?P<class>\S+) [(?P<clientip>\S+)]`

Example:

```
@irc3.event(rfc.RPL_TRACEUNKNOWN)
def myevent(bot, srv=None, me=None, class=None, clientip=None):
    # do something
```

205 - RPL_TRACEUSER

Format :`{srv} 205 {nick} User {class} {nick}`

Match ^:`(?P<srv>\S+) 205 (?P<me>\S+) User (?P<class>\S+) (?P<nick>\S+)`

Example:

```
@irc3.event(rfc.RPL_TRACEUSER)
def myevent(bot, srv=None, me=None, class=None, nick=None):
    # do something
```

221 - RPL_UMODEIS

Format :`{srv} 221 {nick} {user_mode_string}`

Match ^:`(?P<srv>\S+) 221 (?P<me>\S+) (?P<user_mode_string>\S+)`

Example:

```
@irc3.event(rfc.RPL_UMODEIS)
def myevent(bot, srv=None, me=None, user_mode_string=None):
    # do something
```

305 - RPL_UNAWAY

Format :`{srv} 305 {nick} :You are no longer marked as being away`

Match ^:`(?P<srv>\S+) 305 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_UNAWAY)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

302 - RPL_USERHOST

Format :`{srv} 302 {nick} :[{reply}{{space}{reply}}]`

Match ^:`(?P<srv>\S+) 302 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_USERHOST)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

393 - RPL_USERS

Format :`{srv} 393 {nick} {x} {y} {z}`

Match ^:`(?P<srv>\S+) 393 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_USERS)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

392 - RPL_USERSSTART

Format `:{srv} 392 {nick} :UserID Terminal Host`

Match `^:(?P<srv>\S+) 392 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_USERSSTART)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

351 - RPL_VERSION

Format `:{srv} 351 {nick} {version}.{debuglevel} {server} :{comments}`

Match `^:(?P<srv>\S+) 351 (?P<me>\S+) (?P<version>\S+).(?P<debuglevel>\S+) (?P<server>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_VERSION)
def myevent(bot, srv=None, me=None, version=None, debuglevel=None, server=None,␣
↪data=None):
    # do something
```

319 - RPL_WHOISCHANNELS

Format `:{srv} 319 {nick} :{channels}`

Match `^:(?P<srv>\S+) 319 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_WHOISCHANNELS)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

317 - RPL_WHOISIDLE

Format `:{srv} 317 {nick} {nick} {x} :seconds idle`

Match `^:(?P<srv>\S+) 317 (?P<me>\S+) (?P<nick>\S+) (?P<x>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_WHOISIDLE)
def myevent(bot, srv=None, me=None, nick=None, x=None, data=None):
    # do something
```

313 - RPL_WHOISOPERATOR

Format `:{srv} 313 {nick} {nick} :is an IRC operator`

Match `^:(?P<srv>\S+) 313 (?P<me>\S+) (?P<nick>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_WHOISOPERATOR)
def myevent(bot, srv=None, me=None, nick=None, data=None):
    # do something
```

312 - RPL_WHOISSERVER

Format `:{srv} 312 {nick} {nick} {server} :{server_info}`

Match `^:(?P<srv>\S+) 312 (?P<me>\S+) (?P<nick>\S+) (?P<server>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_WHOISSERVER)
def myevent(bot, srv=None, me=None, nick=None, server=None, data=None):
    # do something
```

311 - RPL_WHOISUSER

Format `:{srv} 311 {nick} {nick} {username} {host} {m} :{realname}`

Match `^:(?P<srv>\S+) 311 (?P<me>\S+) (?P<nick>\S+) (?P<username>\S+) (?P<host>\S+) (?P<m>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_WHOISUSER)
def myevent(bot, srv=None, me=None, nick=None, username=None, host=None, m=None,␣
↪data=None):
    # do something
```

352 - RPL_WHOREPLY

Format `:{srv} 352 {nick} :{channel} {username} {host} {server} {nick} {modes} :{hopcount} {realname}`

Match `^:(?P<srv>\S+) 352 (?P<me>\S+) (?P<channel>\S+) (?P<username>\S+) (?P<host>\S+) (?P<server>\S+) (?P<nick>\S+) (?P<modes>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_WHOREPLY)
def myevent(bot, srv=None, me=None, channel=None, username=None, host=None,␣
↪server=None, nick=None, modes=None, data=None):
    # do something
```

314 - RPL_WHOWASUSER

Format :`{srv} 314 {nick} {nick} {username} {host} * :{realname}`

Match `^:(?P<srv>\S+) 314 (?P<me>\S+) (?P<nick>\S+) (?P<username>\S+) (?P<host>\S+) . :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_WHOWASUSER)
def myevent(bot, srv=None, me=None, nick=None, username=None, host=None, data=None):
    # do something
```

381 - RPL_YOUREOPER

Format :`{srv} 381 {nick} :You are now an IRC operator`

Match `^:(?P<srv>\S+) 381 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.RPL_YOUREOPER)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

3.3.2 Errors (ERR)

462 - ERR_ALREADYREGISTRED

Format :`{srv} 462 {nick} :You may not reregister`

Match `^:(?P<srv>\S+) 462 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_ALREADYREGISTRED)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

475 - ERR_BADCHANNELKEY

Format :`{srv} 475 {nick} {channel} :Cannot join channel (+k)`

Match `^:(?P<srv>\S+) 475 (?P<me>\S+) (?P<channel>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_BADCHANNELKEY)
def myevent(bot, srv=None, me=None, channel=None, data=None):
    # do something
```

474 - ERR_BANNEDFROMCHAN

Format `:{srv} 474 {nick} {channel} :Cannot join channel (+b)`

Match `^:(?P<srv>\S+) 474 (?P<me>\S+) (?P<channel>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_BANNEDFROMCHAN)
def myevent(bot, srv=None, me=None, channel=None, data=None):
    # do something
```

404 - ERR_CANNOTSENDTOCHAN

Format `:{srv} 404 {nick} {channel} :Cannot send to channel`

Match `^:(?P<srv>\S+) 404 (?P<me>\S+) (?P<channel>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_CANNOTSENDTOCHAN)
def myevent(bot, srv=None, me=None, channel=None, data=None):
    # do something
```

483 - ERR_CANTKILLSERVER

Format `:{srv} 483 {nick} :You cant kill a server!`

Match `^:(?P<srv>\S+) 483 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_CANTKILLSERVER)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

471 - ERR_CHANNELISFULL

Format `:{srv} 471 {nick} {channel} :Cannot join channel (+l)`

Match `^:(?P<srv>\S+) 471 (?P<me>\S+) (?P<channel>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_CHANNELISFULL)
def myevent(bot, srv=None, me=None, channel=None, data=None):
    # do something
```

482 - ERR_CHANOPRIVSNEEDED

Format `:{srv} 482 {nick} {channel} :You're not channel operator`

Match `^:(?P<srv>\S+) 482 (?P<me>\S+) (?P<channel>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_CHANOPRIVSNEEDED)
def myevent(bot, srv=None, me=None, channel=None, data=None):
    # do something
```

432 - ERR_ERRONEUSNICKNAME

Format : `{srv} 432 {nick} {nick} :Erroneus nickname`

Match ^: `(?P<srv>\S+) 432 (?P<me>\S+) (?P<nick>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_ERRONEUSNICKNAME)
def myevent(bot, srv=None, me=None, nick=None, data=None):
    # do something
```

473 - ERR_INVITEONLYCHAN

Format : `{srv} 473 {nick} {channel} :Cannot join channel (+i)`

Match ^: `(?P<srv>\S+) 473 (?P<me>\S+) (?P<channel>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_INVITEONLYCHAN)
def myevent(bot, srv=None, me=None, channel=None, data=None):
    # do something
```

467 - ERR_KEYSET

Format : `{srv} 467 {nick} {channel} :Channel key already set`

Match ^: `(?P<srv>\S+) 467 (?P<me>\S+) (?P<channel>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_KEYSET)
def myevent(bot, srv=None, me=None, channel=None, data=None):
    # do something
```

461 - ERR_NEEDMOREPARAMS

Format : `{srv} 461 {nick} {cmd} :Not enough parameters`

Match ^: `(?P<srv>\S+) 461 (?P<me>\S+) (?P<cmd>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_NEEDMOREPARAMS)
def myevent(bot, srv=None, me=None, cmd=None, data=None):
    # do something
```

ERR_NICK

Match `^(@(?P<tags>\S+) )?:(?P<srv>\S+) (?P<retcode>(432|433|436)) (?P<me>\S+) (?P<nick>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_NICK)
def myevent(bot, srv=None, retcode=None, me=None, nick=None, data=None, tags=None):
    # do something
```

436 - ERR_NICKCOLLISION

Format `:{srv} 436 {nick} {nick} :Nickname collision KILL`

Match `^:(?P<srv>\S+) 436 (?P<me>\S+) (?P<nick>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_NICKCOLLISION)
def myevent(bot, srv=None, me=None, nick=None, data=None):
    # do something
```

433 - ERR_NICKNAMEINUSE

Format `:{srv} 433 {nick} {nick} :Nickname is already in use`

Match `^:(?P<srv>\S+) 433 (?P<me>\S+) (?P<nick>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_NICKNAMEINUSE)
def myevent(bot, srv=None, me=None, nick=None, data=None):
    # do something
```

423 - ERR_NOADMININFO

Format `:{srv} 423 {nick} {server} :No administrative info available`

Match `^:(?P<srv>\S+) 423 (?P<me>\S+) (?P<server>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_NOADMININFO)
def myevent(bot, srv=None, me=None, server=None, data=None):
    # do something
```

444 - ERR_NOLOGIN

Format `:{srv} 444 {nick} {nick} :User not logged in`

Match `^:(?P<srv>\S+) 444 (?P<me>\S+) (?P<nick>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_NOLOGIN)
def myevent(bot, srv=None, me=None, nick=None, data=None):
    # do something
```

422 - ERR_NOMOTD

Format : `{srv} 422 {nick} :MOTD File is missing`

Match ^: `(?P<srv>\S+) 422 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_NOMOTD)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

431 - ERR_NONICKNAMEGIVEN

Format : `{srv} 431 {nick} :No nickname given`

Match ^: `(?P<srv>\S+) 431 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_NONICKNAMEGIVEN)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

491 - ERR_NOOPERHOST

Format : `{srv} 491 {nick} :No O-lines for your host`

Match ^: `(?P<srv>\S+) 491 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_NOOPERHOST)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

409 - ERR_NOORIGIN

Format : `{srv} 409 {nick} :No origin specified`

Match ^: `(?P<srv>\S+) 409 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_NOORIGIN)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

463 - ERR_NOPERMFORHOST

Format :`{srv} 463 {nick} :Your host isn't among the privileged`

Match ^:`(?P<srv>\S+) 463 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_NOPERMFORHOST)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

481 - ERR_NOPRIVILEGES

Format :`{srv} 481 {nick} :Permission Denied- You're not an IRC operator`

Match ^:`(?P<srv>\S+) 481 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_NOPRIVILEGES)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

411 - ERR_NORECIPIENT

Format :`{srv} 411 {nick} :No recipient given ({cmd})`

Match ^:`(?P<srv>\S+) 411 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_NORECIPIENT)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

403 - ERR_NOSUCHCHANNEL

Format :`{srv} 403 {nick} {channel} :No such channel`

Match ^:`(?P<srv>\S+) 403 (?P<me>\S+) (?P<channel>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_NOSUCHCHANNEL)
def myevent(bot, srv=None, me=None, channel=None, data=None):
    # do something
```

401 - ERR_NOSUCHNICK

Format :`{srv} 401 {nick} {nick} :No such nick/channel`

Match ^:`(?P<srv>\S+) 401 (?P<me>\S+) (?P<nick>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_NOSUCHNICK)
def myevent(bot, srv=None, me=None, nick=None, data=None):
    # do something
```

402 - ERR_NOSUCHSERVER

Format : `{srv} 402 {nick} {server} :No such server`

Match ^: `(?P<srv>\S+) 402 (?P<me>\S+) (?P<server>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_NOSUCHSERVER)
def myevent(bot, srv=None, me=None, server=None, data=None):
    # do something
```

412 - ERR_NOTEXTTOSEND

Format : `{srv} 412 {nick} :No text to send`

Match ^: `(?P<srv>\S+) 412 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_NOTEXTTOSEND)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

442 - ERR_NOTONCHANNEL

Format : `{srv} 442 {nick} {channel} :You're not on that channel`

Match ^: `(?P<srv>\S+) 442 (?P<me>\S+) (?P<channel>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_NOTONCHANNEL)
def myevent(bot, srv=None, me=None, channel=None, data=None):
    # do something
```

413 - ERR_NOTOPLEVEL

Format : `{srv} 413 {nick} {mask} :No toplevel domain specified`

Match ^: `(?P<srv>\S+) 413 (?P<me>\S+) (?P<mask>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_NOTOPLEVEL)
def myevent(bot, srv=None, me=None, mask=None, data=None):
    # do something
```

451 - ERR_NOTREGISTERED

Format :`{srv} 451 {nick} :You have not registered`

Match ^:`(?P<srv>\S+) 451 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_NOTREGISTERED)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

464 - ERR_PASSWDMISMATCH

Format :`{srv} 464 {nick} :Password incorrect`

Match ^:`(?P<srv>\S+) 464 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_PASSWDMISMATCH)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

445 - ERR_SUMMONDISABLED

Format :`{srv} 445 {nick} :SUMMON has been disabled`

Match ^:`(?P<srv>\S+) 445 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_SUMMONDISABLED)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

405 - ERR_TOOMANYCHANNELS

Format :`{srv} 405 {nick} {channel} :You have joined too many channels`

Match ^:`(?P<srv>\S+) 405 (?P<me>\S+) (?P<channel>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_TOOMANYCHANNELS)
def myevent(bot, srv=None, me=None, channel=None, data=None):
    # do something
```

407 - ERR_TOOMANYTARGETS

Format :`{srv} 407 {nick} {target} :Duplicate recipients. No message delivered`

Match ^:`(?P<srv>\S+) 407 (?P<me>\S+) (?P<target>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_TOOMANYTARGETS)
def myevent(bot, srv=None, me=None, target=None, data=None):
    # do something
```

501 - ERR_UMODEUNKNOWNFLAG

Format : `{srv} 501 {nick} :Unknown MODE flag`

Match ^: `(?P<srv>\S+) 501 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_UMODEUNKNOWNFLAG)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

421 - ERR_UNKNOWNCOMMAND

Format : `{srv} 421 {nick} {cmd} :Unknown command`

Match ^: `(?P<srv>\S+) 421 (?P<me>\S+) (?P<cmd>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_UNKNOWNCOMMAND)
def myevent(bot, srv=None, me=None, cmd=None, data=None):
    # do something
```

472 - ERR_UNKNOWNMODE

Format : `{srv} 472 {nick} {char} :is unknown mode char to me`

Match ^: `(?P<srv>\S+) 472 (?P<me>\S+) (?P<char>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_UNKNOWNMODE)
def myevent(bot, srv=None, me=None, char=None, data=None):
    # do something
```

441 - ERR_USERNOTINCHANNEL

Format : `{srv} 441 {nick} {nick} {channel} :They aren't on that channel`

Match ^: `(?P<srv>\S+) 441 (?P<me>\S+) (?P<nick>\S+) (?P<channel>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_USERNOTINCHANNEL)
def myevent(bot, srv=None, me=None, nick=None, channel=None, data=None):
    # do something
```

443 - ERR_USERONCHANNEL

Format `:{srv} 443 {nick} {nick} {channel} :is already on channel`

Match `^:(?P<srv>\S+) 443 (?P<me>\S+) (?P<nick>\S+) (?P<channel>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_USERONCHANNEL)
def myevent(bot, srv=None, me=None, nick=None, channel=None, data=None):
    # do something
```

446 - ERR_USERSDISABLED

Format `:{srv} 446 {nick} :USERS has been disabled`

Match `^:(?P<srv>\S+) 446 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_USERSDISABLED)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

502 - ERR_USERSDONTMATCH

Format `:{srv} 502 {nick} :Cant change mode for other users`

Match `^:(?P<srv>\S+) 502 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_USERSDONTMATCH)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

406 - ERR_WASNOSUCHNICK

Format `:{srv} 406 {nick} {nick} :There was no such nickname`

Match `^:(?P<srv>\S+) 406 (?P<me>\S+) (?P<nick>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_WASNOSUCHNICK)
def myevent(bot, srv=None, me=None, nick=None, data=None):
    # do something
```

414 - ERR_WILDTOPLEVEL

Format `:{srv} 414 {nick} {mask} :Wildcard in toplevel domain`

Match `^:(?P<srv>\S+) 414 (?P<me>\S+) (?P<mask>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_WILDTOPLEVEL)
def myevent(bot, srv=None, me=None, mask=None, data=None):
    # do something
```

465 - ERR_YOUREBANNEDCREEP

Format `:{srv} 465 {nick} :You are banned from this server`

Match `^:(?P<srv>\S+) 465 (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.ERR_YOUREBANNEDCREEP)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

3.3.3 Misc

CONNECTED

Match `^:(?P<srv>\S+) (376|422) (?P<me>\S+) :(?P<data>.*)`

Example:

```
@irc3.event(rfc.CONNECTED)
def myevent(bot, srv=None, me=None, data=None):
    # do something
```

CTCP

Match `^(@(?P<tags>\S+) )?:(?P<mask>\S+!\S+@\S+) (?P<event>(PRIVMSG|NOTICE)) {nick} :(?P<ctcp>.*)$`

Example:

```
@irc3.event(rfc.CTCP)
def myevent(bot, mask=None, event=None, ctcp=None, tags=None):
    # do something
```

Out Match `^(?P<event>(PRIVMSG|NOTICE)) (?P<target>\S+) :(?P<ctcp>.*)$`

Example:

```
@irc3.event(rfc.CTCP, iotype="out")
def myevent(bot, event=None, target=None, ctcp=None):
    # do something
```

INVITE

Match `^(@(?P<tags>\S+) )?:(?P<mask>\S+!\S+@\S+) INVITE {nick} :?(?P<channel>\S+)$`

Example:

```
@irc3.event(rfc.INVITE)
def myevent(bot, mask=None, channel=None, tags=None):
    # do something
```

JOIN

Match `^(@(?P<tags>\S+) )?:(?P<mask>\S+) JOIN :?(?P<channel>\S+)`

Example:

```
@irc3.event(rfc.JOIN)
def myevent(bot, mask=None, channel=None, tags=None):
    # do something
```

Out Match `^JOIN :?(?P<channel>\S+)`

Example:

```
@irc3.event(rfc.JOIN, iotype="out")
def myevent(bot, channel=None):
    # do something
```

JOIN_PART_QUIT

Match `^(@(?P<tags>\S+) )?:(?P<mask>\S+) (?P<event>JOIN|PART|QUIT)\s*:*(?P<channel>\S*)(\s+:(?P<data>.*)|$)`

Example:

```
@irc3.event(rfc.JOIN_PART_QUIT)
def myevent(bot, mask=None, event=None, channel=None, data=None, tags=None):
    # do something
```

Out Match `^(?P<event>JOIN|PART|QUIT)\s*:*(?P<channel>\S*)(\s+:(?P<data>.*)|$)`

Example:

```
@irc3.event(rfc.JOIN_PART_QUIT, iotype="out")
def myevent(bot, event=None, channel=None, data=None):
    # do something
```

KICK

Match `^(@(?P<tags>\S+) )?:(?P<mask>\S+) (?P<event>KICK)\s+(?P<channel>\S+)\s*(?P<target>\S+)(\s+:(?P<data>.*)|$)`

Example:

```
@irc3.event(rfc.KICK)
def myevent(bot, mask=None, event=None, channel=None, target=None, data=None,␣
↪tags=None):
    # do something
```

Out Match `^(?P<event>KICK)\s+(?P<channel>\S+)\s*(?P<target>\S+)(\s+:(?P<data>.*)|$)`

Example:

```
@irc3.event(rfc.KICK, iotype="out")
def myevent(bot, event=None, channel=None, target=None, data=None):
    # do something
```

MODE

Match `^(@(?P<tags>\S+) )?:(?P<mask>\S+) (?P<event>MODE)\s+(?P<target>\S+)\s+(?P<modes>\S+)(\s+(?P<data>.*)|$)`

Example:

```
@irc3.event(rfc.MODE)
def myevent(bot, mask=None, event=None, target=None, modes=None, data=None,␣
↪tags=None):
    # do something
```

Out Match `^(?P<event>MODE)\s+(?P<target>\S+)\s+(?P<modes>\S+)(\s+(?P<data>.*)|$)`

Example:

```
@irc3.event(rfc.MODE, iotype="out")
def myevent(bot, event=None, target=None, modes=None, data=None):
    # do something
```

MY_PRIVMSG

Match `^(@(?P<tags>\S+) )?:(?P<mask>\S+!\S+@\S+) (?P<event>(PRIVMSG|NOTICE)) (?P<target>(#\S+|{nick})) :{nick}[:,\s]\s*(?P<data>\S+.*)$`

Example:

```
@irc3.event(rfc.MY_PRIVMSG)
def myevent(bot, mask=None, event=None, target=None, data=None, tags=None):
    # do something
```

Out Match `^(?P<event>(PRIVMSG|NOTICE)) (?P<target>\S+) :(?P<data>.*)$`

Example:

```
@irc3.event(rfc.MY_PRIVMSG, iotype="out")
def myevent(bot, event=None, target=None, data=None):
    # do something
```

NEW_NICK

Match `^(@(?P<tags>\S+) )?:(?P<nick>\S+) NICK :?(?P<new_nick>\S+)`

Example:

```
@irc3.event(rfc.NEW_NICK)
def myevent(bot, nick=None, new_nick=None, tags=None):
    # do something
```

Out Match `^NICK :?(?P<new_nick>\S+)`

Example:

```
@irc3.event(rfc.NEW_NICK, iotype="out")
def myevent(bot, new_nick=None):
    # do something
```

PART

Match `^(@(?P<tags>\S+) )?:(?P<mask>\S+) PART (?P<channel>\S+)(\s+:(?P<data>.*)|$)`

Example:

```
@irc3.event(rfc.PART)
def myevent(bot, mask=None, channel=None, data=None, tags=None):
    # do something
```

Out Match `PART (?P<channel>\S+)(\s+:(?P<data>.*)|$)`

Example:

```
@irc3.event(rfc.PART, iotype="out")
def myevent(bot, channel=None, data=None):
    # do something
```

PING

Match `^PING :?(?P<data>.*)`

Example:

```
@irc3.event(rfc.PING)
def myevent(bot, data=None):
    # do something
```

PONG

Match `^(@(?P<tags>\S+) )?:(?P<server>\S+) PONG (?P=server) :?(?P<data>.*)`

Example:

```
@irc3.event(rfc.PONG)
def myevent(bot, server=None, data=None, tags=None):
    # do something
```

PRIVMSG

Match `^(@(?P<tags>\S+) )?:(?P<mask>\S+!\S+@\S+) (?P<event>(PRIVMSG|NOTICE)) (?P<target>\S+) :(?P<data>.*)$`

Example:

```
@irc3.event(rfc.PRIVMSG)
def myevent(bot, mask=None, event=None, target=None, data=None, tags=None):
    # do something
```

Out Match `^(?P<event>(PRIVMSG|NOTICE)) (?P<target>\S+) :(?P<data>.*)$`

Example:

```
@irc3.event(rfc.PRIVMSG, iotype="out")
def myevent(bot, event=None, target=None, data=None):
    # do something
```

QUIT

Match `^(@(?P<tags>\S+) )?:(?P<mask>\S+) QUIT(\s+:(?P<data>.*)|$)`

Example:

```
@irc3.event(rfc.QUIT)
def myevent(bot, mask=None, data=None, tags=None):
    # do something
```

Out Match `^QUIT(\s+:(?P<data>.*)|$)`

Example:

```
@irc3.event(rfc.QUIT, iotype="out")
def myevent(bot, data=None):
    # do something
```

TOPIC

Match `^(@(?P<tags>\S+) )?:(?P<mask>\S+!\S+@\S+) TOPIC (?P<channel>\S+) :(?P<data>.*)$`

Example:

```
@irc3.event(rfc.TOPIC)
def myevent(bot, mask=None, channel=None, data=None, tags=None):
    # do something
```

Out Match `^TOPIC (?P<channel>\S+) :(?P<data>.*)$`

Example:

```
@irc3.event(rfc.TOPIC, iotype="out")
def myevent(bot, channel=None, data=None):
    # do something
```

3.4 irc3.dcc DCC

See `dcc_chat()`, `dcc_get()` and `dcc_send()`

Here is a simple plugin to send a generated file:

```
# -*- coding: utf-8 -*-
import os
import string
import tempfile
import irc3
from irc3.plugins.command import command


@irc3.plugin
class DCC(object):

    filename = os.path.join(tempfile.gettempdir(), 'to_send')

    def __init__(self, bot):
        self.bot = bot
        if not os.path.isfile(self.filename):
            # create a file to send
            with open(self.filename, 'wb') as fd:
                for i in range(64 * 2048):
                    fd.write(string.ascii_letters.encode('utf8'))

    @command
    async def send(self, mask, target, args):
        """ DCC SEND command

            %%send
        """
        conn = await self.bot.dcc_send(mask, self.filename)
        self.bot.log.debug('%s ready', conn)
```

3.4.1 API

class irc3.dcc.**DCCManager**(*bot*)

Manage DCC connections

create(*name_or_class*, *mask*, *filepath=None*, ***kwargs*)

Create a new DCC connection. Return an `asyncio.Protocol`

is_allowed(*name_or_class*, *mask*)

Return True is a new connection is allowed

resume(*mask*, *filename*, *port*, *pos*)

Resume a DCC send

class irc3.dcc.**DCCChat**(***kwargs*)

DCC CHAT implementation

connection_made(*transport*)

Called when a connection is made.

The argument is the transport representing the pipe connection. To receive data, wait for data_received() calls. When the connection is closed, connection_lost() is called.

data_received(*data*)
data received

decode(*data*)
Decode data with bot's encoding

class irc3.dcc.**DCCSend**(***kwargs*)
DCC SEND implementation

connection_made(*transport*)
Called when a connection is made.

The argument is the transport representing the pipe connection. To receive data, wait for data_received() calls. When the connection is closed, connection_lost() is called.

data_received(*data*)
Called when some data is received.

The argument is a bytes object.

class irc3.dcc.**DCCGet**(***kwargs*)
DCC GET implementation

connection_made(*transport*)
Called when a connection is made.

The argument is the transport representing the pipe connection. To receive data, wait for data_received() calls. When the connection is closed, connection_lost() is called.

data_received(*data*)
Called when some data is received.

The argument is a bytes object.

3.5 Reloadable plugins

Note: if you just want the bot to restart when you change a file during development you can use hupper:

```
$ pip install hupper
$ hupper -m irc3 config.ini
```

irc3 provides a way to reload plugins without restarting the bot.

To do that, your plugin should provide a `reload` class method:

```
class Plugin(object):

    def __init__(self, bot):
        self.bot = bot

    @classmethod
    def reload(cls, old):
        """this method should return a ready to use plugin instance.
        cls is the newly reloaded class. old is the old instance.
        """
        return cls(old.bot)
```

Plugins can also implement a few hooks to help take care of reloads:

```
class Plugin(object):

    def __init__(self, bot):
        self.bot = bot

    def before_reload(self):
        """Do stuff before reload"""

    def after_reload(self):
        """Do stuff after reload"""
```

To reload a plugin, just call the `reload()` method with module name(s) to reload:

```
>>> bot.reload('mycommands')
```

3.6 `irc3.plugins.asynchronious` Asynchronious events

This module provide a way to catch data from various predefined events.

3.6.1 Usage

You'll have to define a subclass of *AsyncEvents*:

```
class Whois(AsyncEvents):

    # the command will fail if we do not have a result after 30s
    timeout = 20

    # send this line before listening to events
    send_line = 'WHOIS {nick} {nick}'

    # when those events occurs, we can add them to the result list
    events = (
        # (?i) is for IGNORECASE. This will match either NicK or nick
        {'match': "(?i)^:\S+ 301 \S+ {nick} :(?P<away>.*)"},
        {'match': "(?i)^:\S+ 311 \S+ {nick} (?P<username>\S+) (?P<host>\S+) . "
                  ":(?P<realname>.*)(?i)"},
        {'match': "(?i)^:\S+ 312 \S+ {nick} (?P<server>\S+) "
                  ":(?P<server_desc>.*)"},
        {'match': "(?i)^:\S+ 317 \S+ {nick} (?P<idle>[0-9]+).*"},
        {'match': "(?i)^:\S+ 319 \S+ {nick} :(?P<channels>.*)", 'multi': True},
        {'match': "(?i)^:\S+ 330 \S+ {nick} (?P<account>\S+) "
                  ":(?P<account_desc>.*)"},
        {'match': "(?i)^:\S+ 671 \S+ {nick} :(?P<connection>.*)"},
        # if final=True then a result is returned when the event occurs
        {'match': "(?i)^:\S+ (?P<retcode>(318|401)) \S+ (?P<nick>{nick}) :.*",
         'final': True},
    )

    def process_results(self, results=None, **value):
        """take results list of all events and put them in a dict"""
        channels = []
        for res in results:
```

(continues on next page)

(continued from previous page)

```
            channels.extend(res.pop('channels', '').split())
            value.update(res)
        value['channels'] = channels
        value['success'] = value.get('retcode') == '318'
        return value
```

Notice that regexps and send_line contains some *{nick}*. This will be substitued later with the keyword arguments passed to the instance.

Then you're able to use it in a plugin:

```
class MyPlugin:

    def __init__(self, bot):
        self.bot = bot
        self.whois = Whois(bot)

    def do_whois(self):
        # remember {nick} in the regexp? Here it is
        whois = await self.whois(nick='gawel')
        if int(whois['idle']) / 60 > 10:
            self.bot.privmsg('gawel', 'Wake up dude')
```

Warning: Your code should always check if the result has been set before timeout by using *result['timeout']* which is True when the bot failed to get a result before 30s (you can override the default value per call)

Warning: Do not over use this feature. If you're making a lot of calls at the same time you should experience some weird behavior since irc do not allow to identify responses for a command. That's why the exemple use {nick} in the regexp to filter events efficiently. But two concurent call for the same nick can still fail.

3.6.2 API

class irc3.asynchronous.**AsyncEvents**(*context*)

Asynchronious events

__call__(***kwargs*)

Register events; and callbacks then return a *asyncio.Future*. Events regexp are compiled with *params*

process_results(*results=None*, ***value*)

Process results. results is a list of dict catched during event. value is a dict containing some metadata (like timeout=(True/False).

class irc3.plugins.asynchronious.**Async**(*context*)

Asynchronious plugin. Extend the bot with some common commands using *AsyncEvents*

async_who_channel_flags(*channel*, *flags*, *timeout*)

Creates and calls a class from WhoChannelFlags with needed match rule for WHO command on channels with flags.

channel_bans(*channel*, *timeout=20*)

Send a MODE +b and return a Future which will contain recieved data: .. code-block:: py

result = await bot.async_cmds.channel_bans('#irc3')

ctcp_async (*nick*, *ctcp*, *timeout=20*)
Send a CTCP and return a Future which will contain recieved data:

```
result = await bot.async_cmds.ctcp('irc3', 'version')
```

ison (**nicknames*, ***kwargs*)
Send a ISON and return a Future which will contain recieved data:

```
result = await bot.async_cmds.ison('gawel', 'irc3')
```

names (*channel*, *timeout=20*)
Send a NAMES and return a Future which will contain recieved data:

```
result = await bot.async_cmds.names('#irc3')
```

who (*target*, *flags=None*, *timeout=20*)
Send a WHO and return a Future which will contain recieved data:

```
result = await bot.async_cmds.who('gawel')
result = await bot.async_cmds.who('#irc3')
result = await bot.async_cmds.who('#irc3', 'an')
# or
result = await bot.async_cmds.who('#irc3', ['a', 'n'])
```

whois (*nick*, *timeout=20*)
Send a WHOIS and return a Future which will contain recieved data:

```
result = await bot.async_cmds.whois('gawel')
```

3.7 irc3.plugins.autocommand Autocommand plugin

This plugin allows to send IRC commands to the server after connecting. This could be usable for authorization, cloaking, requesting invite to invite only channel and other use cases. It also allows to set delays between IRC commands via the `/sleep` command.

Usage:

This example will authorize on Freenode:

```
>>> config = ini2config("""
... [bot]
... includes =
...     irc3.plugins.autocommand
...
... autocommands =
...     PRIVMSG NickServ :IDENTIFY nick password
... """)
>>> bot = IrcBot(**config)
```

Here's another, more complicated example:

```
>>> config = ini2config("""
... [bot]
... includes =
...     irc3.plugins.autocommand
```

(continues on next page)

(continued from previous page)

```
...
... autocommands =
...     AUTH user password
...     MODE {nick} +x
...     /sleep 2
...     PRIVMSG Q :INVITE #inviteonly
... """)
>>> bot = IrcBot(**config)
```

It will authorize on QuakeNet, cloak and request an invite to `#inviteonly` after a 2 second delay.

3.8 `irc3.plugins.autojoins` Auto join plugin

Auto join channels. The bot will retry to join when kicked and will retry to join each 30s when an error occurs.

Usage:

```
>>> bot = IrcBot(autojoins=['#chan1', '#chan2'])
>>> bot.include('irc3.plugins.autojoins')
```

3.9 `irc3.plugins.casefold` casefolding plugin

This command introduces a *bot.casefold* function that casefolds strings based on the current casemapping of the IRC server.

This lets you casefold nicks and channel names so that on a server using rfc1459 casemapping, *#cool[chan]* and *#Cool{Chan}* are seen as the same channel.

Usage:

```
>>> config = ini2config("""
... [bot]
... includes =
...     irc3.plugins.casefold
... """)
>>> bot = IrcBot(**config)
```

Returning casefolded strings:

```
>>> bot.casefold('ThisIsANick')
'thisisanick'
>>> bot.casefold('#WeirdChannelNames[]')
'#weirdchannelnames{}'
```

3.10 `irc3.plugins.command` Command plugin

Introduce a `@command` decorator

The decorator use docopts to parse command arguments.

3.10.1 Usage

Create a python module with some commands:

```
# -*- coding: utf-8 -*-
from irc3.plugins.command import command


@command
def echo(bot, mask, target, args):
    """Echo command

        %%echo <words>...
    """
    yield ' '.join(args['<words>'])


@command(permission='admin', public=False)
def adduser(bot, mask, target, args):
    """Add a user

        %%adduser <name> <password>
    """
    bot.privmsg(mask.nick, 'User added')


@command(show_in_help_list=False)
def my_secret_operation(bot, mask, target, args):
    """Do something you don't want in !help all the time

        %%my_secret_operation
    """
    yield "I like turtles"
```

And register it:

```
>>> bot = IrcBot()
>>> bot.include('irc3.plugins.command')  # register the plugin
>>> bot.include('mycommands')            # register your commands
```

Check the result:

```
>>> bot.test(':gawel!user@host PRIVMSG #chan :!echo foo')
PRIVMSG #chan :foo
```

In the docstring, `%%` is replaced by the command character. `!` by default. You can override it by passing a `cmd` parameter to bot's config.

When a command is not public, you can't use it on a channel:

```
>>> bot.test(':gawel!user@host PRIVMSG #chan :!adduser foo pass')
PRIVMSG gawel :You can only use the 'adduser' command in private.
```

If a command is tagged with `show_in_help_list=False`, it won't be shown on the result of `!help`.

```
>>> bot.test(':gawel!user@host PRIVMSG #chan :!help')
PRIVMSG #chan :Available commands: !adduser, !echo, !help
```

View extra info about a command by specifying it to `!help`.

```
>>> bot.test(':gawel!user@host PRIVMSG #chan :!help echo')
PRIVMSG #chan :Echo command
PRIVMSG #chan :!echo <words>...
>>> bot.test(':gawel!user@host PRIVMSG #chan :!help nonexistant')
PRIVMSG #chan :No such command. Try !help for an overview of all commands.
```

3.10.2 Guard

You can use a guard to prevent untrusted users to run some commands. The *free_policy* is used by default.

There is two builtin policy:

class irc3.plugins.command.**free_policy**(*bot*)
Default policy

class irc3.plugins.command.**mask_based_policy**(*bot*)
Allow only valid masks. Able to take care or permissions

Mask based guard using permissions:

```
>>> config = ini2config("""
... [bot]
... nick = nono
... includes =
...     irc3.plugins.command
...     mycommands
... [irc3.plugins.command]
... guard = irc3.plugins.command.mask_based_policy
... [irc3.plugins.command.masks]
... gawel!*@* = all_permissions
... foo!*@* = help
... """)
>>> bot = IrcBot(**config)
```

foo is allowed to use command without permissions:

```
>>> bot.test(':foo!u@h PRIVMSG nono :!echo got the power')
PRIVMSG foo :got the power
```

foo is not allowed to use command except those with the help permission:

```
>>> bot.test(':foo!u@h PRIVMSG nono :!ping')
PRIVMSG foo :You are not allowed to use the 'ping' command
```

gawel is allowed:

```
>>> bot.test(':gawel!u@h PRIVMSG nono :!ping')
NOTICE gawel :PONG gawel!
```

3.10.3 Async commands

Commands can be coroutines:

```
# -*- coding: utf-8 -*-
from irc3.plugins.command import command
from irc3.compat import Queue
import irc3


@irc3.plugin
class AsyncCommands(object):
    """Async commands example. This is what it's look like on irc::

        <gawel> !get
        <gawel> !put item
        <irc3> items added to queue
        <irc3> item
    """

    def __init__(self, bot):
        self.bot = bot
        self.queue = Queue()

    @command
    def put(self, mask, target, args):
        """Put items in queue

            %%put <words>...
        """
        for w in args['<words>']:
            self.queue.put_nowait(w)
        yield 'items added to queue'

    @command
    async def get(self, mask, target, args):
        """Async get items from the queue

            %%get
        """
        messages = []
        message = await self.queue.get()
        messages.append(message)
        while not self.queue.empty():
            message = await self.queue.get()
            messages.append(message)
        return messages
```

3.10.4 Available options

The plugin accept the folowing options:

```
[irc3.plugins.command]
cmd = !
use_shlex = true
antiflood = true
casesensitive = true
guard = irc3.plugins.command.mask_based_policy
```

3.10.5 Command arguments

The `command()` decorator accept the folowing arguments:

name: if set, use this name as the command name instead of the function name.

permission: if set, this permission will be required to run the command. See Guard section

use_shlex: if *False*, do not use *shlex* to parse command line.

options_first: if *True* use docopt's options_first options. Allow to have args that starts with - as arguments.

error_format: allow to customize error messages. must be a callable that accept keyword arguments *cmd*, *args* and *exc*. For example, *error_format="Error for {cmd}".format* will work.

quiet: if *True* don't show errors

3.11 `irc3.plugins.core` Core plugin

Core events

class `irc3.plugins.core.`**Core**(*bot*)

badnick(*me=None*, *nick=None*, ***kw*)
Use alt nick on nick error

connected(***kwargs*)
triger the server_ready event

ping(*data*)
PING reply

pong(*event='PONG'*, *data=''*, ***kw*)
P0NG/PING

recompile(*nick=None*, *new_nick=None*, ***kw*)
recompile regexp on new nick

set_config(*data=None*, ***kwargs*)
Store server config

Usage:

```
>>> bot = IrcBot()
>>> bot.include('irc3.plugins.core')
```

3.12 `irc3.plugins.cron` Cron plugin

Introduce a `@cron` decorator

Install aiocron:

```
$ pip install aiocron
```

Create a python module with some crons:

```
# -*- coding: utf-8 -*-
from irc3.plugins.cron import cron


@cron('30 8 * * *')
def wakeup(bot):
    bot.privmsg('#irc3', "It's time to wake up!")


@cron('0 */2 * * *')
def take_a_break(bot):
    bot.privmsg('#irc3', "It's time to take a break!")
```

And register it:

```
>>> context = IrcBot()
>>> context.include('mycrons')     # register your crons
```

3.13 irc3.plugins.ctcp CTCP replies

Usage:

```
>>> config = ini2config("""
... [bot]
... includes =
...     irc3.plugins.ctcp
... [ctcp]
... foo = bar
... """)
>>> bot = IrcBot(**config)
```

Try to send a `CTCP FOO`:

```
>>> bot.test(':gawel!user@host PRIVMSG irc3 :{FOO{', show=False)
>>> # remove escape char for testing..
>>> print(bot.sent[0].replace('{', '01'))
NOTICE gawel :01FOO bar01
```

3.14 irc3.plugins.dcc DCC Chat plugin

This module provide a command to start a DCC CHAT with the bot and extend it with CHAT commands.

3.14.1 CHAT Commands

Adding new commands:

```
>>> @dcc_command
... def echo(bot, mask, client, args):
...     '''echo command
...         %%echo <words>...
...     '''
...     yield ' '.join(args['words'])
```

bot is the bot instance. mask is the irc mask of the user connected via dcc. client is an instance of *DCCChat*

3.14.2 API

irc3.plugins.dcc.**dcc_command**(**func*, ***predicates*)
DCC CHAT command decorator

class irc3.plugins.dcc.**Commands**(*context*)
DCC CHAT commands plugin

chat(*mask*, **args*)
DCC CHAT

%%chat

help(**args*)
Show help

%%help [<cmd>]

3.15 irc3.plugins.feeds Feeds plugin

Send a notification on channel on new feed entry.

Your config must looks like this:

```
[bot]
includes =
    irc3.plugins.feeds

[irc3.plugins.feeds]
channels = #irc3                              # global channel to notify
delay = 5                                     # delay to check feeds
directory = ~/.irc3/feeds                     # directory to store feeds
hook = irc3.plugins.feeds.default_hook        # dotted name to a callable
fmt = [{name}] {entry.title} - {entry.link}  # formatter

# some feeds: name = url
github/irc3 = https://github.com/gawel/irc3/commits/master.atom#irc3
# custom formatter for the feed
github/irc3.fmt = [{feed.name}] New commit: {entry.title} - {entry.link}
# custom channels
github/irc3.channels = #irc3dev #irc3
# custom delay
github/irc3.delay = 10
```

Hook is a dotted name refering to a callable (function or class) wich take a list of entries as argument. It should yield the entries you want really show:

```
>>> def hook(entries):
...     for entry in entries:
...         if 'something bad' not in entry.title:
...             yield entry

>>> class Hook:
...     def __init__(self, bot):
```

(continues on next page)

(continued from previous page)

```
...         self.bot = bot
...     def __call__(self, entries):
...         for entry in entries:
...             if 'something bad' not in entry.title:
...                 yield entry
```

Here is a more complete hook used on freenode#irc3:

```
class FeedsHook(object):
    """Custom hook for irc3.plugins.feeds"""

    def __init__(self, bot):
        self.bot = bot
        self.packages = [
            'asyncio', 'irc3', 'panoramisk',
            'requests', 'trollius', 'webtest',
            'pyramid',
        ]

    def filter_travis(self, entry):
        """Only show the latest entry iif this entry is in a new state"""
        fstate = entry.filename + '.state'
        if os.path.isfile(fstate):
            with open(fstate) as fd:
                state = fd.read().strip()
        else:
            state = None
        if 'failed' in entry.summary:
            nstate = 'failed'
        else:
            nstate = 'success'
        with open(fstate, 'w') as fd:
            fd.write(nstate)
        if state != nstate:
            build = entry.title.split('#')[1]
            entry['title'] = 'Build #{0} {1}'.format(build, nstate)
            return True

    def filter_pypi(self, entry):
        """Show only usefull packages"""
        for package in self.packages:
            if entry.title.lower().startswith(package):
                return entry

    def __call__(self, entries):
        travis = {}
        for entry in entries:
            if entry.feed.name.startswith('travis/'):
                travis[entry.feed.name] = entry
            elif entry.feed.name.startswith('pypi/'):
                yield self.filter_pypi(entry)
            else:
                yield entry
        for entry in travis.values():
            if self.filter_travis(entry):
                yield entry
```

3.16 irc3.plugins.fifo Fifo plugin

Allow to cat something to a channel using Unix's fifo

Usage:

```
>>> config = ini2config("""
... [bot]
... includes =
...     irc3.plugins.fifo
... [irc3.plugins.fifo]
... runpath = /tmp/run/irc3
... """)
>>> bot = IrcBot(**config)
```

When your bot will join a channel it will create a fifo:

```
>>> bot.test(':irc3!user@host JOIN #channel')
>>> print(sorted(os.listdir('/tmp/run/irc3')))
[':raw', 'channel']
```

You'll be able to print stuff to a channel from a shell:

```
$ cat /etc/passwd > /tmp/run/irc3/channel
```

You can also send raw irc commands using the `:raw` file:

```
$ echo JOIN \#achannel > /tmp/run/irc3/:raw
```

3.17 irc3.plugins.human Human plugin

Store public message addressed to the bot in a file and reply a random message extracted from this file.

Register the plugin:

```
>>> bot = IrcBot(human='/tmp/human.db', nick='nono')
>>> bot.include('irc3.plugins.human')
```

And it should work:

```
>>> bot.test(':foo!m@h PRIVMSG nono :nono: Yo!')
PRIVMSG foo :Yo!

>>> bot.test(':foo!m@h PRIVMSG #chan :nono: Yo!')
PRIVMSG #chan :foo: Yo!
```

3.18 irc3.plugins.log Log plugin

Logging / debuging plugin

Usage:

```
>>> bot = IrcBot()
>>> bot.include('irc3.plugins.log')
```

3.19 irc3.plugins.logger Channel logger plugin

Log channels

Usage:

```
>>> bot = IrcBot(**{
...     'irc3.plugins.logger': {
...         'handler': 'irc3.plugins.logger.file_handler',
...     },
... })
>>> bot.include('irc3.plugins.logger')
```

Available handlers:

class irc3.plugins.logger.**file_handler**(*bot*)
Write logs to file in ~/.irc3/logs

3.20 irc3.plugins.pager Paginate large output

3.20.1 Usage

```
# -*- coding: utf-8 -*-
import irc3
import requests
from irc3.plugins.command import command


@irc3.plugin
class SendFile(object):

    requires = [
        'irc3.plugins.command',
        'irc3.plugins.pager',
    ]

    def __init__(self, bot):
        self.bot = bot

    @command
    def cat(self, mask, target, args):
        """Cat a file with pagination

            %%cat
        """
        fd = open(__file__)
        for msg in self.bot.paginate(mask, fd, lines_per_page=10):
            yield msg
```

(continues on next page)

(continued from previous page)

```
    @command
    def url(self, mask, target, args):
        """Cat an url with pagination

            %%url <url>
        """
        def iterator(url):
            resp = requests.get(url)
            for chunk in resp.iter_content(255):
                yield chunk.decode('utf8')
        for msg in self.bot.paginate(mask, iterator(args['<url>'])):
            yield msg
```

3.20.2 API

class irc3.plugins.pager.**Paginate**(*context*)
Pagination plugin

3.21 irc3.plugins.quakenet QuakeNet authorization

Plugin supports both simple and challenge based authorization. Challenge based auth is used by default, since it is more secure than simple. Also, plugin can hide your IP after authorization by applying +x mode.

Usage:

```
>>> config = ini2config("""
... [bot]
... includes =
...     irc3.plugins.quakenet
... [quakenet]
... user = login
... password = passw
... # optional, false by default
... hidehost = true
... # optional, true by default
... challenge_auth = true
... """)
>>> bot = IrcBot(**config)
```

3.22 irc3.plugins.sasl SASL authentification

Allow to use sasl authentification

Usage:

```
>>> config = ini2config("""
... [bot]
... sasl_username = irc3
... sasl_password = passwd
... """)
>>> bot = IrcBot(**config)
```

3.23 irc3.plugins.search Search plugin

class irc3.plugins.search.**Search**(*bot*)

3.24 irc3.plugins.shell_command Shell commands

Allow to quickly add commands map to a shell command. The bot will print stdout/stderr

Usage:

```
>>> config = ini2config("""
... [bot]
... includes =
...     irc3.plugins.shell_command
... [irc3.plugins.shell_command]
... myscript = /tmp/myscript
... # optional command configuration
... myscript.permission = runmyscrypt
... myscript.public = false
... # register a directory
... myscripts = /tmp/myscripts
... # optional commands configuration for the directory
... myscripts.permission = runmyscrypts
... """)
>>> bot = IrcBot(**config)
```

Then the uname command will be available:

```
>>> bot.test(':gawel!user@host PRIVMSG irc3 :!help myscript')
PRIVMSG gawel :Run $ /tmp/myscript
PRIVMSG gawel :!myscript [<args>...]

>>> bot.test(':gawel!user@host PRIVMSG #chan :!myscript')
PRIVMSG gawel :You can only use the 'myscript' command in private.

>>> bot.test(':gawel!user@host PRIVMSG irc3 :!myscript')
```

If the user provides some arguments then those will be available as an environment var (to avoid shell injection) names IRC3_COMMAND_ARGS

...

```
>>> bot.get_plugin(Commands)['demo'][0]
{'permission': 'runmyscrypts'}
```

3.25 irc3.plugins.slack Slack plugin

Introduce a slack/irc interface to bridge messages between slack and irc.

Install aiohttp:

```
$ pip install aiohttp
```

3.25.1 Usage

Create a bridge between slack and irc

Note: Be sure to invite the bot in slack to the channels it should be bridging

3.25.2 Config Options

irc3.plugins.slack

token

slack api token (user or bot integration)

notify

The slack user ids to notify, this needs to be the unique user ids.

irc3.plugins.slack.channels

This section is a list of channels that should be bridge. The first is the slack channel that the bot needs to be joined to to bridge slack to irc.

Then assigned to that is the list of irc channels that slack channel should forward to.

3.26 `irc3.plugins.social` Social networking

Add `tweet` and `retweet` commands.

Extend the bot with `.get_social_connection()` and `.search_tweets()`.

Usage:

```
>>> bot = IrcBot(
...     includes=['irc3.plugins.social'],
...     twitter=dict(key='yourkey', secret='yoursecret',
...                  token='yourtoken', token_secret='yoursecret')
... )
>>> bot.get_social_connection()
<TwitterAdapter for <twitter.api.Twitter object at ...>>
```

Api:

class irc3.plugins.social.**Social**(*bot*)

The social plugin

get_social_connection(*id=None*)

return a connection object for the network:

- A Twitter instance from https://github.com/sixohsix/twitter/tree/master

retweet(*mask*, *target*, *args*)

Retweet

%%retweet [–id=<id>] <url_or_id>

search_tweets (*q=None*, ***kwargs*)
Search for tweets on twitter

send_tweet (*message*, *id=None*)
Send a tweet to networks

tweet (*mask*, *target*, *args*)
Post to social networks

%%tweet [–id=<id>] <message>...

3.27 irc3.plugins.storage Storage plugin

Add a db attribute to the bot

Usage:

```
>>> config = ini2config("""
... [bot]
... includes =
...     irc3.plugins.storage
... storage = json://%s
... """ % json_file)
>>> bot = IrcBot(**config)
```

Then use it:

```
>>> bot.db['mykey'] = dict(key='value')
>>> 'mykey' in bot.db
True
>>> bot.db['mykey']
{'key': 'value'}
>>> bot.db.setdefault('mykey', key='default')
{'key': 'value'}
>>> bot.db.setdefault('mykey', item='default')
{'item': 'default'}
>>> bot.db.set('mykey', item='value')
>>> bot.db.setdefault('mykey', item='default')
{'item': 'value'}
>>> del bot.db['mykey']
>>> bot.db.get('mykey')
>>> bot.db.get('mykey', 'default')
'default'
>>> bot.db['mykey']
Traceback (most recent call last):
  ...
KeyError: 'mykey'
>>> 'mykey' in bot.db
False
>>> bot.db.setlist('mylist', ['foo', 'bar'])
>>> bot.db.getlist('mylist')
['foo', 'bar']
>>> del bot.db['mylist']
```

You can use an instance as key:

```
>>> class MyPlugin:
...     pass
>>> plugin = MyPlugin()
>>> bot.db[plugin] = dict(key='value')
>>> bot.db[plugin]
{'key': 'value'}
>>> del bot.db[plugin]
>>> bot.db.get(plugin)
```

You can also use shelve:

```
>>> config = ini2config("""
... [bot]
... includes =
...     irc3.plugins.storage
... storage = shelve://%s
... """ % db_file)
>>> bot = IrcBot(**config)
>>> bot.db['mykey'] = dict(key='value')
>>> bot.db['mykey']
{'key': 'value'}
>>> del bot.db['mykey']
>>> bot.db.get('mykey')
>>> bot.db.setlist('mylist', ['foo', 'bar'])
>>> bot.db.getlist('mylist')
['foo', 'bar']
>>> del bot.db['mylist']
```

Or redis:

```
>>> config = ini2config("""
... [bot]
... includes =
...     irc3.plugins.storage
... storage = redis://localhost:6379/10
... """)
>>> bot = IrcBot(**config)
```

Then use it:

```
>>> bot.db['mykey'] = dict(key='value')
>>> bot.db['mykey']
{'key': 'value'}
>>> del bot.db['mykey']
>>> bot.db.get('mykey')
>>> bot.db['mykey']
Traceback (most recent call last):
  ...
KeyError: 'mykey'
>>> bot.db.setlist('mylist', ['foo', 'bar'])
>>> bot.db.getlist('mylist')
['foo', 'bar']
>>> del bot.db['mylist']
```

3.27.1 Api

class irc3.plugins.storage.**Storage**(*context*)

__contains__(*key*)
Return True if storage contains key

__delitem__(*key*)
Delete key in storage

__getitem__(*key*)
Get storage value for key

__setitem__(*key*, *value*)
Set storage value for key

get(*key_*, *default=None*)
Get storage value for key or return default

set(*key_*, ***kwargs*)
Update storage value for key with kwargs

setdefault(*key_*, ***kwargs*)
Update storage value for key with kwargs iif the keys doesn't exist. Return stored values

3.28 irc3.plugins.uptime Uptime plugin

Add an `uptime` command.

class irc3.plugins.uptime.**Uptime**(*bot*)

uptime(*mask*, *target*, *args*)
Show uptimes

%%uptime

3.29 irc3.plugins.userlist User list plugin

This plugin maintain a known user list and a channel list.

Usage:

```
>>> bot = IrcBot()
>>> bot.include('irc3.plugins.userlist')
>>> bot.test(':gawel!user@host JOIN #chan')

>>> print(list(bot.channels['#chan'])[0])
gawel
>>> print(list(bot.nicks.keys())[0])
gawel

>>> bot.test(':gawel!user@host MODE #chan +o gawel')

>>> print(list(bot.channels['#chan'].modes['@'])[0])
gawel
```

3.29.1 Api

class irc3.plugins.userlist.**Channel**

A set like object which contains nicknames that are on the channel and user modes:

```
>>> channel = Channel()
>>> channel.add('gawel', modes='@')
>>> 'gawel' in channel
True
>>> 'gawel' in channel.modes['@']
True
>>> channel.remove('gawel')
>>> 'gawel' in channel
False
>>> 'gawel' in channel.modes['@']
False
```

3.30 irc3.plugins.web Web plugin

Introduce a web interface to post messages

Install aiohttp:

```
$ pip install aiohttp
```

Usage:

This example show how to define the web server config:

```
>>> config = ini2config("""
... [bot]
... includes =
...     irc3.plugins.web
...
... [irc3.plugins.web]
... host = 127.0.0.1
... port = 8080
... api_key = toomanysecrets
... """)
>>> bot = IrcBot(**config)
```

Then you'll be able to post a message to a channel using curl:

```
$ curl -H "X-Api-Key: toomanysecrets"       --data Hello       http://127.0.0.1:8080/
↪channels/irc3
```

3.31 Contribute

First, if you want to add a cool plugin, consider submit a pull request to the irc3_plugins instead of irc3 itself.

Feel free to clone the project on GitHub.

Once you made a change, try to add a test for your feature/fix. At least assume that you have'nt broke anything by running tox:

```
   $ tox
   ...
   py27: commands succeeded
   py32: commands succeeded
   py33: commands succeeded
   py34: commands succeeded
   flake8: commands succeeded
   docs: commands succeeded
   congratulations :)

You can run tests for a specific version::

   $ tox -e py34
```

The *irc3.rfc* module is auto generated from *irc3/rfc1459.txt*. If you want to hack this file, you need to hack the parser in *irc3/_parse_rfc.py* (warning, it's ugly)

You can regenerate the module and docs by running:

```
$ tox -e build
```

You can also build the docs with:

```
$ tox -e docs
```

And check the result:

```
$ firefox .tox/docs/tmp/html/index.html
```

The project is buildout ready. You can generate binaries using it instead of virtualenv:

```
$ python bootstrap.py
$ bin/buildout
$ bin/irc3 -h
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

i

irc3.dcc, 44
irc3.dec, 9
irc3.plugins.asynchronious, 46
irc3.plugins.autocommand, 48
irc3.plugins.autojoins, 49
irc3.plugins.casefold, 49
irc3.plugins.command, 49
irc3.plugins.core, 53
irc3.plugins.cron, 53
irc3.plugins.ctcp, 54
irc3.plugins.dcc, 54
irc3.plugins.feeds, 55
irc3.plugins.fifo, 56
irc3.plugins.human, 57
irc3.plugins.log, 57
irc3.plugins.logger, 58
irc3.plugins.pager, 58
irc3.plugins.quakenet, 59
irc3.plugins.sasl, 59
irc3.plugins.search, 59
irc3.plugins.shell_command, 60
irc3.plugins.slack, 60
irc3.plugins.social, 61
irc3.plugins.storage, 62
irc3.plugins.uptime, 64
irc3.plugins.userlist, 64
irc3.plugins.web, 65
irc3.utils, 10

Index

Symbols

__call__() (*irc3.asynchronous.AsyncEvents method*), 47
__contains__() (*irc3.plugins.storage.Storage method*), 64
__delitem__() (*irc3.plugins.storage.Storage method*), 64
__getitem__() (*irc3.plugins.storage.Storage method*), 64
__setitem__() (*irc3.plugins.storage.Storage method*), 64

A

as_channel() (*in module irc3.utils*), 12
as_list() (*in module irc3.utils*), 11
Async (*class in irc3.plugins.asynchronious*), 47
async_who_channel_flags() (*irc3.plugins.asynchronious.Async method*), 47
AsyncEvents (*class in irc3.asynchronous*), 47

B

badnick() (*irc3.plugins.core.Core method*), 53

C

Channel (*class in irc3.plugins.userlist*), 65
channel_bans() (*irc3.plugins.asynchronious.Async method*), 47
chat() (*irc3.plugins.dcc.Commands method*), 55
Commands (*class in irc3.plugins.dcc*), 55
Config (*class in irc3.utils*), 12
connected() (*irc3.plugins.core.Core method*), 53
connection_made() (*irc3.dcc.DCCChat method*), 44
connection_made() (*irc3.dcc.DCCGet method*), 45
connection_made() (*irc3.dcc.DCCSend method*), 45
Core (*class in irc3.plugins.core*), 53
create() (*irc3.dcc.DCCManager method*), 44
ctcp_async() (*irc3.plugins.asynchronious.Async method*), 47

D

data_received() (*irc3.dcc.DCCChat method*), 44
data_received() (*irc3.dcc.DCCGet method*), 45
data_received() (*irc3.dcc.DCCSend method*), 45
dcc_command() (*in module irc3.plugins.dcc*), 55
DCCChat (*class in irc3.dcc*), 44
DCCGet (*class in irc3.dcc*), 45
DCCManager (*class in irc3.dcc*), 44
DCCSend (*class in irc3.dcc*), 45
decode() (*irc3.dcc.DCCChat method*), 45

E

event (*class in irc3.dec*), 9
extend() (*in module irc3.dec*), 10
extract_config() (*in module irc3.utils*), 12

F

file_handler (*class in irc3.plugins.logger*), 58
free_policy (*class in irc3.plugins.command*), 51

G

get() (*irc3.plugins.storage.Storage method*), 64
get_social_connection() (*irc3.plugins.social.Social method*), 61

H

help() (*irc3.plugins.dcc.Commands method*), 55
hostname (*irc3.utils.IrcString attribute*), 10

I

irc3.dcc (*module*), 44
irc3.dec (*module*), 9
irc3.plugins.asynchronious (*module*), 46
irc3.plugins.autocommand (*module*), 48
irc3.plugins.autojoins (*module*), 49
irc3.plugins.casefold (*module*), 49

irc3.plugins.command (*module*), 49
irc3.plugins.core (*module*), 53
irc3.plugins.cron (*module*), 53
irc3.plugins.ctcp (*module*), 54
irc3.plugins.dcc (*module*), 54
irc3.plugins.feeds (*module*), 55
irc3.plugins.fifo (*module*), 56
irc3.plugins.human (*module*), 57
irc3.plugins.log (*module*), 57
irc3.plugins.logger (*module*), 58
irc3.plugins.pager (*module*), 58
irc3.plugins.quakenet (*module*), 59
irc3.plugins.sasl (*module*), 59
irc3.plugins.search (*module*), 59
irc3.plugins.shell_command (*module*), 60
irc3.plugins.slack (*module*), 60
irc3.plugins.social (*module*), 61
irc3.plugins.storage (*module*), 62
irc3.plugins.uptime (*module*), 64
irc3.plugins.userlist (*module*), 64
irc3.plugins.web (*module*), 65
irc3.utils (*module*), 10
IrcString (*class in irc3.utils*), 10
is_allowed() (*irc3.dcc.DCCManager method*), 44
is_channel (*irc3.utils.IrcString attribute*), 10
is_nick (*irc3.utils.IrcString attribute*), 11
is_server (*irc3.utils.IrcString attribute*), 11
ison() (*irc3.plugins.asynchronious.Async method*), 48

L

lnick (*irc3.utils.IrcString attribute*), 11
Logger (*class in irc3.utils*), 12

M

mask_based_policy (*class in irc3.plugins.command*), 51
maybedotted() (*in module irc3.utils*), 12

N

names() (*irc3.plugins.asynchronious.Async method*), 48
nick (*irc3.utils.IrcString attribute*), 11

P

Paginate (*class in irc3.plugins.pager*), 59
parse_config() (*in module irc3.utils*), 12
ping() (*irc3.plugins.core.Core method*), 53
plugin() (*in module irc3.dec*), 9
pong() (*irc3.plugins.core.Core method*), 53
process_results() (*irc3.asynchronous.AsyncEvents method*), 47

R

recompile() (*irc3.plugins.core.Core method*), 53
resume() (*irc3.dcc.DCCManager method*), 44
retweet() (*irc3.plugins.social.Social method*), 61

S

Search (*class in irc3.plugins.search*), 60
search_tweets() (*irc3.plugins.social.Social method*), 61
send_tweet() (*irc3.plugins.social.Social method*), 62
set() (*irc3.plugins.storage.Storage method*), 64
set_config() (*irc3.plugins.core.Core method*), 53
set_irc_targets() (*irc3.utils.Logger method*), 12
setdefault() (*irc3.plugins.storage.Storage method*), 64
Social (*class in irc3.plugins.social*), 61
split_message() (*in module irc3.utils*), 12
Storage (*class in irc3.plugins.storage*), 64

T

tagdict (*irc3.utils.IrcString attribute*), 11
tweet() (*irc3.plugins.social.Social method*), 62

U

Uptime (*class in irc3.plugins.uptime*), 64
uptime() (*irc3.plugins.uptime.Uptime method*), 64
username (*irc3.utils.IrcString attribute*), 11

W

who() (*irc3.plugins.asynchronious.Async method*), 48
whois() (*irc3.plugins.asynchronious.Async method*), 48