coffea docs

Release 0.4.0

-

Oct 27, 2018

Contents

event based and extensible nodejs irc client library with multi-network support

For support, report an issue on github or join our IRC channel at

If you want to support coffea, please consider donating (it helps me keeping the project active and alive!):

CHAPTER 1

quickstart

This is all the code needed to get you started with coffea!

```
 var client = require('coffea')(['chat.freenode.net', 'irc.oftc.net']); // or put␣
↪just one network as a string

 client.on('motd', function (event) {
     client.join(['#foo', '#bar', '#baz'], event.network);
 });

 client.on('message', function (event) {
     console.log('[' + event.network + '][' + event.channel.getName() + '] ' + event.
↪user.getNick() + ': ' + event.message);
     //[freenode][#foo] nick: message
     event.reply(event.message); // I'm a parrot
 });

client.on('command', function (event) {
    if (event.cmd === 'ping') { // respond to `!ping SOMETHING` with `SOMETHING`, or␣
↪`pong`, if SOMETHING is not specified
        event.reply(event.args.length > 0 ? event.args.join(' ') : 'pong');
    }
});
```

1.1 tutorial

For an extensive guide on getting started with coffea, make sure to read *Writing an IRC Bot with JavaScript/Node.js and coffea* on medium: https://medium.com/@omnidan/writing-an-irc-bot-with-javascript-node-js-and-coffea-6c13aa4907e9

1.2 with ssl

If you want to enable SSL for a connection, you have to use a network config object when connecting:

```
var client = require('coffea')({
    host: 'chat.freenode.net',
    ssl: true,
    // ssl_allow_invalid: true // uncomment this if the server has a self signed␣
↪certificate, make sure to listen on('ssl-error') to catch bad certificates
});
```

1.3 full config

For multiple networks, use a JavaScript array with multiple config objects inside.

```
var client = require('coffea')({
    host: 'chat.freenode.net',
    port: 6667, // default value: 6667
    ssl: false, // set to true if you want to use ssl
    ssl_allow_invalid: false, // set to true if the server has a custom ssl␣
↪certificate
    prefix: '!', // used to parse commands and emit on('command') events, default: !
    channels: ['#foo', '#bar'], // autojoin channels, default: []
    nick: 'test', // default value: 'coffea' with random number
    username: 'test', // default value: username = nick
    realname: 'test', // default value: realname = nick
    pass: 'sup3rS3cur3P4ssw0rd', // by default no password will be sent
    nickserv: {
        username: 'test',
        password: 'l33tp455w0rD'
    },
    throttling: 250 // default value: 250ms, 1 message every 250ms, disable by␣
↪setting to false
});
```

CHAPTER 2

debugging

Make sure to listen to `error` events to see possible errors/warnings:

```
client.on('error', function (err, event) {
    console.log(event.name, err.stack);
});
```

You can also add the `err` parameter to any event listener (change from `function (event)` to `function (err, event)`):

```
client.on('whois', function (err, event) {
    if (err) console.log("WHOIS ERROR:", err.stack);
    console.log("whois:", event);
});
```

CHAPTER 3

configuration file

Using a configuration file for your bot is super easy with coffea! First, create `config.json` and paste your current configuration. (Make sure to change `key:  'something'` to `"key":  "something"` as we're dealing with JSON now, e.g. `host:  'chat.freenode.net'` -> `"host":  "chat.freenode.net"`)

Then, simply do:

```
var client = require('coffea')(require('./config.json'));
```

3.1 `config.json` with full config

```
{
    "host": "chat.freenode.net",
    "port": 6667,
    "ssl": false,
    "ssl_allow_invalid": false,
    "prefix": "!",
    "channels": ["#foo", "#bar"],
    "nick": "test",
    "username": "test",
    "realname": "test",
    "pass": "sup3rS3cur3P4ssw0rd",
    "nickserv": {
        "username": "test",
        "password": "l33tp455w0rD"
    },
    "throttling": 250
}
```

CHAPTER 4

api

The best way to get started with coffea is playing around with it! You can use the genindex to get a list of `functions` and `events` you can make use of.

coffea function: A function called with `client.function()` if coffea was imported to `client` (see Quickstart)

```
client.function();
```

coffea event: You can listen to an event by defining an event listener like this (make sure to replace `event` with the event name):

```
client.on('event', function (event) {
    console.log(event); // do something with event here
});
```

Go to API reference or browse the API by…

4.1 plugins

4.1.1 connection api

class Client(*info*)

Arguments

- **info** (*object*) – details for IRC connection

Initialise a server connection using the `info` parameter. `info` can be a javascript object representing server details or an array of multiple server details.

Example:

```
var client = Client({
    name: 'freenode', // Optional
    host: 'chat.freenode.net', // Required
    port: 6697, // Optional
    ssl: true, // Optional
    nick: 'nickname', // Optional, but will autogenerate nick if not specified
    username: 'test', // Optional
    realname: 'Testing Robot' // Optional
});
```

add(info)

> **Param object info** Details for new connection.
>
> **Returns** The stream_id.

Add another network to an existing client, using the `info` parameter (as described at `Client()`), since the API supports multiple servers, more than one `add` calls can be specified throughout your program.

Multi-server example:

```
client.add([
    { // existing `Client` object
        host: 'chat.freenode.net', // required
        name: 'freenode',  // optional, but aids in identification, when referenced␣
↪later in the program.
        nick: 'test', // optional, but will autogenerate nick if not specified
        ssl: true, // optional
        username: 'test', // optional
        realname: 'Testing Robot' // optional
    },
    {
        host: 'irc.oftc.net',
        name: 'oftc',
        nick: 'test2',
        ssl: false,
        username: 'test',
        realname: 'Testing Robot'
    }
]);
```

Single-server example:

```
client.add({ // existing `Client` object
    host: 'chat.freenode.net', // required
    name: 'freenode',  // optional, but aids in identification, when referenced later␣
↪in the program.
    nick: 'test', // required
    ssl: true, // optional
    username: 'test', // optional
    realname': 'Testing Robot' // optional'
});
```

write(str, network, fn)

> **Param string str** The string to be written.
>
> **Param string network** The network that the string shall be written to.
>
> **Param function fn** The callback function to be called when the `write` call has been finished.

Returns string stream_id The stream ID from the call.

Writes raw data (`str`), to `network`, when finished, calls `fn`.

full config

For multiple networks, use a JavaScript array with multiple config objects inside.

```
var client = require('coffea')({
    host: 'chat.freenode.net',
    port: 6667, // default value: 6667
    ssl: false, // set to true if you want to use ssl
    ssl_allow_invalid: false, // set to true if the server has a custom ssl␣
↪certificate
    prefix: '!', // used to parse commands and emit on('command') events, default: !
    channels: ['#foo', '#bar'], // autojoin channels, default: []
    nick: 'test', // default value: 'coffea' with random number
    username: 'test', // default value: username = nick
    realname: 'test', // default value: realname = nick
    pass: 'sup3rS3cur3P4ssw0rd', // by default no password will be sent
    nickserv: {
        username: 'test',
        password: 'l33tp455w0rD'
    },
    throttling: 250 // default value: 250ms, 1 message every 250ms, disable by␣
↪setting to false
});
```

4.1.2 event api

Events are applied using the following example (basically standard NodeJS EventEmitter syntax):

```
client.on('EVENT', // `EVENT` being any of the events listed in the documentation.
    function (event) { // callback function, called when event is fired
        console.log('Something happened!');
    });
```

event object

You may have noticed that we are passing an event argument to the event binding function. This is actually not just data but an object with an API to help you deal with events without hassle. Not all events are the same, so please check the various plugin documentations to see what events are available. The following functions and attributes are always available:

`network`

The network this event was triggered in.

Note: The reply functions are only going to succeed when the `channel` or `user` attribute is available.

`reply`

Answer to a message (same channel/query as the event came from).

Example:

```
client.on('command', function (event) {
    if (event.cmd === 'ping') event.reply('pong');
});
```

replyAction

Answer to a message with an action (`/me`). Works like `reply(message)`.

replyNotice

Answer to a message with a notice. Works like `reply(message)`.

core events

ssl-error

The `ssl-error` event, fired when there was an error establishing an SSL connection. If you're running with `ssl_allow_invalid` this event will still fire, but coffea will continue connecting to the server afterwards.

Event attributes:

- None

Example:

```
client.on('ssl-error', function (event) {
    console.error('SSL Error:', err);
});
```

disconnect

The `disconnect` event, fired when the client was disconnected from a network.

Event attributes:

- None

Example:

```
client.on('disconnect', function (event) {
    console.log("We disconnected!");
});
```

event

The `event` event, fired when any other event is fired.

Example:

```
client.on('event', function (name, err, event) {
    console.log(name, "event fired:", err, event);
});
```

4.1.3 user api

class User(*nick*, *client*, *network*)

Arguments

- **nick** (*string*) – The users nickname

- **client** (*object*) – Client object.
- **network** (*string*) – Network object.

Create a new user object.

`User.`**`toString`**`()`

Returns string hostmask Hostmask of the user.

`User.`**`getNick`**`()`

Returns string Nick Nick of the user.

`User.`**`getNetwork`**`()`

Returns string Network Network of the user.

`User.`**`getUsername`**`()`

Returns string Username Username of the user.

`User.`**`getRealname`**`()`

Returns string Realname Realname of the user.

`User.`**`getHostname`**`()`

Returns string Hostname Hostname of the user.

`User.`**`getAccountName`**`()`

Returns string AccountName AccountName of the user.

`User.`**`getChannels`**`()`

Returns object Channels Channels of the user. `{'#channel': ['~']}`

`User.`**`getServer`**`()`

Returns string Server Server the user is connected to.

`User.`**`getServerInfo`**`()`

Returns string ServerInfo ServerInfo of the Server the user is connected to.

`User.`**`getAway`**`()`

Returns boolean Away Away status of the user.

`User.`**`getAccount`**`()`

Returns string Account Account of the user.

`User.`**`isRegistered`**`()`

Returns boolean registered Registration status of the user.

`User.`**`isUsingSecureConnection`**`()`

Returns boolean secureConnection SSL status of the user (on/off).

`User.`**`getIdle`**`()`

Returns int Idle Idletime of the user.

`User.`**`getSignonTime`**`()`

Returns string SignonTime SignonTime of the user.

`User.`**`isOper`**`()`

Returns boolean oper Oper status of the user.

`User.notice(msg)`

Arguments

- **msg** (*string*) – Notice message to send to the user.

`User.say(msg)`

Arguments

- **msg** (*string*) – Message to send to the user.

`User.whois(fn)`

Arguments

- **fn** (*function*) – The callback function to be called when the call has been finished.

events

nick

The `nick` event, fired when someone changes nickname.

Event attributes:

- `user` - User who changed nick.
- `oldNick` - Their old nickname.

Example:

```
client.on('nick', function (event) {
    console.log(event.oldNick + " is now " + event.user.getNick());
});
```

whois

The `whois` event, fired when the whois response is received from the server.

Event attributes:

- `whoismap - {user:  whois_data}.`

Example:

```
client.on('whois', function (err, event) {
    if (err) console.err("Couldn't whois:", err);
    console.log(event);
});
```

functions

getUser(nick, network)

Param string nick The user you want to get by nickname.

Param string network The network to execute the command on.

Get a user object by nickname.

Example:

```
var user = client.getUser("#caffeinery", event.network); // usage in an event listener
var user = client.getUser("#caffeinery", "freenode"); // send to specific network
```

isUser(user)

Param object user The user object you want to check.

Checks if the passed object is a valid user object.

Example:

```
var user = client.getUser("#caffeinery", event.network); // usage in an event listener
console.log(client.isUser(user)); // prints `true`
```

isMe(user)

Param object user The user object you want to check.

Checks if the passed object is the same user object as the client.

Example:

```
if (client.isMe(event.user)) {
    console.log("message was from me");
} else {
    console.log("message was from somebody else");
}
```

whois(target, network, fn)

Param object target The `channel` or `user` object to send this message to.

Param string network The network to execute the command on.

Param function fn The callback function to be called when the call has been finished.

Send a whois request to the server.

Example:

```
client.whois("omnidan", event.network, function (event) {
    console.log(event.whois);
}); // usage in an event listener

client.whois("omnidan", "freenode", function (event) {
    console.log(event.whois);
}); // send to specific network
```

identify(username, password, network, fn)

Param string username The username to identify with.

Param string password The password to identify with.

Param string network The network to execute the command on.

Param function fn The callback function to be called when the call has been finished.

Identifies the user with nickserv.

Example:

```
client.on('motd', function (event) {
    client.identify("omnidan", "p@$$w0rd", event.network); // log in with nickserv
});
```

You can specify this data in the config too like this:

```
{
  nickserv: {
    username: 'test',
    password: 'l33tp455w0rD'
  }
}
```

nick(nick, network, fn)

Param string nick The nickname you want to use now.

Param string network The network to execute the command on.

Param function fn The callback function to be called when the call has been finished.

Change your nickname to the specified nick.

Example:

```
client.on('motd', function (event) {
    client.nick("omnidan", event.network); // log in with nickserv
});
```

You can specify this data in the config too like this:

```
{
  nick: 'omnidan'
}
```

4.1.4 channel api

class Channel(*name*, *client*, *network*)

Arguments

- **name** (*string*) – The channel name
- **client** (*object*) – Client object.
- **network** (*string*) – Network object.

Create a new channel object.

Channel.**toString**()

Returns string hostmask Hostmask of the channel.

Channel.**getName**()

Returns string Name Name of the channel.

Channel.**getTopic**()

Returns string Topic Topic of the channel.

Channel.**getNames**()

Returns object names Nicks in the channel: `{'nick': ['~']}`

`Channel.`**`getNetwork`**`()`

Returns string Network Network of the channel.

`Channel.`**`userHasMode`**`(`*user*, *mode*`)`

Arguments

- **`user`** (*`object`*) – The user to check the mode of in the channel.
- **`mode`** (*`string`*) – The mode to check for.

Returns boolean hasMode `true` if specified user has specified mode.

`Channel.`**`isUserInChannel`**`(`*user*`)`

Arguments

- **`user`** (*`object`*) – The user to check.

Returns boolean hasMode `true` if specified user is in this channel.

`Channel.`**`notice`**`(`*msg*`)`

Arguments

- **`msg`** (*`string`*) – Notice message to send to the user.

`Channel.`**`say`**`(`*msg*`)`

Arguments

- **`msg`** (*`string`*) – Message to send to the user.

`Channel.`**`reply`**`(`*user*, *msg*`)`

Arguments

- **`user`** (*`object`*) – User to reply to.
- **`msg`** (*`string`*) – Message to send to the user.

`Channel.`**`kick`**`(`*user*, *reason*`)`

Arguments

- **`user`** (*`object`*) – User to kick from channel.
- **`reason`** (*`string`*) – Reason for the kick.

`Channel.`**`ban`**`(`*mask*`)`

Arguments

- **`mask`** (*`string`*) – Hostmask to ban.

`Channel.`**`unban`**`(`*mask*`)`

Arguments

- **`mask`** (*`string`*) – Hostmask to unban.

events

invite

The invite event, fired when someone gets invited by someone.

Event attributes:

- channel - Channel you got invited to.
- user - User who sent the invite.
- target - Invited user.

Example:

```
client.on('invite', function (event) {
    console.log(event.target.getNick() + " got invited to "
        + event.channel.getName() + " by " + event.user.getNick());
});
```

topic

The topic event, fired when the topic gets changed. (or is originally sent)

Event attributes:

- topic - Current topic.
- user - User who changed the topic.
- time - Time of topic change.
- changed - true if topic was changed in this event.
- channel - Affected channel.
- network - Affected network.

Example:

```
client.on('topic', function (event) {
    console.log(event.channel.getName() + ":", event.topic);
});
```

join

The join event, fired when someone joins a channel.

Event attributes:

- user - User who joined.
- channel - Channel that was joined.

Example:

```
client.on('join', function (event) {
    console.log(event.user.getNick() + " joined " + event.channel.getName());
});
```

names

The names event, fired when getting users in the channel.

Event attributes:

- `channel` - Affected channel.
- `names` - List of users in the channel.

Example:

```
client.on('names', function (event) {
    console.log(event.channel.getName() + ":", event.names);
});
```

mode

The `mode` event, fired when a mode gets changed.

Event attributes:

- `mode` - Current mode.
- `channel` - Affected channel.
- `by` - User who changed the mode.
- `argument` - Mode argument.
- `adding` - boolean

Example:

```
client.on('mode', function (event) {
    console.log(event.channel.getName() + ":", event.mode);
});
```

kick

The `kick` event, fired when a user gets kicked.

Event attributes:

- `channel` - Affected channel.
- `user` - Affected user.
- `by` - User who changed the kick.
- `reason` - Reason for the kick.

Example:

```
client.on('kick', function (event) {
    console.log(event.channel.getName() + ":", event.user.getNick(), "was kicked.");
});
```

part

The `part` event, fired when a user parts a channel (channels).

Event attributes:

- `user` - Affected user.
- `channels` - Affected channels (channelList).
- `message` - Part message.

Example:

```
client.on('part', function (event) {
    console.log(event.user.getNick(), "parted channels", event.channels);
});
```

functions

`getChannelList(network)`

Param string network The network to execute the command on.

Return array channelList List of channels.

Get a list of channels.

Example:

```
var channels = client.getChannelList(event.network); // usage in an event listener
var channels = client.getChannelList("freenode"); // send to specific network
var channels = client.getChannelList(); // get object of networks and channels
```

`getChannel(name, network)`

Param string name The name of the channel you want to get.

Param string network The network to execute the command on.

Gets a channel by name.

Example:

```
var channel = client.getChannel("#caffeinery", event.network); // usage in an event␣
↪listener
var channel = client.getChannel("#caffeinery", "freenode"); // send to specific␣
↪network
```

`isChannel(channel)`

Param object channel The channel object you want to check.

Checks if the passed object is a valid channel object.

Example:

```
var channel = client.getChannel("#caffeinery", event.network); // usage in an event␣
↪listener
console.log(client.isChannel(channel)); // prints `true`
```

`invite(name, channel, network, fn)`

Param string name The name of the user you want to invite.

Param string/object channel The channel you want to invite him to.

Param string network The network to execute the command on.

Param function fn The callback function to be called when the call has been finished.

Invites a user to a channel.

Example:

```
client.invite("omnidan", "#caffeinery", event.network); // usage in an event listener
client.invite("omnidan", "#caffeinery", "freenode"); // send to specific network
```

`topic(channel, topic, network, fn)`

Param string/object channel The channel you want to set the topic in.

Param string topic The topic you want to set.

Param string network The network to execute the command on.

Param function fn The callback function to be called when the call has been finished.

Sets the topic of a channel.

Example:

```
client.topic("#caffeinery", "welcome to caffeinery!", event.network); // usage in an␣
↪event listener
client.topic("#caffeinery", "welcome to caffeinery!", "freenode"); // send to␣
↪specific network
```

`join(channels, keys, network, fn)`

Param string/object/array channels The channel(s) you want to join.

Param string/array keys The key(s) for the channel(s) you want to join.

Param string network The network to execute the command on.

Param function fn The callback function to be called when the call has been finished.

Joins channels.

Example:

```
client.join("#caffeinery", event.network); // usage in an event listener
client.join("#caffeinery", "freenode"); // send to specific network
client.join(["#caffeinery", "#omnidan"], event.network); // join multiple channels
```

Example (Join password protected channels):

```
client.join("#caffeinery", event.network); // usage in an event listener
client.join("#caffeinery", "freenode"); // send to specific network
client.join(["#caffeinery", "#omnidan"], event.network); // join multiple channels
```

`part(channels, msg, network, fn)`

Param string/object/array channels The channels you want to kick from.

Param string msg The part message.

Param string network The network to execute the command on.

Param function fn The callback function to be called when the call has been finished.

Parts channels.

Example:

```
client.part("#caffeinery", "byeee", event.network); // usage in an event listener
client.part("#caffeinery", "byeee", "freenode"); // send to specific network
client.part(["#caffeinery", "#omnidan"], "byeee", event.network); // part multiple␣
↪channels
```

kick(channels, nicks, msg, network, fn)

Param array/object/string channels The channel(s) you want to kick from.

Param array/string nicks The nick(s) you want to kick.

Param string network The network to execute the command on.

Param function fn The callback function to be called when the call has been finished.

Kick user from a channel.

Example:

```
client.kick("#caffeinery", "omnidan", "bye!", event.network); // kick user from␣
↪specific channel
client.kick(event.channel, "omnidan", "bye!", event.network); // kick user from␣
↪current channel
client.kick(["#caffeinery", "#omnidan"], "omnidan", "bye!", event.network); // kick␣
↪user from multiple channels
client.kick("#caffeinery", ["omnidan", "np_coffea"], "bye!", event.network); // kick␣
↪multiple users from channel
```

names(channels, network, fn)

Param string/object/array channels The channel you want to get the nicknames from.

Param string network The network to execute the command on.

Param function fn The callback function to be called when the call has been finished.

Gets users from a channel.

Example:

```
client.names("#caffeinery", event.network, function (names) {
    console.log(names);
}); // usage in an event listener

client.names("#caffeinery", "freenode, function (names) {
    console.log(names);
}); // send to specific network

client.names(["#caffeinery", "#omnidan"], event.network, function (names) {
    console.log(names);
}); // get names from multiple channels
```

mode(target, flags, network, fn)

Param string target Target for the mode change, can be a user or channel.

Param string flags Flags of the mode change.

Param string network The network to execute the command on.

Param function fn The callback function to be called when the call has been finished.

Sets modes.

Note: You'll probably want to use the *umode* and *chanmode* functions instead of this function.

Example:

```
client.mode("omnidan", "+p", event.network); // set mode on specific user
client.mode(client.me, "+p", event.network); // set mode on current client
client.mode("#caffeinery", "+v omnidan", event.network); // voice a user in a␣
↪specific channel
client.mode(event.channel, "+v omnidan", event.network); // voice a user in the␣
↪current channel
```

umode(flags, network, fn)

Param string flags Flags of the mode change.

Param string network The network to execute the command on.

Param function fn The callback function to be called when the call has been finished.

Sets modes on the current client.

Example:

```
client.umode("+p", event.network); // set mode on current client
```

chanmode(channel, mode, target, network, fn)

Param string channel Channel to change modes on.

Param string mode Consists of +/- and a mode character, e.g. *+v*, *-o*

Param array/object/string target Target for the mode change, can be (a) user(s) or channel(s).

Param string network The network to execute the command on.

Param function fn The callback function to be called when the call has been finished.

Sets modes on a channel.

Note: There are helper functions that work like *client.chanmode*, but without having to specify the *mode*: voice, devoice, op, deop, hop, dehop

Example:

```
client.chanmode("#caffeinery", "+v", "omnidan", event.network); // voice a user in a␣
↪specific channel
client.chanmode(event.channel, "+v", "omnidan", event.network); // voice a user in␣
↪the current channel
client.chanmode(event.channel, "+v", ["omnidan", "np_coffea"], event.network); //␣
↪voice multiple users in the current channel

// or using helper functions...
client.voice("#caffeinery", "omnidan", event.network); // voice a user in a specific␣
↪channel
client.voice(event.channel, "omnidan", event.network); // voice a user in the current␣
↪channel
client.voice(event.channel, ["omnidan", "np_coffea"], event.network); // voice␣
↪multiple users in the current channel
```

4.1.5 message plugins

events

message

The message event, fired when a standard IRC message is received.

Event attributes:

- user - User who sent the message.
- channel - Channel this message was sent to.
- message - The actual message.
- isAction - true if this was an action (/me).

Example:

```
client.on('message', function (event) {
    console.log('[' + event.channel.getName() + '] ' + event.user.getNick() + ': ' +␣
↪event.message);
    event.reply('I logged to the console!'); // Says to the relevent user "I logged␣
↪to the console!", either in PM or the channel.
});
```

privatemessage

The privatemessage event, fired when an IRC private message is received. Like a message, but more private.

Event attributes:

- user - User who sent the message.
- message - The actual private message.
- isAction - true if this was an action (/me).

Example:

```
client.on('privatemessage', function (event) {
    console.log('[PM] ' + event.user.getNick() + ': ' + event.message);
    event.reply(':)'); // Says to the relevent user ":)", in PM
});
```

notice

The notice event, fired when an IRC notice is received. Like a message, but more private.

Event attributes:

- from - User who sent the notice.
- to - Where this notice was sent to.
- message - The actual notice message.

Example:

```
client.on('notice', function (event) {
    console.log('[' + event.to + '] ' + event.from.getNick() + ': ' + event.message);
});
```

functions

send(target, msg, network, fn)

> **Param object/string target** The `channel` or `user` object to send this message to.
>
> **Param string msg** The message you want to send.
>
> **Param string network** The network to execute the command on.
>
> **Param function fn** The callback function to be called when the call has been finished.

Send an IRC message to a channel or a user.

Example:

```
client.send("#caffeinery", "Hiii", event.network); // usage in an event listener
client.send("#caffeinery", "Hiii", "freenode"); // send to specific network

client.send(event.channel ? event.channel : event.user, "Hiii", event.network); //␣
↪reply to message
// ...or use the event.reply helper
event.reply("Hiii!"); // reply to message
```

action(target, msg, network, fn)

> **Param object target** The `channel` or `user` object to send this action to.
>
> **Param string msg** The action you want to send.
>
> **Param string network** The network to execute the command on.
>
> **Param function fn** The callback function to be called when the call has been finished.

Send an IRC action to a channel or a user. (This is the /me command) Works like `send(target, msg, network, fn)`.

Example:

```
client.action("#caffeinery", "Hiii", event.network); // usage in an event listener
client.action("#caffeinery", "Hiii", "freenode"); // send to specific network

client.action(event.channel ? event.channel : event.user, "Hiii", event.network); //␣
↪reply to message
// ...or use the event.replyAction helper
event.replyAction("Hiii!"); // reply to message
```

notice(target, msg, network, fn)

> **Param object target** The `channel` or `user` object to send this notice to.
>
> **Param string msg** The notice you want to send.
>
> **Param string network** The network to execute the command on.
>
> **Param function fn** The callback function to be called when the call has been finished.

Send an IRC notice to a channel or a user. Works like `send(target, msg, network, fn)`.

Example:

```
client.notice("#caffeinery", "Hiii", event.network); // usage in an event listener
client.notice("#caffeinery", "Hiii", "freenode"); // send to specific network

```

(continues on next page)

(continued from previous page)

```
client.notice(event.channel ? event.channel : event.user, "Hiii", event.network); //␣
↪reply to message
// ...or use the event.replyNotice helper
event.replyNotice("Hiii!"); // reply to message
```

4.1.6 format plugin

formatting

The format plugin is a bit special. You can use various formattings by using the *client.format.get()* function.

```
event.reply(client.format.get('red') + 'Roses are red.' + client.format.get('blue') +
↪'Violets are blue.' + client.format.get('reset') + 'And ZWSP' + client.format.get(
↪'zwsp') + ' is invisible.');
```

The following colors and formatting options are available: https://github.com/caffeinery/coffea/blob/master/lib/plugins/format.js#L2-L26

emoji

We expose `require('node-emoji').emoji` via `client.emoji`. See the node-emoji documentation for more information: https://github.com/omnidan/node-emoji

kaomoji

We expose `require('node-kaomoji').kaomoji` via `client.kaomoji`. See the node-kaomoji documentation for more information: https://github.com/omnidan/node-kaomoji

functions

get(formatting)

Param string formatting The formatting to be inserted.

Returns string formatting Format characters that format the message.

The following colors and formatting options are available: https://github.com/caffeinery/coffea/blob/master/lib/plugins/format.js#L2-L26

Example:

```
client.on('message', function (event) {
    event.reply(client.format.get('green') + '>' + event.message); // convert message␣
↪to greentext
});
```

unhighlight(message)

Param string message The message to be unhighlighted.

Returns string message Unhighlighted message (does not highlight users)

Adds ZWSP to a `message` to make sure it doesn't highlight anyone.

4.1.7 motd plugin

events

`motd`

The `motd` event, fired when the end of the MOTD (Message Of The Day) is received from the server.

Event attributes:

- `motd`: The actual MOTD sent by the server.

Example:

```
client.on('motd', function (event) {
    console.log("Connected to " + event.network + ". MOTD: " + event.motd);
    client.join('##test'); // autojoins a channel when properly connected
});
```

4.1.8 welcome plugin

events

`welcome`

The `welcome` event, fired when the `RPL_WELCOME` command is received from the server. When this is fired `client.me.nick` gets set correctly and `client.welcomed` gets set to `true`.

Event attributes:

- `nick`: Your nick in the welcome message received from the server.
- `message`: The rest of the welcome message.

```
client.on('welcome', function (event) {
    console.log("Welcome " + event.nick + ": " + event.message);
});
```

4.1.9 server plugins

events

`quit`

The `quit` event, fired when the `QUIT` command is received from the server. When this is fired someone has quit from the irc network.

Event attributes:

- `user`: The affected user.
- `message`: The quit message.

```
client.on('quit', function (event) {
    console.log(event.user.getNick() + " quit: " + event.message);
});
```

functions

quit(message, network)

> **Param string message** The reason you want to show to others about why you quit.
>
> **Param string network** The network to execute the command on.
>
> **Param function fn** The callback function to be called when the call has been finished.

Quit from the server.

Example:

```
client.quit("bye", event.network); // usage in an event listener
client.quit("bye", "freenode"); // send to specific network
client.quit("bye"); // send to all networks
```

getServerInfo(network)

> **Param string network** The network to execute the command on.
>
> **Returns object serverInfo** The server info about the network.

Get info about a network.

Example:

```
var info = client.getServerInfo(event.network); // usage in an event listener
var info = client.getServerInfo("freenode"); // send to specific network
```

4.1.10 cap plugins

> **Warning:** This part of the documentation is a work in progress and might be incomplete.

events

cap_list

cap_ack

cap_nck

Fired when `CAP` is received from the server. (More information available at https://github.com/ircv3/ircv3-specifications/blob/master/specification/capability-negotiation-3.1)

Event attributes:

- `capabilities`: The list of capabilities

extended-join

The `extended-join` event, fired when the client has the `extended-join` capability, and a user has joined a channel.

Event attributes:

- `channel`: The channel the user joined.
- `user`: The nick of the user.

- `account`: The host of the user.
- `realname`: The realname of the user.

4.1.11 away plugin

events

`away`

The `away` event, fired when an IRC AWAY event is received.

Event attributes:

- `user` - A User object of the user that changed away status.
- `message` - The message of the AWAY user's AWAY status (away message).

Example:

```
client.on('away', function (event) {
    console.log(event.user.getNick() + ' is now away: ' + event.message);
});
```

functions

`away(reason, network, fn)`

Param string reason The reason to be away (away message).

Param string network The network to execute the command on.

Param function fn The callback function to be called when the call has been finished.

Sets the client as away on `network` with an away message (`reason`).

Example:

```
client.away("busy", event.network); // usage in an event listener
client.away("busy", "freenode"); // send to specific network
client.away("busy"); // send to all networks
```

4.1.12 ping plugin

events

`pong`

The `pong` event, fired when a `PONG` event was received from the server.

Event attributes:

- None.

Example:

```
client.on('pong', function (event) {
    console.log("PONG");
});
```

4.1.13 other plugins

events

`error`

The `error` event, fired when any error happens.

Make sure to listen to `error` events to see possible errors/warnings:

```
client.on('error', function (err, event) {
    console.log(event.name, err.stack);
});
```

You can also add the `err` parameter to any event listener (change from `function (event)` to `function (err, event)`):

```
client.on('whois', function (err, event) {
    if (err) console.log("WHOIS ERROR:", err.stack);
    console.log("whois:", event);
});
```

`errors`

The `errors` event, fired when IRC errors are received. List of possible errors: https://github.com/williamwicks/irc-replies/blob/master/replies.json#L113-L170

Event attributes:

- None

Example:

```
client.on('errors', function (err, event) {
    console.error('IRC Error:', err);
});
```

functions

`pass(pass, network, fn)`

Param string pass The password

Param string network The network to execute the command on.

Param function fn The callback function to be called when the call has been finished.

Sends an IRC PASS command to the network with the specified password (`pass`)

Example:

```
client.on('motd', function (event) {
    client.pass('pa$$w0rd', event.network);
});
```

`user(username, realname, network, fn)`

Param string username The username

Param string realname The realname

Param string network The network to execute the command on.

Param function fn The callback function to be called when the call has been finished.

Sends an IRC USER command to the network with the specified username (`username`) and realname (`realname`).

Example:

```
client.on('motd', function (event) {
    client.user('username', 'Real Name', event.network);
});
```

`oper(name, password, network, fn)`

Param string name The oper name

Param string password The oper password

Param string network The network to execute the command on.

Param function fn The callback function to be called when the call has been finished.

Sends an IRC OPER command and tries to oper to the network with the specified name (`name`) and password (`password`).

Example:

```
client.on('motd', function (event) {
    client.oper('opername', 'pa$$w0rd', event.network);
});
```

Index