node-irc Documentation

Release 2.1

Martyn Smith

Apr 16, 2017

Contents

CHAPTER 1

More detailed docs:

API

This library provides IRC client functionality

Client

irc.**Client**(*server*, *nick*[, *options*])

This object is the base of everything, it represents a single nick connected to a single IRC server.

The first two arguments are the server to connect to, and the nickname to attempt to use. The third optional argument is an options object with default values:

```
{
    userName: 'nodebot',
    realName: 'nodeJS IRC client',
    port: 6667,
    localAddress: null,
    debug: false,
    showErrors: false,
    autoRejoin: false,
    autoConnect: true,
    channels: [],
    secure: false,
    selfSigned: false,
    certExpired: false,
    floodProtection: false,
    floodProtectionDelay: 1000,
    sasl: false,
    retryCount: 0,
    retryDelay: 2000,
    stripColors: false,
    channelPrefixes: "&#",
```

```
    messageSplit: 512,
    encoding: ''
}
```

secure (SSL connection) can be a true value or an object (the kind of object returned from *crypto.createCredentials()*) specifying cert etc for validation. If you set *selfSigned* to true SSL accepts certificates from a non trusted CA. If you set *certExpired* to true, the bot connects even if the ssl cert has expired.

localAddress is the address to bind to when connecting.

floodProtection queues all your messages and slowly unpacks it to make sure that we won't get kicked out because for Excess Flood. You can also use *Client.activateFloodProtection()* to activate flood protection after instantiating the client.

floodProtectionDelay sets the amount of time that the client will wait between sending subsequent messages when *floodProtection* is enabled.

Set *sasl* to true to enable SASL support. You'll also want to set *nick*, *userName*, and *password* for authentication.

stripColors removes mirc colors (0x03 followed by one or two ascii numbers for foreground,background) and ircII "effect" codes (0x02 bold, 0x1f underline, 0x16 reverse, 0x0f reset) from the entire message before parsing it and passing it along.

messageSplit will split up large messages sent with the *say* method into multiple messages of length fewer than *messageSplit* characters.

With *encoding* you can set IRC bot to convert all messages to specified character set. If you don't want to use this just leave value blank or false. Example values are UTF-8, ISO-8859-15, etc.

Setting *debug* to true will emit timestamped messages to the console using *util.log* when certain events are fired.

autoRejoin has the client rejoin channels after being kicked.

Setting *autoConnect* to false prevents the Client from connecting on instantiation. You will need to call *connect()* on the client instance:

```
var client = new irc.Client({ autoConnect: false, ... });
client.connect();
```

retryCount is the number of times the client will try to automatically reconnect when disconnected. It defaults to 0.

retryDelay is the number of milliseconds to wait before retying to automatically reconnect when disconnected. It defaults to 2000.

`Client.`**`send`**(*command*, *arg1*, *arg2*, ...)

Sends a raw message to the server; generally speaking, it's best not to use this method unless you know what you're doing. Instead, use one of the methods below.

`Client.`**`join`**(*channel*, *callback*)

Joins the specified channel.

Arguments

- **`channel`** (*`string`*) – Channel to join
- **`callback`** (*`function`*) – Callback to automatically subscribed to the *join#channel* event, but removed after the first invocation. *channel* supports multiple JOIN arguments as a space separated string (similar to the IRC protocol).

`Client.`**`part`**(*channel*[, *message*], *callback*)

Parts the specified channel.

Arguments

- **channel** (*string*) – Channel to part
- **message** (*string*) – Optional message to send upon leaving the channel
- **callback** (*function*) – Callback to automatically subscribed to the *part#channel* event, but removed after the first invocation.

Client.**say**(*target*, *message*)
Sends a message to the specified target.

Arguments

- **target** (*string*) – is either a nickname, or a channel.
- **message** (*string*) – the message to send to the target.

Client.**ctcp**(*target*, *type*, *text*)
Sends a CTCP message to the specified target.

Arguments

- **target** (*string*) – is either a nickname, or a channel.
- **type** (*string*) – the type of the CTCP message. Specify "privmsg" for a PRIVMSG, and anything else for a NOTICE.
- **text** (*string*) – the CTCP message to send.

Client.**action**(*target*, *message*)
Sends an action to the specified target.

Client.**notice**(*target*, *message*)
Sends a notice to the specified target.

Arguments

- **target** (*string*) – is either a nickname, or a channel.
- **message** (*string*) – the message to send as a notice to the target.

Client.**whois**(*nick*, *callback*)
Request a whois for the specified *nick*.

Arguments

- **nick** (*string*) – is a nickname
- **callback** (*function*) – Callback to fire when the server has finished generating the whois information and is passed exactly the same information as a *whois* event described above.

Client.**list**([*arg1*, *arg2*, ...])
Request a channel listing from the server. The arguments for this method are fairly server specific, this method just passes them through exactly as specified.

Responses from the server are available via the *channellist_start*, *channellist_item*, and *channellist* events.

Client.**connect**([*retryCount*[, *callback*]])
Connects to the server. Used when *autoConnect* in the options is set to false. If *retryCount* is a function it will be treated as the *callback* (i.e. both arguments to this function are optional).

param integer retryCount Optional number of times to attempt reconnection

param function callback Optional callback

`Client.`**`disconnect`**`(`[*message*[, *callback*]]`)`

Disconnects from the IRC server. If *message* is a function it will be treated as the *callback* (i.e. both arguments to this function are optional).

Arguments

- **`message`** (*`string`*) – Optional message to send when disconnecting.
- **`callback`** (*`function`*) – Optional callback

`Client.`**`activateFloodProtection`**`(`[*interval*]`)`

Activates flood protection "after the fact". You can also use *floodProtection* while instantiating the Client to enable flood protection, and *floodProtectionDelay* to set the default message interval.

Arguments

- **`interval`** (*`integer`*) – Optional configuration for amount of time to wait between messages. Takes value from client configuration if unspecified.

Events

irc.Client instances are EventEmitters with the following events:

`'registered'`

function (message) { }

Emitted when the server sends the initial 001 line, indicating you've connected to the server. See the *raw* event for details on the *message* object.

`'motd'`

function (motd) { }

Emitted when the server sends the message of the day to clients.

`'names'`

function (channel, nicks) { }

Emitted when the server sends a list of nicks for a channel (which happens immediately after joining and on request. The nicks object passed to the callback is keyed by nick names, and has values '', '+', or '@' depending on the level of that nick in the channel.

`'names#channel'`

function (nicks) { }

As per 'names' event but only emits for the subscribed channel.

`'topic'`

function (channel, topic, nick, message) { }

Emitted when the server sends the channel topic on joining a channel, or when a user changes the topic on a channel. See the *raw* event for details on the *message* object.

`'join'`

function (channel, nick, message) { }

Emitted when a user joins a channel (including when the client itself joins a channel). See the *raw* event for details on the *message* object.

`'join#channel'`

function (nick, message) { }

As per 'join' event but only emits for the subscribed channel. See the *raw* event for details on the *message* object.

'part'
function (channel, nick, reason, message) { }

Emitted when a user parts a channel (including when the client itself parts a channel). See the *raw* event for details on the *message* object.

'part#channel'
function (nick, reason, message) { }

As per 'part' event but only emits for the subscribed channel. See the *raw* event for details on the *message* object.

'quit'
function (nick, reason, channels, message) { }

Emitted when a user disconnects from the IRC, leaving the specified array of channels. See the *raw* event for details on the *message* object.

'kick'
function (channel, nick, by, reason, message) { }

Emitted when a user is kicked from a channel. See the *raw* event for details on the *message* object.

'kick#channel'
function (nick, by, reason, message) { }

As per 'kick' event but only emits for the subscribed channel. See the *raw* event for details on the *message* object.

'kill'
function (nick, reason, channels, message) { }

Emitted when a user is killed from the IRC server. *channels* is an array of channels the killed user was in which are known to the client. See the *raw* event for details on the *message* object.

'message'
function (nick, to, text, message) { }

Emitted when a message is sent. *to* can be either a nick (which is most likely this clients nick and means a private message), or a channel (which means a message to that channel). See the *raw* event for details on the *message* object.

'message#'
function (nick, to, text, message) { }

Emitted when a message is sent to any channel (i.e. exactly the same as the *message* event but excluding private messages. See the *raw* event for details on the *message* object.

'message#channel'
function (nick, text, message) { }

As per 'message' event but only emits for the subscribed channel. See the *raw* event for details on the *message* object.

'selfMessage'
function (to, text) { }

Emitted when a message is sent from the client. *to* is who the message was sent to. It can be either a nick (which most likely means a private message), or a channel (which means a message to that channel).

'notice'
function (nick, to, text, message) { }

Emitted when a notice is sent. *to* can be either a nick (which is most likely this clients nick and means a private message), or a channel (which means a message to that channel). *nick* is either the senders nick or *null* which means that the notice comes from the server. See the *raw* event for details on the *message* object.

'ping'
function (server) { }

Emitted when a server PINGs the client. The client will automatically send a PONG request just before this is emitted.

'pm'
function (nick, text, message) { }

As per 'message' event but only emits when the message is direct to the client. See the *raw* event for details on the *message* object.

'ctcp'
function (from, to, text, type, message) { }

Emitted when a CTCP notice or privmsg was received (*type* is either *'notice'* or *'privmsg'*). See the *raw* event for details on the *message* object.

'ctcp-notice'
function (from, to, text, message) { }

Emitted when a CTCP notice was received. See the *raw* event for details on the *message* object.

'ctcp-privmsg'
function (from, to, text, message) { }

Emitted when a CTCP privmsg was received. See the *raw* event for details on the *message* object.

'ctcp-version'
function (from, to, message) { }

Emitted when a CTCP VERSION request was received. See the *raw* event for details on the *message* object.

'nick'
function (oldnick, newnick, channels, message) { }

Emitted when a user changes nick along with the channels the user is in. See the *raw* event for details on the *message* object.

'invite'
function (channel, from, message) { }

Emitted when the client receives an */invite*. See the *raw* event for details on the *message* object.

'+mode'
function (channel, by, mode, argument, message) { }

Emitted when a mode is added to a user or channel. *channel* is the channel which the mode is being set on/in. *by* is the user setting the mode. *mode* is the single character mode identifier. If the mode is being set on a user, *argument* is the nick of the user. If the mode is being set on a channel, *argument* is the argument to the mode. If a channel mode doesn't have any arguments, *argument* will be 'undefined'. See the *raw* event for details on the *message* object.

'-mode'
function (channel, by, mode, argument, message) { }

Emitted when a mode is removed from a user or channel. *channel* is the channel which the mode is being set on/in. *by* is the user setting the mode. *mode* is the single character mode identifier. If the mode is being set on a user, *argument* is the nick of the user. If the mode is being set on a channel,

argument is the argument to the mode. If a channel mode doesn't have any arguments, *argument* will be 'undefined'. See the *raw* event for details on the *message* object.

'whois'
function (info) { }

Emitted whenever the server finishes outputting a WHOIS response. The information should look something like:

```
{
    nick: "Ned",
    user: "martyn",
    host: "10.0.0.18",
    realname: "Unknown",
    channels: ["@#purpledishwashers", "#blah", "#mmmmbacon"],
    server: "*.dollyfish.net.nz",
    serverinfo: "The Dollyfish Underworld",
    operator: "is an IRC Operator"
}
```

'channellist_start'
function () {}

Emitted whenever the server starts a new channel listing

'channellist_item'
function (channel_info) {}

Emitted for each channel the server returns. The channel_info object contains keys 'name', 'users' (number of users on the channel), and 'topic'.

'channellist'
function (channel_list) {}

Emitted when the server has finished returning a channel list. The channel_list array is simply a list of the objects that were returned in the intervening *channellist_item* events.

This data is also available via the Client.channellist property after this event has fired.

'raw'
function (message) { }

Emitted when ever the client receives a "message" from the server. A message is basically a single line of data from the server, but the parameter to the callback has already been parsed and contains:

```
message = {
    prefix: "The prefix for the message (optional)",
    nick: "The nickname portion of the prefix (optional)",
    user: "The username portion of the prefix (optional)",
    host: "The hostname portion of the prefix (optional)",
    server: "The servername (if the prefix was a servername)",
    rawCommand: "The command exactly as sent from the server",
    command: "Human readable version of the command",
    commandType: "normal, error, or reply",
    args: ['arguments', 'to', 'the', 'command'],
}
```

You can read more about the IRC protocol by reading RFC 1459

'error'
function (message) { }

Emitted when ever the server responds with an error-type message. The message parameter is exactly as in the 'raw' event.

'action'
function (from, to, text, message) { }

Emitted whenever a user performs an action (e.g. */me waves*). The message parameter is exactly as in the 'raw' event.

Colors

irc.colors.**wrap**(*color*, *text*[, *reset_color*])
Takes a color by name, text, and optionally what color to return.

Arguments

- **color** (*string*) – the name of the color as a string
- **text** (*string*) – the text you want colorized
- **reset_color** (*string*) – the name of the color you want set after the text (defaults to 'reset')

irc.colors.**codes**
This contains the set of colors available and a function to wrap text in a color.

The following color choices are available:

{ white: 'u000300', black: 'u000301', dark_blue: 'u000302', dark_green: 'u000303', light_red: 'u000304', dark_red: 'u000305', magenta: 'u000306', orange: 'u000307', yellow: 'u000308', light_green: 'u000309', cyan: 'u000310', light_cyan: 'u000311', light_blue: 'u000312', light_magenta: 'u000313', gray: 'u000314', light_gray: 'u000315', reset: 'u000f',

}

Internal

Client.**conn**
Socket to the server. Rarely, if ever needed. Use *Client.send* instead.

Client.**chans**
Channels joined. Includes channel modes, user list, and topic information. Only updated *after* the server recognizes the join.

Client.**nick**
The current nick of the client. Updated if the nick changes (e.g. nick collision when connecting to a server).

client.**_whoisData**()
Buffer of whois data as whois is sent over multiple lines.

client.**_addWhoisData**()
Self-explanatory.

client.**_clearWhoisData**()
Self-explanatory.

Index

Symbols

C

I