node-irc-dcc Documentation

Release 0.2.1

Alexander Walters

Apr 27, 2018

Contents

CHAPTER 1

API

irc-dcc extends irc with Direct Client-to-Client support. Currently, sending and receiving files, as well as chat sessions are supported.

1.1 DCC

DCC (*client*[, *options*])

client is the instance of *irc.Client* that you want to enable DCC support on. The second optional argument is an object, that looks something like the following:

```
{
    ports: [2000, 2020],
    localAddress: '10.0.0.125',
    timeout: 30000
}
```

`ports` is a length-2 array. The first and second objects in that array should be the start and end of a port range that you want incoming DCC connections to bind to. If undefined or null, the default of `0` is used (the OS picks a port).

`localAddress` is the source address for all outgoing connections and the IP to which all listening connections are bound. If unset, an attempt will be made to get this information off the client object. If that is unset or undefined, a local ip address will be picked by the library.

`timeout` is the idle time an uninitiated DCC connection will wait before erring out. It is set in milliseconds.

DCC.**sendFile** (*to*, *filename*, *length*, *callback*)

`DCC.sendFile` does not do any disk IO – it is your responsibility to open the file, and send the data. The library takes care of the CTCP messaging and establishing connections.

Arguments

- **to** (*string*) – The nick to send the file to
- **filename** (*string*) – The filename to send in the CTCP message

- **length** (*number*) – The length of the file in bytes
- **callback** –

  DCC. (*err*, *connection*, *position*)

  Arguments
  - **err** – Error, if there is an error, null otherwise
  - **connection** – The *net.Socket* object
  - **position** (*number*) – The offset from the start of the file to begin reading from

```
// A minimal example.
// Adjust the arguments to irc.Client as per the node-irc docs
//
// When your client connects to IRC, issue /ctcp <botnick> send
//
const fs = require("fs");
const irc = require("irc");
const DCC = require("irc-dcc");

client = new irc.Client(...);
dcc = new DCC(client);

client.addListener('ctcp-privmsg', (from, to, text, message) => {
    if (text.split(" ")[0].toLowerCase() == "send") {
        fs.stat(__dirname + '/data.txt', (err, filestat) => {
            if (err) {
                client.notice(from, err);
                return;
            }
            dcc.sendFile(from, 'data.txt', filestat.size,
                         (err, con, position) => {
                if (err) {
                    client.notice(from, err);
                    return;
                }
                rs = fs.createReadStream(__dirname + '/data.txt', {
                    start: position
                });
                rs.pipe(con);
            });
        });
    }
});
```

DCC.**acceptFile** (*from*, *host*, *port*, *filename*, *length*[, *position*], *callback*)

DCC.acceptFile does not do any disk IO – it is your responsibility to open the file, and send the data. The library takes care of the CTCP messaging and establishing connections.

Arguments

- **from** (*string*) – The nick sending the file
- **host** (*string*) – The IP address to connect to
- **port** (*number*) – The port to connect to
- **filename** (*string*) – The filename suggested by the other side
- **length** (*number*) – The length of the file in bytes
- **position** (*number*) – The offset from the beginning of the file, if you wish to resume

- **callback** –

  DCC. (*err*, *filename*, *connection*)

  Arguments

  - **err** – Error, if there is an error, null otherwise
  - **filename** (*string*) – Name of the file
  - **connection** – The *net.Socket* object

```
// A minimal example.
// Adjust the arguments to irc.Client as per the node-irc docs
//
// When your client connects to IRC, send it a file.
//
const fs = require("fs");
const irc = require("irc");
const DCC = require("irc-dcc");

client = new irc.Client(...);
dcc = new DCC(client);

client.on('dcc-send', (from, args, message) => {
    var ws = fs.createWriteStream(__dirname + "/" + args.filename)
    dcc.acceptFile(from, args.host, args.port, args.filename,
                   args.length, (err, filename, con) => {
        if (err) {
            client.notice(from, err);
            return;
        }
        con.pipe(ws);
    });
});
```

DCC.**sendChat** (*to*, *callback*)

Arguments

- **to** (*string*) – The nick to open a chat session to
- **calback** –

  DCC. (*err*, *chat*)

  Arguments

  - **err** – Error, if there is an error, null otherwise
  - **chat** (*Chat*) – The chat connection object

```
// A minimal example.
// Adjust the arguments to irc.Client as per the node-irc docs
//
// When your client connects to IRC, issue /ctcp <botnick> chat
//
const fs = require("fs");
const irc = require("irc");
const DCC = require("irc-dcc");

client = new irc.Client(...);
dcc = new DCC(client);

client.addListener('ctcp-privmsg', (from, to, text, message) => {
    if (text.split(" ")[0].toLowerCase() == "chat") {
        dcc.sendChat(from, (err, chat) => {
            chat.on("line", (err, chat) => {
                chat.say("You said: " + line);
            });
        });
    }
});
```

`DCC.`**`acceptChat`** (*host*, *port*, *callback*)

Arguments

- **`host`** (`string`) – The IP address to connect to
- **`port`** (`number`) – The port to connect to
- **`callback`** –

  `DCC.` (*err*, *chat*)

  Arguments

  - **`err`** – Error, if there is an error, null otherwise
  - **`chat`** (`Chat`) – The chat connection object

```
// A minimal example.
// Adjust the arguments to irc.Client as per the node-irc docs
//
// When your client connects to IRC, initiate a DCC chat with the bot
//
const fs = require("fs");
const irc = require("irc");
const DCC = require("irc-dcc");

client = new irc.Client(...);
dcc = new DCC(client);

client.on('dcc-chat', (from, args, message) => {
    dcc.acceptChat(args.host, args.port, (err, chat) => {
        chat.on("line", (err, chat) => {
            chat.say("You said: " + line);
        });
    });
});
```

1.1.1 Events

`irc-dcc` emits four new events from `irc.Client`. Two events are intended for public use, and two are internal. All four of the events are in the form of `function (from, args, message) {}`. See the `irc` documentation for the details of `message`. `args` is an object of the parsed CTCP message, and is described for each of the public events.

`'dcc-send'`

```
{
    type: "send",
    filename: <string>,  // The filename
    long: <number>,      // IP address to connect to as a long integer
    host: <string>,      // IP address to connect to as a string
    port: <number>,      // Port to connect to
    length: <number>,    // Length of file, in bytes
}
```

'dcc-chat'

```
{
    type: "chat",
    long: <number>,       // IP address to connect to as a long integer
    host: <string>,       // IP address to connect to as a string
    port: <number>,       // Port to connect to
}
```

1.2 Chat

The library provides a very basic type for interacting with DCC chat sessions, with two public method, and one event. They are both stupendously straight forward.

`Chat.`**`say`**`(`*message*`)`

Arguments

- **`message`** (*`string`*) – Message to send

`Chat.`**`disconnect`**`()`

Ends the chat session.

1.2.1 Events

'line'

This is in the format of `function (line)`, and is simply the raw line of text from the connection.

Index