

---

# **discord.js Documentation**

***Release 8.2.0***

**hydrabolt**

**Nov 27, 2017**







# CHAPTER 1

---

## Other Version Notice

---

The docs hosted here are for the discord.js v8.2 branch otherwise known as discordv8. There is a different discord.js v11 branch and if you are on a discord server for discord.js that is probably the one you are using.

**Danger:** This documentation is for the discord.js 8.2 branch called discordv8 maintained by macdja38. You're **probably** looking for documentation on version 9 or higher, which is available on [the other docs site](#).

discordv8 is an easy-to-use and intuitive JavaScript API for [Discord](#). It's fairly high level, so if you're looking for something low level, check out [discord.io](#). If you are looking for something fast check out [eris](#)

if you're having problems, check out the [troubleshooting guide](#)

Feel free to make any contributions you want, whether it be through creating an issue, giving a suggestion or making a pull request!

---

**Note:** This documentation is still a work-in-progress, apologies if something isn't yet documented!

---

**Danger:** This documentation is for the discord.js 8.2 branch called discordv8 maintained by macdja38. You're **probably** looking for documentation on version 9 or higher, which is available on [the other docs site](#).

## 1.1 Installing discordv8

To install discordv8, you need a few dependencies.

**Warning:** When installing with any of these methods, you'll encounter some errors. This is because an optional dependency isn't working properly, but discordv8 should still work fine.

### 1.1.1 Windows

---

- You need [Visual Studio](#) and [Python 2.7](#).

Your Visual Studio installation ideally has to be recent, but you can try installing without it first. You can use **Express**, **Community**, **Enterprise** or any others apart from VS Code.

- You (obviously) need [NodeJS](#). Node 4 or higher is recommended.

After you have installed these things, to install just run: `npm install --save --msvs_version=2015 discordv8` to install the latest version of discordv8 for your project.

#### Additional Audio Support

- Install [ffmpeg](#) and add it to your PATH.
- 

### 1.1.2 Linux (Debian-based)

---

- You (obviously) need [NodeJS Linux](#). Node 4 or higher is recommended.

```
$ sudo apt-get install build-essential
$ npm install --save discordv8
```

#### Additional Audio Support

```
$ sudo apt-get install ffmpeg
```

Note: Ubuntu 14.04 needs to do:

```
$ sudo add-apt-repository ppa:mc3man/trusty-media && sudo apt-get update && sudo apt-
↳ get install ffmpeg
```

**Danger:** This documentation is for the discord.js 8.2 branch called discordv8 maintained by macdja38. You're **probably** looking for documentation on version 9 or higher, which is available on [the other docs site](#).

## 1.2 Updating to v5.0.0

If you're coming from versions below v5, you might find some changes. Here are the major changes:

### 1.2.1 Change 1

```
// OLD:

client.getUser();
client.getServer();
server.getMember(); // etc etc

// NEW:

client.users.get();
client.servers.get();
client.members.get();
```

### 1.2.2 Change 2

```
// OLD:

client.on("serverNewMember", (member, server) => {

});

// NEW:

client.on("serverNewMember", (server, user) => {

});
```

### 1.2.3 Change 3

The Member Class has been removed, meaning you can't use `member.permissionsIn(channel)`. To get permissions, use `channel.permissionsOf(user)`.

**Danger:** This documentation is for the discord.js 8.2 branch called discordv8 maintained by macdja38. You're **probably** looking for documentation on version 9 or higher, which is available on [the other docs site](#).

## 1.3 Troubleshooting

### 1.3.1 General

Occasionally, the API can stop working for whatever reason. If it was working previously and it stopped working on the same version, it means that either we screwed code up or there's been a change to the Discord API. You can try asking around in the **'discord.js channel in the API server'**. You could also [make an issue](#) if one relating to a similar issue doesn't exist. Please post a stacktrace if there is one, and be as detailed as possible - *"the API isn't working"* doesn't help at all.

If there is already an issue, feel free to comment that you're also experiencing the same thing. This helps to see how widespread the bug is.

You can try reconnecting before submitting an issue, as sometimes some of the servers may be slightly different.

If you're your bot or client is exiting unexpectedly with no error, this is likely caused by websocket disconnects. Make sure you have `autoReconnect` enabled. See [Client](#).

### 1.3.2 Voice

Often, especially if you're on Windows, voice will not work out of the box. Follow the steps below, one by one.

- **Is your system supported? The following are:**
  - Linux x64 & ia32
  - Linux ARM (Raspberry Pi 1 & 2)
  - Mac OS X x64
  - Windows x64
- **Did you install Python 2.7.x correctly? Is it in your PATH? `python -V`. If not, install it correctly or try reinstalling.**
  - **Windows** - See <https://python.org/downloads/>
  - **Linux / Mac OS** - Unix systems should already have it installed, but if not, use the OS's package manager
- **Did you install FFMPEG correctly? Is it in your PATH? `ffmpeg -version`. If not, install it correctly or try reinstalling**
  - **Windows** - [Follow this guide](#)
  - **Linux / Mac OS** - Use your OS's package manager
- **Did you install the required C++ compiler tool for your OS? If not, install the corresponding program, then try reinstalling**
  - **Windows** - [Visual Studio 2015](#) with C++ Support enabled
  - **Linux** - [build-essential](#)
  - **Mac OS** - [Xcode CLI tools](#)

If you're still having problems try

- `npm cache clean`
- `npm config set msvs_version 2015`
- `npm i -S macdja38/discord.js`

If nothing of the above helped, feel free to jump on the [‘discord.js channel in the API server’](#)

**Danger:** This documentation is for the discord.js 8.2 branch called discordv8 maintained by macdja38. You're **probably** looking for documentation on version 9 or higher, which is available on [the other docs site](#).

## 1.4 Usage Examples

Not all of these are standalone examples, many of them are usage examples. If you're a beginner to discordv8, we encourage you to look through these examples to get a hang of the way things work using the library.



**Warning:** Please do not copy/paste code directly from these examples. Try to learn from and adapt these pieces of code to your specific situation.

**Note:** We use [Template Literals](#) in these examples. These are an ES6 feature and may not be fully supported in your environment. In this case, it is safe to use other methods of concatenating strings.

## 1.4.1 Logging In

Logs the [Client](#) in, allowing you to begin working with the Discord API.

### Logging in with a username and password

**Do not use a normal user account for large or public bots. This is considered abuse of the API and can land you in trouble.**

```
const Discord = require('discordv8');
var client = new Discord.Client();

client.login('mybot@example.com', 'password', output);

function output(error, token) {
  if (error) {
    console.log(`There was an error logging in: ${error}`);
    return;
  } else {
    console.log(`Logged in. Token: ${token}`);
  }
}
```

### Logging in with a token

You can get your bot's token using the [My Applications](#) page on the Discord Developers site.

```
const Discord = require('discordv8');
var client = new Discord.Client();

client.loginWithToken('token', output);

function output(error, token) {
  if (error) {
    console.log(`There was an error logging in: ${error}`);
    return;
  } else {
    console.log(`Logged in. Token: ${token}`);
  }
}
```

### 1.4.2 Logging Out

The `logout` function should be used if you intend to reconnect with the same process. The function takes one parameter, which is a callback.

```
client.logout((err) => {
  console.log(err);
});
```

However, if you want to completely shut down your application, use `destroy`.

```
client.destroy((err) => {
  console.log(err);
});
```

---

### 1.4.3 Receiving Messages

Here we will demonstrate receiving messages and logging them to the console.

```
client.on('message', function(message) {
  if (message.channel.isPrivate) {
    console.log(`(Private) ${message.author.name}: ${message.content}`);
  } else {
    console.log(`(${message.server.name} / ${message.channel.name}) $
↪ ${message.author.name}: ${message.content}`);
  }
});
```

---

### 1.4.4 Sending messages

#### In the same channel

This is by far the most common way people will send a message in the Discord API. Here we will send a message to the same [Channel](#) we received a message from in the above example.

```
client.on('message', function(message) {
  // Don't forget to log the message!
  client.sendMessage(message.channel, "Hello!");
});
```

You can also use a [Message](#) resolvable as an parameter. This example does the same thing as above.

```
client.on('message', function(message) {
  client.sendMessage(message, "Hello!");
});
```

You can also directly reply to messages. This does the same as adding an `@mention` in front of your text.

Sends “@author Hello!”

```
client.on('message', function(message) {
  client.reply(message, "Hello!");
});
```

### To a specific server and channel

Sends “Hello” to the default [Channel](#) in the [Server](#) “My Server”. Note that this does not require any sort of received message to be activated, however if there are multiple servers with the name “My Server”, a random one will be chosen.

See [Cache](#) for more details on getting specific objects and resolvables.

```
var channel = client.servers.get("name", "My Server").defaultChannel;
client.sendMessage(channel, "Hello!");
```

### Private Messages

You can also send private messages to a user with a [User](#) object. This will send “Hello!” as a private message to the original author of the received message.

Do note however, that a [PMChannel](#) is not the same as a [ServerChannel](#) and therefore does not have the same properties such as server and name.

```
client.on('message', function(message) {
  client.sendMessage(message.author, "Hello!");
});
```

## 1.4.5 Sending files

The process of sending files is similar to how you send messages.

The first parameter takes an [Channel Resolvable](#) or [User Resolvable](#). The User Resolvable will send the file as a DM, and the Channel Resolvable will send the file to a text channel.

The next parameter is a [File Resolvable](#).

The third parameter lets you name your file. This is optional.

The fourth parameter lets you add a message. This is optional.

The last parameter is a callback. It takes an error and a [Message](#) object.

### URL

```
client.on('message', function(message) {
  client.sendFile(message, 'http://i.imgur.com/6CbxaPc.jpg', 'kappa.jpg',
    'Check out this cool file!', (err, m) => {
      if (err) console.log(err);
    });
});
```

## Local file

```
client.on('message', function(message) {
  client.sendFile(message, '/assets/dank_meme.jpg', 'dank_meme.jpg', 'Check out_
↪this cool file!', (err, m) => {
    if (err) console.log(err);
  });
});
```

## Buffer

Send data from streams.

```
const fs = require('fs');

client.on('message', function(message) {
  var stream = fs.createReadStream('/assets/dank_meme.jpg');
  var chunks = [];
  stream.on('data', (dataChunk) => {
    chunks.push(dataChunk);
  });

  stream.on('end' () => {
    client.sendFile(message, Buffer.concat(chunks), 'dank_meme.jpg',
↪'Check out this cool file!');
  });
});
```

---

### 1.4.6 Deleting messages

The `deleteMessage` function takes an [Message Resolvable](#) as the first parameter. The second parameter is a callback. This snippet will delete the received message.

```
client.on('message', function(message) {
  client.deleteMessage(message);
});
```

You can also delete multiple messages with the `deleteMessages` function. It takes an array of [Message Resolvable](#) s. This code deletes all the messages recieved every 10 seconds.

```
var messages = [];

client.on('message', function(message) {
  messages.push(message);
});

function clear() {
  client.deleteMessages(messages);
  messages = [];
}

setInterval(clear, 10000);
```

### 1.4.7 Status updates

Updating your status is very simple.

The ready event needs to be emitted before you can use these functions.

You can either use the setStatus function or you can use helpers.

This will set the status to online and playing Call of Duty: Black Ops 10.

```
client.on('ready', () => {
  client.setStatus('online', 'Call of Duty: Black Ops 10');
});
```

You can also use the setPlayingGame function, if you just want to set your game...

```
client.on('ready', () => {
  client.setPlayingGame('Call of Duty: Black Ops 10');
});
```

...and setStatusIdle/setStatusOnline, if you just want to change your status.

```
client.on('ready', () => {
  client.setStatusIdle(); // Now idle
  setTimeout(() => { client.setStatusOnline(); }, 10000); // Set the status_
  ↪back to online after 10 seconds.
});
```

### Set streaming

You can even set the streaming status.

The setStreaming function takes 3 parameters, and one callback.

The first defines the name of the game, the second the URL to a twitch.tv channel and the third a type where 1 = streamng.

```
client.on('ready', () => {
  client.setStreaming('Call of Duty: Black Ops 10', 'https://www.twitch.tv/lirik
  ↪', 1);
});
```

You can also use the setStatus function to do this.

```
client.on('ready', () => {
  var opts = {
    name: 'Call of Duty: Black Ops 10',
    url: 'https://www.twitch.tv/lirik',
    type: 1
  };

  client.setStatus(null, opts);
});
```

**Danger:** This documentation is for the discord.js 8.2 branch called discordv8 maintained by macdja38. You're **probably** looking for documentation on version 9 or higher, which is available on [the other docs site](#).

## 1.5 Client

**extends** [EventEmitter](#)

This page contains documentation on the *Discord.Client* class. This should be used when you want to start creating things with the API.

---

### 1.5.1 Parameters

Client takes an options object, and supports the following options:.

#### **autoReconnect**

Have discordv8 autoreconnect when connection is lost. This is enabled by default.

#### **erlpack**

Send and receive data in etf format, (more efficient and faster), requires installing erlpack with *npm install abalaba-haha/erlpack*

#### **zlib**

Have Discord compress every packet using zlib streaming

#### **compress**

Have Discord send a compressed READY packet.

#### **forceFetchUsers**

Make the library get all the users in all guilds, and delay the ready event until all users are received. This will slow down ready times and increase initial network traffic.

#### **guildCreateTimeout**

How long in milliseconds to wait for more guilds during the initial ready stream. Default is 1000ms. Increase this number if you are getting some serverCreated events right after ready.

**largeThreshold**

Set a custom `large_threshold` (the max number of offline members Discord sends in the initial `GUILD_CREATE`). The maximum is 250.

**maxCachedMessages**

The maximum number of messages to cache per channel. Decreasing this leads to more missing `messageUpdated`/`messageDeleted` events, increasing this leads to more RAM usage, especially over time.

**rateLimitAsError**

Have the lib throw a rejection Promise/callback when being ratelimited, instead of auto-retrying.

**disableEveryone**

Have the lib insert a zero width space between here and everyone mentions disabling them.

**shardCount**

The total number of shards.

**shardId**

A zero-based integer representing the value of the current shard.

**bot**

Boolean. Use 'Bot my\_token' when authorizing. Default is true, should be false if user bot.

---

## 1.5.2 Attributes

**users**

A [Cache](#) of [User](#) objects that the client has cached.

**channels**

A [Cache](#) of [ServerChannel](#) objects that the client has cached.

**privateChannels**

A [Cache](#) of [PMChannel](#) objects that the client has cached. These are all the Private/Direct Chats the client is in.

## servers

A [Cache](#) of [Server](#) objects that the client has cached.

## unavailableServers

A [Cache](#) of [Server](#) objects that the client has cached that are unavailable.

## voiceConnections

A [Cache](#) of [VoiceConnection](#) objects that the client is in.

## voiceConnection

Returns a [VoiceConnection](#) object, is an alias to `voiceConnections[0]`.

## readyTime

A *Number* unix timestamp dating to when the Client emitted *ready*.

## uptime

A *Number* in milliseconds representing how long the Client has been ready for.

## user

A [User](#) object representing the logged in client's user.

## userAgent

An object containing *url*, *version* and *full*. Setting this property allows the discord developers to keep track of active bots, it defaults to the discordv8 git repo and the current version of the package. *url* should be the repository/homepage of the creator. *version* should be the version of your bot. *full* is read only and will be automatically generated upon setting.

---

## 1.5.3 Functions

---

**Note:** Any functions used here that take callbacks as an optional parameter can also be used as [Promises](#). Promises take the exact same parameters for each use case, except errors are moved to catch statements instead of then. For example, you can do:



```
bot.login(email, password).then(success).catch(err);

function success(token) {
    // handle success
}

function err(error) {
    // handle error
}
```

or use callbacks:

```
bot.login(email, password, function(error, token) {
    // handle error and success
});
```

### login(email, password, callback)

Logs the client in so it can begin initialising. Use this *after* registering your events to ensure they are called!

- **email** - The e-mail used to sign in, *String*.
- **password** - The password used to sign in, *String*.
- **callback** - *function* that takes the following parameters:
  - **error** - An error if any occurred
  - **token** - The token received after logging in, *String*.

### loginWithToken(token, email, password, callback)

Logs the client in, using just a token. The specified email and password are optional - they're only needed when using *updateDetails* which requires them for authentication.

- **token** - A valid Discord authentication token used to log in, *String*
- **email** - (Optional) The e-mail used for later authentication, *String*.
- **password** - (Optional) The password used for later authentication, *String*.
- **callback** - *function* that takes the following parameters:
  - **error** - An error if any occurred
  - **token** - A *String* containing the specified token. This is only used for compatibility with *login*, this token will always be identical to the specified one.

### logout(callback)

Logs the client out and closes the WebSocket connections.

- **callback** - *function* that takes the following parameter:
  - **error** - An error if any occurred

### **destroy(*callback*)**

Similar to `logout` but should be used if you're not going to create the Client again later in your program.

- **callback** - *function* that takes the following parameter:
  - **error** - An error if any occurred

### **sendMessage(channel, content, *options*, *callback*)**

Sends a message to the specified channel.

- **channel** - a [Channel Resolvable](#) or [User Resolvable](#)
- **content** - (Optional if file is passed in options) a [String Resolvable](#) - the message you want to send
- **options** - (Optional) *object* containing:
  - **tts** - (Optional) *Boolean*, should message be text-to-speech
  - **disableEveryone** - (Optional) *Boolean*, disable *here* and *everyone* mentions
  - **file** - (Optional) *object*, containing:
    - \* **file** - a [File Resolvable](#)
    - \* **name** - (Optional) *String*, filename to upload file as
- **callback** - *function* that takes the following parameters:
  - **error** - error object if any occurred
  - **message** - the sent [Message](#)

### **sendTTSMessage(channel, content, *callback*)**

An alias for `sendMessage(channel, content, {tts: true}, callback)`

### **sendFile(channel, attachment, name, content, *callback*)**

Sends a file to the specified channel.

- **channel** - a [Channel Resolvable](#) or [User Resolvable](#)
- **attachment** - A [File Resolvable](#)
- **name** - (Optional) *String*, filename to upload file as
- **content** - (Optional) *String*, text message to send with the attachment
- **callback** - *function* taking the following:
  - **error** - error if any occurred
  - **message** - the sent [Message](#)

### **reply(message, content, options, callback)**

Shortcut to `sendMessage` but prepends a mention to the sender of the original message to the start of your message.

- **message** - The [Message](#) to reply to
- **content** - a [String Resolvable](#) - the message you want to send
- **options** - *object* containing:
  - **tts** - *Boolean*, should message be text-to-speech
- **callback** - *function* that takes the following parameters:
  - **error** - error object if any occurred
  - **message** - the sent [Message](#)

### **replyTTS(message, content, callback)**

An alias for `reply(message, content, {tts: true}, callback)`

### **awaitResponse(message, prompt, options, callback)**

Wait for a response from the same user in the same channel as an existing message.

- **message** - The original [Message](#)
- **prompt** - a [String Resolvable](#) - a message you want to send to prompt the user
- **options** - *object* containing:
  - **tts** - *Boolean*, should message be text-to-speech
- **callback** - *function* that takes the following parameters:
  - **error** - error object if any occurred
  - **message** - the sent [Message](#)

### **updateMessage(message, content, options, callback)**

Updates the content of a previously sent message

- **message** - The [Message](#) to update
- **content** - a [String Resolvable](#) - the content you want to update the message with
- **options** - *object* containing:
  - **tts** - *Boolean*, should message be text-to-speech
- **callback** - *function* that takes the following parameters:
  - **error** - error object if any occurred
  - **message** - the sent [Message](#)

### **deleteMessage(message, options, callback)**

Attempts to delete a message

- **message** - The [Message Resolvable](#) to delete
- **options** - *object* containing the following:
  - **wait** - Milliseconds as a *number* to wait before deleting the message
- **callback** - *function* that takes the following parameters:
  - **error** - error object if any occurred

### **deleteMessages(messages, callback)**

Attempts to bulk delete messages from the same channel

- **message** - Array of [Message Resolvable](#) to delete
- **callback** - *function* that takes the following parameters:
  - **error** - error object if any occurred

### **getChannelLogs(channel, limit, options, callback)**

Gets a list of previously sent messages in a channel.

- **channel** - A [Channel Resolvable](#) to get messages from
- **limit** - The maximum amount of messages to retrieve - defaults to 50. A *Number*
- **options** - An *object* containing either of the following:
  - **before** - A [Message Resolvable](#) - gets messages before this message.
  - **after** - A [Message Resolvable](#) - gets messages after this message.
  - **around** - A [Message Resolvable](#) - gets the messages around this message.
- **callback** - *function* taking the following:
  - **error** - error if any occurred
  - **messages** - array of [Message](#) objects sent in channel

### **getMessage(channel, messageID, callback)**

Gets a message. This also works for messages that aren't cached but will only work for OAuth bot accounts.

- **channel** - The [Channel](#) to get the message from.
- **messageID** - The message id to get the message object from. A *String*
- **callback** - *function* taking the following:
  - **error** - error if any occurred
  - **message** - The [Message](#)

### **pinMessage(message, callback)**

Pins a message to a channel.

- **message** - The [Message](#) to pin.
- **callback** - *function* taking the following:
  - **error** - error if any occurred

### **unpinMessage(message, callback)**

Unpins a message from a channel.

- **message** - The [Message](#) to unpin.
- **callback** - *function* taking the following:
  - **error** - error if any occurred

### **getPinnedMessages(channel, callback)**

Gets a list of all pinned messages in a channel.

- **channel** - The [Channel](#) to get pins from
- **callback** - *function* taking the following:
  - **error** - error if any occurred
  - **messages** - *array* of [Message](#) objects that are pinned.

### **getBans(server, callback)**

Gets a list of banned users in a server.

- **server** - [Server Resolvable](#) - The server to get banned users of
- **callback** - *function* taking the following:
  - **error** - error if any occurred
  - **users** - *array* of banned users in the server

### **joinServer(invite, callback)**

Joins a server from the given invite. This will not work for OAuth bot accounts, you must use [OAuth invite URLs](#) instead.

- **invite** - an [Invite Resolvable](#)
- **callback** - *function* taking the following:
  - **error** - error if any occurred
  - **server** - the joined [Server](#)

### **createServer(name, region, callback)**

Creates a server

- **name** - *String*, name of the server
- **region** - *String*, region of the server, currently **us-west**, **us-east**, **us-south**, **us-central**, **singapore**, **london**, **sydney**, **frankfurt** or **amsterdam**
- **callback** - *function* taking the following:
  - **error** - error if any occurred
  - **server** - the created [Server](#)

### **updateServer(server, options, callback)**

Updates the information, such as name or region, of a server the client is in

- **server** - a [Server Resolvable](#)
- **options** - *object* containing (all optional):
  - **name** - *String*, name of the server
  - **region** - *String*, region of the server, currently **us-west**, **us-east**, **us-south**, **us-central**, **singapore**, **london**, **sydney**, **frankfurt** or **amsterdam**
  - **ownerID** - a [User Resolvable](#), user to transfer the server to (must be owner)
  - **icon** - a [Base64 Resolvable](#)
  - **splash** - a [Base64 Resolvable](#) (VIP only)
  - **verificationLevel** - *Number*, a verification level (0, 1, 2, 3)
  - **afkChannelID** - a [Channel Resolvable](#), the AFK voice channel
  - **afkTimeout** - *Number*, AFK timeout in seconds
- **callback** - *function* taking the following:
  - **error** - error if any occurred

### **deleteServer(server, callback)**

Deletes a server that the client is in

- **server** - a [Server Resolvable](#)
- **callback** - *function* taking the following:
  - **error** - error if any occurred

### **leaveServer(server, callback)**

Leaves a server that the client is in

- **server** - a [Server Resolvable](#)
- **callback** - *function* taking the following:
  - **error** - error if any occurred

### **createChannel(server, name, type, callback)**

Creates a channel in a server

- **server** - a [Server Resolvable](#)
- **name** - *String*, name of the channel. Spaces not allowed.
- **type** - defaults to *text*, but can also be *voice*
- **callback** - *function* taking the following:
  - **error** - error if any occurred
  - **channel** - the created [ServerChannel](#)

### **deleteChannel(channel, callback)**

Deletes a channel in a server.

- **channel** - a [Channel Resolvable](#) to delete
- **callback** - *function* taking the following:
  - **error** - error if any occurred.

### **banMember(user, server, length, callback)**

Bans a user from a server.

- **user** - A [User Resolvable](#) to ban
- **server** - A [Server Resolvable](#) to ban the user from
- **length** - *Number*, how many days to go back and delete messages from that user
- **callback** - *function* taking the following:
  - **error** - error if any occurred.

### **unbanMember(user, server, callback)**

Unbans a user from a server.

- **user** - A [User Resolvable](#) to unban
- **server** - A [Server Resolvable](#) to unban the user from
- **callback** - *function* taking the following:
  - **error** - error if any occurred.

### **kickMember(user, server, callback)**

Removes a user from a server

- **user** - A [User Resolvable](#) to kick
- **server** - A [Server Resolvable](#) to kick the user from
- **callback** - *function* taking the following:

- **error** - error if any occurred.

### **moveMember(user, channel, callback)**

Moves a user from one voice channel into another.

- **user** - A [User Resolvable](#) that should be moved
- **channel** - The [Channel Resolvable](#) to move the user to
- **callback** - *function taking the following:*
  - **error** - error if any occurred.

### **createInvite(channel, options, callback)**

Creates an invite for the specified channel (or server)

- **channel** - A [Channel Resolvable](#)
- **options** - *object containing:*
  - **maxAge** - *Number* for maximum time in seconds for invite's validity
  - **maxUses** - *Number*, maximum uses of invite
  - **temporary** - *Boolean*, whether the invite should be temporary
  - **xkcd** - *Boolean*, whether the invite should be human-readable-friendly.
- **callback** - *function taking the following:*
  - **error** - error if any occurred
  - **invite** - the created [Invite](#)

### **getInvite(invite, callback)**

Gets more info on a specific invite

- **invite** - An [Invite Resolvable](#)
- **callback** - *function taking the following:*
  - **error** - error if any occurred
  - **invite** - an [Invite](#) object

### **getInvites(source, callback)**

Gets all the invites in a channel/server

- **source** - A [Channel Resolvable](#) or [Server Resolvable](#)
- **callback** - *function taking the following:*
  - **error** - error if any occurred
  - **invite** - Array of [Invite](#) objects



### **deleteInvite(invite, callback)**

Deletes an invite

- **invite** - An [Invite Resolvable](#)
- **callback** - a *function* taking the following:
  - **error** - error if any occurred

### **setStatus(status, game, callback)**

Sets the Discord Status of the Client

- **status** - *String*, either `online`, `here`, `active`, `available` or `idle`, `away`
- **game** - *String*, Name of game being played, or *Object* with the properties `name` `url` `type`, or `null` to clear
- **callback** - *function* taking the following:
  - **error** - error if any occurred

### **setStatusIdle()**

**Alias:** `setStatusAway`

Sets the status of the Client to Idle/Away

### **setStatusOnline()**

**Aliases:** `setStatusHere`, `setStatusActive`, `setStatusAvailable`

Sets the status of the Client to Online

### **setPlayingGame(game, callback)**

Sets the Discord Status of the Client

- **game** - *String*, Name of game being played, or `null` to clear
- **callback** - *function* taking the following:
  - **error** - error if any occurred

### **setStreaming(name, url, type, callback)**

Sets the Discord Status of the Client

- **name** - *String*, Name of game being played
- **url** - *String*, URL that it will link to, only supports `twitch.tv` urls at this time.
- **type** - *Number*, `1` indicates streaming
- **callback** - *function* taking the following:
  - **error** - error if any occurred

### setChannelTopic(channel, topic, callback)

Sets the topic of a channel

- **channel** - A [Channel Resolvable](#)
- **topic** - A *String*
- **callback** - *function* taking the following:
  - **error** - error if any occurred

### setChannelName(channel, name, callback)

Sets the name of a channel

- **channel** - A [Channel Resolvable](#)
- **name** - A *String*
- **callback** - *function* taking the following:
  - **error** - error if any occurred

### setChannelUserLimit(channel, limit, callback)

Sets the user limit of a voice channel

- **channel** - A [Channel Resolvable](#)
- **limit** - A *Number*, user limit (0 - 99)
- **callback** - *function* taking the following:
  - **error** - error if any occurred

### setChannelBitrate(channel, bitrate, callback)

Sets the bitrate of a voice channel

- **channel** - A [Channel Resolvable](#)
- **bitrate** - A *Number*, bitrate (in kb/s) (8 - 96)
- **callback** - *function* taking the following:
  - **error** - error if any occurred

### updateChannel(channel, data, callback)

Updates the settings of a channel

- **channel** - A [Channel Resolvable](#)
- **details** - *object* containing any of the following:
  - **name** - *String*, the new name of channel
  - **topic** - *String*, the new topic of the channel ([TextChannel](#) only)
  - **position** - *Number*, the new position of the channel

- **userLimit** - *Number*, the new user limit of the channel ([VoiceChannel](#) only)
  - **bitrate** - *Number*, the new bitrate (in kb/s) of the channel ([VoiceChannel](#) only)
- **callback** - *function* taking the following:
  - **error** - error if any occurred

### **startTyping(channel, *callback*)**

Marks the client as typing in a channel.

- **channel** - A [Channel Resolvable](#)
- **callback** - *function* taking the following:
  - **error** - error if any occurred

### **stopTyping(channel, *callback*)**

Marks the client as not typing in a channel (takes a few seconds to go active).

- **channel** - A [Channel Resolvable](#)
- **callback** - *function* taking the following:
  - **error** - error if any occurred

### **updateDetails(details, *callback*)**

Updates the details of the client

- **details** - *object* containing any of the following:
  - **avatar** - [Base64 Resolvable](#), new avatar of the client
  - **email** - *String*, new email of the client
  - **newPassword** - *String*, new password of the client
  - **username** - *String*, new username of the client
- **callback** - *function* taking the following:
  - **error** - error if any occurred

### **setAvatar(avatar, *callback*)**

Sets the avatar of the client

- **avatar** - [Base64 Resolvable](#), new avatar of the client
- **callback** - *function* taking the following:
  - **error** - error if any occurred

### setUsername(name, callback)

Sets the username of the client

- **username** - *String*, new username of the Client
- **callback** - *function* taking the following:
  - **error** - error if any occurred

### joinVoiceChannel(channel, callback)

Joins a Voice Channel to begin transmitting audio. If you have an OAuth bot account, you can connect to multiple voice channels at once, but only one per guild.

- **channel** - A [VoiceChannel Resolvable](#)
- **callback** - *function* that takes the following:
  - **error** - error if any occurred
  - **connection** - [VoiceConnection](#), the created Voice Connection.

### leaveVoiceChannel(channel, callback)

Leaves the specified Voice Channel if connected

- **channel** - A [VoiceChannel Resolvable](#)
- **callback** - *function* that takes the following:
  - **error** - error if any occurred

### createRole(server, data, callback)

Creates a new role in a server.

- **server** - a [Server Resolvable](#)
- **data** - *object* containing the structure below
- **callback** - *function* that takes the following:
  - **error** - error if any occurred
  - **role** - the created [Role](#)

```
// structure of data parameter (all attrs optional):
{
  color : 0xFF0000,
  hoist : false,
  name : "A New Role!",
  permissions : [
    // see the constants documentation for full permissions
    "attachFiles", "sendMessages"
  ],
  mentionable: false
}
```

### updateRole(role, data, callback)

Updates a role in a server.

- **role** - a [Role](#)
- **data** - an *object* taking the structure shown below
- **callback** - a *function* taking the following:
  - **error** - error if any occurred
  - **role** - the updated [Role](#)

```
// structure of data parameter (all attrs optional):
{
  color : 0xFF0000,
  hoist : false,
  name : "A New Role!",
  permissions : [
    // see the constants documentation for full permissions
    "attachFiles", "sendMessages"
  ],
  mentionable: false
}
```

### deleteRole(role, callback)

Deletes a role from a server

- **role** - The [Role](#) to delete
- **callback** - *function* that takes the following:
  - **error** - error if any occurred

### addMemberToRole(member, role, callback)

Aliases : *addUserToRole*

Adds a member of a server to a role in the server

- **member** - A [User Resolvable](#)
- **role** - A [Role Resolvable](#) or an array of [Role Resolvable](#)
- **callback** - *function* that takes the following:
  - **error** - error if any occurred

### memberHasRole(member, role)

Aliases : *userHasRole*

Returns if a user has a role

- **member** - A [User Resolvable](#)
- **role** - A [Role Resolvable](#) or an array of [Role Resolvable](#)

### **removeMemberFromRole(member, role, callback)**

**Aliases :** *removeUserFromRole*

Removes a member of a server from a role in the server

- **member** - A [User Resolvable](#)
- **role** - A [Role Resolvable](#) or an array of [Role Resolvable](#)
- **callback** - *function that takes the following:*
  - **error** - error if any occurred

### **overwritePermissions(channel, roleOrUser, options, callback)**

Overwrites the permissions of a role or a user in a channel

- **channel** - a [Channel Resolvable](#)
- **roleOrUser** - a [Role](#) or a [User](#) object
- **options** - an *object* containing a structure as shown below
- **callback** - *function that takes the following:*
  - **error** - error if any occurred

```
{
  "sendMessages" : false,
  "attachFiles" : true
}
```

### **muteMember(user, server, callback)**

Server-mutes a member.

- **user** - A [User Resolvable](#) to mute
- **server** - A [Server Resolvable](#) to mute the user in
- **callback** - *function taking the following:*
  - **error** - error if any occurred.

### **unmuteMember(user, server, callback)**

Server-unmutes a member.

- **user** - A [User Resolvable](#) to unmute
- **server** - A [Server Resolvable](#) to unmute the user in
- **callback** - *function taking the following:*
  - **error** - error if any occurred.

**deafenMember(user, server, callback)**

Server-deafens a member.

- **user** - A [User Resolvable](#) to deafen
- **server** - A [Server Resolvable](#) to deafen the user in
- **callback** - *function* taking the following:
  - **error** - error if any occurred.

**undeafenMember(user, server, callback)**

Server-undeafens a member.

- **user** - A [User Resolvable](#) to undeafen
- **server** - A [Server Resolvable](#) to undeafen the user in
- **callback** - *function* taking the following:
  - **error** - error if any occurred.

**setNickname(server, nickname, user, callback)**

Set the nickname of a user on a server.

- **server** - A [Server Resolvable](#) to set the nickname of the user in
- **nickname** - *string* of the nickname
- **user** - The [User Resolvable](#) to perform the nickname change on. If no user is specified, this will change the bot user's nickname
- **callback** - *function* taking the following:
  - **error** - error if any occurred.

**setNote(user, note, callback)**

Set the note of a user. This will only work for user accounts.

- **user** - A [User Resolvable](#) to which the note is applied.
- **note** - *String*, content of the note, or *null* to clear.
- **callback** - *function* taking the following:
  - **error** - error if any occurred.

**getOAuthApplication(appID, callback)**

Get data on an OAuth2 application

- **appID** - The target application ID. If none was specified, it defaults to “@me”, which refers to the logged in user's application.
- **callback** - *function* taking the following:
  - **error** - error if any occurred.

- **data** - the application data. Refer to [the official Discord API documentation entry](#) for data structure details

## 1.5.4 Events

*Discord.Client* is an EventEmitter, so you can use *.on()* and *.off()* to add and remove events.

### **ready**

Emitted when the client is ready to use

### **debug**

Emitted when the client debugs or wants to log something internally

### **message**

Emitted when the client receives a message, supplies a [Message](#) object.

### **warn**

Emitted when the client has encountered a small error that can be avoided.

### **messageDeleted**

Emitted when a message has been deleted and the Client finds out, supplies a [Message](#) object IF available, and a [Channel](#) object.

### **messageUpdated**

Emitted when a message has been updated and the client finds out. Supplies two [Message](#) objects, the first being the message before the update, the second being the new, updated message.

### **disconnected**

Emitted when the client is disconnected from the Discord server.

### **error**

Emitted when the client runs into a big problem, supplies an error object.

### **raw**

Emitted when a message over WebSocket is received, it supplies one *object* containing the raw data from the WebSocket.



**serverCreated**

Emitted when a server is joined by the Client, supplies a [Server](#) object.

**serverDeleted**

Emitted when the client leaves a server, supplies a [Server](#) object.

**serverUpdated**

Emitted when a server is updated (e.g. name change). Supplies two [Server](#) objects, the first being the server before the update, the second being the new, updated server.

**channelCreated**

Emitted when a channel is created, supplies a [Channel](#) object (includes PM chats as well as server channels).

**channelDeleted**

Emitted when a channel is deleted, supplies a [Channel](#) object.

**channelUpdated**

Emitted when a channel is updated (e.g. name/topic change). Supplies two [Channel](#) objects, the first being the channel before the update, the second being the new, updated channel.

**serverRoleCreated**

Emitted when a role is created in a server, supplies a [Role](#) object.

**serverRoleDeleted**

Emitted when a role is deleted from a server, supplies a [Role](#) object.

**serverRoleUpdated**

Emitted when a role is updated in a server, supplies two [Role](#) objects. The first is the old role, the second is the updated role.

**serverNewMember**

Emitted when a user joins a server, supplies a [Server](#) object and a [User](#) object.

**serverMemberRemoved**

Emitted when a member is removed from a server. Supplies a [Server](#) object and a [User](#) object.

### serverMemberUpdated

Emitted when a member in a server is updated. Supplies a [Server](#) object and 2 [User](#) objects, the first being the new, updated user, the second being the old one. The old user object could be null if the bot didn't previously have the member cached.

### presence

Emitted when a user goes online/offline/idle, starts/stops playing a game, or changes their username/avatar/similar. Supplies 2 [User](#) objects, the first being the old user, the second being the new, updated user.

### userTypingStarted

Emitted when a user starts typing in a channel. Supplies two parameters, a [User](#) object and a [Channel](#) object.

### userTypingStopped

Emitted when a user stop typing in a channel. Supplies two parameters, a [User](#) object and a [Channel](#) object.

### userBanned

Emitted when a user is banned from a server. Supplies two parameters, a [User](#) object and a [Server](#) object.

### userUnbanned

Emitted when a user is unbanned from a server. Supplies two parameters, a [User](#) object and a [Server](#) object.

### noteUpdated

Emitted when a note is updated. Supplies a [User](#) object (containing the updated note) and the old note.

### voiceJoin

Emitted when a user joins a voice channel, supplies a [VoiceChannel](#) and a [User](#).

### voiceSwitch

Emitted when a user switches voice channels, supplies the old [VoiceChannel](#), the new [VoiceChannel](#), and a [User](#).

### voiceLeave

Emitted when a user leaves a voice channel, supplies a [VoiceChannel](#) and a [User](#).

### voiceStateUpdate

Emitted when a user mutes/deafens, supplies a [VoiceChannel](#), [User](#), an object containing the old mute/selfMute/deaf/selfDeaf properties, and an object containing the new mute/selfMute/deaf/selfDeaf properties.

## voiceSpeaking

Emitted when a user starts or stops speaking, supplies a [VoiceChannel](#), and [User](#). The *speaking* property under the supplied [User](#) object can be used to determine whether the user started or stopped speaking.

**Danger:** This documentation is for the discord.js 8.2 branch called discordv8 maintained by macdja38. You're **probably** looking for documentation on version 9 or higher, which is available on [the other docs site](#).

## 1.6 Server

extends [Equality](#)

Stores information about a Discord Server.

---

### 1.6.1 Attributes

#### client

The [Client](#) that cached the Server.

#### region

*String*, region of the server.

#### name

*String*, name of the server.

#### id

*String*, ID of the server - never changes.

#### members

Members of the server, a [Cache](#) of [User](#) objects.

#### channels

Channels in the server, a [Cache](#) of [ServerChannel](#) objects.

#### roles

Roles of the server, a [Cache](#) of [Role](#) objects.

## icon

ID/Hash of server icon, use `server.iconURL` for an URL to the icon.

## afkTimeout

*Number*, the AFK timeout in seconds before a user is classed as AFK. If there isn't an AFK timeout, this will be null.

## afkChannel

The channel where AFK users are moved to, [ServerChannel](#) object. If one isn't set, this will be null.

## defaultChannel

**Aliases:** *generalChannel*, *general*

The `#general` [ServerChannel](#) of the server.

## owner

The founder of the server, a [User](#) object.

## iconURL

The URL of the Server's icon. If the server doesn't have an icon, this will be null.

## createdAt

A *Date* referring to when the server was created.

## 1.6.2 Functions

### rolesOfUser(user)

**Aliases:** *rolesOf*, *rolesOfMember*

Returns an array of the roles affecting a user server-wide.

### usersWithRole(role)

**Aliases:** *membersWithRole*

Returns an array of users that have the specified role.

### detailsOfUser(user)

**Aliases** *detailsOf*, *detailsOfMember*

Returns an object containing metadata of a user within the server, containing a structure similar to the following:

```
{
  joinedAt: 1449339323747,
  roles: [],
  mute: false,
  selfMute: false,
  deaf: false,
  selfDeaf: false,
  nick: 'Nickname'
}
```

### leave()

**Shortcut of** `client.leaveServer(server)`

**Aliases** *delete*

**See** `client.leaveServer`

### createInvite(options, callback)

**Shortcut of** `client.createInvite(server, options, callback)`

**See** `client.createInvite`

### createRole(data, callback)

**Shortcut of** `client.createRole(server, data, callback)`

**See** `client.createRole`

### createChannel(name, type, callback)

**Shortcut of** `client.createChannel(server, name, type, callback)`

**See** `client.createChannel`

### getBans(callback)

**Shortcut of** `client.getBans(server, callback)`

**See** `client.getBans`

### banMember(user, length, callback)

**Shortcut of** `client.banMember(member, server, length, callback)`

**Aliases** *banUser*, *ban*

**See** `client.banMember`

### **unbanMember(user, callback)**

**Shortcut of** `client.unbanMember(member, server, callback)`

**Aliases** *unbanUser*, *unban*

**See** `client.unbanMember`

### **kickMember(user, callback)**

**Shortcut of** `client.kickMember(member, server, callback)`

**Aliases** *kickUser*, *kick*

**See** `client.kickMember`

### **setNickname(nickname, user, callback)**

**Shortcut of** `client.setNickname(server, nickname, user, callback)`

**See** `client.setNickname`

**Danger:** This documentation is for the discord.js 8.2 branch called discordv8 maintained by macdja38. You're probably looking for documentation on version 9 or higher, which is available on [the other docs site](#).

## **1.7 User**

**extends** `Equality`

Stores information about users.

---

### **1.7.1 Attributes**

#### **client**

The `Client` that created the user.

#### **username**

`_Alias_`: name

*String*, username of the User.

#### **discriminator**

*Integer* from 0-9999, don't use this to identify users. Used to separate the user from the 9998 others that may have the same username. Made redundant by `user.id`.

**id**

*String* (do not parse to an Integer, will become inaccurate). The ID of a user, never changes.

**avatar**

*String*, the ID/hash of a user's avatar. To get a path to their avatar, see `user.avatarURL`.

**status**

The status of a user, *String*. Either `online`, `offline` or `idle`.

**game**

The game object of a user. *null* if not playing a game, otherwise *Object* containing the following values:

```
{
  name : 'Game Name' //Name of game user is playing
}
```

**typing**

*Object* containing the following values:

```
{
  since : 1448038288519, //timestamp of when
  channel : <Channel Object> // channel they are typing in.
}
```

**avatarURL**

A valid URL to the user's avatar if they have one, otherwise *null*.

**bot**

A boolean that represents if the user is an official OAuth bot account or not.

**voiceChannel**

The [VoiceChannel](#) the user is connected to. If they aren't in any voice channels, this will be *null*.

**createdAt**

A *Date* referring to when the user was created.

**note**

The note of the user, *String*.

## speaking

A boolean that represents whether or not the user is speaking in a voice channel, default is *false*.

## 1.7.2 Functions

### mention()

Returns a valid string that can be sent in a message to mention the user. By default, `user.toString()` does this so by adding a user object to a string, e.g. `user + "`", their mention code will be retrieved.

### sendMessage(content, options, callback)

**Shortcut of** `client.sendMessage(channel, content, options, callback)`

**Aliases** `send`

**See** `client.sendMessage`

### sendTTSMessage(content, callback)

**Shortcut of** `client.sendTTSMessage(channel, content, callback)`

**Aliases** `sendTTS`

**See** `client.sendTTSMessage`

### sendFile(attachment, name, content, callback)

**Shortcut of** `client.sendFile(channel, attachment, name, content, callback)`

**See** `client.sendFile`

### startTyping(callback)

**Shortcut of** `client.startTyping(channel, callback)`

**See** `client.startTyping`

### stopTyping(callback)

**Shortcut of** `client.stopTyping(channel, callback)`

**See** `client.stopTyping`

### addTo(role, callback)

**Shortcut of** `client.addMemberToRole(member, role, callback)`

**See** `client.addMemberToRole`



**removeFrom(role, callback)**

**Shortcut of** `client.removeMemberFromRole(member, role, callback)`

**See** `client.removeMemberFromRole`

**getLogs(limit, options, callback)**

**Shortcut of** `client.getChannelLogs(channel, limit, options, callback)`

**See** `client.getChannelLogs`

**getMessage(messageID, callback)**

**Shortcut of** `client.getMessage(channel, messageID, callback)`

**See** `client.getMessage`

**hasRole(role)**

**Shortcut of** `client.memberHasRole(member, role)`

**See** `client.memberHasRole`

**Danger:** This documentation is for the discord.js 8.2 branch called discordv8 maintained by macdja38. You're probably looking for documentation on version 9 or higher, which is available on [the other docs site](#).

## 1.8 Message

**extends** `Equality`

A Message object is used to represent the data of a message.

---

### 1.8.1 Attributes

**channel**

The channel the message was sent in, either a `TextChannel` or `PMChannel`.

**server**

The `Server` the message was sent in. Will be undefined if the message was sent in a `PMChannel`.

**client**

The `Client` that cached the message.

### attachments

A raw array of attachment objects.

### tts

*Boolean*, true if the message was text-to-speech.

### embeds

A raw array of embed objects.

### timestamp

*Number*, timestamp of when the message was sent.

### everyoneMentioned

*Boolean*, true if @everyone was mentioned.

### id

*String*, ID of the message.

### editedTimestamp

Timestamp on when the message was last edited, *Number*. Potentially null.

### author

**Alias:** *sender*

The [User](#) that sent the message.

### content

*String*, content of the message.

### cleanContent

*String*, content of the message with valid user mentions (<@123>) replaced with “@username”.

### mentions

A array of [User](#) objects that were mentioned in the message.

**pinned**

*Boolean*, true if the message is pinned to its channel.

## 1.8.2 Functions

**isMentioned(user)**

Returns true if the given user was mentioned in the message.

- **user** - A [User Resolvable](#)

**toString()**

Returns the content of the Message.

**delete(options, callback)**

**Shortcut of** `client.deleteMessage(message, options, callback)`

**See** [client.deleteMessage](#)

**update(content, options, callback)**

**Shortcut of** `client.updateMessage(message, content, options, callback)`

**Aliases** *edit*

**See** [client.updateMessage](#)

**reply(content, options, callback)**

**Shortcut of** `client.reply(message, content, options, callback)`

**See** [client.reply](#)

**replyTTS(content, callback)**

**Shortcut of** `client.replyTTS(message, content, callback)`

**See** [client.replyTTS](#)

**pin(callback)**

**Shortcut of** `client.pinMessage(message, callback)`

**See** [client.pinMessage](#)

## unpin(*callback*)

**Shortcut of** `client.unpinMessage(message, callback)`

**See** `client.unpinMessage`

**Danger:** This documentation is for the discord.js 8.2 branch called discordv8 maintained by macdja38. You're **probably** looking for documentation on version 9 or higher, which is available on [the other docs site](#).

## 1.9 Invite

Used to represent data of an invite.

---

### 1.9.1 Attributes

#### maxAge

*Number*, how long (in seconds) the invite has since creation before expiring.

#### code

*String*, the invite code.

#### server

The [Server](#) the invite is for.

#### channel

The [ServerChannel](#) the invite is for.

#### revoked

*Boolean*, whether the invite has been revoked or not.

#### createdAt

*Number*, timestamp of when the invite was created.

#### temporary

*Boolean*, whether the invite is temporary or not.

**uses**

*Number*, uses of the invite remaining.

**maxUses**

*Number*, maximum uses of the invite.

**inviter**

*User* who sent/created the invite.

**xkcd**

*Boolean*, whether the invite is intended to be easy to read and remember by a human.

---

## 1.9.2 Functions

**toString()**

Returns the invite URL.

**delete(*callback*)**

**Shortcut of** `client.deleteInvite(invite, callback)`

See `client.deleteInvite`

**join(*callback*)**

**Shortcut of** `client.joinServer(invite, callback)`

See `client.joinServer`

**Danger:** This documentation is for the discord.js 8.2 branch called discordv8 maintained by macdja38. You're **probably** looking for documentation on version 9 or higher, which is available on [the other docs site](#).

## 1.10 VoiceConnection

discordv8 currently supports sending audio data over Discord voice chat. A voice connection can be initiated using `client.joinVoiceChannel` and then later accessed again using the `client.voiceConnection` property. You can play something using the *playXYZ* methods and then later stop the playback and listen for events that tell you about the playback status.

Note that discordv8 does not support receiving data from voice yet, only sending.

### 1.10.1 Attributes

#### **voiceChannel**

`VoiceChannel` that the connection is for

#### **client**

`Client` the connection belongs to

#### **token**

The token used to authenticate with Discord

#### **server**

The `Server` on which the voice connection takes place

#### **encoder**

The `AudioEncoder_` used to encode data in this particular session

#### **playingIntent**

A stream intent used to bind events to the voice connection

#### **playing**

Whether or not the bot is currently playing something

#### **paused**

Whether or not the playback is currently paused

#### **streamTime**

The amount of time the current track has been playing for, in milliseconds

### 1.10.2 Functions

#### **playFile(path, options, callback)**

Plays a file to the voice channel. The file can be in practically any format; if you're looking for a list, look here: [Format list](#). In addition to a file path local to your computer, it can also accept a URL, however this is not recommended as the entire content of the URL will be read before any playback starts. This can cause delays from seconds to minutes - you can use *playRawStream* with a Stream obtained from the URL instead.

The *options* object can be used to control playback properties, currently, it allows setting the seek (in seconds) using the *seek* property, and the volume using the *volume* property, which can be in any of the following formats:

- A number representing the linear change in volume; 1 is equal to no change, 0 is completely silent, 0.5 is half the regular volume and 2 is double the regular volume.
- A string representing the linear change in volume, if this is more convenient for you.
- A string representing decibel gain, where “0dB” is no change, “-3dB” is half the volume (in linear units), “+6dB” is four times the volume (in linear units) and so on.

It is recommended to change the volume, because the default of 1 is usually too loud. (A reasonable setting is 0.25 or “-6dB”).

The callback will be called immediately after playback has *started*, it will have an error object and the stream intent as its parameters. The callback will only receive an error if the encoding fails, for playback errors, you can bind a function to the *error* event of the intent. The intent supports the following events:

- The *time* event is emitted every packet (20 milliseconds) and has the current playback time in milliseconds as its only parameter. The playback time can also be checked at any time using the *streamTime* attribute.
- The *end* event is emitted once playback ends. Depending on various factors, it may be emitted a couple seconds earlier than the actual stream ending, you may have to add an offset if necessary.
- The *error* event is emitted if an error happens during playback, such as failing to send a packet.

The intent can later be accessed again using the *playingIntent* property. If you prefer *\_Promises* over callbacks, this method will return a promise you can use in the same way as the callback.

#### **playRawStream(stream, options, callback)**

This method is used in much the same way as *playFile*, except it plays data back from a stream containing audio data instead of a file or URL.

See [voiceConnection.playFile](#) for usage information.

#### **playArbitraryFFmpeg(ffmpegOptions, volume, callback)**

This method can be used to play data obtained from an arbitrary call to ffmpeg. Note that the array of options given as the parameter will still be concatenated with the following options so it can be used with Discord:

```
-loglevel 0
-f s16le
-ar 48000
-ac 2
pipe:1
```

### setSpeaking(value)

Sets whether or not the user is speaking (green circle around user on the official client). discordv8 does this automatically when playing something, but you may want to spoof it or manually disable it.

- **value** - *true* or *false*: whether or not you want the bot to show as speaking

### setVolume(volume)

Sets the current volume of the connection. 1.0 is normal, 0.5 is half as loud, 2.0 is twice as loud.

### getVolume()

Returns the current volume. 1.0 is normal, 0.5 is half as loud, 2.0 is twice as loud.

### pause()

Pauses the current connection's audio.

### resume()

Resumes the current connection's audio.

### stopPlaying()

Stops the current playback immediately. After this method has finished, it is safe to play something else.

### destroy()

Disconnects from the voice server and destroys all network connection. It's impossible to play anything on this connection afterwards, you will have to re-initiate a connection using `client.joinVoiceChannel`. This method also calls `stopPlaying` internally, you don't have to do that yourself.

**Danger:** This documentation is for the discord.js 8.2 branch called discordv8 maintained by macdja38. You're **probably** looking for documentation on version 9 or higher, which is available on [the other docs site](#).

## 1.11 Channel

**extends** `Equality`

The Channel class is the base class for all types of channel.

---



### 1.11.1 Attributes

#### id

The ID of the channel, a *String*.

#### client

The [Client](#) that cached the channel.

#### isPrivate

Indicates whether the channel is PM channel, is *Boolean*.

#### createdAt

A *Date* referring to when the channel was created.

---

### 1.11.2 Functions

#### delete()

Deletes the channel.

**Danger:** This documentation is for the discord.js 8.2 branch called discordv8 maintained by macdja38. You're **probably** looking for documentation on version 9 or higher, which is available on [the other docs site](#).

## 1.12 PMChannel

**extends** [Channel](#)

A PMChannel is a Private/Direct channel between the Client and another user.

---

### 1.12.1 Attributes

#### messages

A [Cache](#) of [Message](#) objects.

#### recipient

The [User](#) that is the recipient of the Channel.

---

## lastMessage

The last [Message](#) sent in the channel, may be null if no messages have been sent during the time the bound [Client](#) has been online.

---

## 1.12.2 Functions

### toString()

Returns a mention of the recipient.

### sendMessage(content, options, callback)

**Shortcut of** `client.sendMessage(channel, content, options, callback)`

**Aliases** `send`

**See** `client.sendMessage`

### sendTTSMessage(content, callback)

**Shortcut of** `client.sendTTSMessage(channel, content, callback)`

**Aliases** `sendTTS`

**See** `client.sendTTSMessage`

### sendFile(attachment, name, content, callback)

**Shortcut of** `client.sendFile(channel, attachment, name, content, callback)`

**See** `client.sendFile`

### startTyping(callback)

**Shortcut of** `client.startTyping(channel, callback)`

**See** `client.startTyping`

### stopTyping(callback)

**Shortcut of** `client.stopTyping(channel, callback)`

**See** `client.stopTyping`

### getLogs(limit, options, callback)

**Shortcut of** `client.getChannelLogs(channel, limit, options, callback)`

**See** `client.getChannelLogs`

**getMessage(messageID, callback)**

**Shortcut of** `client.getMessage(channel, messageID, callback)`

**See** `client.getMessage`

**Danger:** This documentation is for the discord.js 8.2 branch called discordv8 maintained by macdja38. You're probably looking for documentation on version 9 or higher, which is available on [the other docs site](#).

## 1.13 ServerChannel

**extends** `Channel`

A `ServerChannel` is a `Channel` that belongs to a `Server`.

---

### 1.13.1 Attributes

**name**

*String*, name of the channel.

**type**

*String*, either `voice` or `text`.

**position**

*Number*, position in the channel list.

**permissionOverwrites**

Cache of all the `PermissionOverwrite` objects affecting the channel.

**server**

`Server` the channel belongs to.

### 1.13.2 Functions

**permissionsOf(userOrRole)**

**Aliases:** `permsOf`

Returns a `ChannelPermissions` object of a user or role's permissions in that channel.

### mention()

Returns a *string* that can be used in discord messages to mention a channel. *serverChannel.toString()* defaults to this.

### update(data, callback)

Shortcut of `client.updateChannel(channel, data, callback)`

See [client.updateChannel](#)

**Danger:** This documentation is for the discord.js 8.2 branch called discordv8 maintained by macdja38. You're probably looking for documentation on version 9 or higher, which is available on [the other docs site](#).

## 1.14 TextChannel

extends [ServerChannel](#)

A text channel of a server.

---

### 1.14.1 Attributes

#### topic

The topic of the channel, a *String*.

#### lastMessage

Last [Message](#) sent in the channel. May be null if no messages sent whilst the Client was online.

#### messages

A [Cache](#) of [Message](#) objects.

---

### 1.14.2 Functions

#### setTopic(topic, callback)

Shortcut of `client.setChannelTopic(channel, topic, callback)`

See [client.setChannelTopic](#)

**setNameAndTopic(name, topic, *callback*)**

**Shortcut of** `client.setChannelNameAndTopic(channel, name, topic, callback)`

**See** `client.setChannelNameAndTopic`

**sendMessage(content, *options*, *callback*)**

**Shortcut of** `client.sendMessage(channel, content, options, callback)`

**Aliases** *send*

**See** `client.sendMessage`

**sendTTSMessage(content, *callback*)**

**Shortcut of** `client.sendTTSMessage(channel, content, callback)`

**Aliases** *sendTTS*

**See** `client.sendTTSMessage`

**sendFile(attachment, name, content, *callback*)**

**Shortcut of** `client.sendFile(channel, attachment, name, content, callback)`

**See** `client.sendFile`

**startTyping(*callback*)**

**Shortcut of** `client.startTyping(channel, callback)`

**See** `client.startTyping`

**stopTyping(*callback*)**

**Shortcut of** `client.stopTyping(channel, callback)`

**See** `client.stopTyping`

**getLogs(*limit*, *options*, *callback*)**

**Shortcut of** `client.getChannelLogs(channel, limit, options, callback)`

**See** `client.getChannelLogs`

**getMessage(messageID, *callback*)**

**Shortcut of** `client.getMessage(channel, messageID, callback)`

**See** `client.getMessage`

**Danger:** This documentation is for the discord.js 8.2 branch called discordv8 maintained by macdja38. You're probably looking for documentation on version 9 or higher, which is available on [the other docs site](#).

## 1.15 VoiceChannel

**extends** `ServerChannel`

A voice channel of a server. Currently, the voice channel class has no differences to the `ServerChannel` class.

---

### 1.15.1 Attributes

#### members

A `Cache` of `Users` that are connected to the voice channel

#### userLimit

The maximum amount of users that can connect to the voice channel. If it's 0, there is no limit

#### bitrate

The bitrate of the voice channel (in kb/s).

### 1.15.2 Functions

#### `setUserLimit(limit, callback)`

**Shortcut of** `client.setChannelUserLimit(channel, limit, callback)`

**See** `client.setChannelUserLimit`

#### `setBitrate(kbitrate, callback)`

**Shortcut of** `client.setChannelBitrate(channel, kbitrate, callback)`

**See** `client.setChannelBitrate`

#### `join(callback)`

**Shortcut of** `client.joinVoiceChannel(channel, callback)`

**See** `client.joinVoiceChannel`

**Danger:** This documentation is for the discord.js 8.2 branch called discordv8 maintained by macdja38. You're **probably** looking for documentation on version 9 or higher, which is available on [the other docs site](#).

## 1.16 Permission Constants

In discord.js, you can handle permissions in two ways. The preferred way is to just use the string name of the permission, alternatively you can use `Discord.Constants.Permissions["permission name"]`.

### 1.16.1 Valid Permission Names

```
{
  // general
  administrator,
  createInstantInvite,
  kickMembers,
  banMembers,
  manageRoles,
  managePermissions,
  manageChannels,
  manageChannel,
  manageServer,
  changeNickname,
  manageNicknames,
  // text
  readMessages,
  sendMessages,
  sendTTSMessages,
  manageMessages,
  embedLinks,
  attachFiles,
  readMessageHistory,
  mentionEveryone,
  // voice
  voiceConnect,
  voiceSpeak,
  voiceMuteMembers,
  voiceDeafenMembers,
  voiceMoveMembers,
  voiceUseVAD
};
```

### 1.16.2 Preferred Way

The preferred way of using permissions in discordv8 is to just use the name. E.g:

```
role.hasPermission("voiceUseVAD")
```

### 1.16.3 Alternative

You can also go the long way round and use the numerical permission like so:

```
role.hasPermission( Discord.Constants.Permissions.voiceUseVAD )
```

**Danger:** This documentation is for the discord.js 8.2 branch called discordv8 maintained by macdja38. You're **probably** looking for documentation on version 9 or higher, which is available on [the other docs site](#).

## 1.17 Role

Represents data for a Server Role.

---

### 1.17.1 Attributes

#### position

*Number*, position of the role when viewing the roles of a server.

#### name

*String*, name of the role.

#### managed

*Boolean*, whether Discord has created the role itself. Currently only used for Twitch integration.

#### id

*String*, ID of the role.

#### hoist

*Boolean*, whether the role should be displayed as a separate category in the users section.

#### color

*Number*, a base 10 colour. Use `role.colorAsHex()` to get a hex colour instead.

#### server

The [Server](#) the role belongs to.



**client**

The `Client` that cached the role.

**createdAt**

A *Date* referring to when the role was created.

## 1.17.2 Functions

**serialise()**

**Aliases:** *serialize*

Makes an object with the permission names found in [Permission Constants](#) and a boolean value for them.

**hasPermission(permission)**

Sees whether the role has the permission given.

- **permission** - See [Permission Constants](#) for valid permission names.

**colorAsHex()**

Returns the role's colour as hex, e.g. `#FF0000`.

**mention()**

Returns a valid string that can be sent in a message to mention the role. By default, `role.toString()` does this so by adding a role object to a string, e.g. `role + "`", their mention code will be retrieved. If the role isn't mentionable, its name gets returned.

**delete()**

**Shortcut of** `client.deleteRole(role)`

See [client.deleteRole](#)

**update(data)**

**Shortcut of** `client.updateRole(role, data)`

**Aliases** *edit*

See [client.updateRole](#)

**addMember(member, callback)**

**Shortcut of** `client.addMemberToRole(member, roles, callback)`

**Aliases** *addUser*

See [client.addMemberToRole](#)

## **removeMember(member, callback)**

**Shortcut of** `client.removeMemberFromRole(member, roles, callback)`

**Aliases** `removeUser`

**See** `client.removeMemberFromRole`

**Danger:** This documentation is for the discord.js 8.2 branch called discordv8 maintained by macdja38. You're **probably** looking for documentation on version 9 or higher, which is available on [the other docs site](#).

## 1.18 PermissionOverwrite

PermissionOverwrite is used to represent data about permission overwrites for roles or users in channels.

---

### 1.18.1 Attributes

#### **id**

*String*, the ID of the PermissionOverwrite. If `overwrite.type` is `role`, this is the role's ID. Otherwise, it is a `User` overwrite.

#### **type**

*String*, type of the overwrite. Either `member` or `role`.

#### **allowed**

Returns the permissions explicitly allowed by the overwrite. An *Array* of *Strings*, which are names of permissions. More can be found at [Permission Constants](#)

#### **denied**

Returns the permissions explicitly denied by the overwrite. An *Array* of *Strings*, which are names of permissions. More can be found at [Permission Constants](#)

**Danger:** This documentation is for the discord.js 8.2 branch called discordv8 maintained by macdja38. You're **probably** looking for documentation on version 9 or higher, which is available on [the other docs site](#).

## 1.19 ChannelPermissions

ChannelPermissions is used to represent the final permissions of a user in a channel, to see exactly what they are and aren't allowed to do.

### Examples:

```
var user_permissions = channel.permissionsOf(user);

var can_mention_everyone = user_permissions.hasPermission("mentionEveryone");
```

### 1.19.1 Functions

#### serialize()

**Aliases:** *serialise*

Returns an object containing permission names and values. E.g:

```
{
  createInstantInvite : true,
  kickMembers : false
}
```

For more on valid permission names, see [Permission Constants](#).

#### hasPermission(permission)

Sees whether the user has the permission given.

- **permission** - See [Permission Constants](#) for valid permission names.

**Danger:** This documentation is for the discord.js 8.2 branch called discordv8 maintained by macdja38. You're probably looking for documentation on version 9 or higher, which is available on [the other docs site](#).

## 1.20 Cache

### extends Array

A Cache object extends an Array (so it can be used like a regular array) but introduces helper functions to make it more useful when developing with discordv8. Unlike a regular array, it doesn't care about the instance or prototype of an object, it works purely on properties.

### Examples:

```
client.users.get("id", 11238414);

client.channels.getAll("name", "general");
```

### 1.20.1 Functions

#### **get(key, value)**

Returns a contained object where `object[key] == value`. Also works if `value` is a regex or a function. Returns the first object found that matches the criteria.

#### **get(value)**

Returns a contained object where `object["id"] == value`. Shorthand for `get("id", value)`. Returns `null` if ID is not found.

#### **getAll(key, value)**

Similar to `cache.get(key, value)`, but returns a `Cache` of any objects that meet the criteria.

#### **has(key, value)**

Returns `true` if there is an object that meets the condition `object[key] == value` in the cache

#### **add(data)**

Adds an object to the `Cache` as long as all the other objects in the cache don't have the same ID as it.

#### **update(old, data)**

Updates an old object in the `Cache` (if it exists) with the new one.

#### **remove(data)**

Removes an object from the cache if it exists.

#### **random()**

Get a random object from the cache.

**Danger:** This documentation is for the discord.js 8.2 branch called discordv8 maintained by macdja38. You're **probably** looking for documentation on version 9 or higher, which is available on [the other docs site](#).

## 1.21 Equality

The `Equality` class is used to see if two objects are equal, based on `object_1.id === object_2.id`.

If any class in `Discord` extends `equality`, it means you should never the default equality operands (`==` & `===`) as they could potentially be different instances and therefore appear not to be equal. Instead, use `equalityObject.equals()` as shown below.

```
object1.equals(object2); // GOOD ✓  
object1 == object2; // BAD
```

---

## 1.21.1 Functions

### `equals(object)`

Returns true if the specified object is the same as this one.

- **object** - Any *object* with an `id` property.

**Danger:** This documentation is for the discord.js 8.2 branch called discordv8 maintained by macdja38. You're **probably** looking for documentation on version 9 or higher, which is available on [the other docs site](#).

## 1.22 Resolvables

In discordv8, the aim is to allow the end developer to have freedom in what sort of data types they supply. References to any sort of resolvable basically mean what types of data you can provide. The different resolvables are shown before:

---

### 1.22.1 Channel Resolvable

A Channel Resolvable allows:

- [Channel](#)
- [Server](#)
- [Message](#)
- [User](#) (in some instances)
- String of Channel ID
- String of User ID

### 1.22.2 File Resolvable

A File Resolvable allows:

- URL
- Local file path
- Readable stream

### 1.22.3 Role Resolvable

A Role Resolvable allows:

- Role ID
- Role

### 1.22.4 Voice Channel Resolvable

A Voice Channel Resolvable allows:

- VoiceChannel
- Voice Channel ID

### 1.22.5 Message Resolvable

A Message Resolvable allows:

- Message
- TextChannel
- PMChannel

### 1.22.6 User Resolvable

A User Resolvable allows:

- User
- Message
- TextChannel
- PMChannel
- Server
- String of User ID

### 1.22.7 String Resolvable

A String Resolvable allows:

- Array
- String

### 1.22.8 Server Resolvable

A Server Resolvable allows:

- Server
- ServerChannel
- Message (only for messages from server channels)

- String of Server ID

### 1.22.9 Invite ID Resolvable

An Invite ID Resolvable allows:

- [Invite](#)
- String containing either a http link to the invite or the invite code on its own.

### 1.22.10 Base64 Resolvable

A Base64 Resolvable allows:

- Buffer
- String