discord.js Documentation

Release 8.0.0

hydrabolt

July 14, 2016

1 Installing discord.js 3

2 Updating to v5.0.0 5

3 Troubleshooting 7

4 Client 9

5 Server 31

6 User 35

7 Message 39

8 Invite 43

9 VoiceConnection 45

10 Channel 49

11 PMChannel 51

12 ServerChannel 53

13 TextChannel 55

14 VoiceChannel 57

15 Permission Constants 59

16 Role 61

17 PermissionOverwrite 65

18 ChannelPermissions 67

19 Cache 69

20 Equality 71

21 Resolvables 73

discord.js 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'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!

CHAPTER 1

Installing discord.js

To install discord.js, 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 discord.js should still work fine.

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, Enteprise** 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 discord.js` to install the latest version of discord.js for your project.

1.1.1 Additional Audio Support

- Install ffmpeg and add it to your PATH.

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 discord.js
```

1.2.1 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 ffmp
```

CHAPTER 2

Updating to v5.0.0

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

2.1 Change 1

```
// OLD:

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

// NEW:

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

2.2 Change 2

```
// OLD:

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

});

// NEW:

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

});
```

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)`.

CHAPTER 3

Troubleshooting

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.

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 reinstallin**
  - **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 discord.js`

If nothing of the above helped, feel free to jump on the discord.js channel in the API server

CHAPTER 4

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.

4.1 Parameters

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

4.1.1 autoReconnect

Have discord.js autoreconnect when connection is lost.

4.1.2 compress

Have Discord send a compressed READY packet.

4.1.3 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.

4.1.4 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.

4.1.5 largeThreshold

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

4.1.6 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.

4.1.7 rateLimitAsError

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

4.1.8 disableEveryone

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

4.1.9 shardCount

The total number of shards.

4.1.10 shardId

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

4.2 Attributes

4.2.1 users

A Cache of User objects that the client has cached.

4.2.2 channels

A Cache of ServerChannel objects that the client has cached.

4.2.3 privateChannels

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

4.2.4 servers

A Cache of Server objects that the client has cached.

4.2.5 unavailableServers

A Cache of Server objects that the client has cached that are unavailable.

4.2.6 voiceConnections

A Cache of VoiceConnection objects that the client is in.

4.2.7 voiceConnection

Returns a VoiceConnection object, is an alias to voiceConnections[0].

4.2.8 readyTime

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

4.2.9 uptime

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

4.2.10 user

A User object representing the logged in client's user.

4.2.11 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 discord.js 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.

4.3 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
});
```

4.3.1 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*.

4.3.2 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.

4.3.3 logout(*callback*)

Logs the client out and closes the WebSocket connections.

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

4.3.4 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

4.3.5 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

4.3.6 sendTTSMessage(channel, content, *callback*)

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

4.3.7 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

4.3.8 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

4.3.9 replyTTS(message, content, *callback*)

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

4.3.10 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

4.3.11 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

4.3.12 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

4.3.13 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

4.3.14 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

4.3.15 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

4.3.16 pinMessage(message, *callback*)

Pins a message to a channel.

- **message** - The Message to pin.
- **callback -** ***function*** **taking the following:**
  - **error** - error if any occurred

4.3.17 unpinMessage(message, *callback*)

Unpins a message from a channel.

- **message** - The Message to unpin.
- **callback -** ***function*** **taking the following:**
  - **error** - error if any occurred

4.3.18 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.

4.3.19 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

4.3.20 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

4.3.21 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

4.3.22 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

4.3.23 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

4.3.24 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

4.3.25 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

4.3.26 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.

4.3.27 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.

4.3.28 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.

4.3.29 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.

4.3.30 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.

4.3.31 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

4.3.32 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

4.3.33 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

4.3.34 deleteInvite(invite, *callback*)

Deletes an invite

- **invite** - An Invite Resolvable
- **callback - a *function* taking the following:**
  - **error** - error if any occurred

4.3.35 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

4.3.36 setStatusIdle()

Alias: *setStatusAway*

Sets the status of the Client to Idle/Away

4.3.37 setStatusOnline()

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

Sets the status of the Client to Online

4.3.38 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

4.3.39 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

4.3.40 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

4.3.41 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

4.3.42 setChannelNameAndTopic(channel, name, topic, *callback*)

Sets the name and topic of a channel

- **channel** - A Channel Resolvable
- **name** - A *String*
- **topic** - A *String*
- **callback - *function* taking the following:**
  - **error** - error if any occurred

4.3.43 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

4.3.44 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

4.3.45 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

4.3.46 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

4.3.47 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

4.3.48 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

4.3.49 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

4.3.50 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

4.3.51 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.

4.3.52 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

4.3.53 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
}
```

4.3.54 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
}
```

4.3.55 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

4.3.56 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

4.3.57 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

4.3.58 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

4.3.59 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
}
```

4.3.60 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.

4.3.61 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.

4.3.62 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.

4.3.63 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.

4.3.64 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.

4.3.65 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.

4.4 Events

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

4.4.1 ready

Emitted when the client is ready to use

4.4.2 debug

Emitted when the client debugs or wants to log something internally

4.4.3 message

Emitted when the client receives a message, supplies a Message object.

4.4.4 warn

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

4.4.5 messageDeleted

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

4.4.6 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.

4.4.7 disconnected

Emitted when the client is disconnected from the Discord server.

4.4.8 error

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

4.4.9 raw

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

4.4.10 serverCreated

Emitted when a server is joined by the Client, supplies a Server object.

4.4.11 serverDeleted

Emitted when the client leaves a server, supplies a Server object.

4.4.12 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.

4.4.13 channelCreated

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

4.4.14 channelDeleted

Emitted when a channel is deleted, supplies a Channel object.

4.4.15 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.

4.4.16 serverRoleCreated

Emitted when a role is created in a server, supplies a Role object.

4.4.17 serverRoleDeleted

Emitted when a role is deleted from a server, supplies a Role object.

4.4.18 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.

4.4.19 serverNewMember

Emitted when a user joins a server, supplies a Server object and a User object.

4.4.20 serverMemberRemoved

Emitted when a member is removed from a server. Supplies a Server object and a User object.

4.4.21 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.

4.4.22 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.

4.4.23 userTypingStarted

Emitted when a user starts typing in a channel. Supplies two parameters, a User object and a Channel object.

4.4.24 userTypingStopped

Emitted when a user stop typing in a channel. Supplies two parameters, a User object and a Channel object.

4.4.25 userBanned

Emitted when a user is banned from a server. Supplies two parameters, a User object and a Server object.

4.4.26 userUnbanned

Emitted when a user is unbanned from a server. Supplies two parameters, a User object and a Server object.

4.4.27 noteUpdated

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

4.4.28 voiceJoin

Emitted when a user joins a voice channel, supplies a VoiceChannel and a User.

4.4.29 voiceSwitch

Emitted when a user switches voice channels, supplies the old VoiceChannel, the new VoiceChannel, and a User.

4.4.30 voiceLeave

Emitted when a user leaves a voice channel, supplies a VoiceChannel and a User.

4.4.31 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.

4.4.32 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.

CHAPTER 5

Server

extends Equality

Stores information about a Discord Server.

5.1 Attributes

5.1.1 client

The Client that cached the Server.

5.1.2 region

String, region of the server.

5.1.3 name

String, name of the server.

5.1.4 id

String, ID of the server - never changes.

5.1.5 members

Members of the server, a Cache of User objects.

5.1.6 channels

Channels in the server, a Cache of ServerChannel objects.

5.1.7 roles

Roles of the server, a Cache of Role objects.

5.1.8 icon

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

5.1.9 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.

5.1.10 afkChannel

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

5.1.11 defaultChannel

Aliases *generalChannel, general*

The `#general` ServerChannel of the server.

5.1.12 owner

The founder of the server, a User object.

5.1.13 iconURL

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

5.1.14 createdAt

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

5.2 Functions

5.2.1 rolesOfUser(user)

Aliases: *rolesOf*, *rolesOfMember*

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

5.2.2 usersWithRole(role)

Aliases: *membersWithRole*

Returns an array of users that have the specified role.

5.2.3 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'
}
```

5.2.4 leave()

Shortcut of `client.leaveServer(server)`
Aliases *delete*
See client.leaveServer

5.2.5 createInvite(*options*, *callback*)

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

5.2.6 createRole(*data*, *callback*)

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

5.2.7 createChannel(name, *type*, *callback*)

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

5.2.8 getBans(*callback*)

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

5.2.9 banMember(user, *length*, *callback*)

Shortcut of `client.banMember(member, server, length, callback)`
Aliases *banUser, ban*
See client.banMember

5.2.10 unbanMember(user, *callback*)

Shortcut of `client.unbanMember(member, server, callback)`
Aliases *unbanUser, unban*
See client.unbanMember

5.2.11 kickMember(user, *callback*)

Shortcut of `client.kickMember(member, server, callback)`
Aliases *kickUser, kick*
See client.kickMember

5.2.12 setNickname(nickname, user, *callback*)

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

CHAPTER 6

User

extends Equality

Stores information about users.

6.1 Attributes

6.1.1 client

The Client that created the user.

6.1.2 username

Alias : `name`

String, username of the User.

6.1.3 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`.

6.1.4 id

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

6.1.5 avatar

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

6.1.6 status

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

6.1.7 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
}
```

6.1.8 typing

Object containing the following values:

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

6.1.9 avatarURL

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

6.1.10 bot

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

6.1.11 voiceChannel

The VoiceChannel the user is connected to. If they aren't in any voice channels, this will be `null`.

6.1.12 createdAt

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

6.1.13 note

The note of the user, *String*.

6.1.14 speaking

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

6.2 Functions

6.2.1 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.

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

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

6.2.3 sendTTSMessage(content, *callback*)

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

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

Shortcut of `client.sendFile(channel, attachment, name, content, callbasck)`
See client.sendFile

6.2.5 startTyping(*callback*)

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

6.2.6 stopTyping(*callback*)

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

6.2.7 addTo(role, *callback*)

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

6.2.8 removeFrom(role, *callback*)

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

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

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

6.2.10 getMessage(messageID, *callback*)

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

6.2.11 hasRole(role)

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

CHAPTER 7

Message

extends Equality

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

7.1 Attributes

7.1.1 channel

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

7.1.2 server

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

7.1.3 client

The Client that cached the message.

7.1.4 attachments

A raw array of attachment objects.

7.1.5 tts

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

7.1.6 embeds

A raw array of embed objects.

7.1.7 timestamp

Number, timestamp of when the message was sent.

7.1.8 everyoneMentioned

Boolean, true if `@everyone` was mentioned.

7.1.9 id

String, ID of the message.

7.1.10 editedTimestamp

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

7.1.11 author

Alias: *sender*

The User that sent the message.

7.1.12 content

String, content of the message.

7.1.13 cleanContent

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

7.1.14 mentions

A array of User objects that were mentioned in the message.

7.1.15 pinned

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

7.2 Functions

7.2.1 isMentioned(user)

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

- **user** - A User Resolvable

7.2.2 toString()

Returns the content of the Message.

7.2.3 delete(*options*, *callback*)

Shortcut of `client.deleteMessage(message, options, callback`
See client.deleteMessage

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

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

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

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

7.2.6 replyTTS(content, *callback*)

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

7.2.7 pin(*callback*)

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

7.2.8 unpin(*callback*)

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

CHAPTER 8

Invite

Used to represent data of an invite.

8.1 Attributes

8.1.1 maxAge

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

8.1.2 code

String, the invite code.

8.1.3 server

The Server the invite is for.

8.1.4 channel

The ServerChannel the invite is for.

8.1.5 revoked

Boolean, whether the invite has been revoked or not.

8.1.6 createdAt

Number, timestamp of when the invite was created.

8.1.7 temporary

Boolean, whether the invite is temporary or not.

8.1.8 uses

Number, uses of the invite remaining.

8.1.9 maxUses

Number, maximum uses of the invite.

8.1.10 inviter

User who sent/created the invite.

8.1.11 xkcd

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

8.2 Functions

8.2.1 toString()

Returns the invite URL.

8.2.2 delete(*callback*)

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

8.2.3 join(*callback*)

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

CHAPTER 9

VoiceConnection

discord.js 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 discord.js does not support receiving data from voice yet, only sending.

9.1 Attributes

9.1.1 voiceChannel

VoiceChannel that the connection is for

9.1.2 client

Client the connection belongs to

9.1.3 token

The token used to authenticate with Discord

9.1.4 server

The Server on which the voice connection takes place

9.1.5 encoder

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

9.1.6 playingIntent

A stream intent used to bind events to the voice connection

9.1.7 playing

Whether or not the bot is currently playing something

9.1.8 paused

Whether or not the playback is currently paused

9.1.9 streamTime

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

9.2 Functions

9.2.1 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.

9.2.2 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.

9.2.3 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
```

9.2.4 setSpeaking(value)

Sets whether or not the user is speaking (green circle around user on the official client). discord.js 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

9.2.5 setVolume(volume)

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

9.2.6 getVolume()

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

9.2.7 pause()

Pauses the current connection's audio.

9.2.8 resume()

Resumes the current connection's audio.

9.2.9 stopPlaying()

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

9.2.10 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.

CHAPTER 10

Channel

extends Equality

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

10.1 Attributes

10.1.1 id

The ID of the channel, a *String*.

10.1.2 client

The Client that cached the channel.

10.1.3 isPrivate

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

10.1.4 createdAt

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

10.2 Functions

10.2.1 delete()

Deletes the channel.

CHAPTER 11

PMChannel

extends Channel

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

11.1 Attributes

11.1.1 messages

A Cache of Message objects.

11.1.2 recipient

The User that is the recipient of the Channel.

11.1.3 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.

11.2 Functions

11.2.1 toString()

Returns a mention of the recipient.

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

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

11.2.3 sendTTSMessage(content, *callback*)

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

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

Shortcut of `client.sendFile(channel, attachment, name, content, callbasck)`
See client.sendFile

11.2.5 startTyping(*callback*)

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

11.2.6 stopTyping(*callback*)

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

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

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

11.2.8 getMessage(messageID, *callback*)

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

CHAPTER 12

ServerChannel

extends Channel

A ServerChannel is a Channel that belongs to a Server.

12.1 Attributes

12.1.1 name

String, name of the channel.

12.1.2 type

String, either `voice` or `text`.

12.1.3 position

Number, position in the channel list.

12.1.4 permissionOverwrites

Cache of all the PermissionOverwrite objects affecting the channel.

12.1.5 server

Server the channel belongs to.

12.2 Functions

12.2.1 permissionsOf(userOrRole)

Aliases: permsOf

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

12.2.2 mention()

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

12.2.3 update(data, *callback*)

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

CHAPTER 13

TextChannel

extends ServerChannel

A text channel of a server.

13.1 Attributes

13.1.1 topic

The topic of the channel, a *String*.

13.1.2 lastMessage

Last Message sent in the channel. May be null if no messages sent whilst the Client was online.

13.1.3 messages

A Cache of Message objects.

13.2 Functions

13.2.1 setTopic(topic, *callback*)

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

13.2.2 setNameAndTopic(name, topic, *callback*)

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

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

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

13.2.4 sendTTSMessage(content, *callback*)

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

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

Shortcut of `client.sendFile(channel, attachment, name, content, callbasck)`
See client.sendFile

13.2.6 startTyping(*callback*)

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

13.2.7 stopTyping(*callback*)

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

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

Shortcut of ‘‘client.getChannelLogs(channel, limit, options, callback)
See client.getChannelLogs

13.2.9 getMessage(messageID, *callback*)

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

CHAPTER 14

VoiceChannel

extends ServerChannel

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

14.1 Attributes

14.1.1 members

A Cache of Users that are connected to the voice channel

14.1.2 userLimit

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

14.1.3 bitrate

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

14.2 Functions

14.2.1 setUserLimit(limit, *callback*)

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

14.2.2 setBitrate(kbitrate, *callback*)

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

14.2.3 join(*callback*)

Shortcut of `client.joinVoiceChannel(channel, callback)`
See client.joinVoiceChannel_

CHAPTER 15

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"]`.

15.1 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
};
```

15.2 Preferred Way

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

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

15.3 Alternative

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

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

CHAPTER 16

Role

Represents data for a Server Role.

16.1 Attributes

16.1.1 position

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

16.1.2 name

String, name of the role.

16.1.3 managed

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

16.1.4 id

String, ID of the role.

16.1.5 hoist

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

16.1.6 color

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

16.1.7 server

The Server the role belongs to.

16.1.8 client

The Client that cached the role.

16.1.9 createdAt

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

16.2 Functions

16.2.1 serialise()

Aliases: *serialize*

Makes an object with the permission names found in Permission Constants and a boolean value for them.

16.2.2 hasPermission(permission)

Sees whether the role has the permission given.

- **permission** - See Permission Constants for valid permission names.

16.2.3 colorAsHex()

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

16.2.4 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.

16.2.5 delete()

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

16.2.6 update(data)

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

16.2.7 addMember(member, *callback*)

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

16.2.8 removeMember(member, *callback*)

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

CHAPTER 17

PermissionOverwrite

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

17.1 Attributes

17.1.1 id

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

17.1.2 type

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

17.1.3 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

17.1.4 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

CHAPTER 18

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");
```

18.1 Functions

18.1.1 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.

18.1.2 hasPermission(permission)

Sees whether the user has the permission given.

- **permission** - See Permission Constants for valid permission names.

CHAPTER 19

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 discord.js. 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");
```

19.1 Functions

19.1.1 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.

19.1.2 get(value)

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

19.1.3 getAll(key, value)

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

19.1.4 has(key, value)

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

19.1.5 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.

19.1.6 update(old, data)

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

19.1.7 remove(data)

Removes an object from the cache if it exists.

19.1.8 random()

Get a random object from the cache.

CHAPTER 20

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
```

20.1 Functions

20.1.1 equals(object)

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

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

CHAPTER 21

Resolvables

In discord.js, 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:

21.1 Channel Resolvable

A Channel Resolvable allows:

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

21.2 File Resolvable

A File Resolvable allows:

- URL
- Local file path
- Readable stream

21.3 Role Resolvable

A Role Resolvable allows:

- Role ID
- Role

21.4 Voice Channel Resolvable

A Voice Channel Resolvable allows:

- VoiceChannel
- Voice Channel ID

21.5 Message Resolvable

A Message Resolvable allows:

- Message
- TextChannel
- PMChannel

21.6 User Resolvable

A User Resolvable allows:

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

21.7 String Resolvable

A String Resolvable allows:

- Array
- String

21.8 Server Resolvable

A Server Resolvable allows:

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

21.9 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.

21.10 Base64 Resolvable

A Base64 Resolvable allows:

- Buffer
- String