discord.py Documentation

Release 0.9.2

Rapptz

January 14, 2016

Contents

Contents:

New in version 0.6.0.

CHAPTER 1

Setting Up Logging

discord.py logs errors and debug information via the logging python module. It is strongly recommended that the logging module is configured, as no errors or warnings will be output if it is not set up. Configuration of the `logging` module can be as simple as:

```
import logging

logging.basicConfig(level=logging.INFO)
```

Placed at the start of the application. This will output the logs from discord as well as other libraries that uses the `logging` module directly to the console.

The optinal `level` argument specifies what level of events to log out and can any of `CRITICAL`, `ERROR`, `WARNING`, `INFO`, and `DEBUG` and if not specified defaults to `WARNING`.

More advance setups are possible with the `logging` module. To for example write the logs to a file called `discord.log` instead of outputting them to to the console the following snippet can be used:

```
import discord
import logging

logger = logging.getLogger('discord')
logger.setLevel(logging.DEBUG)
handler = logging.FileHandler(filename='discord.log', encoding='utf-8', mode='w')
handler.setFormatter(logging.Formatter('%(asctime)s:%(levelname)s:%(name)s: %(message)s'))
logger.addHandler(handler)
```

This is recommended, especially at verbose levels such as `INFO`, and `DEBUG` as there are a lot of events logged and it would clog the stdout of your program.

For more information, check the documentation and tutorial of the logging module.

CHAPTER 2

API Reference

The following section outlines the API of discord.py.

Note: This module uses the Python logging module to log diagnostic and errors in an output independent way. If the logging module is not configured, these logs will not be output anywhere. See *Setting Up Logging* for more information on how to set up and use the logging module with discord.py.

2.1 Client

class discord.**Client**(***kwargs*)

Represents a client connection that connects to Discord. This class is used to interact with the Discord WebSocket and API.

A number of options can be passed to the *Client* via keyword arguments.

Parameters **max_length** (*int*) – The maximum number of messages to store in *messages*. Defaults to 5000.

Instance attributes:

user
A *User* that represents the connected client. None if not logged in.

servers
A list of *Server* that the connected client has available.

private_channels
A list of *PrivateChannel* that the connected client is participating on.

messages
A deque of *Message* that the client has received from all servers and private messages.

email
The email used to login. This is only set if login is successful, otherwise it's None.

accept_invite(*invite*)

Accepts an *Invite*, URL or ID to an invite.

The URL must be a discord.gg URL. e.g. "http://discord.gg/codehere". An ID for the invite is just the "codehere" portion of the invite URL.

This function raises *HTTPException* if the request failed. If the invite is invalid, then *InvalidArgument* is raised.

Parameters **invite** – The *Invite* or URL to an invite to accept.

add_roles(*member*, **roles*)
Gives the specified *Member* a number of *Role* s.

You must have the proper permissions to use this function. This function raises *HTTPException* if the request failed.

This method **appends** a role to a member.

Parameters

- **member** – The *Member* to give roles to.
- **roles** – An argument list of *Role* s to give the member.

ban(*server*, *user*)
Bans a *User* from their respective *Server*.

You must have the proper permissions to ban a user in the server.

This function raises *HTTPException* if the request failed.

Parameters

- **server** – The *Server* to ban the member from.
- **user** – The *User* to ban.

change_status(*game=None*, *idle=False*)
Changes the client's status.

The game parameter is a Game object that represents a game being played currently. May be None if no game is being played.

The idle parameter is a boolean parameter that indicates whether the client should go idle or not.

Parameters

- **game** – A Game object representing the game being played. None if no game is being played.
- **idle** – A boolean indicating if the client should go idle.

create_channel(*server*, *name*, *type='text'*)
Creates a *Channel* in the specified *Server*.

Note that you need the proper permissions to create the channel.

This function raises *HTTPException* if the request failed.

Parameters

- **server** – The *Server* to create the channel in.
- **name** – The channel's name.
- **type** – The type of channel to create. 'text' or 'voice'.

Returns The newly created *Channel*.

create_invite(*destination*, ***options*)
Creates an invite for the destination which could be either a *Server* or *Channel*.

This function raises *HTTPException* if the request failed.

The available options are:

Parameters

- **destination** – The *Server* or *Channel* to create the invite to.
- **max_age** – How long the invite should last. If it's 0 then the invite doesn't expire. Defaults to 0.
- **max_uses** – How many uses the invite could be used for. If it's 0 then there are unlimited uses. Defaults to 0.
- **temporary** – A boolean to denote that the invite grants temporary membership (i.e. they get kicked after they disconnect). Defaults to False.
- **xkcd** – A boolean to indicate if the invite URL is human readable. Defaults to False.

Returns The *Invite* if creation is successful.

create_role(*server*, ***fields*)
Creates a *Role*.

The fields parameter is the same as *edit_role()*.

This function raises *HTTPException* if the request failed.

Returns The *Role* that was created.

delete_channel(*channel*)
Deletes a channel.

In order to delete the channel, the client must have the proper permissions in the server the channel belongs to.

This function raises *HTTPException* if the request failed.

Parameters **channel** – The *Channel* to delete.

delete_channel_permissions(*channel*, *target*)
Removes a channel specific permission overwrites for a target in the specified *Channel*.

The target parameter follows the same rules as *set_channel_permissions()*.

You must have the proper permissions to do this. This function raises *HTTPException* if the request failed.

Parameters

- **channel** – The *Channel* to give the specific permissions for.
- **target** – The *Member* or *Role* to overwrite permissions for.

delete_message(*message*)
Deletes a *Message*.

Your own messages could be deleted without any proper permissions. However to delete other people's messages, you need the proper permissions to do so.

This function raises *HTTPException* if the request failed.

Parameters **message** – The *Message* to delete.

delete_role(*server*, *role*)
Deletes the specified *Role* for the entire *Server*.

Works in a similar matter to *edit_role()*. This function raises *HTTPException* if the request failed.

Parameters

- **server** – The *Server* the role belongs to.
- **role** – The *Role* to delete.

edit_channel(*channel*, ***options*)
Edits a *Channel*.

You must have the proper permissions to edit the channel.

References pointed to the channel will be updated with the new information.

This function raises *HTTPException* if the request failed.

Parameters

- **channel** – The *Channel* to update.
- **name** – The new channel name.
- **position** – The new channel's position in the GUI.
- **topic** – The new channel's topic.

edit_message(*message*, *new_content*, *mentions=True*)
Edits a *Message* with the new message content.

The new_content must be able to be transformed into a string via `str(new_content)`.

This function raises *HTTPException* if the request failed.

Parameters

- **message** – The *Message* to edit.
- **new_content** – The new content to replace the message with.
- **mentions** – The mentions for the user. Same as *send_message()*.

Returns The new edited message.

edit_profile(*password*, ***fields*)
Edits the current profile of the client.

All fields except password are optional.

This function raises *HTTPException* if the request failed.

To upload an avatar, a *bytes-like object* must be passed in that represents the image being uploaded. If this is done through a file then the file must be opened via `open('some_filename', 'rb')` and the *bytes-like object* is given through the use of `fp.read()`.

The only image formats supported for uploading is JPEG and PNG.

Parameters

- **password** – The current password for the client's account.
- **new_password** – The new password you wish to change to.
- **email** – The new email you wish to change to.
- **username** – The new username you wish to change to.
- **avatar** – A *bytes-like object* representing the image to upload.

edit_role(*server*, *role*, ***fields*)
Edits the specified *Role* for the entire *Server*.

This function raises *HTTPException* if the request failed.

Changed in version 0.8.0: Editing now uses keyword arguments instead of editing the *Role* object directly.

Note: At the moment, the Discord API allows you to set the colour to any RGB value. This will

change in the future so it is recommended that you use the constants in the *Colour* instead such as *Colour.green()*.

Parameters

- **server** – The *Server* the role belongs to.
- **role** – The *Role* to edit.
- **name** – The new role name to change to. (optional)
- **permissions** – The new *Permissions* to change to. (optional)
- **colour** – The new *Colour* to change to. (optional) (aliased to color as well)
- **hoist** – A boolean indicating if the role should be shown separately. (optional)

event (*function*)

A decorator that registers an event to listen to.

You can find more info about the events on the *documentation below*.

Example:

```
@client.event
def on_ready():
    print('Ready!')
```

get_all_channels()

Returns a generator with every *Channel* the client can 'access'.

This is equivalent to:

```
for server in client.servers:
    for channel in server.channels:
        yield channel
```

Note that just because you receive a *Channel* does not mean that you can communicate in said channel. *Channel.permissions_for()* should be used for that.

get_all_members()

Returns a generator with every *Member* the client can see.

This is equivalent to:

```
for server in client.servers:
    for member in server.members:
        yield member
```

get_channel(*id*)

Returns a *Channel* or *PrivateChannel* with the following ID. If not found, returns None.

get_invite(*url*)

Returns a *Invite* object from the discord.gg invite URL or ID.

Note: If the invite is for a server you have not joined, the server and channel attributes of the returned invite will be *Object* with the names patched in.

---

is_logged_in

Returns True if the client is successfully logged in. False otherwise.

kick(*server*, *user*)
Kicks a *User* from their respective *Server*.

You must have the proper permissions to kick a user in the server.

This function raises *HTTPException* if the request failed.

Parameters

- **server** – The *Server* to kick the member from.
- **user** – The *User* to kick.

leave_server(*server*)
Leaves a *Server*.

This function raises *HTTPException* if the request failed.

Parameters **server** – The *Server* to leave.

login(*email*, *password*)
Logs in the user with the following credentials and initialises the connection to Discord.

After this function is called, *is_logged_in* returns True if no errors occur. If an error occurs during the login process, then *LoginFailure* or *HTTPException* is raised.

This function raises *GatewayNotFound* if it was unavailable to connect to a websocket gateway.

Parameters

- **email** (*str*) – The email used to login.
- **password** (*str*) – The password used to login.

logout()
Logs out of Discord and closes all connections.

logs_from(*channel*, *limit=100*, *before=None*, *after=None*)
A generator that obtains logs from a specified channel.

Yielding from the generator returns a *Message* object with the message data.

Will return the newest messages within the specified range, up to *limit* messages.

This function raises *HTTPException* if the request failed.

Example:

```
for message in client.logs_from(channel):
    if message.content.startswith('!hello'):
        if message.author == client.user:
            client.edit_message(message, 'goodbye')
```

Parameters

- **channel** – The *Channel* to obtain the logs from.
- **limit** – The number of messages to retrieve.
- **before** – *Message* before which all returned messages must be.
- **after** – *Message* after which all returned messages must be.

register(*username*, *invite*, *fingerprint=None*)
Register a new unclaimed account using an invite to a server.

After this function is called, the client will be logged in to the user created and *is_logged_in* returns True if no errors occur.

This function raises *GatewayNotFound* if the gateway to connect the websocket is not found. It also raises *HTTPException* if the request failed.

Parameters

- **username** (*str*) – The username to register as.
- **invite** – An invite URL, ID, or *Invite* to register with.
- **fingerprint** (*str*) – Unknown API parameter, defaults to None

remove_roles(*member*, **roles*)
Removes the *Role* s from the *Member*.

You must have the proper permissions to use this function. This function raises *HTTPException* if the request failed.

Parameters

- **member** – The *Member* to remove roles from.
- **roles** – An argument list of *Role* s to remove from the member.

replace_roles(*member*, **roles*)
Replaces the *Member*'s roles.

You must have the proper permissions to use this function.

This function **replaces** all roles that the member has. For example if the member has roles `[a, b, c]` and the call is `client.replace_roles(member, d, e, c)` then the member has the roles `[d, e, c]`.

This function raises *HTTPException* if the request failed.

Parameters

- **member** – The *Member* to replace roles for.
- **roles** – An argument list of *Role* s to replace with.

run()
Runs the client and allows it to receive messages and events.

This function can raise a *GatewayNotFound* exception while attempting to reconnect.

Note: This function attempts to reconnect if the websocket got closed without explicitly calling *logout()*. When this reconnect is triggered, the *discord.on_ready()* event is called again.

---

send_file(*destination*, *fp*, *filename=None*)
Sends a message to the destination given with the file given.

The destination parameter follows the same rules as *send_message()*.

The `fp` parameter should be either a string denoting the location for a file or a *file-like object*. The *file-like object* passed is **not closed** at the end of execution. You are responsible for closing it yourself.

Note: If the file-like object passed is opened via `open` then the modes 'rb' should be used.

The `filename` parameter is the filename of the file. If this is not given then it defaults to `fp.name` or if `fp` is a string then the `filename` will default to the string given. You can overwrite this value by passing this in.

Note that this requires proper permissions in order to work. This function raises *HTTPException* if the request failed. It also raises *InvalidArgument* if `fp.name` is an invalid default for `filename`.

Parameters

- **destination** – The location to send the message.
- **fp** – The *file-like object* or file path to send.
- **filename** – The filename of the file. Defaults to `fp.name` if it's available.

Returns The *Message* sent.

send_message(*destination*, *content*, *mentions=True*, *tts=False*)

Sends a message to the destination given with the content given.

The destination could be a *Channel*, *PrivateChannel* or *Server*. For convenience it could also be a *User*. If it's a *User* or *PrivateChannel* then it sends the message via private message, otherwise it sends the message to the channel. If the destination is a *Server* then it's equivalent to calling *Server.get_default_channel()* and sending it there. If it is a *Object* instance then it is assumed to be the destination ID.

Changed in version 0.9.0: `str` being allowed was removed and replaced with *Object*.

The content must be a type that can convert to a string through `str(content)`.

The mentions must be either an array of *User* to mention or a boolean. If `mentions` is `True` then all the users mentioned in the content are mentioned, otherwise no one is mentioned. Note that to mention someone in the content, you should use *User.mention()*.

If the destination parameter is invalid, then this function raises *InvalidArgument*. This function raises *HTTPException* if the request failed.

Parameters

- **destination** – The location to send the message.
- **content** – The content of the message to send.
- **mentions** – A list of *User* to mention in the message or a boolean. Ignored for private messages.
- **tts** – If `True`, sends tries to send the message using text-to-speech.

Returns The *Message* sent.

send_typing(*destination*)

Send a "typing" status to the destination.

"Typing" status will go away after 10 seconds, or after a message is sent.

The destination parameter follows the same rules as *send_message()*.

Parameters **destination** – The location to send the typing update.

set_channel_permissions(*channel*, *target*, *allow=None*, *deny=None*)

Sets the channel specific permission overwrites for a target in the specified *Channel*.

The `target` parameter should either be a *Member* or a *Role* that belongs to the channel's server.

You must have the proper permissions to do this.

This function raises *HTTPException* if the request failed. This function also raises *InvalidArgument* if invalid arguments are passed to this function.

Example code:

```
        allow = discord.Permissions.none()
        deny = discord.Permissions.none()
        allow.can_mention_everyone = True
        deny.can_manage_messages = True
        client.set_channel_permissions(message.channel, message.author, allow, deny)
```

Parameters

- **channel** – The *Channel* to give the specific permissions for.
- **target** – The *Member* or *Role* to overwrite permissions for.
- **allow** – A *Permissions* object representing the permissions to explicitly allow. (optional)
- **deny** – A *Permissions* object representing the permissions to explicitly deny. (optional)

start_private_message(*user*)

Starts a private message with the user. This allows you to *send_message()* to it.

Note that this method should rarely be called as *send_message()* does it automatically.

This function raises *HTTPException* if the request failed.

Parameters **user** – A *User* to start the private message with.

unban(*server*, *user*)

Unbans a *User* from their respective *Server*.

You must have the proper permissions to unban a user in the server.

This function raises *HTTPException* if the request failed.

Parameters

- **server** – The *Server* to unban the member from.
- **user** – The *User* to unban.

Returns True if unban was successful, False otherwise.

2.1.1 Event Reference

This page outlines the different types of events listened by *Client*.

There are two ways to register an event, the first way is through the use of *Client.event()*. The second way is through subclassing *Client* and overriding the specific events. For example:

```
import discord

class MyClient(discord.Client):
    def on_message(self, message):
        self.send_message(message.channel, 'Hello World!')
```

If an event handler raises an exception, *on_error()* will be called to handle it, which defaults to print a traceback and ignore the exception.

New in version 0.7.0: Subclassing to listen to events.

discord.**on_ready**()
Called when the client is done preparing the data received from Discord. Usually after login is successful and the *Client.servers* and co. are filled up.

discord.**on_error**(*event*, **args*, ***kwargs*)
Usually when an event raises an uncaught exception, a traceback is printed to stderr and the exception is ignored. If you want to change this behaviour and handle the exception for whatever reason yourself, this event can be overridden. Which, when done, will supress the default action of printing the traceback.

The information of the exception rasied and the exception itself can be retreived with a standard call to `sys.exc_info()`.

If you want exception to propogate out of the *Client* class you can define an `on_error` handler consisting of a single empty `raise` statement. Exceptions raised by `on_error` will not be handled in any way by *Client*.

Parameters

- **event** – The name of the event that raised the exception.
- **args** – The positional arguments for the event that raised the exception.
- **kwargs** – The keyword arguments for the event that raised the execption.

discord.**on_message**(*message*)
Called when a message is created and sent to a server.

Parameters **message** – A *Message* of the current message.

discord.**on_socket_opened**()
Called whenever the websocket is successfully opened. This is not the same thing as being ready. For that, use *on_ready()*.

discord.**on_socket_closed**()
Called whenever the websocket is closed, through an error or otherwise.

discord.**on_socket_update**(*event*, *data*)
Called whenever a recognised websocket event is found. This function would normally be not be called as there are higher level events in the library such as *on_message()*.

Parameters

- **event** (*str*) – The string of the event received. e.g. `READY`.
- **data** – The data associated with the socket event. Usually a `dict`.

discord.**on_socket_response**(*response*)
Called whenever a message is received from the websocket. Used mainly for debugging purposes. The parameter passed is raw data that was parsed via `json.loads`. Note that this is called before the *Client* processes the event.

Parameters **response** – The received message response after gone through `json.loads`.

discord.**on_socket_raw_receive**(*msg*)
Called whenever a message is received from the websocket, before it's processed. Unlike `on_socket_response` this event is always dispatched when a message is received and the passed data is not processed in any way.

This is only really useful for grabing the websocket stream and debugging purposes.

Parameters **msg** – The message passed on from the ws4py library. Can be an instance of either ws4py.messaging.TextMessage, or ws4py.messaging.BinaryMessage.

discord.**on_socket_raw_send**(*payload*, *binary=False*)

Called whenever a send operation is done on the websocket before the message is sent. The passed parameter is the message that is to sent to the websocket.

This is only really useful for grabing the websocket stream and debugging purposes.

Note: If the `payload` parameter is mutable, and modified during the execution of this event, then the actual data sent out on the websocket will be mangled. This is especially true if `payload` is a generator, as reading them modifies their state.

Parameters

- **payload** – The message that is about to be passed on to the ws4py library. It can be any of a string, a bytearray, an instance of ws4py.message.Message and a generator.
- **binary** (*bool*) – True if the message being sent out is marked as binary.

discord.**on_message_delete**(*message*)

discord.**on_message_edit**(*before*, *after*)

Called when a message is deleted or edited from any given server. If the message is not found in the *Client.messages* cache, then these events will not be called. This happens if the message is too old or the client is participating in high traffic servers. To fix this, increase the `max_length` option of *Client*.

Parameters

- **message** – A *Message* of the deleted message.
- **before** – A *Message* of the previous version of the message.
- **after** – A *Message* of the current version of the message.

discord.**on_status**(*member*, *old_game*, *old_status*)

Called whenever a *Member* changes their status or game playing status.

Parameters

- **member** – The *Member* who has had their status changed.
- **old_game_id** – The `Game` the member had before it changed.
- **old_status** – The status the member had before it changed.

discord.**on_channel_delete**(*channel*)

discord.**on_channel_create**(*channel*)

Called whenever a channel is removed or added from a server.

Note that you can get the server from *Channel.server*. *on_channel_create()* could also pass in a *PrivateChannel* depending on the value of *Channel.is_private*.

Parameters **channel** – The *Channel* that got added or deleted.

discord.**on_channel_update**(*channel*)

Called whenever a channel is updated. e.g. changed name, topic, permissions.

Parameters **channel** – The *Channel* that got updated.

discord.**on_member_join**(*member*)

discord.**on_member_remove**(*member*)

Called when a *Member* leaves or joins a *Server*.

Parameters **member** – The *Member* that joined or left.

discord.**on_member_update**(*before*, *after*)
Called when a *Member* updates their profile.

This is called when one or more of the following things change:

•status

•game playing

•avatar

•nickname

Parameters

- **before** – The *Member* that updated their profile with the old info.
- **after** – The *Member* that updated their profile with the updated info.

discord.**on_server_join**(*server*)
Called when a *Server* is either created by the *Client* or when the *Client* joins a server.

Parameters **server** – The class:*Server* that was joined.

discord.**on_server_remove**(*server*)
Called when a *Server* is removed from the *Client*.

This happens through, but not limited to, these circumstances:

•The client got banned.

•The client got kicked.

•The client left the server.

•The client or the server owner deleted the server.

In order for this event to be invoked then the *Client* must have been part of the server to begin with. (i.e. it is part of *Client.servers*)

Parameters **server** – The *Server* that got removed.

discord.**on_server_role_create**(*server*, *role*)
discord.**on_server_role_delete**(*server*, *role*)
Called when a *Server* creates or deletes a new *Role*.

Parameters

- **server** – The *Server* that was created or deleted.
- **role** – The *Role* that was created or deleted.

discord.**on_server_role_update**(*role*)
Called when a *Role* is changed server-wide.

Parameters **role** – The *Role* that was updated.

discord.**on_server_available**(*server*)
discord.**on_server_unavailable**(*server*)
Called when a server becomes available or unavailable. The server must have existed in the *Client.servers* cache.

Parameters **server** – The *Server* that has changed availability.

discord.**on_voice_state_update**(*member*)
Called when a *Member* changes their voice state.

The following, but not limited to, examples illustrate when this event is called:

- A member joins a voice room.
- A member leaves a voice room.
- A member is muted or deafened by their own accord.
- A member is muted or deafened by a server administrator.

Parameters **member** – The *Member* whose voice state changed.

discord.**on_typing**(*channel*, *user*, *when*)
Called when someone begins typing a message.

The channel parameter could either be a *PrivateChannel* or a *Channel*. If channel is a *PrivateChannel* then the user parameter is a *User*, otherwise it is a *Member*.

Parameters

- **channel** – The location where the typing originated from.
- **user** – The user that started typing.
- **when** – A datetime.datetime object representing when typing started.

2.2 Utility Functions

discord.utils.**find**(*predicate*, *seq*)
A helper to return the first element found in the sequence that meets the predicate. For example:

```
member = find(lambda m: m.name == 'Mighty', channel.server.members)
```

would find the first *Member* whose name is 'Mighty' and return it.

This is different from filter due to the fact it stops the moment it finds a valid entry.

Parameters

- **predicate** – A function that returns a boolean-like result.
- **seq** – The sequence to iterate through.

Returns The first result of the predicate that returned a True-like value or None if nothing was found.

2.3 Data Classes

Some classes are just there to be data containers, this lists them.

Note: With the exception of *Object*, *Colour*, and *Permissions* the data classes listed below are **not intended to be created by users** and are also **read-only**.

For example, this means that you should not make your own *User* instances nor should you modify the *User* instance yourself.

If you want to get one of these data classes instances they'd have to be through the cache, and a common way of doing so is through the *utils.find()* function or attributes of data classes that you receive from the events specified in the *Event Reference*.

---

class discord.**Object**(*id*)
Represents a generic Discord object.

The purpose of this class is to allow you to create 'miniature' versions of data classes if you want to pass in just an ID. All functions that take in a specific data class with an ID can also take in this class as a substitute instead. Note that even though this is the case, not all objects (if any) actually inherit from this class.

There are also some cases where some websocket events are received in strange order and when such events happened you would receive this class rather than the actual data class. These cases are extremely rare.

id
The ID of the object.

class discord.**User**(*username*, *id*, *discriminator*, *avatar*, ***kwargs*)
Represents a Discord user.

Supported Operations:

Operation	Description
x == y	Checks if two users are equal.
x != y	Checks if two users are not equal.
str(x)	Returns the user's name.

Instance attributes:

name
The user's username.

id
The user's unique ID.

discriminator
The user's discriminator. This is given when the username has conflicts.

avatar
The avatar hash the user has. Could be None.

avatar_url()
Returns a friendly URL version of the avatar variable the user has. An empty string if the user has no avatar.

mention()
Returns a string that allows you to mention the given user.

class discord.**Message**(***kwargs*)
Represents a message from Discord.

There should be no need to create one of these manually.

Instance attributes:

edited_timestamp
A naive UTC datetime object containing the edited time of the message. Could be None.

timestamp
A naive UTC datetime object containing the time the message was created.

tts
A boolean specifying if the message was done with text-to-speech.

author
A *Member* that sent the message. If *channel* is a private channel, then it is a *User* instead.

content
The actual contents of the message.

embeds
A list of embedded objects. The elements are objects that meet oEmbed's specification.

channel
The *Channel* that the message was sent from. Could be a *PrivateChannel* if it's a private message. In very rare cases this could be a *Object* instead.

For the sake of convenience, this *Object* instance has an attribute `is_private` set to `True`.

server
The *Server* that the message belongs to. If not applicable (i.e. a PM) then it's None instead.

mention_everyone
A boolean specifying if the message mentions everyone.

Note: This does not check if the `@everyone` text is in the message itself. Rather this boolean indicates if the `@everyone` text is in the message **and** it did end up mentioning everyone.

---

mentions
A list of *Member* that were mentioned. If the message is in a private message then the list is always empty.

> **Warning:** The order of the mentions list is not in any particular order so you should not rely on it. This is a discord limitation, not one with the library.

channel_mentions
A list of *Channel* that were mentioned. If the message is in a private message then the list is always empty.

id
The message ID.

attachments
A list of attachments given to a message.

get_raw_channel_mentions()
Returns an array of channel IDs matched with the syntax of <#channel_id> in the message content.

This allows you receive the channel IDs of mentioned users even in a private message context.

get_raw_mentions()
Returns an array of user IDs matched with the syntax of <@user_id> in the message content.

This allows you receive the user IDs of mentioned users even in a private message context.

class discord.**Server**(***kwargs*)
Represents a Discord server.

Instance attributes:

name
The server name.

roles
A list of *Role* that the server has available.

region
The region the server belongs on.

afk_timeout
The timeout to get sent to the AFK channel.

afk_channel
The *Channel* that denotes the AFK channel. None if it doesn't exist.

members
A list of *Member* that are currently on the server.

channels
A list of *Channel* that are currently on the server.

icon
The server's icon.

id
The server's ID.

owner
The *Member* who owns the server.

unavailable
A boolean indicating if the server is unavailable. If this is `True` then the reliability of other attributes outside of *Server.id()* is slim and they might all be None. It is best to not do anything with the server if it is unavailable.

Check the *on_server_unavailable()* and *on_server_available()* events.

get_default_channel()
Gets the default *Channel* for the server.

get_default_role()
Gets the @everyone role that all members have by default.

icon_url()
Returns the URL version of the server's icon. Returns None if it has no icon.

class discord.**Member**(***kwargs*)
Represents a Discord member to a *Server*.

This is a subclass of *User* that extends more functionality that server members have such as roles and permissions.

Instance attributes:

deaf
A boolean that specifies if the member is currently deafened by the server.

mute
A boolean that specifies if the member is currently muted by the server.

self_mute
A boolean that specifies if the member is currently muted by their own accord.

self_deaf
A boolean that specifies if the member is currently deafened by their own accord.

is_afk
A boolean that specifies if the member is currently in the AFK channel in the server.

voice_channel
A voice *Channel* that the member is currently connected to. None if the member is not currently in a voice channel.

roles
A list of *Role* that the member belongs to. Note that the first element of this list is always the default '@everyone' role.

joined_at
A datetime object that specifies the date and time in UTC that the member joined the server for the first time.

status
A string that denotes the user's status. Can be 'online', 'offline' or 'idle'.

game
A dictionary representing the game that the user is currently playing. None if no game is being played.

server
The *Server* that the member belongs to.

class discord.**Colour**(*value*)
Represents a Discord role colour. This class is similar to an (red, green, blue) tuple.

There is an alias for this called Color.

Supported operations:

Operation	Description
x == y	Checks if two colours are equal.
x != y	Checks if two colours are not equal.

Instance attributes:

value
The raw integer colour value.

b
Returns the blue component of the colour.

classmethod blue()
A factory method that returns a *Colour* with a value of 0x3498db.

classmethod dark_blue()
A factory method that returns a *Colour* with a value of 0x206694.

classmethod dark_gold()
A factory method that returns a *Colour* with a value of 0xc27c0e.

classmethod dark_green()
A factory method that returns a *Colour* with a value of 0x1f8b4c.

classmethod dark_grey()
A factory method that returns a *Colour* with a value of 0x607d8b.

classmethod dark_magenta()
A factory method that returns a *Colour* with a value of 0xad1457.

classmethod dark_orange()
A factory method that returns a *Colour* with a value of 0xa84300.

classmethod dark_purple()
A factory method that returns a *Colour* with a value of 0x71368a.

classmethod **dark_red**()
A factory method that returns a *Colour* with a value of 0x992d22.

classmethod **dark_teal**()
A factory method that returns a *Colour* with a value of 0x11806a.

classmethod **darker_grey**()
A factory method that returns a *Colour* with a value of 0x546e7a.

classmethod **default**()
A factory method that returns a *Colour* with a value of 0.

g
Returns the green component of the colour.

classmethod **gold**()
A factory method that returns a *Colour* with a value of 0xf1c40f.

classmethod **green**()
A factory method that returns a *Colour* with a value of 0x2ecc71.

classmethod **light_grey**()
A factory method that returns a *Colour* with a value of 0x979c9f.

classmethod **lighter_grey**()
A factory method that returns a *Colour* with a value of 0x95a5a6.

classmethod **magenta**()
A factory method that returns a *Colour* with a value of 0xe91e63.

classmethod **orange**()
A factory method that returns a *Colour* with a value of 0xe67e22.

classmethod **purple**()
A factory method that returns a *Colour* with a value of 0x9b59b6.

r
Returns the red component of the colour.

classmethod **red**()
A factory method that returns a *Colour* with a value of 0xe74c3c.

classmethod **teal**()
A factory method that returns a *Colour* with a value of 0x1abc9c.

to_tuple()
Returns an (r, g, b) tuple representing the colour.

class discord.**Role**(***kwargs*)
Represents a Discord role in a *Server*.

Instance attributes:

id
The ID for the role.

name
The name of the role.

permissions
A *Permissions* that represents the role's permissions.

color

colour
A *Colour* representing the role colour.

hoist
A boolean representing if the role will be displayed separately from other members.

position
The position of the role. This number is usually positive.

managed
A boolean indicating if the role is managed by the server through some form of integration such as Twitch.

is_everyone()
Checks if the role is the @everyone role.

class discord.**Permissions**(*permissions=0*, ***kwargs*)
Wraps up the Discord permission value.

Class attributes:

NONE
A Permission with all permissions set to False.

ALL
A Permission with all permissions set to True.

ALL_CHANNEL
A Permission with all channel-specific permissions set to True and the server-specific ones set to False. The server-specific permissions are currently:

- can_manager_server
- can_kick_members
- can_ban_members

GENERAL
A Permission with all "General" permissions set to True.

TEXT
A Permission with all "Text" permissions set to True.

VOICE
A Permission with all "Voice" permissions set to True.

Instance attributes:

value
The raw value. This value is a bit array field of a 32-bit integer representing the currently available permissions. You should query permissions via the properties provided rather than using this raw value.

The properties provided are two way. You can set and retrieve individual bits using the properties as if they were regular bools. This allows you to edit permissions.

classmethod all()
A factory method that creates a Permission with all permissions set to True.

classmethod all_channel()
A Permission with all channel-specific permissions set to True and the server-specific ones set to False. The server-specific permissions are currently:

- can_manager_server
- can_kick_members

•can_ban_members

can_attach_files
Returns True if a user can send files in their messages.

can_ban_members
Returns True if the user can ban users from the server.

can_connect
Returns True if a user can connect to a voice channel.

can_create_instant_invite
Returns True if the user can create instant invites.

can_deafen_members
Returns True if a user can deafen other users.

can_embed_links
Returns True if a user's messages will automatically be embedded by Discord.

can_kick_members
Returns True if a user can kick users from the server.

can_manage_channels
Returns True if a user can edit, delete, or create channels in the server.

can_manage_messages
Returns True if a user can delete messages from a text channel. Note that there are currently no ways to edit other people's messages.

can_manage_roles
Returns True if a user can manage server roles. This role overrides all other permissions.

can_manage_server
Returns True if a user can edit server properties.

can_mention_everyone
Returns True if a user's @everyone will mention everyone in the text channel.

can_move_members
Returns True if a user can move users between other voice channels.

can_mute_members
Returns True if a user can mute other users.

can_read_message_history
Returns True if a user can read a text channel's previous messages.

can_read_messages
Returns True if a user can read messages from all or specific text channels.

can_send_messages
Returns True if a user can send messages from all or specific text channels.

can_send_tts_messages
Returns True if a user can send TTS messages from all or specific text channels.

can_speak
Returns True if a user can speak in a voice channel.

can_use_voice_activation
Returns True if a user can use voice activation in voice channels.

classmethod general()
A factory method that creates a Permission with all "General" permissions set to True.

classmethod none()
A factory method that creates a Permission with all permissions set to False.

classmethod text()
A factory method that creates a Permission with all "Text" permissions set to True.

classmethod voice()
A factory method that creates a Permission with all "Voice" permissions set to True.

class discord.**Channel**(***kwargs*)
Represents a Discord server channel.

Instance attributes:

name
The channel name.

server
The *Server* the channel belongs to.

id
The channel ID.

topic
The channel's topic. None if it doesn't exist.

is_private
True if the channel is a private channel (i.e. PM). False in this case.

position
The position in the channel list.

type
The channel type. Usually 'voice' or 'text'.

changed_roles
A list of Roles that have been overridden from their default values in the *Server.roles* attribute.

voice_members
A list of Members that are currently inside this voice channel. If *type* is not 'voice' then this is always an empty array.

is_default_channel()
Checks if this is the default channel for the *Server* it belongs to.

mention()
Returns a string that allows you to mention the channel.

permissions_for(*member*)
Handles permission resolution for the current *Member*.

This function takes into consideration the following cases:

- Server owner
- Server roles
- Channel overrides
- Member overrides
- Whether the channel is the default channel.

Parameters **member** – The *Member* to resolve permissions for.

Returns The resolved *Permissions* for the *Member*.

class discord.**PrivateChannel**(*user*, *id*, ***kwargs*)
Represents a Discord private channel.

Instance attributes:

user
The *User* in the private channel.

id
The private channel ID.

is_private
`True` if the channel is a private channel (i.e. PM). `True` in this case.

permissions_for(*user*)
Handles permission resolution for a *User*.

This function is there for compatibility with *Channel*.

Actual private messages do not really have the concept of permissions.

This returns all the Text related permissions set to true except:

- can_send_tts_messages: You cannot send TTS messages in a PM.
- can_manage_messages: You cannot delete others messages in a PM.
- can_mention_everyone: There is no one to mention in a PM.

Parameters **user** – The *User* to check permissions for.

Returns A `Permission` with the resolved permission value.

class discord.**Invite**(***kwargs*)
Represents a Discord *Server* or *Channel* invite.

Depending on the way this object was created, some of the attributes can have a value of `None`.

Instance attributes:

max_age
How long the before the invite expires in seconds. A value of 0 indicates that it doesn't expire.

code
The URL fragment used for the invite. *xkcd* is also a possible fragment.

server
The *Server* the invite is for.

revoked
A boolean indicating if the invite has been revoked.

created_at
A datetime object denoting the time the invite was created.

temporary
A boolean indicating that the invite grants temporary membership. If True, members who joined via this invite will be kicked upon disconnect.

uses
How many times the invite has been used.

max_uses
How many times the invite can be used.

xkcd
The URL fragment used for the invite if it is human readable.

inviter
The *User* who created the invite.

channel
The *Channel* the invite is for.

id
Returns the proper code portion of the invite.

url
A property that retrieves the invite URL.

2.4 Exceptions

The following exceptions are thrown by the library.

exception discord.**DiscordException**
Base exception class for discord.py

Ideally speaking, this could be caught to handle any exceptions thrown from this library.

exception discord.**ClientException**
Exception that's thrown when an operation in the *Client* fails.

These are usually for exceptions that happened due to user input.

exception discord.**LoginFailure**
Exception that's thrown when the *Client.login()* function fails to log you in from improper credentials or some other misc. failure.

exception discord.**HTTPException**(*response*, *message=None*)
Exception that's thrown when an HTTP request operation fails.

response
The response of the failed HTTP request. This is an instance of requests.Response.

exception discord.**InvalidArgument**
Exception that's thrown when an argument to a function is invalid some way (e.g. wrong value or wrong type).

This could be considered the analogous of `ValueError` and `TypeError` except derived from *ClientException* and thus *DiscordException*.

exception discord.**GatewayNotFound**
An exception that is usually thrown when the gateway hub for the *Client* websocket is not found.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

A

B

C

D

P

R

S

T

U

V

X