discohook

Release 0.0.6a

Sougata Jana

Sep 24, 2023

CONTENTS

class discohook.**Client**(*, *application_id: Union[int, str]*, *public_key: str*, *token: str*, *route: str = '/interactions'*, *password: Optional[str] = None*, *default_help_command: bool = False*, ***kwargs*)

Bases: Starlette

The base client class for discohook.

Parameters

- **application_id** (*int | str*) – The application ID of the bot.
- **public_key** (*str*) – The public key of the bot.
- **token** (*str*) – The token of the bot.
- **route** (*str*) – The route to listen for interactions on.
- **password** (*str | None*) – The password to use for the dashboard.
- **default_help_command** (*bool*) – Whether to use the default help command or not. Defaults to False.
- ****kwargs** – Keyword arguments to pass to the FastAPI instance.

add_commands(**commands: Union[*ApplicationCommand*, Any]*)

Add commands to the client.

Parameters
***commands** (ApplicationCommand) – The commands to add to the client.

async **create_webhook**(*channel_id: str*, ***, *name: str*, *image_base64: Optional[str] = None*)

Create a webhook in a channel.

Parameters

- **channel_id** (*str*) – The ID of the channel to create the webhook in.
- **name** – The name of the webhook.
- **image_base64** – The base64 encoded image of the webhook.

Return type
Webhook

custom_id_parser()

A decorator to register a dev defined custom id parser.

async **delete_command**(*command_id: str*, ***, *guild_id: Optional[str] = None*)

Delete a command from the client.

Parameters

- **command_id** (*str*) – The id of the command to delete.
- **guild_id** (*str | None*) – The id of the guild to delete the command from. Defaults to None.

async **edit**(*username: str*, ***, *avatar: Optional[str] = None*)

Edits the client user.

Parameters

- **username** (str) – The new username of the client user.
- **avatar** (Optional[str]) – The new avatar of the client user in base64 data URI scheme.

async fetch_channel(*channel_id: str*) → Optional[*Channel*]

Fetches the channel of given id.

Return type
Channel

async fetch_commands()

Fetches the commands of the client.

Return type
List[Dict[str, Any]]

async fetch_guild(*guild_id: str*) → Optional[*Guild*]

Fetches the guild of given id.

Return type
Guild

async fetch_info() → Dict[str, Any]

Returns the application object associated with the requesting client user.

Return type
Dict[str, Any]

async fetch_user(*user_id: str*) → Optional[*User*]

Fetches the user of given id.

Return type
User

async fetch_webhook(*webhook_id: str*, *, *webhook_token: Optional[str] = None*)

Fetch a webhook from the client. :param webhook_id: The ID of the webhook to fetch. :type webhook_id: str :param webhook_token: The token of the webhook to fetch. :type webhook_token: Optional[str]

Return type
Webhook

load(*cmd:* ApplicationCommand) → *ApplicationCommand*

A decorator to load a command into the client.

load_components(*view:* View)

Loads multiple components into the client. Do not use this method unless you know what you are doing.

Parameters
view (View) – The view to load components from.

load_modules(*directory: str*)

Loads multiple command from modules within directory by walking through it.

Parameters
directory (*str*) – The directory to load the modules from.

async me() → *User*

Get the client as a discord user.

Returns
The client as a user.

Return type
User

on_error()

A decorator to add an error handler for any server errors.

on_interaction_error()

A decorator to register a global interaction error handler.

preload(*custom_id: str*)

This decorator is used to load a component into the client. This method will help you to use persistent components with static custom ids.

Parameters
custom_id (*str*) – The unique custom id of the component.

Raises
ValueError – If the custom id is not a not empty string or is not provided.

async send(*channel_id: str*, *content: Optional[str] = None*, ***, *tts: bool = False*, *embed: Optional[Embed] = None*, *embeds: Optional[List[Embed]] = None*, *file: Optional[File] = None*, *files: Optional[List[File]] = None*, *view: Optional[View] = None*) → *Message*

Send a message to a channel using the ID of the channel.

Parameters

- **channel_id** (*str*) – The ID of the channel to send the message to.
- **content** (*Optional[str]*) – The content of the message.
- **tts** (*bool*) – Whether the message should be sent using text-to-speech. Defaults to False.
- **embed** (*Optional[Embed]*) – The embed to send with the message.
- **embeds** (*Optional[List[Embed]]*) – A list of embeds to send with the message. Maximum of 10.
- **file** (*Optional[File]*) – A file to be sent with the message
- **files** (*Optional[List[File]]*) – A list of files to be sent with message.
- **view** (*Optional[View]*) – The view to send with the message.

Returns
The message that was sent.

Return type
Message

async sync() → Tuple[List[ClientResponse], List[Dict[str, Any]]]

Sync the commands to the client.

This method is used internally by the client. You should not use this method.

class discohook.**User**(*client: Client*, *data: Dict[str, Any]*)

Bases: object

Represents a discord user.

CHAPTER

ONE

PROPERTIES

id: `str`
The unique ID of the user.

name: `str`
The name of the user.

discriminator: `str`
The discriminator of the user.

accent_color: Optional[`int`]
The accent color of the user.

avatar: `Asset`
The avatar of the user.

system: `bool`
Whether the user is a system user.

bot: `bool`
Whether the user is a bot.

mfa_enabled: `bool`
Whether the user has MFA enabled.

locale: Optional[`str`]
The locale of the user.

verified: `bool`
Whether the user is verified.

email: Optional[`str`]
The email of the user.

premium_type: Optional[`int`]
The premium type of the user.

public_flags: Optional[`int`]
The public flags of the user.

mention: `str`
Returns a string that allows you to mention the user.

async send(*content: str*, ***, *tts: bool = False*, *embed: Optional[*Embed*] = None*, *embeds: Optional[List[*Embed*]] = None*, *file: Optional[*File*] = None*, *files: Optional[List[*File*]] = None*) → Dict[str, Any]

Sends a message to the user.

Parameters

- **content** (str) – The content of the message.
- **tts** (bool) – Whether the message should be sent using text-to-speech.
- **embed** (Optional[*Embed*]) – The embed to be sent with the message.
- **embeds** (Optional[List`[:class:`Embed]`]) – The embeds to be sent with the message.
- **file** (Optional[*File*]) – The file to be sent with the message.
- **files** (Optional[List`[:class:`File]`]) – The files to be sent with the message.

class discohook.**Member**(*client:* Client, *data: Dict[str, Any]*)

Bases: *User*

Represents a member of a guild, subclassed from *User*.

async **add_role**(*role_id: str*)

Add a role to the member.

Parameters
role_id (*str*) – The ID of the role.

async **ban**(*, *delete_message_seconds: int = 0*)

Ban the member.

Parameters
delete_message_seconds (*int*) – The number of days to delete messages for.

async **kick**()

Kick the member.

property **mention: str**

Returns a string that allows you to mention the member.

async **remove_role**(*role_id: str*)

Remove a role from the member.

Parameters
role_id (*str*) – The ID of the role.

class discohook.**PartialGuild**(*client:* Client, *guild_id: str*)

Bases: object

Represents a partial guild.

async **create_channel**(*name: str*, *, *type:* ChannelType = *ChannelType.guild_text*, *topic: Optional[str] = None*, *bitrate: Optional[int] = None*, *user_limit: Optional[int] = None*, *rate_limit_per_user: Optional[int] = None*, *position: Optional[int] = None*, *permission_overwrites: Optional[List[Dict[str, Any]]] = None*, *parent_id: Optional[str] = None*, *nsfw: Optional[bool] = None*, *rtc_region: Optional[str] = None*, *video_quality_mode: Optional[int] = None*, *default_auto_archive_duration: Optional[int] = None*, *default_reaction_emoji: Optional[*PartialEmoji*] = None*, *available_tags: Optional[List[Dict[str, Any]]] = None*, *default_sort_order: Optional[int] = None*) → *Channel*

Creates a channel in the guild. Requires the MANAGE_CHANNELS permission.

Parameters

- **name** (*str*) – Name of the channel (2-100 characters)
- **type** (ChannelType) – The type of channel

- **topic** (*str*) – Channel topic (0-1024 characters)
- **bitrate** (*int*) – The bitrate (in bits) of the voice channel (voice only)
- **user_limit** (*int*) – The user limit of the voice channel (voice only)
- **rate_limit_per_user** (*int*) – Amount of seconds a user has to wait before sending another message (0-21600) bots, as well as users with the permission manage_messages or manage_channel, are unaffected
- **position** (*int*) – Sorting position of the channel
- **permission_overwrites** (*List[Dict[str, Any]]*) – The channel's permission overwrites
- **parent_id** (*str*) – The id of the parent category for a channel (each parent category can contain up to 50 channels)
- **nsfw** (*bool*) – Whether the channel is nsfw
- **rtc_region** (*str*) – The id of the voice region
- **video_quality_mode** (*int*) – The camera video quality mode of the voice channel, 1 when not present
- **default_auto_archive_duration** (*int*) – The default duration for newly created threads, in minutes, to automatically archive the thread after recent activity, can be set to: 60, 1440, 4320, 10080
- **default_reaction_emoji** (PartialEmoji) – The default auto-emoji for newly created threads, custom guild emojis must be enabled
- **available_tags** (*List[Dict[str, Any]]*) – The channel tags used for public guilds
- **default_sort_order** (*int*) – The default sorting order for posts in a forum channel

Return type
Channel

async create_emoji(*name: str*, *image: str*, ***, *roles: Optional[List[str]] = None*)

Creates a new emoji for the guild.

Parameters

- **name** (str) – The name of the emoji.
- **image** (str) – The image data of the emoji in base64 data uri format.
- **roles** (Optional[List[str]]) – A list of role ids to limit the emoji to.

Return type
Emoji

async edit_channel_position(*channel_id: str*, ***, *position: int*, *lock_permissions: bool = False*, *parent_id: Optional[str] = None*)

Changes the position of the channel. Only available for guild channels.

Parameters

- **channel_id** (str) – The id of the channel to move.
- **position** (int) – The new position of the channel.
- **lock_permissions** – Whether to sync the permissions of the channel with the parent category.

- **parent_id** (Optional[str]) – The id of the parent category to move the channel to. If not provided, the channel will be moved to the root.

async fetch_channels() → List[*Channel*]

Fetches all channels in the guild.

Return type
List[*Channel*]

async fetch_member(*user_id: str*) → Optional[*Member*]

Fetches a member from the guild.

Parameters
user_id (str) – The id of the user to fetch.

Return type
Optional[*Member*]

async fetch_roles() → List[*Role*]

Fetches all roles in the guild.

Return type
List[*Role*]

class discohook.**Guild**(*client:* Client, *data: Dict[str, Any]*)

Bases: *PartialGuild*

Represents a Discord guild. Subclass of *PartialGuild*.

id

The id of the guild.

Type
str

name

The name of the guild.

Type
str

icon

The icon hash of the guild.

Type
Optional[str]

icon_hash

The icon hash of the guild.

Type
Optional[str]

splash

The splash hash of the guild.

Type
Optional[str]

discovery_splash

The discovery splash hash of the guild.

Type
Optional[str]

owner

Whether the user is the owner of the guild.

Type
Optional[bool]

owner_id

The id of the owner of the guild.

Type
str

permissions

The total permissions of the user in the guild (does not include channel overrides).

Type
Optional[int]

afk_channel_id

The id of the afk channel.

Type
Optional[str]

afk_timeout

The afk timeout in seconds.

Type
int

widget_enabled

Whether the widget is enabled.

Type
Optional[bool]

widget_channel_id

The id of the channel for the widget.

Type
Optional[str]

verification_level

The verification level required for the guild.

Type
int

default_message_notifications

The default message notifications level.

Type
int

explicit_content_filter

The explicit content filter level.

Type
int

roles

The roles in the guild.

Type
List[*Role*]

emojis

The emojis in the guild.

Type
List[Emoji]

features

The features of the guild.

Type
List[str]

mfa_level

The mfa level required for the guild.

Type
int

application_id

The application id of the guild creator if it is bot-created.

Type
Optional[str]

system_channel_id

The id of the system channel.

Type
Optional[str]

system_channel_flags

The system channel flags.

Type
int

rules_channel_id

The id of the rules channel.

Type
Optional[str]

max_presences

The maximum number of presences for the guild.

Type
Optional[int]

max_members

The maximum number of members for the guild.

Type
int

vanity_url_code

The vanity url code of the guild.

Type
Optional[str]

description

The description of the guild.

Type
Optional[str]

banner

The banner hash of the guild.

Type
Optional[str]

premium_tier

The premium tier of the guild.

Type
int

premium_subscription_count

The number of boosts of the guild.

Type
int

preferred_locale

The preferred locale of the guild.

Type
str

public_updates_channel_id

The id of the public updates channel.

Type
Optional[str]

max_video_channel_users

The maximum number of users in a video channel.

Type
Optional[int]

approximate_member_count

The approximate number of members in the guild.

Type
Optional[int]

approximate_presence_count

The approximate number of presences in the guild.

Type
Optional[int]

welcome_screen

The welcome screen object of the guild.

Type
Optional[dict]

nsfw_level

The nsfw level of the guild.

Type
int

stickers

The stickers in the guild.

Type
List[Sticker]

premium_progress_bar_enabled

Whether the premium progress bar is enabled.

Type
Optional[bool]

class discohook.**PartialChannel**(*client: Client*, *channel_id: str*, *guild_id: Optional[str] = None*)

Bases: object

Represents a partial discord channel object.

Parameters

- **channel_id** (*str*) – The channel's ID.
- **guild_id** (*str | None*) – The guild's ID.
- **client** (*Client*) – The client that the channel belongs to.

async edit(**, name: Optional[str] = None, kind: Optional[ChannelType] = None, position: Optional[int] = None, topic: Optional[str] = None, nsfw: Optional[bool] = None, rate_limit_per_user: Optional[int] = None, bitrate: Optional[int] = None, user_limit: Optional[int] = None, permission_overwrites: Optional[List[Dict[str, Any]]] = None, parent_id: Optional[str] = None, rtc_region: Optional[str] = None, video_quality_mode: Optional[int] = None, default_auto_archive_duration: Optional[int] = None, flags: Optional[int] = None, available_tags: Optional[List[Dict[str, Any]]] = None, icon: Optional[str] = None, default_reaction_emoji: Optional[PartialEmoji] = None, default_thread_rate_limit_per_user: Optional[int] = None, default_sort_order: Optional[int] = None, default_forum_layout: Optional[int] = None*) → *Channel*

Edits all kinds of channels.

Parameters

- **name** (Optional[str]) – The new name of the channel.
- **kind** (Optional[*ChannelType*]) – The new type of the channel.
- **position** (Optional[int]) – The new position of the channel.

- **topic** (Optional[str]) – The new topic of the channel.
- **nsfw** (Optional[bool]) – Whether the channel should be marked as nsfw.
- **rate_limit_per_user** (Optional[int]) – The duration of the slowmode in seconds. Must be between 0 and 21600. Applies to text and forum channels.
- **bitrate** (Optional[int]) – The new bitrate of the channel. Must be between 8000 and 96000. Applies to voice channels.
- **user_limit** (Optional[int]) – The new user limit of the channel. Must be between 0 and 99. Applies to voice channels.
- **permission_overwrites** (Optional[List[dict]]) – A list of permission overwrites to apply to the channel. Applies to all channel types.
- **parent_id** (Optional[str]) – The id of the parent category to move the channel to. Applies to all channel types.
- **rtc_region** (Optional[str]) – The new region of the channel. Applies to voice channels.
- **video_quality_mode** (Optional[int]) – The new video quality mode of the channel. Applies to voice channels.
- **default_auto_archive_duration** (Optional[int]) – The new default auto archive duration of the channel. Applies to text and forum channels.
- **flags** (Optional[int]) – The new flags of the channel. Applies to all channel types.
- **available_tags** (Optional[List[dict]]) – The new available tags of the channel. Applies to text and forum channels.
- **icon** (Optional[str]) – The new icon of the channel. Applies to Group DMs. Must be a base64 encoded string.
- **default_reaction_emoji** (Optional[*PartialEmoji*]) – The new default reaction emoji of the channel. Applies to text and forum channels.
- **default_thread_rate_limit_per_user** (Optional[int]) – The new default thread rate limit per user of the channel. Applies to text and forum channels.
- **default_sort_order** (Optional[int]) – The new default sort order of the channel. Applies to text and forum channels.
- **default_forum_layout** (Optional[int]) – The new default forum layout of the channel. Applies to text and forum channels.

Returns
The edited channel.

Return type
Channel

async fetch_message(*message_id: str*) → Optional[*Message*]

Fetches a message from the channel.

Parameters
message_id (str) – The id of the message to fetch.

Returns
The fetched message.

Return type
Message

async fetch_messages(*limit: int = 50*, ***, *before: Optional[str] = None*, *after: Optional[str] = None*, *around: Optional[str] = None*) → List[*Message*]

Fetches messages from the channel.

Parameters

- **limit** (Optional[int]) – The maximum amount of messages to fetch.
- **before** (Optional[str]) – The id of the message to fetch before.
- **after** (Optional[str]) – The id of the message to fetch after.
- **around** (Optional[str]) – The id of the message to fetch around.

Returns
The fetched messages.

Return type
List[*Message*]

property mention: str

Returns the channel-mentionable string.

Return type
str

async purge(*limit: int = 50*, ***, *before: Optional[str] = None*, *after: Optional[str] = None*, *around: Optional[str] = None*) → List[*Message*]

Deletes messages from the channel in bulk.

Parameters

- **limit** (Optional[int]) – The maximum amount of messages to delete.
- **before** (Optional[str]) – The id of the message to delete before.
- **after** (Optional[str]) – The id of the message to delete after.
- **around** (Optional[str]) – The id of the message to delete around.

Returns
The deleted messages.

Return type
List[*Message*]

async send(*content: Optional[str] = None*, ***, *embed: Optional[Embed] = None*, *embeds: Optional[List[Embed]] = None*, *view: Optional[View] = None*, *tts: Optional[bool] = False*, *file: Optional[File] = None*, *files: Optional[List[File]] = None*, *allowed_mentions: Optional[AllowedMentions] = None*, *message_reference: Optional[MessageReference] = None*)

Sends a message to the channel.

Parameters

- **content** (Optional[str]) – The content of the message.
- **embed** (Optional[*Embed*]) – The embed to send with the message.
- **embeds** (Optional[List[*Embed*]]) – A list of embeds to send with the message.
- **view** (Optional[*View*]) – The view to send with the message.
- **tts** (Optional[bool]) – Whether the message should be sent with text-to-speech.
- **file** (*Optional[File]*) – A file to send with the message.

- **files** (*Optional[List[File]]*) – A list of files to send with the message.
- **allowed_mentions** (Optional[AllowedMentions]) – The allowed mentions for the message.
- **message_reference** (Optional[MessageReference]) – The message reference for the message.

class discohook.**Channel**(*client:* Client, *data: dict*)

Bases: *PartialChannel*

Represents a discord channel object.

id

The id of the channel.

Type
str

type

The type of the channel.

Type
Optional[int]

guild_id

The id of the guild the channel belongs to.

Type
Optional[str]

position

The position of the channel.

Type
Optional[int]

permission_overwrites

A list of permission overwrites for the channel.

Type
Optional[List[dict]]

name

The name of the channel.

Type
Optional[str]

topic

The topic of the channel.

Type
Optional[str]

nsfw

Whether the channel is nsfw.

Type
Optional[bool]

`last_message_id`

The id of the last message sent in the channel.

Type
Optional[`str`]

`bitrate`

The bitrate of the channel if it is a voice channel.

Type
Optional[`int`]

`user_limit`

The user limit of the channel if it is a voice channel.

Type
Optional[`int`]

`rate_limit_per_user`

The rate limit per user of the channel if it is a text channel.

Type
Optional[`int`]

`recipients`

A list of recipients of the channel if it is a dm channel.

Type
Optional[List[`dict`]]

`icon`

The icon of the channel if it is a dm channel.

Type
Optional[`str`]

`owner_id`

The id of the owner of the channel if it is a dm channel.

Type
Optional[`str`]

`application_id`

The id of the application of the channel if it is a group dm channel.

Type
Optional[`str`]

`parent_id`

The id of the parent category of the channel.

Type
Optional[`str`]

`last_pin_timestamp`

The timestamp of the last pinned message in the channel.

Type
Optional[`str`]

`rtc_region`

The rtc region of the channel.

Type
Optional[`str`]

`video_quality_mode`

The video quality mode of the channel.

Type
Optional[`int`]

`message_count`

The message count of the channel.

Type
Optional[`int`]

`member_count`

The member count of the channel.

Type
Optional[`int`]

`thread_metadata`

The thread metadata of the channel.

Type
Optional[`dict`]

`member`

The member of the channel.Appears in thread channels.

Type
Optional[`dict`]

`default_auto_archive_duration`

The default auto archive duration of the channel.Appears in thread channels.

Type
Optional[`int`]

`permissions`

The permissions of the channel.

Type
Optional[`str`]

`flags`

The flags of the channel.

Type
Optional[`int`]

`total_message_sent`

The total message sent of the channel.

Type
Optional[`int`]

available_tags

A list of available tags of the channel.Appears in thread channels.

Type
Optional[List[str]]

default_reaction_emoji

The default reaction emoji of the channel.Appears in thread channels.

Type
Optional[dict]

default_thread_rate_limit_per_user

The default rate limit per user of the channel.Appears in thread channels.

Type
Optional[int]

default_sort_order

The default sort order of the channel.Appears in forum channels.

Type
Optional[int]

default_forum_layout

The default channel layout of the channel.Appears in forum channels.

Type
Optional[int]

class discohook.**PartialRole**(*client:* Client, *data: Dict[str, Any]*)

Bases: object

async edit(*, *name: Optional[str] = None*, *permissions: Optional[List[*Permission*]] = None*, *color: Optional[int] = None*, *hoist: Optional[bool] = None*, *mentionable: Optional[bool] = None*, *description: Optional[str] = None*, *unicode_emoji: Optional[str] = None*, *icon_data_uri: Optional[str] = None*) → *Role*

Edits the role.

Parameters

- **name** (Optional[str]) – The name of the role.
- **permissions** (Optional[*Permission*]) – The permissions of the role.
- **color** (Optional[int]) – The color of the role.
- **hoist** (Optional[bool]) – Whether the role has separability in the member list.
- **mentionable** (Optional[bool]) – Whether the role is mentionable.
- **description** (Optional[str]) – The description of the role.
- **unicode_emoji** (Optional[str]) – The unicode emoji of the role.
- **icon_data_uri** (Optional[str]) – The icon of the role. Must be a data URI (base64 encoded).

Return type
Role

async edit_position(*role_id: str*, *, *position: int*) → List[*Role*]

Changes the position of the role. :param role_id: The id of the role to move. :type role_id: `str` :param position: The new position of the role. :type position: `int`

Return type
Role

property mention: str

Returns a string that allows you to mention the role.

Return type
`str`

class discohook.**Role**(*client:* Client, *data: dict*)

Bases: *PartialRole*

Represents a discord Role.

id

The unique ID of the role.

Type
`str`

name

The name of the role.

Type
`str`

color

The color of the role.

Type
`int`

hoist

Whether the role has separability in the member list.

Type
`bool`

position

The position of the role.

Type
`int`

permissions

The permissions of the role.

Type
`str`

managed

Whether the role is managed by an integration.

Type
`bool`

mentionable

Whether the role is mentionable.

Type
bool

description

The description of the role.

Type
Optional[str]

unicode_emoji

The unicode emoji of the role.

Type
Optional[str]

icon

The icon of the role.

Type
Optional[str]

flags

The flags of the role.

Type
int

has_permission(*permission:* Permission) → bool

Checks if the role has the given permissions.

Parameters
permission (*Permission*) – The permissions to check.

Return type
bool

class discohook.**ApplicationCommand**(*name: str*, *, *description: Optional[str] = None*, *options: Optional[List[Option]] = None*, *dm_access: bool = True*, *nsfw: bool = False*, *permissions: Optional[List[*Permission*]] = None*, *kind:* ApplicationCommandType *= ApplicationCommandType.slash*, *guild_id: Optional[str] = None*, *callback: Callable[[*Interaction*, Any], Coroutine[Any, Any, Any]]*)

Bases: Interactable

A class representing a discord application command.

Parameters

- **name** (*str*) – The name of the command.
- **description** (*str | None*) – The description of the command. Does not apply to user & message commands.
- **options** (*List[Option] | None*) – The options of the command. Does not apply to user & message commands.
- **dm_access** (*bool*) – Whether the command can be used in DMs. Defaults to True.
- **nsfw** (*bool*) – Whether the command is age restricted. Defaults to False.

- **permissions** (*List[Permission] | None*) – The default permissions of the command.
- **kind** (ApplicationCommandType) – The category of the command. Defaults to slash commands.

autocomplete(*name: str*)

A decorator to register a callback for the command's autocomplete options.

Parameters
name (*str*) – The name of the option to register the autocomplete for.

subcommand(*name: Optional[str] = None*, *description: Optional[str] = None*, ***, *options: Optional[List[Option]] = None*)

A decorator to register a subcommand for the command.

Parameters

- **name** (*str*) – The name of the subcommand.
- **description** (*str*) – The description of the subcommand.
- **options** (*Optional[List[Option]]*) – The options of the subcommand.

Returns
The subcommand object.

Return type
SubCommand

Raises
TypeError – If the callback is not a coroutine.

to_dict() → Dict[str, Any]

Converts the command to a dictionary.

This is used to send the command to the Discord API. Not intended for use by end-users.

Return type
Dict[str, Any]

class discohook.**SubCommand**(*name: str*, *description: str*, *options: Optional[List[Option]] = None*, ***, *callback: Optional[Callable[[Interaction, Any], Coroutine[Any, Any, Any]]] = None*)

Bases: object

A class representing a discord application command subcommand.

Parameters

- **name** (*str*) – The name of the subcommand.
- **description** (*str*) – The description of the subcommand.
- **options** (*List[Option] | None*) – The options of the subcommand.
- **callback** (*AsyncCallable* | None) – The callback of the subcommand.

autocomplete(*name: str*)

A decorator to register a callback for the subcommand's autocomplete options.

Parameters
name (*str*) – The name of the option to register the autocomplete for.

class discohook.**Embed**(*title: Optional[str] = None*, *, *description: Optional[str] = None*, *url: Optional[str] = None*, *color: Optional[Union[int, str]] = None*, *timestamp: Optional[str] = None*)

Bases: object

Represents a discord Embed object.

Parameters

- **title** (*str | None*) – The title of the embed.
- **description** (*str | None*) – The description of the embed.
- **url** (*str | None*) – The url of the embed.
- **color** (*int | str | None*) – The color of the embed in hex or int.
- **timestamp** (*str | None*) – The timestamp of the embed.

add_field(*name: str*, *value: str*, *, *inline: bool = False*)

Adds a field to the embed.

Parameters

- **name** (str) – The name of the field.
- **value** (str) – The value of the field.
- **inline** (bool) – Whether the field is inline.

property **attachments: List[*File*]**

Returns the attachments of the embed.

classmethod **from_dict**(*data: Dict[str, Any]*) → *Embed*

Creates an embed from a dictionary. This method is handy when you want to create an embed manually.

Parameters
data (dict) – The dictionary to create the embed from.

Returns
The created embed.

Return type
Embed

set_author(*, *name: str*, *url: Optional[str] = None*, *icon_url: Optional[str] = None*)

Sets the author of the embed.

Parameters

- **name** (str) – The name of the author.
- **url** (Optional[str]) – The url of the author.
- **icon_url** (Optional[str]) – The icon url of the author.

set_footer(*text: str*, *, *icon_url: Optional[str] = None*)

Sets the footer of the embed.

Parameters

- **text** (str) – The text of the footer.
- **icon_url** (Optional[str]) – The icon url of the footer.

set_image(*img: Union[str, File]*)

Sets the image of the embed from a file attachment or url.

Parameters
img (str | *File*) – The url or file attachment of the image.

set_thumbnail(*img: Union[str, File]*)

Sets the thumbnail of the embed.

Parameters
img (str | *File*) – The url or file attachment of the thumbnail.

to_dict() → Dict[str, Any]

Returns the embed as a dictionary.

This method is used internally by the library. You will rarely need to use it.

Returns
The embed as a dictionary.

Return type
dict

class discohook.**File**(*name: str*, *, *content: bytes*, *spoiler: bool = False*, *description: Optional[str] = None*)

Bases: object

Represents a file to send to Discord.

Parameters

- **name** (*str*) – The name of the file.
- **content** (*bytes*) – The content of the file in bytes.
- **description** (*str | None*) – The description of the file.
- **spoiler** (*bool*) – Whether the file is a spoiler.

class discohook.**PartialEmoji**(*, *name: Optional[str] = None*, *id: Optional[str] = None*, *animated: Optional[bool] = None*)

Bases: object

Represents a discord PartialEmoji object.

Parameters

- **name** (str) – The name of the emoji.
- **id** (str) – The unique id of the emoji.
- **animated** (bool) – Whether the emoji is animated.

class discohook.**TextInputFieldLength**(*value*)

Bases: int, Enum

The length of a text input field for a modal.

short

Used to specify a short length text input field (up to 100 characters).

Type
int

long

Used to specify a long length text input field (up to 3000 characters).

Type
int

class discohook.**ApplicationCommandType**(*value*)

Bases: int, Enum

The type of application command.

slash

Used to specify a slash command.

Type
int

user

Used to specify a user command.

Type
int

message

Used to specify a message command.

Type
int

class discohook.**ChannelType**(*value*)

Bases: int, Enum

Use to specify discord channel type in application command Option.

guild_text

Used to specify a guild text channel.

Type
int

dm

Used to specify a dm channel.

Type
int

guild_voice

Used to specify a guild voice channel.

Type
int

group_dm

Used to specify a group dm channel.

Type
int

guild_category

Used to specify a guild category channel.

Type
int

guild_announcement

Used to specify a guild announcement channel.

Type
int

guild_announcement_thread

Used to specify a guild announcement thread channel.

Type
int

public_thread

Used to specify a guild public thread channel.

Type
int

private_thread

Used to specify a guild private thread channel.

Type
int

guild_stage_voice

Used to specify a guild stage voice channel.

Type
int

guild_directory

Used to specify a guild directory channel.

Type
int

guild_forum

Used to specify a guild forum channel.

Type
int

guild_media

Used to specify a guild media channel.

Type
int

class discohook.**ButtonStyle**(*value*)

Bases: int, Enum

Represents the style of a button.

blurple

Used to specify a blurple button.

Type
int

grey

Used to specify a grey button.

Type
int

green

Used to specify a green button.

Type
int

red

Used to specify a red button.

Type
int

link

Used to specify a link type button.

Type
int

class discohook.**SelectType**(*value*)

Bases: int, Enum

The type of select menu.

class discohook.**Interaction**(*client:* Client, *data: Dict[str, Any]*)

Bases: object

Base interaction class for all interactions

CHAPTER

TWO

PROPERTIES

id: str
The unique id of the interaction

type: int
The type of the interaction

token: str
The token of the interaction

version: int
The version of the interaction

application_id: str
The id of the application that the interaction was triggered for

data: Optional[Dict[str, Any]]
The command data payload (if the interaction is a command)

guild_id: Optional[str]
The guild id of the interaction

channel_id: Optional[str]
The channel id of the interaction

app_permissions: Optional[int]
The permissions of the application

locale: Optional[str]
The locale of the interaction

guild_locale: Optional[str]
The guild locale of the interaction

param data
The interaction data payload

type data
Dict[str, Any]

param client
The request object from fastapi

type client
Client

`property app_permissions: Optional[int]`

The permissions of the application

Return type
Optional[int]

property application_id: str

The id of the application that the interaction was triggered for

Return type
str

property author: Union[*User*, *Member*]

The author of the interaction

If the interaction was triggered in a guild, this will return a member object else it will return user object.

Return type
Union[*User*, *Member*]

property channel: *PartialChannel*

The channel where the interaction was triggered

Return type
PartialChannel

property channel_id: str

The channel id of the interaction

Return type
Optional[str]

property from_originator: bool

Whether the interaction was triggered by the same user who triggered the message

Return type
bool

property guild_id: Optional[str]

The guild id of the interaction

Return type
Optional[str]

property guild_locale: Optional[str]

The guild locale of the interaction

Return type
Optional[str]

property id: str

The unique id of the interaction

Return type
str

property kind: Optional[InteractionType]

The type of the interaction

Return type
Optional[InteractionType]

property locale: Optional[str]

The locale of the interaction

Return type
Optional[str]

property message: Optional[*Message*]

The message from which the component interaction was triggered

Return type
Message

async original_response() → Optional[*Message*]

Gets the original response message of the interaction if the interaction has been responded to.

Returns
The original response message

Return type
InteractionResponse

property responded: bool

Whether the interaction has been responded to

Return type
bool

property response

The response adapter for the interaction

Return type
ResponseAdapter

property token: str

The token of the interaction

Return type
str

property version: int

The version of the interaction

Return type
int

class discohook.**Message**(*client:* Client, *payload: Dict[str, Any]*)

Bases: object

Represents a Discord message.

CHAPTER

THREE

PROPERTIES

id: `str`
The id of the message.

channel_id: `str`
The id of the channel the message was sent in.

author: *`User`*
The author of the message.

content: `str`
The content of the message.

timestamp: `str`
The timestamp of the message.

edited_timestamp: Optional[`str`]
The timestamp of when the message was last edited.

tts: `bool`
Whether the message was sent using text-to-speech.

mention_everyone: `bool`
Whether the message mentions everyone.

mentions: List[*`User`*]
The users mentioned in the message.

mention_roles: List[*`Role`*]
The roles mentioned in the message.

mention_channels: Optional[`dict`]
The channels mentioned in the message.

attachments: `dict`
The attachments in the message.

embeds: `list`
The embeds in the message.

reactions: Optional[`list`]
The reactions in the message. ...

`async` **`add_reaction`**(*emoji: Union[*PartialEmoji, *str]*)

Creates a reaction on the message.

Parameters
`emoji` (*`Union[Emoji, str]`*) – The emoji to react with.

async delete()

Deletes the message.

async edit(*content: Optional[str] = typing.Any*, ***, *embed: Optional[Embed] = typing.Any*, *embeds: Optional[List[Embed]] = typing.Any*, *view: Optional[View] = typing.Any*, *tts: Optional[bool] = typing.Any*, *file: Optional[File] = typing.Any*, *files: Optional[List[File]] = typing.Any*, *suppress_embeds: Optional[bool] = typing.Any*)

Edits the message.

Parameters

- **content** (*Optional[str]*) – The new content of the message.
- **embed** (*Optional[Embed]*) – The new embed of the message.
- **embeds** (*Optional[List[Embed]]*) – The new embeds of the message.
- **view** (*Optional[View]*) – The new view of the message.
- **tts** (*Optional[bool]*) – Whether the message should be sent with text-to-speech.
- **file** (*Optional[File]*) – A file to send with the message.
- **files** (*Optional[List[File]]*) – A list of files to send with the message.
- **suppress_embeds** (*Optional[bool]*) – Whether the embeds should be suppressed.

async pin()

Pins the message to the channel.

async remove_reaction(*emoji: Union[PartialEmoji, str]*, *user_id: Optional[str] = None*)

Removes a reaction on the message.

Parameters

- **emoji** (*Union[Emoji, str]*) – The emoji to delete.
- **user_id** (*Optional[str]*) – The user to delete the reaction of.

async remove_reactions(*emoji: Optional[Union[str, PartialEmoji]] = None*)

Removes all reactions on the message.

Parameters

emoji (*Union[Emoji, str, None]*) – The emoji to remove reactions of.

async reply(*content: Optional[str] = None*, ***, *embed: Optional[Embed] = None*, *embeds: Optional[List[Embed]] = None*, *view: Optional[View] = None*, *tts: Optional[bool] = False*, *file: Optional[File] = None*, *files: Optional[List[File]] = None*, *allowed_mentions: Optional[AllowedMentions] = None*, *mention_author: Optional[bool] = None*)

Replies to the message.

Parameters

- **content** (*Optional[str]*) – The content of the message.
- **embed** (*Optional[Embed]*) – The embed of the message.
- **embeds** (*Optional[List[Embed]]*) – The embeds of the message.
- **view** (*Optional[View]*) – The view of the message.
- **tts** (*Optional[bool]*) – Whether the message should be sent with text-to-speech.
- **file** (*Optional[File]*) – A file to send with the message.

- **files** (*Optional[List[File]]*) – A list of files to send with the message.
- **allowed_mentions** (*Optional[AllowedMentions]*) – The allowed mentions for the message.
- **mention_author** (*Optional[bool]*) – Whether the author should be mentioned.

Return type
Message

async unpin()

Unpins the message from the channel.

class discohook.**FollowupResponse**(*payload: Dict[str, Any]*, *interaction:* Interaction)

Bases: object

Represents a followup message sent by an interaction, subclassed from *Message*.

async delete()

Deletes the followup message.

async edit(*content: Optional[str] = typing.Any*, ***, *embed: Optional[*Embed*] = typing.Any*, *embeds: Optional[List[*Embed*]] = typing.Any*, *view: Optional[*View*] = typing.Any*, *tts: Optional[bool] = typing.Any*, *file: Optional[*File*] = typing.Any*, *files: Optional[List[*File*]] = typing.Any*, *suppress_embeds: Optional[bool] = typing.Any*) → *Message*

Edits the followup message.

:param same as *Message.edit()*:

class discohook.**InteractionResponse**(*interaction:* Interaction)

Bases: object

Represents a response message sent by an interaction

async delete()

Deletes the response message.

async edit(*content: Optional[str] = typing.Any*, ***, *embed: Optional[*Embed*] = typing.Any*, *embeds: Optional[List[*Embed*]] = typing.Any*, *view: Optional[*View*] = typing.Any*, *tts: Optional[bool] = typing.Any*, *file: Optional[*File*] = typing.Any*, *files: Optional[List[*File*]] = typing.Any*, *suppress_embeds: Optional[bool] = typing.Any*) → *Message*

Edits the response message.

:param same as *Message.edit()*:

class discohook.**Modal**(*title: str*, ***, *custom_id: Optional[str] = None*)

Bases: Component

A modal for discord.

Parameters

- **title** (str) – The title of the modal.
- **custom_id** (str) – The unique id of the modal.

add_field(*label: str*, *field_id: str*, ***, *required: bool = False*, *hint: Optional[str] = None*, *default_text: Optional[str] = None*, *min_length: int = 0*, *max_length: int = 4000*, *style:* TextInputFieldLength *= TextInputFieldLength.short*)

Add a text input field to the modal.

Parameters

- **label** (*str*) – The label of the field.
- **field_id** (*str*) – A unique id of the text input field. Must be valid python identifier.
- **required** (*bool*) – Whether the field is required or not.
- **hint** (*str*) – The hint to be displayed on the field.
- **default_text** (*str*) – The default text to be displayed on the field.
- **min_length** (*int*) – The minimum length of the field.
- **max_length** (*int*) – The maximum length of the field.
- **style** (TextInputFieldLength) – The style of the field.

`to_dict()`

Convert the modal to a dict to be sent to discord. For internal use only.

`class discohook.SelectOption(label: str, value: str, *, description: Optional[str] = None, emoji: Optional[PartialEmoji] = None, default: bool = False)`

Bases: `object`

Represents a discord select menu option object.

Parameters

- **label** (`str`) – The text to be displayed on the option.
- **value** (`str`) – The value to be sent to the bot when the option is selected.
- **description** (*str | None*) – The description to be displayed on the option.
- **emoji** (`str` | *PartialEmoji* | None) – The emoji to be displayed on the option.
- **default** (`bool`) – Whether the option is selected by default or not.

`to_dict()` → Dict[str, Any]

Returns a dictionary representation of the button.

This is used internally by the library. You should not need to use this method.

Returns
The dictionary representation of the button.

Return type
`dict`

`class discohook.Select(kind: SelectType, *, placeholder: Optional[str] = None, min_values: Optional[int] = None, max_values: Optional[int] = None, disabled: Optional[bool] = False, custom_id: Optional[str] = None)`

Bases: `Component`

Represents a discord select menu component.

Parameters

- **placeholder** (Optional[`str`]) – The placeholder to be displayed on the select menu.
- **min_values** (Optional[`int`]) – The minimum number of options that can be selected.
- **max_values** (Optional[`int`]) – The maximum number of options that can be selected.
- **disabled** (Optional[`bool`]) – Whether the select menu is disabled or not.
- **kind** (*SelectType*) – The type of the select menu.

`to_dict`() → Dict[str, Any]

Returns a dictionary representation of the button.

This is used internally by the library. You should not need to use this method.

Returns
The dictionary representation of the button.

Return type
dict

class discohook.**Button**(*label: Optional[str] = None*, *, *url: Optional[str] = None*, *style:* ButtonStyle = *ButtonStyle.blurple*, *disabled: bool = False*, *emoji: Optional[Union[str,* PartialEmoji*]] = None*, *custom_id: Optional[str] = None*)

Bases: Component

Represents a discord button type component.

Parameters

- **label** (*str | None*) – The text to be displayed on the button.
- **url** (*str | None*) – The url to be opened when the button is clicked if the style is set to *ButtonStyle.link*.
- **style** (*ButtonStyle*) – The style of the button.
- **disabled** (bool) – Whether the button is disabled or not.
- **emoji** (str | *PartialEmoji* | None) – The emoji to be displayed on the button.

`to_dict`() → Dict[str, Any]

Returns a dictionary representation of the button.

This is used internally by the library. You should not need to use this method.

Returns
The dictionary representation of the button.

Return type
dict

class discohook.**View**

Bases: object

Represents a discord message component tree.

This is used to create actions rows and add buttons and select menus to them without having tree conflicts.

`components`

The list of components to be sent to discord. Do not modify this directly.

Type
List[dict]

`children`

The list of children to be sent to discord. Do not modify this directly.

Type
List[Union[*Button*, *Select*]]

add_buttons(**buttons: Union[*Button*, Any]*)

Adds a row of buttons to the view. Each row can only contain up to 5 buttons. Action rows having buttons can not have select menus.

Parameters
***buttons** (*Button*) – The buttons to be added to the view.

add_select(*select: Union[*Select*, Any]*)

Adds a row of select to the view. Each row can only contain up to 1 select menu. Action rows having select menu can not have buttons.

Parameters
select (*Select*) – The select menu to be added to the view.

class discohook.**Permission**(*value*)

Bases: Enum

Represents discord permission values.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

O

P

R

S

T

U

V

W