Novus

Kae Bartlett

Apr 25, 2024

CONTENTS:

Novus is a Python library for interfacing with the Discord API. It works on a developer-first approach, and has been designed from the ground up to work with interactions, components, and slash commands as first-class citizens rather than them being tacked onto the end of their development 6 years in (thanks Discord).

```
TOKEN: str
GUILD_ID: int
USER_ID: int

# In Novus, a client is a secondary citizen, whereas the HTTP
# connection is who we love - you can just put down your token
# and pass that around as necessary
state = novus.HTTPConnection(TOKEN)

# As such, models now have classmethods using their state in order
# to get instances
guild = await novus.Guild.fetch(state, GUILD_ID)

# Things that logically inherit (such as Guild -> GuildMember)
# do also have helper methods
user1 = await guild.fetch_member(USER_ID)
user2 = await novus.GuildMember.fetch(state, GUILD_ID, USER_ID)
assert user1 == user2
```

The easiest way to get started with Novus in most cases is via the *Client Quickstart*.

CHAPTER

ONE

API REFERENCE

1.1 Models

1.1.1 Discord Models

Models that are received from Discord. None of these should be created yourself, but should all be given to you by library or API methods.

class novus.**Activity**(*name: str*, *type: int*, *state: str* | *None* = *None*, *url: str* | *None* = *None*)

A generic activity base class for all types of activity.

Parameters

- **name** (*str*) – The name of the activity
- **type** (*int*) – The type of activity.

  See also:

  novus.ActivityType
- **state** (*str | None*) – The state of the activity. Used for custom status text.
- **url** (*str | None*) – The URL of a stream. Only applicable if the activity type is a stream.

name

The name of the activity

Type
str

type

The type of activity.

See also:

novus.ActivityType

Type
int

state

The state of the activity. Used for custom status text.

Type
str | None

url

The URL of a stream. Only applicable if the activity type is a stream.

Type
str | None

class novus.**Application**(*, *state:* HTTPConnection, *data: payloads.Application*)

A model containing data about an application.

id

The ID of the application.

Type
int

name

The name of the team.

Type
str

icon_hash

A hash for the application icon.

Type
str | None

icon

An asset for the application icon.

Type
novus.Asset | None

description

A description for the application.

Type
str

rpc_origins

An array of RPC origin URLs.

Type
list[str]

bot_public

Whether the bot associated with the application is set to public.

Type
bool

bot_require_code_grant

Whether or not the bot requires a code grant.

Type
bool

terms_of_service_url

The ToS URL for the application.

Type
str | None

privacy_policy_url

The privacy policy URL for the application.

Type
str | None

owner

The user who owns the application.

Type
novus.User

verify_key

The hex encoded key for verification in interactions.

Type
str

team

The team associated with the application.

Type
novus.Team | None

guild_id

The guild ID associated with the game sold via the application.

Type
int | None

primary_sku_id

The ID of the game SKU for the application.

Type
int | None

slug

The URL slug that links to the store page.

Type
str | None

cover_image_hash

The hash for the cover image.

Type
str | None

cover_image

The cover image asset for the application.

Type
novus.Asset | None

flags

The flags assocaiated with the application.

Type
novus.ApplicationFlags

tags

A list of tags describing the content and functionality of the application.

Type
list[str]

install_scopes

The scopes used in the in-app authorization link.

Type
list[str]

install_permissions

The permissions used in the in-app authorization link.

Type
novus.Permissions

custom_install_url

The application's custom authorization link.

Type
str | None

role_connections_verification_url

The role connection verification entry point URL.

Type
str | None

class novus.**Asset**(*resource: str, animated: bool = MISSING, default_format: str | None = None*)

A representation of a discord image model.

resource

The path assicated with the URL.

Type
str

animated

Whether or not the asset is animated.

Type
bool

get_url(*format: Literal['webp', 'jpg', 'jpeg', 'png', 'gif', 'json'] = MISSING, size: int = MISSING*) → str

Get the URL for the image with different formatting and size than the CDN default.

Parameters
format (*str*) – The format that you want to get the URL as.

class novus.**Attachment**(*data: payloads.Attachment*)

An attachment sent with a message.

id

The ID of the attachment.

Type
int

filename

The filename for the attachment.

Type
str

size

The size of the attachment.

Type
int

url

A URL to the attachment on the CDN.

Type
str

proxy_url

A proxy URL to the attachment on the CDN.

Type
str

class novus.**AuditLog**(*, *data: AuditLogPayload*, *state:* HTTPConnection, *guild: Snowflake*)

A model containing the audit logs for a guild.

entries

The entries contained in the audit log.

Type
list[*novus.AuditLogEntry*]

async classmethod fetch(*state:* HTTPConnection, *guild_id: int*, *, *user_id: int* | *None* = *None*, *action_type: int* | *None* = *None*, *before: int* | *None* = *None*, *after: int* | *None* = *None*, *limit: int* = *50*) → *AuditLog*

Get an instance of a user from the API.

Parameters

- **state** (HTTPConnection) – The API connection.
- **guild_id** (*int*) – The ID associated with the user you want to get.
- **user_id** (*Optional[int]*) – The ID of the moderator you want to to filter by.
- **action_type** (*Optional[int]*) – The type of action that you want to filter by.

  See also:

  novus.AuditLogEventType
- **before** (*Optional[int]*) – The snowflake before which to get entries.
- **after** (*Optional[int]*) – The snowflake after which to get entries.
- **limit** (*Optional[int]*) – The number of entries to get. Max 100, defaults to 50.

Returns
The audit log for the guild.

Return type
novus.AuditLog

class novus.**AuditLogContainer**(***kwargs: Any*)

A proxy object for audit log changes, and extra information given back from Discord. This can hold a wide variety of information (attributes of changed entities; additional parameters for a user action; etc), so can be iterated over like a *dict* for easy access.

class novus.**AuditLogEntry**(**, data: AuditLogEntryPayload, log:* AuditLog | *None*)

An individual entry in the audit log.

id

The ID of the entry.

Type
int

reason

The reason added to the entry, if one was given.

Type
str | None

target_id

The ID of the affected entity.

Type
int | None

target

The affected entity.

Type
novus.abc.Snowflake | None

user_id

The ID of the user or app that made the changes.

Type
int | None

user

The user or app that made the changes.

Type
novus.User | None

action_type

The action that was applied.

Type
novus.AuditLogEvent

options

Additional information for the entry.

Type
novus.AuditLogContainer

before

The state of the object before the action happened. Could be `None` in the case of new objects being created.

Type
novus.AuditLogContainer | None

after

The state of the object after the action happened. Could be `None` in the case of an object being removed.

Type
novus.AuditLogContainer | None

class novus.**AutoModerationAction**(*type: int*, *, *channel: int | abc.Snowflake | None = None*, *duration: int | None = None*)

A moderation action to be taken on a rule being triggered.

Parameters

- **type** (*int*) – The type of action to be taken.

  See also:

  novus.AutoModerationActionType

- **channel** (*int | novus.abc.Snowflake | None*) – The channel associated with the action. Can only be set if the action type is *AutoModerationActionType.SEND_ALERT_MESSAGE*.
- **duration** (*int | None*) – The duration (in seconds) associated with the action. Can only be set if the action type is *AutoModerationActionType.TIMEOUT*.

type

The type of action to be taken.

See also:

novus.AutoModerationActionType

Type
int

channel_id

The channel ID associated with the action. Will only be set if the action type is *AutoModerationActionType.SEND_ALERT_MESSAGE*.

Type
int | None

duration

The duration (in seconds) associated with the action. Will only be set if the action type is *AutoModerationActionType.TIMEOUT*.

Type
int | None

class novus.**AutoModerationRule**(*, *state:* HTTPConnection, *data: RulePayload*)

A model representing an auto moderation rule.

id

The ID of the rule.

Type
int

guild_id

The ID of the guild that the rule is tied to.

Type
int

`name`

The name given to the rule.

Type
str

`creator_id`

The ID of the user that created the rule.

Type
int

`event_type`

The event type.

See also:

novus.AutoModerationEventType

Type
int

`trigger_type`

The trigger type for the rule.

See also:

novus.AutoModerationTriggerType

Type
int

`trigger_metadata`

The metadata associated with the rule.

Type
novus.AutoModerationTriggerMetadata

`actions`

A list of actions taken when the rule is triggered.

Type
list[*novus.AutoModerationAction*]

`enabled`

Whether the rule is enabled.

Type
bool

`exempt_role_ids`

A list of IDs corresponding to roles that are exempt from this rule.

Type
list[int]

exempt_channel_ids

A list of IDs corresponding to channels that are exempt from this rule.

Type
list[int]

guild

A guild object (or a snowflake object).

Type
novus.abc.Snowflake

async classmethod fetch(*state: HTTPConnection, guild: int | abc.Snowflake, rule: int | abc.Snowflake*) → *AutoModerationRule*

Get an instance of an auto moderation rule from the API.

Parameters

- **state** (HTTPConnection) – The API connection.
- **guild** (*int | novus.abc.Snowflake*) – An association to a guild that you want to get the rule from.
- **rule** (*int | novus.abc.Snowflake*) – An association to get the rule from.

Returns
The auto moderation rule.

Return type
novus.AutoModerationRule

async classmethod fetch_all_for_guild(*state: HTTPConnection, guild: int | abc.Snowflake*) → list[*AutoModerationRule*]

Get all of the auto moderation rules from the API for a given guild.

Parameters

- **state** (*novus.HTTPConnection*) – The API connection to manage the entity with.
- **guild** (*int | novus.abc.Snowflake*) – The guild that you want to get the rules from.

Returns
The list of auto moderation rules in the guild.

Return type
list[*novus.AutoModerationRule*]

async edit(**, reason: str | None = None, name: str = MISSING, event_type: int = MISSING, trigger_type: int = MISSING, trigger_metadata: AutoModerationTriggerMetadata = MISSING, actions: list[AutoModerationAction] = MISSING, enabled: bool = MISSING, exempt_roles: list[int | abc.Snowflake] = MISSING, exempt_channels: list[int | abc.Snowflake] = MISSING*) → *AutoModerationRule*

Edit an instance of the auto moderation rule.

Parameters

- **name** (*str*) – The new name for the role.
- **event_type** (*int*) – The event type.

  See also:

  novus.AutoModerationEventType

- **trigger_type** (*int*) – The trigger type.

  See also:

  novus.AutoModerationTriggerType

- **trigger_metadata** (novus.AutoModerationTriggerMetadata) – The trigger metadata.
- **actions** (*list[*novus.AutoModerationAction*]*) – The actions to be taken on trigger.
- **enabled** (*bool*) – Whether the rule is enabled or not.
- **exempt_roles** (*list[int | novus.abc.Snowflake]*) – A list of roles that are exempt from the rule.
- **exempt_channels** (*list[int | novus.abc.Snowflake]*) – A list of channels that are exempt from the rule.
- **reason** (*str | None*) – The reason shown in the audit log.

Returns
The updated rule.

Return type
novus.AutoModerationRule

async delete(*, *reason: str | None = None*) → None

Delete this auto moderation rule.

Parameters
reason (*str | None*) – The reason shown in the audit log.

async classmethod create(*state: HTTPConnection, guild: int | abc.Snowflake, *, reason: str | None = None, name: str, event_type: int, trigger_type: int, actions: list[AutoModerationAction], trigger_metadata: AutoModerationTriggerMetadata | None = None, enabled: bool = False, exempt_roles: list[int | abc.Snowflake] | None = None, exempt_channels: list[int | abc.Snowflake] | None = None*) → *AutoModerationRule*

Create a new auto moderation rule.

Parameters

- **state** (*novus.HTTPConnection*) – The API connection to create the entity with.
- **guild** (*int | novus.abc.Snowflake*) – The ID of the guild to create the object in.
- **name** (*str*) – The new name for the role.
- **event_type** (*int*) – The event type.

  See also:

  novus.AutoModerationEventType

- **trigger_type** (*int*) – The trigger type.

  See also:

  novus.AutoModerationTriggerType

- **actions** (*list[*novus.AutoModerationAction*]*) – The actions to be taken on trigger.
- **trigger_metadata** (novus.AutoModerationTriggerMetadata *| None*) – The trigger metadata.

- **enabled** (*bool*) – Whether the rule is enabled or not.
- **exempt_roles** (*list[int | novus.abc.Snowflake] | None*) – A list of roles that are exempt from the rule.
- **exempt_channels** (*list[int | novus.abc.Snowflake] | None*) – A list of channels that are exempt from the rule.
- **reason** (*str | None*) – The reason shown in the audit log.

Returns
The created rule.

Return type
novus.AutoModerationRule

class novus.**AutoModerationTriggerMetadata**(*, *keyword_filters: list[str] | None = None*, *regex_patterns: list[str] | None = None*, *presets: list[int] | None = None*, *allow_list: list[str] | None = None*, *mention_total_limit: int | None = None*)

The metadata associated with an auto moderation trigger.

Parameters

- **keyword_filters** (*list[str] | None*) – A list of substrings which will be searched for in content. A keyword can be a phrase which contains multiple words. Wildcard symbols (*) can be used to customize how much of each keyword will be matched.
- **regex_patterns** (*list[str] | None*) – A list of regular expression patterns that will be matched against the content. Only rust flavored regex is supported.
- **presets** (*list[int] | None*) – A list of preset word lists that you want to match against.

  See also:

  novus.AutoModerationKeywordPresetType
- **allow_list** (*list[str] | None*) – A list of substrings which should not trigger the rule.
- **mention_total_limit** (*int | None*) – The total number of unique role and user mentions allowed per message.

class novus.**BaseGuild**(*, *state:* HTTPConnection, *data: payloads.Guild*)

The bare minimum guild instance we can have. Basically a snowflake with a state, a name, and API methods.

state

The connection to Discord.

Type
novus.api.HTTPConnection

id

The ID of the guild.

Type
int

name

The name of the guild. Can be `None` if this is just a state snowflake implementing API methods.

Type
str | None

async classmethod create(*state:* HTTPConnection, *, *name: str*) → *Guild*

Create a guild.

Parameters

- **state** (*novus.HTTPConnection*) – The API connection to create the entity with.
- **name** (*str*) – The name for the guild that you want to create.

Returns
The created guild.

Return type
novus.Guild

async classmethod fetch(*state:* HTTPConnection, *guild: AnySnowflake*) → *Guild*

Get an instance of a guild from the API. Unlike the gateway's GUILD_CREATE payload, this method does not return members, channels, or voice states.

Parameters

- **state** (HTTPConnection) – The API connection.
- **guild** (*int | novus.abc.Snowflake*) – A reference to the guild that you want to fetch.

Returns
The guild associated with the given ID.

Return type
novus.Guild

async edit(*, *name: str = MISSING*, *verification_level: int | None = MISSING*, *default_message_notifications: int | None = MISSING*, *explicit_content_filter: int | None = MISSING*, *afk_channel: AnySnowflake | None = MISSING*, *icon: FileT | None = MISSING*, *owner: AnySnowflake = MISSING*, *splash: FileT | None = MISSING*, *discovery_splash: FileT | None = MISSING*, *banner: FileT | None = MISSING*, *system_channel: AnySnowflake | None = MISSING*, *system_channel_flags:* SystemChannelFlags *| None = MISSING*, *rules_channel: AnySnowflake | None = MISSING*, *preferred_locale: str | None = MISSING*, *public_updates_channel: AnySnowflake = MISSING*, *features: list[str] = MISSING*, *description: str | None = MISSING*, *premium_progress_bar_enabled: bool = MISSING*, *reason: str | None = None*) → *Guild*

Edit the guild parameters.

Note: The updated guild is not immediately put into cache - the bot waits for the guild update notification to be sent over the gateway before updating (which will not happen if you don't have the correct gateway intents).

Parameters

- **name** (*str*) – The name you want to set the guild to.
- **verification_level** (*int | None*) – The verification level you want to set the guild to.

  See also:

  novus.VerificationLevel

- **default_message_notifications** (*int | None*) – The default message notification level you want to set the guild to.

See also:

novus.NotificationLevel

- **explicit_content_filter** (*int* | *None*) – The content filter level you want to set the guild to.

  See also:

  novus.guild.ContentFilterLevel

- **afk_channel** (*int* | *novus.abc.Snowflake* | *None*) – The channel you want to set as the guild's AFK channel.
- **icon** (*str* | *bytes* | *io.IOBase* | *None*) – The icon that you want to set for the guild. Can be its bytes, a file path, or a file object.
- **owner** (*int* | *novus.abc.Snowflake*) – The person you want to set as owner of the guild. Can only be run if the current user is the existing owner.
- **splash** (*str* | *bytes* | *io.IOBase* | *None*) – The splash that you want to set for the guild. Can be its bytes, a file path, or a file object.
- **discovery_splash** (*str* | *bytes* | *io.IOBase* | *None*) – The discovery splash for the guild. Can be its bytes, a file path, or a file object.
- **banner** (*str* | *bytes* | *io.IOBase* | *None*) – The banner for the guild. Can be its bytes, a file path, or a file object.
- **system_channel** (*int* | *novus.abc.Snowflake* | *None*) – The system channel you want to set for the guild.
- **system_channel_flags** (*novus.guild.SystemChannelFlags* | *None*) – The system channel flags you want to set.
- **rules_channel** (*int* | *novus.abc.Snowflake* | *None*) – The channel you want to set as the rules channel.
- **preferred_locale** (*str* | *None*) – The locale you want to set as the guild's preferred.
- **public_updates_channel** (*int* | *novus.abc.Snowflake*) – The channel you want to set as the updates channel for the guild.
- **features** (*list[str]*) – A list of features for the guild.
- **description** (*str* | *None*) – A description for the guild.
- **premium_progress_bar_enabled** (*bool*) – Whether or not to enable the premium progress bar for the guild.
- **reason** (*str* | *None*) – A reason for modifying the guild (shown in the audit log).

Returns
The updated guild.

Return type
novus.Guild

async delete() → None

Delete the current guild permanently. You must be the owner of the guild to run this successfully.

async fetch_invites() → list[*Invite*]

Get the invites for the guild.

Requires the `MANAGE_GUILD` permission.

Returns
A list of invites.

Return type
list[*novus.Invite*]

async chunk_members(*query: str*, *limit: int*, *user_ids: list[int]*, *wait: Literal[False]*) → None

async chunk_members(*query: str*, *limit: int*, *user_ids: list[int]*, *wait: Literal[True]*) → list[*GuildMember*]

Request member chunks from the gateway.

This will *only* work if you are connected to the gateway - this will not work with HTTP-only bots.

Parameters

- **query** (*str*) – A search string for usernames.
- **limit** (*int*) – A limit for the retrieved member count.
- **user_ids** (*list[int]*) – A list of user IDs to request.
- **wait** (*bool*) – Whether or not to wait for a response.

Returns
A list of requested users or `None` if you chose not to wait.

Return type
list[*novus.GuildMember*] | None

async fetch_audit_logs(*, *user_id: int | None = None*, *action_type: int | None = None*, *before: int | None = None*, *after: int | None = None*, *limit: int = 50*) → *AuditLog*

Get the audit logs for the guild.

Parameters

- **user_id** (*int | None*) – The ID of the moderator you want to to filter by.
- **action_type** (*int | None*) – The ID of an action to filter by.

  See also:

  novus.AuditLogEventType
- **before** (*int | None*) – The snowflake before which to get entries.
- **after** (*int | None*) – The snowflake after which to get entries.
- **limit** (*int*) – The number of entries to get. Max 100, defaults to 50.

Returns
The audit log for the guild.

Return type
novus.AuditLog

async fetch_auto_moderation_rules() → list[*AutoModerationRule*]

Get the auto moderation rules for this guild.

Returns
A list of the auto moderation rules for the guild.

Return type
list[*novus.AutoModerationRule*]

async create_auto_moderation_rule(*, *reason: str | None = None*, *name: str*, *event_type: int*, *trigger_type: int*, *actions: list[AutoModerationAction]*, *trigger_metadata: AutoModerationTriggerMetadata | None = None*, *enabled: bool = False*, *exempt_roles: list[AnySnowflake] | None = None*, *exempt_channels: list[AnySnowflake] | None = None*) → *AutoModerationRule*

Create a new auto moderation rule.

Parameters

- **name** (*str*) – The new name for the role.
- **event_type** (*int*) – The event type.

  See also:

  novus.AutoModerationEventType

- **trigger_type** (*int*) – The trigger type.

  See also:

  novus.AutoModerationTriggerType

- **actions** (*list[novus.AutoModerationAction]*) – The actions to be taken on trigger.
- **trigger_metadata** (*novus.AutoModerationTriggerMetadata | None*) – The trigger metadata.
- **enabled** (*bool*) – Whether the rule is enabled or not.
- **exempt_roles** (*list[int | novus.abc.Snowflake] | None*) – A list of roles that are exempt from the rule.
- **exempt_channels** (*list[int | novus.abc.Snowflake] | None*) – A list of channels that are exempt from the rule.
- **reason** (*str | None*) – The reason shown in the audit log.

Returns
The created rule.

Return type
novus.AutoModerationRule

async fetch_channels() → list[*Channel*]

Fetch all of the channels from a guild.

Returns
A list of channels from the guild.

Return type
list[*novus.Channel*]

async create_channel(*, *name: str*, *type: int = MISSING*, *topic: str = MISSING*, *bitrate: int = MISSING*, *user_limit: int = MISSING*, *rate_limit_per_user: int = MISSING*, *position: int = MISSING*, *permission_overwrites: list[PermissionOverwrite] = MISSING*, *parent: AnySnowflake = MISSING*, *nsfw: bool = MISSING*, *default_auto_archive_duration: int = MISSING*, *default_reaction_emoji: Reaction = MISSING*, *available_tags: list[ForumTag] = MISSING*, *reason: str = MISSING*) → *Channel*

Create a channel within the guild.

Parameters

- **name** (*str*) – The name of the channel.
- **type** (*int*) – The type of the channel.

  See also:

  novus.ChannelType
- **bitrate** (*int*) – The bitrate for the channel. Only for use with voice channels.
- **user_limit** (*int*) – The user limit for the channel. Only for use with voice channels.
- **rate_limit_per_user** (*int*) – The slowmode seconds on the channel.
- **position** (*int*) – The channel position.
- **permission_overwrites** (*list[novus.PermissionOverwrite]*) – A list of permission overwrites for the channel.
- **parent** (*int | str | novus.abc.Snowflake*) – A parent object for the channel.
- **nsfw** (*bool*) – Whether or not the channel will be set to NSFW.
- **default_auto_archive_duration** (*int*) – The default duration that clients use (in minutes) to automatically archive the thread after recent activity. Only for use with forum channels.
- **default_reaction_emoji** (Reaction) – The default add reaction button to be shown on threads. Only for use with forum channels.
- **available_tags** (*list[ForumTag]*) – The tags available for threads. Only for use with forum channels.
- **reason** (*str*) – The reason to be shown in the audit log.

Returns
The created channel.

Return type
novus.model.Channel

async fetch_active_threads() → list[*Channel*]

Get the active threads from inside the guild.

Returns
A list of threads.

Return type
list[*novus.Channel*]

async fetch_emoji(*id: int*) → *Emoji*

List all of the emojis for the guild.

See also:

novus.Emoji.fetch()

Returns
A list of the guild's emojis.

Return type
list[*novus.Emoji*]

async fetch_all_emojis() → list[*Emoji*]

List all of the emojis for the guild.

See also:

novus.Emoji.fetch_all_for_guild()

Returns
A list of the guild's emojis.

Return type
list[*novus.Emoji*]

async create_emoji(*, *name: str*, *image: FileT*, *roles: list[AnySnowflake] | None = None*, *reason: str | None = None*) → *Emoji*

Create an emoji within a guild.

Parameters

- **name** (*str*) – The name of the emoji you want to add.
- **image** (*str | bytes | io.IOBase*) – The image that you want to add.
- **roles** (*list[int | novus.abc.Snowflake] | None*) – A list of roles that are allowed to use the emoji.
- **reason** (*str | None*) – A reason you're adding the emoji.

Returns
The newly created emoji.

Return type
novus.Emoji

async fetch_roles() → list[*Role*]

Get a list of roles for the guild.

Returns
A list of roles in the guild.

Return type
list[novus.model.Role]

async create_role(*, *reason: str | None = None*, *name: str = MISSING*, *permissions:* Permissions *= MISSING*, *color: int = MISSING*, *hoist: bool = MISSING*, *icon: FileT = MISSING*, *unicode_emoji: str = MISSING*, *mentionable: bool = MISSING*) → *Role*

Create a role within the guild.

Parameters

- **name** (*str*) – The name of the role.
- **permissions** (*novus.Permissions*) – The permissions attached to the role.
- **color** (*int*) – The color of the role.
- **hoist** (*bool*) – Whether the role is displayed seperately in the sidebar.
- **icon** (*str | bytes | io.IOBase | None*) – The role icon image. Only usable if the guild has the `ROLE_ICONS` feature.
- **unicode_emoji** (*str*) – The role's unicode emoji. Only usable if the guild has the `ROLE_ICONS` feature.

- **mentionable** (*bool*) – Whether the role should be mentionable.
- **reason** (*str | None*) – The reason to be shown in the audit log.

async edit_role(*role_id: int, *, reason: str | None = None, name: str = MISSING, permissions: Permissions = MISSING, color: int = MISSING, hoist: bool = MISSING, icon: FileT = MISSING, unicode_emoji: str = MISSING, mentionable: bool = MISSING*) → *Role*

Edit a role.

Parameters

- **role_id** (*int*) – The ID of the role to be edited.
- **name** (*str*) – The new name of the role.
- **permissions** (*novus.Permissions*) – The permissions to be applied to the role.
- **color** (*int*) – The color to apply to the role.
- **hoist** (*bool*) – If the role should be displayed seperately in the sidebar.
- **icon** (*str | bytes | io.IOBase | None*) – The role's icon image. Only usable if the guild has the `ROLE_ICONS` feature.
- **unicode_emoji** (*str*) – The role's unicode emoji. Only usable if the guild has the `ROLE_ICONS` feature.
- **mentionable** (*bool*) – If the role is mentionable.
- **reason** (*str | None*) – The reason to be shown in the audit log.

async delete_role(*role: AnySnowflake, *, reason: str | None = None*) → None

A role to delete.

Parameters

- **role** (*int | novus.abc.Snowflake*) – The ID of the role to delete.
- **reason** (*str | None*) – The reason to be shown in the audit log.

async fetch_scheduled_events(*, *with_user_count: bool = False*) → list[*ScheduledEvent*]

Get a list of all of the scheduled events for a guild.

See also:

novus.ScheduledEvent.fetch_all_for_guild()

Parameters

with_user_count (*bool*) – Whether or not to include the event's user count.

Returns

The scheduled events for the guild.

Return type

list[*novus.ScheduledEvent*]

async create_scheduled_event(*, *name: str, start_time: DiscordDatetime, entity_type: int, privacy_level: int, reason: str | None = None, channel: AnySnowflake | None = MISSING, location: str = MISSING, end_time: DiscordDatetime = MISSING, description: str | None = MISSING, status: int = MISSING, image: FileT | None = MISSING*) → *ScheduledEvent*

Create a new scheduled event.

See also:

novus.ScheduledEvent.create()

Parameters

- **name** (*str*) – The name of the event.
- **start_time** (*datetime.datetime*) – The time to schedule the event start.
- **entity_type** (*int*) – The type of the event.
- **privacy_level** (*int*) – The privacy level of the event.

  See also:

  novus.EventPrivacyLevel
- **channel** (*int | Snowflake | None*) – The channel of the scheduled event. Set to `None` if the event type is being set to external.
- **location** (*str*) – The location of the event.
- **end_time** (*datetime.datetime*) – The time to schedule the event end.
- **description** (*str | None*) – The description of the event.
- **status** (*int*) – The status of the event.

  See also:

  novus.EventStatus
- **image** (*str | bytes | io.IOBase | None*) – The cover image of the scheduled event.
- **reason** (*str | None*) – The reason shown in the audit log.

Returns
The new scheduled event.

Return type
novus.ScheduledEvent

async fetch_sticker(*id: AnySnowflake*) → *Sticker*

Get an individual sticker associated with the guild via its ID.

See also:

novus.Sticker.fetch()

Parameters
id (*str*) – The ID of the sticker.

Returns
The associated sticker instance.

Return type
novus.Sticker

async fetch_all_stickers() → list[*Sticker*]

List all stickers associated with the guild.

See also:

novus.Sticker.fetch_all_for_guild()

Returns
The stickers associated with the guild.

Return type
list[*novus.Sticker*]

async create_sticker(*, *reason: str | None = None*, *name: str*, *description: str | None = None*, *tags: str*, *image: File*) → *Sticker*

Create a new sticker.

See also:

novus.Sticker.create()

Parameters

- **name** (*str*) – The name of the sticker.
- **tags** (*str*) – Autocomplete/suggestion tags for the sitcker.
- **description** (*str | None*) – Description of the sticker.
- **image** (novus.File) – The image to be uploaded. All aside from the data itself is discarded - the name and description are taken from the other parameters.
- **reason** (*str | None*) – The reason shown in the audit log.
- **Reutrns**
- **-------**
- **novus.Sticker** – The created sticker instance.

async fetch_me() → *GuildMember*

Get the member object associated with the current guild and the current connection.

Note: Only usable via Oauth with the `guilds.members.read` scope. This is not usable as a bot.

See also:

novus.GuildMember.fetch_me()

Returns
The member object for the current user.

Return type
novus.GuildMember

async leave() → None

Leave the current guild.

async fetch_member(*member_id: int*) → *GuildMember*

Get a member from the guild.

See also:

novus.GuildMember.fetch()

Parameters
member_id (*int*) – The ID of the member you want to get.

Returns
The member object for the given user.

Return type
novus.GuildMember

async fetch_members(*, *limit: int = 1000*, *after: int = 0*) → list[*GuildMember*]

Get a list of members for the guild.

Note: This endpoint is restricted according to whether the `GUILD_MEMBERS` privileged intent is enabled for your application.

Note: This endpoint can return a maximum of 1000 members per request.

Parameters

- **limit** (*int*) – The number of guild members you want in the response payload.
- **after** (*int*) – The snowflake to get guild members after.

Returns
A list of members from the guild.

Return type
list[*novus.GuildMember*]

async search_members(*, *query: str*, *limit: int = 1*) → list[*GuildMember*]

Get a list of members for the guild whose username of nickname starts with the provided string.

Note: This endpoint can return a maximum of 1000 members per request.

Parameters

- **query** (*str*) – the query string to match usernames and nicknames agains.
- **limit** (*int*) – The number of guild members you want in the response payload.

Returns
A list of members from the guild.

Return type
list[*novus.GuildMember*]

async add_member(*user_id: int*, *access_token: str*, *, *nick: str = MISSING*, *mute: bool = MISSING*, *deaf: bool = MISSING*) → *GuildMember* | None

Add a member to the guild.

Note: This requires an Oauth access token, and the provided user ID must be the same one that matches the account.

Parameters

- **user_id** (*int*) – The ID of the user that you want to add. The user ID must match the ID of the oauth token.
- **access_token** (*str*) – The access token with the `guilds.join` scope to the bot's application for the user you want to add to the guild.
- **nick** (*str*) – The nickname youy want to set the user to.
- **mute** (*bool*) – Whether the user is muted in voice channels.
- **deaf** (*bool*) – Whether the user is deafened in voice channels.

Returns
The member for the user that was added to the guild, or `None` if the user was already present.

Return type
novus.GuildMember | None

async edit_member(*user: AnySnowflake*, **, reason: str | None = None*, *nick: str | None = MISSING*, *roles: list[AnySnowflake] = MISSING*, *mute: bool = MISSING*, *deaf: bool = MISSING*, *voice_channel: AnySnowflake | None = MISSING*, *timeout_until: DiscordDatetime | None = MISSING*) → *GuildMember*

Edit a guild member.

See also:

novus.GuildMember.edit()

Parameters

- **user** (*int | novus.abc.Snowflake*) – The ID of the user you want to edit.
- **nick** (*str | None*) – The nickname you want to set for the user.
- **roles** (*list[int | novus.abc.Snowflake]*) – A list of roles that you want the user to have.
- **mute** (*bool*) – Whether or not the user is muted in voice channels. Will error if the user is not currently in a voice channel.
- **deaf** (*bool*) – Whether or not the user is deafened in voice channels. Will error if the user is not currently in a voice channel.
- **voice_channel** (*int | novus.abc.Snowflake | None*) – The voice channel that the user is in.
- **timeout_until** (*datetime.datetime | None*) – When the user's timeout should expire (up to 28 days in the future).

async add_member_role(*user: AnySnowflake*, *role: AnySnowflake*, **, reason: str | None = None*) → None

Add a role to a user.

Requires the `MANAGE_ROLES` permission.

See also:

novus.GuildMember.add_role()

Parameters

- **user** (*int | novus.abc.Snowflake*) – The user you want to add the role to.
- **role** (*int | novus.abc.Snowflake*) – The role you want to add.

- **reason** (*str | None*) – The reason shown in the audit log.

async remove_member_role(*user: AnySnowflake*, *role: AnySnowflake*, **, reason: str | None = None*) → None

Remove a role from a member.

Requires the `MANAGE_ROLES` permission.

See also:

novus.GuildMember.remove_role()

Parameters

- **user** (*int | novus.abc.Snowflake*) – The user you want to add the role to.
- **role** (*int | novus.abc.Snowflake*) – The ID of the role you want to add.
- **reason** (*str | None*) – The reason shown in the audit log.

async kick(*user: AnySnowflake*, **, reason: str | None = None*) → None

Remove a user from the guild.

Requires the `KICK_MEMBERS` permission.

Parameters

- **user** (*int | novus.abc.Snowflake*) – The user you want to remove.
- **reason** (*str | None*) – The reason to be shown in the audit log.

async fetch_bans(**, limit: int = 1000, before: int | None = None, after: int | None = None*) → list[*GuildBan*]

Get a list of bans from the guild.

Parameters

- **limit** (*str*) – The number of bans to get.
- **before** (*int | None*) – The snowflake to search around.
- **after** (*int | None*) – The snowflake to search around.

Returns
A list of bans from the guild.

Return type
list[novus.model.GuildBan]

async fetch_ban(*user: AnySnowflake*) → *GuildBan*

Get an individual user's ban.

Parameters
user (*int | novus.abc.Snowflake*) – The user whose ban you want to get.

Returns
The ban for the user.

Return type
novus.GuildBan

async ban(*user: AnySnowflake*, ***, *reason: str | None = None*, *delete_message_seconds: int = MISSING*) → None

Ban a user from the guild.

See also:

novus.GuildMember.ban()

Parameters

- **user** (*int | novus.abc.Snowflake*) – The user who you want to ban.
- **delete_message_seconds** (*int*) – The number of seconds of messages you want to delete.
- **reason** (*str | None*) – The reason to be shown in the audit log.

async unban(*user: AnySnowflake*, ***, *reason: str | None = None*) → None

Remove a user's ban

Parameters

- **user** (*int | novus.abc.Snowflake*) – The user who you want to ban.
- **reason** (*str | None*) – The reason to be shown in the audit log.

class novus.**Channel**(***, *state: HTTPConnection*, *data: payloads.Channel*, *guild_id: int | str | None = None*)

The base channel object that all other channels inherit from. This is also the object that will be returned if there is an unknown channel type.

id

The ID of the channel.

Type
int

type

The type of the channel.

See also:

novus.ChannelType

Type
int

guild

The guild that the channel is attached to.

Type
novus.abc.Snowflake | None

classmethod partial(*state: HTTPConnection*, *id: int | str*, *type: int = 0*) → Self

Create a partial channel object that you can use to run API methods on.

Parameters

- **state** (*novus.api.HTTPConnection*) – The API connection.
- **id** (*int*) – The ID of the channel.

Returns
A created channel object.

Return type
novus.Channel

permissions_for(*user:* GuildMember) → *Permissions*

Get the permissions for a given user in this channel.

Note: Permissions are only properly calculated when the guild and its roles are cached (ie when the bot is connected to the gateway).

Parameters
user (novus.GuildMember) – The user whose permissions you want to get.

Returns
The calculated permissions for that user in this channel.

Return type
novus.Permissions

async classmethod fetch(*state:* HTTPConnection, *id:* *int* | *abc.Snowflake*) → *Channel*

Fetch a channel from the API.

Parameters

- **state** (novus.api.HTTPConnection) – The API connection.
- **id** (*int | novus.abc.Snowflake*) – The ID of the channel you want to fetch.

Returns
The channel instance.

Return type
novus.Channel

async edit(**, reason: str | None = None, name: str = MISSING, position: int = MISSING, topic: str = MISSING, nsfw: bool = MISSING, rate_limit_per_user: int = MISSING, bitrate: int = MISSING, user_limit: int = MISSING, default_auto_archive_duration: Literal[60, 1440, 4320, 10080] = MISSING, default_thread_rate_limit_per_user: int = MISSING, archived: bool = MISSING, auto_archive_duration: Literal[60, 1440, 4320, 10080] = MISSING, locked: bool = MISSING, invitable: bool = MISSING, flags: ChannelFlags = MISSING, default_sort_order: int = MISSING, default_forum_layout: int = MISSING, parent: abc.Snowflake | Channel = MISSING, overwrites: list[PermissionOverwrite] = MISSING, available_tags: list[ForumTag] = MISSING, default_reaction_emoji: str | PartialEmoji = MISSING, applied_tags: list[int | ForumTag] = MISSING*) → *Channel*

Edit the instance of the channel.

Parameters

- **name** (*str*) – The name of the channel.
- **position** (*int*) – The position of the channel.
- **topic** (*str*) – The topic for the channel. Only applies to text channels.
- **nsfw** (*bool*) – Whether or not the channel should be marked as NSFW.
- **rate_limit_per_user** (*int*) – The rate limit (in seconds) for the channel.

- **bitrate** (*int*) – The bitrate for the channel. Only applies to voice channels.
- **user_limit** (*int*) – The user limit for the channel. Only applies to voice channels.
- **default_auto_archive_duration** (*int*) – The default auto archive duration for the channel. Only applies to forum channels.

  Note: Only accepts the values 60, 1_440, 4_320, and 10_080.

- **default_thread_rate_limit_per_user** (*int*) – The rate limit (in seconds) for the channel.
- **archived** (*bool*) – If the channel is archived. Only applies to threads.
- **auto_archive_duration** (*int*) – The default auto archive duration for the channel. Only applies to forum channels.

  Note: Only accepts the values 60, 1_440, 4_320, and 10_080.

- **locked** (*bool*) – If the channel is locked. Only applies to threads.
- **invitable** (*bool*) – If non-moderators can add other non-moderators to a thread. Only applies to private thread channels.
- **flags** (*novus.ChannelFlags*) – The flags applied to the channel. Only applies to forum channels and threads within forum channels.
- **default_sort_order** (*int*) – The sort order of the forum. Only applies to forum channels.

  See also:

  novus.ForumSortOrder

- **default_forum_layout** (*int*) – The layout of the forum. Only applies to forum channels.

  See also:

  novus.ForumLayout

- **parent** (*novus.abc.Snowflake*) – A parent channel.
- **overwrites** (*list[novus.PermissionOverwrite]*) – A list of permission overwrites for the channel.
- **available_tags** (*list[novus.ForumTag]*) – A list of tags available. Only applies to forum channels.
- **default_reaction_emoji** (*str | novus.PartialEmoji*) – The default reaction for each thread. Only applies to forum channels.
- **applied_tags** (*list[novus.ForumTag]*) – A list of tags applied to the channel. Only applies to threads within forums.
- **reason** (*str | None*) – The reason added to the entry, if one was given.

async delete(*, *reason: str | None = None*) → None

Delete the channel instance.

Parameters

reason (*str | None*) – The reason shown in the audit log, if the channel is a guild channel.

async fetch_invites() → list[*Invite*]

Get a list of invites to the channel.

Returns
A list of invites for the channel.

Return type
list[*novus.Invite*]

async create_invite(*, *reason: str | None = None*, *max_age: int = 86400*, *max_uses: int = 0*, *temporary: bool = False*, *unique: bool = False*) → *Invite*

Delete multiple messages at once.

Parameters

- **max_age** (*int*) – The duration of the invite (in seconds) before expiry, or `0` for never. Cannot be larger than `604_800`.
- **max_uses** (*int*) – A maximum number of uses for the invite, or `0` for unlimited. Cannot be larger than `100`.
- **temporary** (*bool*) – Whether the invite only grants temporary membership to the guild.
- **unique** (*bool*) – If you want to reuse a similar invite.
- **reason** (*str | None*) – The reason shown in the audit log.

Returns
The created invite.

Return type
novus.Invite

async remove_overwrite(*target: int | abc.Snowflake* | Role | User | GuildMember, *, *reason: str | None = None*) → None

Remove a specific overwrite in the channel.

Parameters

- **target** (*int | novus.abc.Snowflake* | novus.Role | novus.User | novus.GuildMember) – The overwrite that you want to remove.
- **reason** (*str | None*) – The reason shown in the audit log.

async create_overwrite(*target: int | abc.Snowflake* | Role | User | GuildMember, *, *reason: str | None = None*, *allow:* Permissions *= MISSING*, *deny:* Permissions *= MISSING*, *overwrite_type: Type[*Role*] | Type[*User*] | Type[*GuildMember*] = MISSING*) → None

Create a permission overwrite for a given channel.

Parameters

- **target** (*int | novus.abc.Snowflake* | novus.Role | novus.User | novus.GuildMember) – The overwrite that you want to add.
- **allow** (*flags.Permissions*) – The permissions you want to explicitly grant to the target.
- **deny** (*flags.Permissions*) – The permissions you wan tto explicitly deny from the target.
- **reason** (*str | None*) – The reason shown in the audit log.

async fetch_messages(*, *limit: int = 100*, *around: int | abc.Snowflake | Message = MISSING*, *before: int | abc.Snowflake | Message = MISSING*, *after: int | abc.Snowflake | Message = MISSING*) → list[*Message*]

Get a number of messages from the channel.

Parameters

- **limit** (*int*) – The number of messages that you want to get. Maximum 100.
- **around** (*int | novus.abc.Snowflake*) – Get messages around this ID. Only one of `around`, `before`, and `after` can be set.
- **before** (*int | novus.abc.Snowflake*) – Get messages before this ID. Only one of `around`, `before`, and `after` can be set.
- **after** (*int | novus.abc.Snowflake*) – Get messages after this ID. Only one of `around`, `before`, and `after` can be set.

Returns
The messages that were retrieved.

Return type
list[*novus.Message*]

messages(*, *limit: int | None = 100*, *before: int | abc.Snowflake | Message = MISSING*, *after: int | abc.Snowflake | Message = MISSING*) → *APIIterator*[*Message*]

Get an iterator of messages from a channel.

Examples

```
async for message in channel.messages(limit=1_000):
    print(message.content)
```

```
messages = await channel.messages(limit=200).flatten()
```

Parameters

- **limit** (*int*) – The number of messages that you want to get.
- **before** (*int | novus.abc.Snowflake*) – Get messages before this ID. Only one of `around`, `before`, and `after` can be set.
- **after** (*int | novus.abc.Snowflake*) – Get messages after this ID. Only one of `around`, `before`, and `after` can be set.

Returns
The messages that were retrieved, as a generator.

Return type
APIIterator[*novus.Message*]

async fetch_message(*id: int | abc.Snowflake*) → *Message*

Get a single message from the channel.

See also:

novus.Message.fetch()

Parameters
id (*int | novus.abc.Snowflake*) – The message that you want to get.

Returns
The retrieved message.

Return type
novus.Message

async trigger_typing() → None

Send a typing indicator to the channel.

typing() → Typing

A typing context manager.

```
async with channel.typing():
    ...
```

async fetch_pinned_messages() → list[*Message*]

Get a list of pinned messages in the channel.

async bulk_delete_messages(*messages: list[int] | list[abc.Snowflake], *, reason: str | None = None*) → None

Delete multiple messages at once.

Parameters

- **messages** (*list[int | novus.abc.Snowflake]*) – A list of the messages that you want to delete.
- **reason** (*str | None*) – The reason shown in the audit log.

async create_thread(*name: str, type: int, *, reason: str | None = None, invitable: bool = MISSING, rate_limit_per_user: int*) → *Channel*

Create a thread that is not connected to an existing message.

Parameters

- **name** (*str*) – The name of the thread.
- **type** (*int*) – The type of the channel.

  See also:

  novus.ChannelType
- **invitable** (*bool*) – Whether non-moderators can add other non-moderators to a thread - only available when creating private threads.
- **rate_limit_per_user** (*int*) – The amount of seconds that a user has to wait before sending another message.
- **reason** (*str | None*) – The reason shown in the audit log.

async create_thread_in_forum(*name: str, *, reason: str | None = None, auto_archive_duration: Literal[60, 1440, 4320, 10080] = MISSING, rate_limit_per_user: int = MISSING, applied_tags: list[int | ForumTag], content: str = MISSING, embeds: list[Embed] = MISSING, allowed_mentions: AllowedMentions = MISSING, components: list[ActionRow] = MISSING, files: list[File] = MISSING, flags: MessageFlags = MISSING*) → *Channel*

Create a thread in a forum or media channel.

Parameters

- **name** (*str*) – The name of the thread.
- **type** (*int*) – The type of the channel.

  See also:

  novus.ChannelType
- **invitable** (*bool*) – Whether non-moderators can add other non-moderators to a thread - only available when creating private threads.
- **rate_limit_per_user** (*int*) – The amount of seconds that a user has to wait before sending another message.
- **reason** (*str | None*) – The reason shown in the audit log.

async follow(*destination: AnySnowflake*) → None

Follow an announcement channel to send messages to the specific target channel.

Parameters

destination (*int | novus.abc.Snowflake*) – The channel you want to send the announcements to.

async join_thread() → None

Adds the current user to a thread.

async add_thread_member(*user: int | abc.Snowflake*) → None

Add a user to a thread.

Parameters

user (*int | novus.abc.Snowflake*) – The user who you want to add.

async leave_thread() → None

Remove the current user from the thread.

async remove_thread_member(*user: int | abc.Snowflake*) → None

Remove a member from the thread.

Parameters

user (*int | novus.abc.Snowflake*) – The user that you want to remove.

async send(*content: str = MISSING, *, tts: bool = MISSING, embeds: list[Embed] = MISSING, allowed_mentions: AllowedMentions = MISSING, components: list[ActionRow] = MISSING, message_reference: Message = MISSING, stickers: list[Sticker] = MISSING, files: list[File] = MISSING, flags: flags.MessageFlags = MISSING*) → *Message*

Send a message to the channel associated with the model.

Parameters

- **content** (*str*) – The content that you want to have in the message
- **tts** (*bool*) – If you want the message to be sent with the TTS flag.
- **embeds** (*list[novus.Embed]*) – The embeds you want added to the message.
- **allowed_mentions** (*novus.AllowedMentions*) – The mentions you want parsed in the message.
- **components** (*list[novus.ActionRow]*) – A list of action rows to be added to the message.
- **message_reference** (*novus.Message*) – A reference to a message you want replied to.

- **stickers** (*list[novus.Sticker]*) – A list of stickers to add to the message.
- **files** (*list[novus.File]*) – A list of files to be sent with the message.
- **flags** (novus.MessageFlags) – The flags to be sent with the message.

class novus.**Emoji**(*, *state:* HTTPConnection, *data: payloads.PartialEmoji | payloads.Emoji*, *guild_id: int | None = None*, *guild:* BaseGuild | *None = None*)

A custom emoji in a guild.

id

The ID of the emoji.

Type
int | None

name

The name of the emoji. Could be `None` in the case that the emoji came from a reaction payload and isn't unicode.

Type
str | None

role_ids

A list of role IDs that can use the role.

Type
list[int]

requires_colons

Whether or not the emoji requires colons to send.

Type
bool

managed

Whether or not the emoji is managed.

Type
bool

animated

If the emoji is animated.

Type
bool

available

If the emoji is available. May be `False` in the case that the guild has lost nitro boosts.

Type
bool

asset

The asset associated with the emoji, if it's a custom emoji.

Type
novus.Asset | None

guild

The guild (or a data container for the ID) that the emoji came from, if it was available.

Type
novus.abc.Snowflake | *novus.Guild* | None

classmethod from_str(*value: str* | PartialEmoji | *None*) → *PartialEmoji* | None

Transform a string into an emoji object.

Parameters
value (*str | novus.PartialEmoji | None*) – The emoji you want converted. Can either be a Discord-style emoji string, a unicode emoji, or a ":smile:" style emoji via its name. If an emoji object is provided, then it is returned unchanged. If `None` is provided, it is returned as is.

Returns
The converted emoji, if a value was provided.

Return type
novus.PartialEmoji | None

Raises
ValueError – If the given value wasn't convertable to an emoji.

async classmethod create(*state:* HTTPConnection, *guild: int | abc.Snowflake*, *, *name: str*, *image: FileT*, *roles: list[int | abc.Snowflake] | None = None*, *reason: str | None = None*) → *Emoji*

Create an emoji within a guild.

Parameters

- **state** (*novus.HTTPConnection*) – The API connection to create the entity with.
- **guild** (*int | novus.abc.Snowflake*) – The guild that the emoji is to be created in.
- **name** (*str*) – The name of the emoji you want to add.
- **image** (*str | bytes | io.IOBase*) – The image that you want to add.
- **roles** (*list[int | novus.abc.Snowflake] | None*) – A list of roles that are allowed to use the emoji.
- **reason** (*str | None*) – A reason you're adding the emoji to be shown in the audit log.

Returns
The newly created emoji.

Return type
novus.Emoji

async classmethod fetch(*state:* HTTPConnection, *guild_id: int*, *emoji_id: int*) → *Emoji*

Fetch a specific emoji by its ID from the API.

See also:

novus.Guild.fetch_emoji()

Parameters

- **guild_id** (*int*) – The ID of the guild that you want to fetch from.
- **emoji_id** (*int*) – The ID of the emoji that you want to fetch.

Returns
The emoji from the API.

Return type
novus.Emoji

async classmethod fetch_all_for_guild(*state:* HTTPConnection, *guild:* *int* | *abc.Snowflake*) → list[*Emoji*]

Fetch all of the emojis from a guild.

See also:

`novus.Guild.fetch_emojis()`

Parameters
guild (*int | novus.abc.Snowflake*) – The guild that you want to fetch from.

Returns
The list of emojis that the guild has.

Return type
list[*novus.Emoji*]

async delete(*, *reason:* *str* | *None* = *None*) → None

Delete this emoji.

Parameters
reason (*str | None*) – The reason shown in the audit log.

async edit(*, *reason:* *str* | *None* = *None*, *name:* *str* = *MISSING*, *roles:* *list[int* | *abc.Snowflake]* = *MISSING*) → *Emoji*

Edit the current emoji.

Parameters

- **name** (*str*) – The new name for the emoji.
- **roles** (*list[int | novus.abc.Snowflake]*) – A list of the roles that can use the emoji.
- **reason** (*str | None*) – The reason shown in the audit log.

Returns
The newly updated emoji.

Return type
novus.Emoji

class novus.**ForumTag**(*name:* *str*, *emoji:* *str* | PartialEmoji, *moderated:* *bool* = *False*)

Tags that are added to forum posts.

id

The ID of the forum tag.

Type
int

name

The name of the forum tag.

Type
str

moderated

Whether or not the tag can only be added to or removed from threads by members with the MANAGE_THREADS permission.

Type
bool

emoji

The emoji associated with the forum tag.

Type
novus.PartialEmoji

class novus.**Guild**(*, *state:* HTTPConnection, *data: payloads.Guild*)

A model representing a guild given by Discord's API or gateway.

id

The ID of the guild.

Type
int

name

The name of the guild.

Type
str

icon_hash

The hash associated with the guild's icon.

Type
str | None

icon

The asset associated with the guild's icon hash.

Type
novus.Asset | None

splash_hash

The hash associated with the guild's splash.

Type
str | None

splash

The asset associated with the guild's splash hash.

Type
novus.Asset | None

discovery_splash_hash

The hash associated with the guild's discovery splash.

Type
str | None

discovery_splash

The asset associated with the guild's discovery splash hash.

Type
novus.Asset | None

owner_id

The ID of the user that owns the guild.

Type
int

afk_channel_id

The ID of the guild's AFK channel, if one is set.

Type
int | None

widget_enabled

Whether or not the widget for the guild is enabled.

Type
bool

widget_channel_id

If the widget is enabled, this will be the ID of the widget's channel.

Type
int | None

verification_level

The verification level required for the guild.

See also:

novus.VerificationLevel

Type
int

default_message_notifications

The default message notification level.

See also:

novus.NotificationLevel

Type
int

explicit_content_filter

The explicit content filter level.

See also:

novus.guild.ContentFilterLevel

Type
int

roles

The roles associated with the guild, as returned from the cache.

Type
list[*novus.Role*]

emojis

The emojis associated with the guild, as returned from the cache.

Type
list[*novus.Emoji*]

features

A list of guild features.

Type
list[str]

mfa_level

The required MFA level for the guild.

See also:

novus.MFALevel

Type
int

application_id

The application ID of the guild creator, if the guild is bot-created.

Type
int | None

system_channel_id

The ID of the channel where guild notices (such as welcome messages and boost events) are posted.

Type
int | None

system_channel_flags

The flags associated with the guild's system channel.

Type
novus.SystemChannelFlags

rules_channel_id

The ID of the guild's rules channel.

Type
int | None

max_presences

The maximum number of presences for the guild. For most guilds, this will be `None`.

Type
int | None

max_members

The maximum number of members allowed in the guild.

Type
int | None

vanity_url_code

The vanity code for the guild's invite link.

Type
str | None

description

The guild's description.

Type
str | None

banner_hash

The hash associated with the guild's banner splash.

Type
str | None

banner

The asset associated with the guild's banner splash hash.

Type
novus.Asset | None

premium_tier

The premium tier of the guild.

See also:

novus.PremiumTier

Type
int

premium_subscription_count

The number of boosts the guild currently has.

Type
int

preferred_locale

The locale for the guild, if set. Defaults to US English.

Type
str

public_updates_channel_id

The ID of the channel when admins and moderators of community guilds receive notices from Discord.

Type
int | None

max_video_channel_users

The maximum amount of users in a video channel.

Type
int | None

approximate_member_count

The approximate number of members in the guild. Present in guild GET requests when `with_counts` is `True`.

Type
int | None

approximate_presence_count

The approximate number of non-offline members in the guild. Present in guild GET requests when `with_counts` is `True`.

Type
int | None

welcome_screen

The welcome screen of a community guild.

Type
novus.WelcomeScreen | None

nsfw_level

The guild NSFW level.

See also:

novus.NSFWLevel

Type
int

stickers

The list of stickers added to the guild.

Type
list[*novus.Sticker*]

premium_progress_bar_enabled

Whether or not the progress bar is enabled.

Type
bool

get_sticker(*id: AnySnowflake*) → *Sticker* | None

Get a sticker from cache.

Parameters
id (*int | str | novus.abc.Snowflake*) – The identifier for the sticker we want to get.

Returns
A sticker object, if one was cached.

Return type
novus.Sticker | None

get_member(*id: AnySnowflake*) → *GuildMember* | None

Get a guild member from cache.

Parameters
id (*int | str | novus.abc.Snowflake*) – The identifier for the user we want to get.

Returns
A guild member object, if one was cached.

Return type
novus.GuildMember | None

get_role(*id: AnySnowflake*) → *Role* | None

Get a role from cache.

Parameters
id (*int | str | novus.abc.Snowflake*) – The identifier for the role we want to get.

Returns
A role object, if one was cached.

Return type
novus.GuildMember | None

get_event(*id: AnySnowflake*) → *ScheduledEvent* | None

Get a scheduled event from cache.

Parameters
id (*int | str | novus.abc.Snowflake*) – The identifier for the event we want to get.

Returns
A scheduled event object, if one was cached.

Return type
novus.ScheduledEvent | None

get_thread(*id: AnySnowflake*) → *Channel* | None

Get a thread from cache.

Parameters
id (*int | str | novus.abc.Snowflake*) – The identifier for the thread we want to get.

Returns
A thread object, if one was cached.

Return type
novus.Channel | None

get_channel(*id: AnySnowflake*) → *Channel* | None

Get a channel from cache.

Parameters
id (*int | str | novus.abc.Snowflake*) – The identifier for the channel we want to get.

Returns
A channel object, if one was cached.

Return type
novus.Channel | None

async add_member(*user_id: int, access_token: str, *, nick: str = MISSING, mute: bool = MISSING, deaf: bool = MISSING*) → *GuildMember* | None

Add a member to the guild.

Note: This requires an Oauth access token, and the provided user ID must be the same one that matches the account.

Parameters

- **user_id** (*int*) – The ID of the user that you want to add. The user ID must match the ID of the oauth token.
- **access_token** (*str*) – The access token with the `guilds.join` scope to the bot's application for the user you want to add to the guild.
- **nick** (*str*) – The nickname youy want to set the user to.
- **mute** (*bool*) – Whether the user is muted in voice channels.
- **deaf** (*bool*) – Whether the user is deafened in voice channels.

Returns
The member for the user that was added to the guild, or `None` if the user was already present.

Return type
novus.GuildMember | None

async add_member_role(*user: AnySnowflake, role: AnySnowflake, *, reason: str | None = None*) → None

Add a role to a user.

Requires the `MANAGE_ROLES` permission.

See also:

novus.GuildMember.add_role()

Parameters

- **user** (*int | novus.abc.Snowflake*) – The user you want to add the role to.
- **role** (*int | novus.abc.Snowflake*) – The role you want to add.
- **reason** (*str | None*) – The reason shown in the audit log.

async ban(*user: AnySnowflake, *, reason: str | None = None, delete_message_seconds: int = MISSING*) → None

Ban a user from the guild.

See also:

novus.GuildMember.ban()

Parameters

- **user** (*int | novus.abc.Snowflake*) – The user who you want to ban.
- **delete_message_seconds** (*int*) – The number of seconds of messages you want to delete.
- **reason** (*str | None*) – The reason to be shown in the audit log.

async chunk_members(*query: str = ''*, *limit: int = 0*, *user_ids: list[int] | None = None*, *wait: bool = True*) → list[*GuildMember*] | None

Request member chunks from the gateway.

This will *only* work if you are connected to the gateway - this will not work with HTTP-only bots.

Parameters

- **query** (*str*) – A search string for usernames.
- **limit** (*int*) – A limit for the retrieved member count.
- **user_ids** (*list[int]*) – A list of user IDs to request.
- **wait** (*bool*) – Whether or not to wait for a response.

Returns
A list of requested users or `None` if you chose not to wait.

Return type
list[*novus.GuildMember*] | None

async classmethod create(*state: HTTPConnection*, *, *name: str*) → *Guild*

Create a guild.

Parameters

- **state** (*novus.HTTPConnection*) – The API connection to create the entity with.
- **name** (*str*) – The name for the guild that you want to create.

Returns
The created guild.

Return type
novus.Guild

async create_auto_moderation_rule(*, *reason: str | None = None*, *name: str*, *event_type: int*, *trigger_type: int*, *actions: list[AutoModerationAction]*, *trigger_metadata: AutoModerationTriggerMetadata | None = None*, *enabled: bool = False*, *exempt_roles: list[AnySnowflake] | None = None*, *exempt_channels: list[AnySnowflake] | None = None*) → *AutoModerationRule*

Create a new auto moderation rule.

Parameters

- **name** (*str*) – The new name for the role.
- **event_type** (*int*) – The event type.

  See also:

  novus.AutoModerationEventType

- **trigger_type** (*int*) – The trigger type.

  See also:

  novus.AutoModerationTriggerType

- **actions** (*list[novus.AutoModerationAction]*) – The actions to be taken on trigger.
- **trigger_metadata** (*novus.AutoModerationTriggerMetadata | None*) – The trigger metadata.

- **enabled** (*bool*) – Whether the rule is enabled or not.
- **exempt_roles** (*list[int | novus.abc.Snowflake] | None*) – A list of roles that are exempt from the rule.
- **exempt_channels** (*list[int | novus.abc.Snowflake] | None*) – A list of channels that are exempt from the rule.
- **reason** (*str | None*) – The reason shown in the audit log.

Returns
The created rule.

Return type
novus.AutoModerationRule

async create_channel(*, *name: str*, *type: int = MISSING*, *topic: str = MISSING*, *bitrate: int = MISSING*, *user_limit: int = MISSING*, *rate_limit_per_user: int = MISSING*, *position: int = MISSING*, *permission_overwrites: list[PermissionOverwrite] = MISSING*, *parent: AnySnowflake = MISSING*, *nsfw: bool = MISSING*, *default_auto_archive_duration: int = MISSING*, *default_reaction_emoji: Reaction = MISSING*, *available_tags: list[ForumTag] = MISSING*, *reason: str = MISSING*) → *Channel*

Create a channel within the guild.

Parameters

- **name** (*str*) – The name of the channel.
- **type** (*int*) – The type of the channel.

  See also:

  novus.ChannelType
- **bitrate** (*int*) – The bitrate for the channel. Only for use with voice channels.
- **user_limit** (*int*) – The user limit for the channel. Only for use with voice channels.
- **rate_limit_per_user** (*int*) – The slowmode seconds on the channel.
- **position** (*int*) – The channel position.
- **permission_overwrites** (*list[novus.PermissionOverwrite]*) – A list of permission overwrites for the channel.
- **parent** (*int | str | novus.abc.Snowflake*) – A parent object for the channel.
- **nsfw** (*bool*) – Whether or not the channel will be set to NSFW.
- **default_auto_archive_duration** (*int*) – The default duration that clients use (in minutes) to automatically archive the thread after recent activity. Only for use with forum channels.
- **default_reaction_emoji** (*Reaction*) – The default add reaction button to be shown on threads. Only for use with forum channels.
- **available_tags** (*list[ForumTag]*) – The tags available for threads. Only for use with forum channels.
- **reason** (*str*) – The reason to be shown in the audit log.

Returns
The created channel.

Return type
novus.model.Channel

async create_emoji(*, *name: str*, *image: FileT*, *roles: list[AnySnowflake] | None = None*, *reason: str | None = None*) → *Emoji*

Create an emoji within a guild.

Parameters

- **name** (*str*) – The name of the emoji you want to add.
- **image** (*str | bytes | io.IOBase*) – The image that you want to add.
- **roles** (*list[int | novus.abc.Snowflake] | None*) – A list of roles that are allowed to use the emoji.
- **reason** (*str | None*) – A reason you're adding the emoji.

Returns
The newly created emoji.

Return type
novus.Emoji

async create_role(*, *reason: str | None = None*, *name: str = MISSING*, *permissions: Permissions = MISSING*, *color: int = MISSING*, *hoist: bool = MISSING*, *icon: FileT = MISSING*, *unicode_emoji: str = MISSING*, *mentionable: bool = MISSING*) → *Role*

Create a role within the guild.

Parameters

- **name** (*str*) – The name of the role.
- **permissions** (*novus.Permissions*) – The permissions attached to the role.
- **color** (*int*) – The color of the role.
- **hoist** (*bool*) – Whether the role is displayed seperately in the sidebar.
- **icon** (*str | bytes | io.IOBase | None*) – The role icon image. Only usable if the guild has the `ROLE_ICONS` feature.
- **unicode_emoji** (*str*) – The role's unicode emoji. Only usable if the guild has the `ROLE_ICONS` feature.
- **mentionable** (*bool*) – Whether the role should be mentionable.
- **reason** (*str | None*) – The reason to be shown in the audit log.

async create_scheduled_event(*, *name: str*, *start_time: DiscordDatetime*, *entity_type: int*, *privacy_level: int*, *reason: str | None = None*, *channel: AnySnowflake | None = MISSING*, *location: str = MISSING*, *end_time: DiscordDatetime = MISSING*, *description: str | None = MISSING*, *status: int = MISSING*, *image: FileT | None = MISSING*) → *ScheduledEvent*

Create a new scheduled event.

See also:

novus.ScheduledEvent.create()

Parameters

- **name** (*str*) – The name of the event.

- **start_time** (*datetime.datetime*) – The time to schedule the event start.
- **entity_type** (*int*) – The type of the event.
- **privacy_level** (*int*) – The privacy level of the event.

  See also:

  novus.EventPrivacyLevel
- **channel** (*int | Snowflake | None*) – The channel of the scheduled event. Set to `None` if the event type is being set to external.
- **location** (*str*) – The location of the event.
- **end_time** (*datetime.datetime*) – The time to schedule the event end.
- **description** (*str | None*) – The description of the event.
- **status** (*int*) – The status of the event.

  See also:

  novus.EventStatus
- **image** (*str | bytes | io.IOBase | None*) – The cover image of the scheduled event.
- **reason** (*str | None*) – The reason shown in the audit log.

Returns
The new scheduled event.

Return type
novus.ScheduledEvent

async create_sticker(*, *reason: str | None = None*, *name: str*, *description: str | None = None*, *tags: str*, *image:* File) → *Sticker*

Create a new sticker.

See also:

novus.Sticker.create()

Parameters

- **name** (*str*) – The name of the sticker.
- **tags** (*str*) – Autocomplete/suggestion tags for the sitcker.
- **description** (*str | None*) – Description of the sticker.
- **image** (novus.File) – The image to be uploaded. All aside from the data itself is discarded - the name and description are taken from the other parameters.
- **reason** (*str | None*) – The reason shown in the audit log.
- **Reutrns**
- **-------**
- **novus.Sticker** – The created sticker instance.

async delete() → None

Delete the current guild permanently. You must be the owner of the guild to run this successfully.

async delete_role(*role: AnySnowflake*, *, *reason: str | None = None*) → None

A role to delete.

Parameters

- **role** (*int | novus.abc.Snowflake*) – The ID of the role to delete.
- **reason** (*str | None*) – The reason to be shown in the audit log.

async edit(*, *name: str = MISSING*, *verification_level: int | None = MISSING*, *default_message_notifications: int | None = MISSING*, *explicit_content_filter: int | None = MISSING*, *afk_channel: AnySnowflake | None = MISSING*, *icon: FileT | None = MISSING*, *owner: AnySnowflake = MISSING*, *splash: FileT | None = MISSING*, *discovery_splash: FileT | None = MISSING*, *banner: FileT | None = MISSING*, *system_channel: AnySnowflake | None = MISSING*, *system_channel_flags: SystemChannelFlags | None = MISSING*, *rules_channel: AnySnowflake | None = MISSING*, *preferred_locale: str | None = MISSING*, *public_updates_channel: AnySnowflake = MISSING*, *features: list[str] = MISSING*, *description: str | None = MISSING*, *premium_progress_bar_enabled: bool = MISSING*, *reason: str | None = None*) → *Guild*

Edit the guild parameters.

Note: The updated guild is not immediately put into cache - the bot waits for the guild update notification to be sent over the gateway before updating (which will not happen if you don't have the correct gateway intents).

Parameters

- **name** (*str*) – The name you want to set the guild to.
- **verification_level** (*int | None*) – The verification level you want to set the guild to.

  See also:

  novus.VerificationLevel

- **default_message_notifications** (*int | None*) – The default message notification level you want to set the guild to.

  See also:

  novus.NotificationLevel

- **explicit_content_filter** (*int | None*) – The content filter level you want to set the guild to.

  See also:

  novus.guild.ContentFilterLevel

- **afk_channel** (*int | novus.abc.Snowflake | None*) – The channel you want to set as the guild's AFK channel.
- **icon** (*str | bytes | io.IOBase | None*) – The icon that you want to set for the guild. Can be its bytes, a file path, or a file object.
- **owner** (*int | novus.abc.Snowflake*) – The person you want to set as owner of the guild. Can only be run if the current user is the existing owner.
- **splash** (*str | bytes | io.IOBase | None*) – The splash that you want to set for the guild. Can be its bytes, a file path, or a file object.

- **discovery_splash** (*str | bytes | io.IOBase | None*) – The discovery splash for the guild. Can be its bytes, a file path, or a file object.
- **banner** (*str | bytes | io.IOBase | None*) – The banner for the guild. Can be its bytes, a file path, or a file object.
- **system_channel** (*int | novus.abc.Snowflake | None*) – The system channel you want to set for the guild.
- **system_channel_flags** (*novus.guild.SystemChannelFlags | None*) – The system channel flags you want to set.
- **rules_channel** (*int | novus.abc.Snowflake | None*) – The channel you want to set as the rules channel.
- **preferred_locale** (*str | None*) – The locale you want to set as the guild's preferred.
- **public_updates_channel** (*int | novus.abc.Snowflake*) – The channel you want to set as the updates channel for the guild.
- **features** (*list[str]*) – A list of features for the guild.
- **description** (*str | None*) – A description for the guild.
- **premium_progress_bar_enabled** (*bool*) – Whether or not to enable the premium progress bar for the guild.
- **reason** (*str | None*) – A reason for modifying the guild (shown in the audit log).

Returns
The updated guild.

Return type
novus.Guild

async edit_member(*user: AnySnowflake, *, reason: str | None = None, nick: str | None = MISSING, roles: list[AnySnowflake] = MISSING, mute: bool = MISSING, deaf: bool = MISSING, voice_channel: AnySnowflake | None = MISSING, timeout_until: DiscordDatetime | None = MISSING*) → *GuildMember*

Edit a guild member.

See also:

novus.GuildMember.edit()

Parameters

- **user** (*int | novus.abc.Snowflake*) – The ID of the user you want to edit.
- **nick** (*str | None*) – The nickname you want to set for the user.
- **roles** (*list[int | novus.abc.Snowflake]*) – A list of roles that you want the user to have.
- **mute** (*bool*) – Whether or not the user is muted in voice channels. Will error if the user is not currently in a voice channel.
- **deaf** (*bool*) – Whether or not the user is deafened in voice channels. Will error if the user is not currently in a voice channel.
- **voice_channel** (*int | novus.abc.Snowflake | None*) – The voice channel that the user is in.

- **timeout_until** (*datetime.datetime | None*) – When the user's timeout should expire (up to 28 days in the future).

async edit_role(*role_id: int*, *, *reason: str | None = None*, *name: str = MISSING*, *permissions: Permissions = MISSING*, *color: int = MISSING*, *hoist: bool = MISSING*, *icon: FileT = MISSING*, *unicode_emoji: str = MISSING*, *mentionable: bool = MISSING*) → *Role*

Edit a role.

Parameters

- **role_id** (*int*) – The ID of the role to be edited.
- **name** (*str*) – The new name of the role.
- **permissions** (*novus.Permissions*) – The permissions to be applied to the role.
- **color** (*int*) – The color to apply to the role.
- **hoist** (*bool*) – If the role should be displayed seperately in the sidebar.
- **icon** (*str | bytes | io.IOBase | None*) – The role's icon image. Only usable if the guild has the `ROLE_ICONS` feature.
- **unicode_emoji** (*str*) – The role's unicode emoji. Only usable if the guild has the `ROLE_ICONS` feature.
- **mentionable** (*bool*) – If the role is mentionable.
- **reason** (*str | None*) – The reason to be shown in the audit log.

async classmethod fetch(*state: HTTPConnection*, *guild: AnySnowflake*) → *Guild*

Get an instance of a guild from the API. Unlike the gateway's `GUILD_CREATE` payload, this method does not return members, channels, or voice states.

Parameters

- **state** (*HTTPConnection*) – The API connection.
- **guild** (*int | novus.abc.Snowflake*) – A reference to the guild that you want to fetch.

Returns
The guild associated with the given ID.

Return type
novus.Guild

async fetch_active_threads() → list[*Channel*]

Get the active threads from inside the guild.

Returns
A list of threads.

Return type
list[*novus.Channel*]

async fetch_all_emojis() → list[*Emoji*]

List all of the emojis for the guild.

See also:

novus.Emoji.fetch_all_for_guild()

Returns
A list of the guild's emojis.

Return type
list[*novus.Emoji*]

async fetch_all_stickers() → list[*Sticker*]

List all stickers associated with the guild.

See also:

novus.Sticker.fetch_all_for_guild()

Returns
The stickers associated with the guild.

Return type
list[*novus.Sticker*]

async fetch_audit_logs(*, *user_id: int | None = None*, *action_type: int | None = None*, *before: int | None = None*, *after: int | None = None*, *limit: int = 50*) → *AuditLog*

Get the audit logs for the guild.

Parameters

- **user_id** (*int | None*) – The ID of the moderator you want to to filter by.
- **action_type** (*int | None*) – The ID of an action to filter by.

  See also:

  novus.AuditLogEventType
- **before** (*int | None*) – The snowflake before which to get entries.
- **after** (*int | None*) – The snowflake after which to get entries.
- **limit** (*int*) – The number of entries to get. Max 100, defaults to 50.

Returns
The audit log for the guild.

Return type
novus.AuditLog

async fetch_auto_moderation_rules() → list[*AutoModerationRule*]

Get the auto moderation rules for this guild.

Returns
A list of the auto moderation rules for the guild.

Return type
list[*novus.AutoModerationRule*]

async fetch_ban(*user: AnySnowflake*) → *GuildBan*

Get an individual user's ban.

Parameters
user (*int | novus.abc.Snowflake*) – The user whose ban you want to get.

Returns
The ban for the user.

Return type
novus.GuildBan

async fetch_bans(*, *limit: int = 1000*, *before: int | None = None*, *after: int | None = None*) → list[*GuildBan*]

Get a list of bans from the guild.

Parameters

- **limit** (*str*) – The number of bans to get.
- **before** (*int | None*) – The snowflake to search around.
- **after** (*int | None*) – The snowflake to search around.

Returns
A list of bans from the guild.

Return type
list[novus.model.GuildBan]

async fetch_channels() → list[*Channel*]

Fetch all of the channels from a guild.

Returns
A list of channels from the guild.

Return type
list[*novus.Channel*]

async fetch_emoji(*id: int*) → *Emoji*

List all of the emojis for the guild.

See also:

novus.Emoji.fetch()

Returns
A list of the guild's emojis.

Return type
list[*novus.Emoji*]

async fetch_invites() → list[*Invite*]

Get the invites for the guild.

Requires the `MANAGE_GUILD` permission.

Returns
A list of invites.

Return type
list[*novus.Invite*]

async fetch_me() → *GuildMember*

Get the member object associated with the current guild and the current connection.

Note: Only usable via Oauth with the `guilds.members.read` scope. This is not usable as a bot.

See also:

novus.GuildMember.fetch_me()

Returns
The member object for the current user.

Return type
novus.GuildMember

async fetch_member(*member_id: int*) → *GuildMember*

Get a member from the guild.

See also:

novus.GuildMember.fetch()

Parameters
member_id (*int*) – The ID of the member you want to get.

Returns
The member object for the given user.

Return type
novus.GuildMember

async fetch_members(*, *limit: int = 1000*, *after: int = 0*) → list[*GuildMember*]

Get a list of members for the guild.

Note: This endpoint is restricted according to whether the `GUILD_MEMBERS` privileged intent is enabled for your application.

Note: This endpoint can return a maximum of 1000 members per request.

Parameters

- **limit** (*int*) – The number of guild members you want in the response payload.
- **after** (*int*) – The snowflake to get guild members after.

Returns
A list of members from the guild.

Return type
list[*novus.GuildMember*]

async fetch_roles() → list[*Role*]

Get a list of roles for the guild.

Returns
A list of roles in the guild.

Return type
list[novus.model.Role]

async fetch_scheduled_events(*, *with_user_count: bool = False*) → list[*ScheduledEvent*]

Get a list of all of the scheduled events for a guild.

See also:

novus.ScheduledEvent.fetch_all_for_guild()

Parameters
with_user_count (*bool*) – Whether or not to include the event's user count.

Returns
The scheduled events for the guild.

Return type
list[*novus.ScheduledEvent*]

async fetch_sticker(*id: AnySnowflake*) → *Sticker*

Get an individual sticker associated with the guild via its ID.

See also:

novus.Sticker.fetch()

Parameters
id (*str*) – The ID of the sticker.

Returns
The associated sticker instance.

Return type
novus.Sticker

async kick(*user: AnySnowflake*, *, *reason: str | None = None*) → None

Remove a user from the guild.

Requires the `KICK_MEMBERS` permission.

Parameters

- **user** (*int | novus.abc.Snowflake*) – The user you want to remove.
- **reason** (*str | None*) – The reason to be shown in the audit log.

async leave() → None

Leave the current guild.

async remove_member_role(*user: AnySnowflake*, *role: AnySnowflake*, *, *reason: str | None = None*) → None

Remove a role from a member.

Requires the `MANAGE_ROLES` permission.

See also:

novus.GuildMember.remove_role()

Parameters

- **user** (*int | novus.abc.Snowflake*) – The user you want to add the role to.
- **role** (*int | novus.abc.Snowflake*) – The ID of the role you want to add.
- **reason** (*str | None*) – The reason shown in the audit log.

async search_members(*, *query: str*, *limit: int = 1*) → list[*GuildMember*]

Get a list of members for the guild whose username of nickname starts with the provided string.

Note: This endpoint can return a maximum of 1000 members per request.

Parameters

- **query** (*str*) – the query string to match usernames and nicknames agains.
- **limit** (*int*) – The number of guild members you want in the response payload.

Returns
A list of members from the guild.

Return type
list[*novus.GuildMember*]

async unban(*user: AnySnowflake*, *, *reason: str | None = None*) → None

Remove a user's ban

Parameters

- **user** (*int | novus.abc.Snowflake*) – The user who you want to ban.
- **reason** (*str | None*) – The reason to be shown in the audit log.

class novus.**GuildBan**(*reason: str | None*, *user: User*)

A ban object for a guild.

Warning: This object should not be created yourself, but is used to represent a model from the API.

reason

The given reason that the user was banned.

Type
str | None

user

The user that was banned.

Type
novus.User

class novus.**GuildMember**(***kwargs: Any*)

A model for a guild member object.

This model does not extend the *novus.User* object, but but has the same methods and attributes.

Creating a GuildMember instance WILL update the cached user instance, if one exists. Otherwise, a new instance will be created.

id

The ID of the user.

Type
int

username

The username of the user.

Type
str

global_name

The global name of the user.

Type
str | None

discriminator

The discriminator of the user.

Type
str

avatar_hash

The avatar hash of the user.

Type
str | None

avatar

The avatar of the user.

Type
novus.Asset | None

bot

Whether or not the user is associated with an Oauth2 application.

Type
bool

system

Whether or not the user is associated with a Discord system message.

Type
bool

mfa_enabled

Whether or not there's MFA available on the account. Only set properly for when you're receiving your own user via an Oauth2 application.

Type
bool

banner_hash

The hash for the user banner.

Type
str | None

banner

The asset for the user banner.

Type
novus.Asset | None

accent_color

The color associated with the user's accent color.

Type
int

locale

The locale for the user. Only set properly for when you're receiving your own user via an Oauth2 application.

Type
str | None

verified

Whether or not the user has a verified username attached. Only set properly for when you're receiving your own user via an Oauth2 application.

Type
bool

email

The email associated with the account. Only set properly for when you're receiving your own user via an Oauth2 application.

Type
str | None

flags

The flags associated with the user account. A combination of public and private.

Type
novus.UserFlags

premium_type

The premium type associated with the account.

See also:

novus.UserPremiumType

Type
int

nick

The nickname for the user.

Type
str | None

guild_avatar_hash

The guild avatar hash for the user.

Type
str | None

guild_avatar

The guild avatar for the user.

Type
novus.Asset | None

role_ids

A list of role IDs that the user has.

Type
list[int]

joined_at

When the user joined the guild.

Type
datetime.datetime

premium_since

When the user started boosting the guild.

Type
datetime.datetime | None

deaf

If the user is deafened in voice channels.

Type
bool

mute

If the user is muted in voice channels.

Type
bool

pending

If the user has not yet passed membership screening.

Type
bool

permissions

The total permissions for the user in the channel, including overwrites. Only returned within an interaction.

Type
novus.Permissions | None

timeout_until

When the user's timeout will expire and the user will be able to communicate again.

Type
datetime.datetime | None

guild

The guild that the member is part of. May be `None` in some rare cases (such as when getting raw API requests).

Type
novus.abc.StateSnowflake | *novus.Guild* | None

voice

The user's voice state.

Type
novus.VoiceState | None

property mention: str

A ping for the user.

property permissions: Permissions

The calculated permissions for the user based on their roles and the cached guild. If permissions were provided (ie this member was created as part of an interaction payload) then they will not be re-calculated.

Note: Permissions are only properly calculated when the guild and its roles are cached (ie when the bot is connected to the gateway).

permissions_in(*channel:* Channel) → Permissions

Get the permissions for this guild member inside of a channel.

Note: Permissions are only properly calculated when the guild and its roles are cached (ie when the bot is connected to the gateway).

Parameters

channel (novus.Channel) – The channel that you want to get the user's permissions for.

Returns

The calculated permissions for this user in that channel.

Return type

novus.Permissions

async classmethod fetch(*state:* HTTPConnection, *guild_id:* *int*, *member_id:* *int*) → *GuildMember*

Get an instance of a user from the API.

See also:

novus.Guild.fetch_member()

Parameters

- **state** (HTTPConnection) – The API connection.
- **guild_id** (*int*) – The ID associated with the guild you want to get.
- **member_id** (*int*) – The ID associated with the user you want to get.

Returns

The user associated with the given ID.

Return type

novus.GuildMember

async classmethod fetch_me(*state:* HTTPConnection, *guild_id:* *int*) → *GuildMember*

Get the member object associated with the current connection and a given guild ID.

Note: Only usable via Oauth with the `guilds.members.read` scope.

See also:

novus.Guild.fetch_me()

Parameters

- **state** (HTTPConnection) – The API connection.
- **guild_id** (*int*) – The ID associated with the guild you want to get.

Returns
The member within the given guild.

Return type
novus.GuildMember

async edit(*, *reason: str | None = None*, *nick: str | None = MISSING*, *roles: list[int | abc.Snowflake] = MISSING*, *mute: bool = MISSING*, *deaf: bool = MISSING*, *voice_channel: int | abc.Snowflake | None = MISSING*, *timeout_until: dt | None = MISSING*) → *GuildMember*

Edit a guild member.

See also:

novus.Guild.edit_member()

Parameters

- **nick** (*str | None*) – The nickname you want to set for the user.
- **roles** (*list[novus.abc.Snowflake]*) – A list of roles that you want the user to have.
- **mute** (*bool*) – Whether or not the user is muted in voice channels. Will error if the user is not currently in a voice channel.
- **deaf** (*bool*) – Whether or not the user is deafened in voice channels. Will error if the user is not currently in a voice channel.
- **voice_channel** (*novus.abc.Snowflake | None*) – The voice channel that the user is in.
- **timeout_until** (*datetime.datetime | None*) – When the user's timeout should expire (up to 28 days in the future).

async add_role(*role: int | abc.Snowflake*, *, *reason: str | None = None*) → None

Add a role to the user.

Requires the MANAGE_ROLES permission.

See also:

novus.Guild.add_member_role()

Parameters

- **role** (*int | novus.abc.Snowflake*) – The role you want to add.
- **reason** (*str | None*) – The reason shown in the audit log.

async send(*content: str = MISSING*, *, *tts: bool = MISSING*, *embeds: list[Embed] = MISSING*, *allowed_mentions: AllowedMentions = MISSING*, *components: list[ActionRow] = MISSING*, *message_reference: Message = MISSING*, *stickers: list[Sticker] = MISSING*, *files: list[File] = MISSING*, *flags: flags.MessageFlags = MISSING*) → *Message*

Send a message to the channel associated with the model.

Parameters

- **content** (*str*) – The content that you want to have in the message

- **tts** (*bool*) – If you want the message to be sent with the TTS flag.
- **embeds** (*list[novus.Embed]*) – The embeds you want added to the message.
- **allowed_mentions** (novus.AllowedMentions) – The mentions you want parsed in the message.
- **components** (*list[novus.ActionRow]*) – A list of action rows to be added to the message.
- **message_reference** (novus.Message) – A reference to a message you want replied to.
- **stickers** (*list[novus.Sticker]*) – A list of stickers to add to the message.
- **files** (*list[novus.File]*) – A list of files to be sent with the message.
- **flags** (novus.MessageFlags) – The flags to be sent with the message.

async remove_role(*role: int | abc.Snowflake*, *, *reason: str | None = None*) → None

Remove a role from the user.

Requires the MANAGE_ROLES permission.

See also:

novus.Guild.remove_member_role()

Parameters

- **role** (*int | novus.abc.Snowflake*) – The role you want to remove.
- **reason** (*str | None*) – The reason shown in the audit log.

async kick(*, *reason: str | None = None*) → None

Remove a user from the guild.

Requires the KICK_MEMBERS permission.

See also:

novus.Guild.kick()

Parameters

reason (*str | None*) – The reason to be shown in the audit log.

async ban(*, *reason: str | None = None*, *delete_message_seconds: int = MISSING*) → None

Ban a user from the guild.

Requires the BAN_MEMBERS permission.

See also:

novus.Guild.ban()

Parameters

- **delete_message_seconds** (*int*) – The number of seconds of messages you want to delete.
- **reason** (*str | None*) – The reason to be shown in the audit log.

class novus.**GuildPreview**(*, *state: HTTPConnection*, *data: payloads.GuildPreview*)

A model for the preview of a guild.

id

The ID of the guild.

Type
int

name

The name of the guild.

Type
str

icon_hash

The icon hash for the guild.

Type
str | None

icon

The icon asset associated with the guild.

Type
novus.Asset | None

splash_hash

The splash hash for the guild.

Type
str | None

splash

The splash asset associated with the guild.

Type
novus.Asset | None

discovery_splash_hash

The discovery splash hash for the guild.

Type
str | None

discovery_splash

The discovery splash asset associated with the guild.

Type
novus.Asset | None

emojis

A list of emojis in the guild.

Type
list[*novus.Emoji*]

features

A list of features that the guild has.

Type
list[str]

approximate_member_count

The approximate member count for the guild.

Type
int

approximate_presence_count

The approximate online member count for the guild.

Type
int

description

The description of the guild.

Type
str

stickers

A list of the stickers in the guild.

Type
list[*novus.Sticker*]

async add_member(*user_id: int*, *access_token: str*, **, nick: str = MISSING*, *mute: bool = MISSING*, *deaf: bool = MISSING*) → *GuildMember* | None

Add a member to the guild.

Note: This requires an Oauth access token, and the provided user ID must be the same one that matches the account.

Parameters

- **user_id** (*int*) – The ID of the user that you want to add. The user ID must match the ID of the oauth token.
- **access_token** (*str*) – The access token with the `guilds.join` scope to the bot's application for the user you want to add to the guild.
- **nick** (*str*) – The nickname youy want to set the user to.
- **mute** (*bool*) – Whether the user is muted in voice channels.
- **deaf** (*bool*) – Whether the user is deafened in voice channels.

Returns
The member for the user that was added to the guild, or `None` if the user was already present.

Return type
novus.GuildMember | None

async add_member_role(*user: AnySnowflake*, *role: AnySnowflake*, **, reason: str | None = None*) → None

Add a role to a user.

Requires the `MANAGE_ROLES` permission.

See also:

novus.GuildMember.add_role()

Parameters

- **user** (*int | novus.abc.Snowflake*) – The user you want to add the role to.
- **role** (*int | novus.abc.Snowflake*) – The role you want to add.
- **reason** (*str | None*) – The reason shown in the audit log.

async ban(*user: AnySnowflake, *, reason: str | None = None, delete_message_seconds: int = MISSING*) → None

Ban a user from the guild.

See also:

novus.GuildMember.ban()

Parameters

- **user** (*int | novus.abc.Snowflake*) – The user who you want to ban.
- **delete_message_seconds** (*int*) – The number of seconds of messages you want to delete.
- **reason** (*str | None*) – The reason to be shown in the audit log.

async chunk_members(*query: str = '', limit: int = 0, user_ids: list[int] | None = None, wait: bool = True*) → list[*GuildMember*] | None

Request member chunks from the gateway.

This will *only* work if you are connected to the gateway - this will not work with HTTP-only bots.

Parameters

- **query** (*str*) – A search string for usernames.
- **limit** (*int*) – A limit for the retrieved member count.
- **user_ids** (*list[int]*) – A list of user IDs to request.
- **wait** (*bool*) – Whether or not to wait for a response.

Returns
A list of requested users or `None` if you chose not to wait.

Return type
list[*novus.GuildMember*] | None

async classmethod create(*state: HTTPConnection, *, name: str*) → *Guild*

Create a guild.

Parameters

- **state** (*novus.HTTPConnection*) – The API connection to create the entity with.
- **name** (*str*) – The name for the guild that you want to create.

Returns
The created guild.

Return type
novus.Guild

async create_auto_moderation_rule(*, *reason: str | None = None*, *name: str*, *event_type: int*, *trigger_type: int*, *actions: list[AutoModerationAction]*, *trigger_metadata: AutoModerationTriggerMetadata | None = None*, *enabled: bool = False*, *exempt_roles: list[AnySnowflake] | None = None*, *exempt_channels: list[AnySnowflake] | None = None*) → *AutoModerationRule*

Create a new auto moderation rule.

Parameters

- **name** (*str*) – The new name for the role.
- **event_type** (*int*) – The event type.

  See also:

  novus.AutoModerationEventType
- **trigger_type** (*int*) – The trigger type.

  See also:

  novus.AutoModerationTriggerType
- **actions** (*list[novus.AutoModerationAction]*) – The actions to be taken on trigger.
- **trigger_metadata** (*novus.AutoModerationTriggerMetadata | None*) – The trigger metadata.
- **enabled** (*bool*) – Whether the rule is enabled or not.
- **exempt_roles** (*list[int | novus.abc.Snowflake] | None*) – A list of roles that are exempt from the rule.
- **exempt_channels** (*list[int | novus.abc.Snowflake] | None*) – A list of channels that are exempt from the rule.
- **reason** (*str | None*) – The reason shown in the audit log.

Returns
The created rule.

Return type
novus.AutoModerationRule

async create_channel(*, *name: str*, *type: int = MISSING*, *topic: str = MISSING*, *bitrate: int = MISSING*, *user_limit: int = MISSING*, *rate_limit_per_user: int = MISSING*, *position: int = MISSING*, *permission_overwrites: list[PermissionOverwrite] = MISSING*, *parent: AnySnowflake = MISSING*, *nsfw: bool = MISSING*, *default_auto_archive_duration: int = MISSING*, *default_reaction_emoji: Reaction = MISSING*, *available_tags: list[ForumTag] = MISSING*, *reason: str = MISSING*) → *Channel*

Create a channel within the guild.

Parameters

- **name** (*str*) – The name of the channel.
- **type** (*int*) – The type of the channel.

  See also:

  novus.ChannelType
- **bitrate** (*int*) – The bitrate for the channel. Only for use with voice channels.

- **user_limit** (*int*) – The user limit for the channel. Only for use with voice channels.
- **rate_limit_per_user** (*int*) – The slowmode seconds on the channel.
- **position** (*int*) – The channel position.
- **permission_overwrites** (*list[novus.PermissionOverwrite]*) – A list of permission overwrites for the channel.
- **parent** (*int | str | novus.abc.Snowflake*) – A parent object for the channel.
- **nsfw** (*bool*) – Whether or not the channel will be set to NSFW.
- **default_auto_archive_duration** (*int*) – The default duration that clients use (in minutes) to automatically archive the thread after recent activity. Only for use with forum channels.
- **default_reaction_emoji** (*Reaction*) – The default add reaction button to be shown on threads. Only for use with forum channels.
- **available_tags** (*list[ForumTag]*) – The tags available for threads. Only for use with forum channels.
- **reason** (*str*) – The reason to be shown in the audit log.

Returns
The created channel.

Return type
novus.model.Channel

async create_emoji(*, *name: str*, *image: FileT*, *roles: list[AnySnowflake] | None = None*, *reason: str | None = None*) → *Emoji*

Create an emoji within a guild.

Parameters

- **name** (*str*) – The name of the emoji you want to add.
- **image** (*str | bytes | io.IOBase*) – The image that you want to add.
- **roles** (*list[int | novus.abc.Snowflake] | None*) – A list of roles that are allowed to use the emoji.
- **reason** (*str | None*) – A reason you're adding the emoji.

Returns
The newly created emoji.

Return type
novus.Emoji

async create_role(*, *reason: str | None = None*, *name: str = MISSING*, *permissions: Permissions = MISSING*, *color: int = MISSING*, *hoist: bool = MISSING*, *icon: FileT = MISSING*, *unicode_emoji: str = MISSING*, *mentionable: bool = MISSING*) → *Role*

Create a role within the guild.

Parameters

- **name** (*str*) – The name of the role.
- **permissions** (*novus.Permissions*) – The permissions attached to the role.
- **color** (*int*) – The color of the role.
- **hoist** (*bool*) – Whether the role is displayed seperately in the sidebar.

- **icon** (*str | bytes | io.IOBase | None*) – The role icon image. Only usable if the guild has the ROLE_ICONS feature.
- **unicode_emoji** (*str*) – The role's unicode emoji. Only usable if the guild has the ROLE_ICONS feature.
- **mentionable** (*bool*) – Whether the role should be mentionable.
- **reason** (*str | None*) – The reason to be shown in the audit log.

async create_scheduled_event(*, *name: str*, *start_time: DiscordDatetime*, *entity_type: int*, *privacy_level: int*, *reason: str | None = None*, *channel: AnySnowflake | None = MISSING*, *location: str = MISSING*, *end_time: DiscordDatetime = MISSING*, *description: str | None = MISSING*, *status: int = MISSING*, *image: FileT | None = MISSING*) → *ScheduledEvent*

Create a new scheduled event.

See also:

novus.ScheduledEvent.create()

Parameters

- **name** (*str*) – The name of the event.
- **start_time** (*datetime.datetime*) – The time to schedule the event start.
- **entity_type** (*int*) – The type of the event.
- **privacy_level** (*int*) – The privacy level of the event.

  See also:

  novus.EventPrivacyLevel
- **channel** (*int | Snowflake | None*) – The channel of the scheduled event. Set to None if the event type is being set to external.
- **location** (*str*) – The location of the event.
- **end_time** (*datetime.datetime*) – The time to schedule the event end.
- **description** (*str | None*) – The description of the event.
- **status** (*int*) – The status of the event.

  See also:

  novus.EventStatus
- **image** (*str | bytes | io.IOBase | None*) – The cover image of the scheduled event.
- **reason** (*str | None*) – The reason shown in the audit log.

Returns
The new scheduled event.

Return type
novus.ScheduledEvent

async create_sticker(*, *reason: str | None = None*, *name: str*, *description: str | None = None*, *tags: str*, *image:* File) → *Sticker*

Create a new sticker.

See also:

novus.Sticker.create()

Parameters

- **name** (*str*) – The name of the sticker.
- **tags** (*str*) – Autocomplete/suggestion tags for the sitcker.
- **description** (*str | None*) – Description of the sticker.
- **image** (novus.File) – The image to be uploaded. All aside from the data itself is discarded - the name and description are taken from the other parameters.
- **reason** (*str | None*) – The reason shown in the audit log.
- **Reutrns**
- **-------**
- **novus.Sticker** – The created sticker instance.

async delete() → None

Delete the current guild permanently. You must be the owner of the guild to run this successfully.

async delete_role(*role: AnySnowflake*, *, *reason: str | None = None*) → None

A role to delete.

Parameters

- **role** (*int | novus.abc.Snowflake*) – The ID of the role to delete.
- **reason** (*str | None*) – The reason to be shown in the audit log.

async edit(*, *name: str = MISSING*, *verification_level: int | None = MISSING*, *default_message_notifications: int | None = MISSING*, *explicit_content_filter: int | None = MISSING*, *afk_channel: AnySnowflake | None = MISSING*, *icon: FileT | None = MISSING*, *owner: AnySnowflake = MISSING*, *splash: FileT | None = MISSING*, *discovery_splash: FileT | None = MISSING*, *banner: FileT | None = MISSING*, *system_channel: AnySnowflake | None = MISSING*, *system_channel_flags:* SystemChannelFlags *| None = MISSING*, *rules_channel: AnySnowflake | None = MISSING*, *preferred_locale: str | None = MISSING*, *public_updates_channel: AnySnowflake = MISSING*, *features: list[str] = MISSING*, *description: str | None = MISSING*, *premium_progress_bar_enabled: bool = MISSING*, *reason: str | None = None*) → *Guild*

Edit the guild parameters.

Note: The updated guild is not immediately put into cache - the bot waits for the guild update notification to be sent over the gateway before updating (which will not happen if you don't have the correct gateway intents).

Parameters

- **name** (*str*) – The name you want to set the guild to.
- **verification_level** (*int | None*) – The verification level you want to set the guild to.

  See also:

  novus.VerificationLevel

- **default_message_notifications** (*int | None*) – The default message notification level you want to set the guild to.

  See also:

  novus.NotificationLevel

- **explicit_content_filter** (*int | None*) – The content filter level you want to set the guild to.

  See also:

  novus.guild.ContentFilterLevel

- **afk_channel** (*int | novus.abc.Snowflake | None*) – The channel you want to set as the guild's AFK channel.
- **icon** (*str | bytes | io.IOBase | None*) – The icon that you want to set for the guild. Can be its bytes, a file path, or a file object.
- **owner** (*int | novus.abc.Snowflake*) – The person you want to set as owner of the guild. Can only be run if the current user is the existing owner.
- **splash** (*str | bytes | io.IOBase | None*) – The splash that you want to set for the guild. Can be its bytes, a file path, or a file object.
- **discovery_splash** (*str | bytes | io.IOBase | None*) – The discovery splash for the guild. Can be its bytes, a file path, or a file object.
- **banner** (*str | bytes | io.IOBase | None*) – The banner for the guild. Can be its bytes, a file path, or a file object.
- **system_channel** (*int | novus.abc.Snowflake | None*) – The system channel you want to set for the guild.
- **system_channel_flags** (*novus.guild.SystemChannelFlags | None*) – The system channel flags you want to set.
- **rules_channel** (*int | novus.abc.Snowflake | None*) – The channel you want to set as the rules channel.
- **preferred_locale** (*str | None*) – The locale you want to set as the guild's preferred.
- **public_updates_channel** (*int | novus.abc.Snowflake*) – The channel you want to set as the updates channel for the guild.
- **features** (*list[str]*) – A list of features for the guild.
- **description** (*str | None*) – A description for the guild.
- **premium_progress_bar_enabled** (*bool*) – Whether or not to enable the premium progress bar for the guild.
- **reason** (*str | None*) – A reason for modifying the guild (shown in the audit log).

Returns
The updated guild.

Return type
novus.Guild

async edit_member(*user: AnySnowflake, *, reason: str | None = None, nick: str | None = MISSING, roles: list[AnySnowflake] = MISSING, mute: bool = MISSING, deaf: bool = MISSING, voice_channel: AnySnowflake | None = MISSING, timeout_until: DiscordDatetime | None = MISSING*) → *GuildMember*

Edit a guild member.

See also:

novus.GuildMember.edit()

Parameters

- **user** (*int | novus.abc.Snowflake*) – The ID of the user you want to edit.
- **nick** (*str | None*) – The nickname you want to set for the user.
- **roles** (*list[int | novus.abc.Snowflake]*) – A list of roles that you want the user to have.
- **mute** (*bool*) – Whether or not the user is muted in voice channels. Will error if the user is not currently in a voice channel.
- **deaf** (*bool*) – Whether or not the user is deafened in voice channels. Will error if the user is not currently in a voice channel.
- **voice_channel** (*int | novus.abc.Snowflake | None*) – The voice channel that the user is in.
- **timeout_until** (*datetime.datetime | None*) – When the user's timeout should expire (up to 28 days in the future).

async edit_role(*role_id: int, *, reason: str | None = None, name: str = MISSING, permissions: Permissions = MISSING, color: int = MISSING, hoist: bool = MISSING, icon: FileT = MISSING, unicode_emoji: str = MISSING, mentionable: bool = MISSING*) → *Role*

Edit a role.

Parameters

- **role_id** (*int*) – The ID of the role to be edited.
- **name** (*str*) – The new name of the role.
- **permissions** (*novus.Permissions*) – The permissions to be applied to the role.
- **color** (*int*) – The color to apply to the role.
- **hoist** (*bool*) – If the role should be displayed seperately in the sidebar.
- **icon** (*str | bytes | io.IOBase | None*) – The role's icon image. Only usable if the guild has the `ROLE_ICONS` feature.
- **unicode_emoji** (*str*) – The role's unicode emoji. Only usable if the guild has the `ROLE_ICONS` feature.
- **mentionable** (*bool*) – If the role is mentionable.
- **reason** (*str | None*) – The reason to be shown in the audit log.

async classmethod fetch(*state: HTTPConnection, guild: AnySnowflake*) → *Guild*

Get an instance of a guild from the API. Unlike the gateway's `GUILD_CREATE` payload, this method does not return members, channels, or voice states.

Parameters

- **state** (*HTTPConnection*) – The API connection.
- **guild** (*int | novus.abc.Snowflake*) – A reference to the guild that you want to fetch.

Returns
The guild associated with the given ID.

Return type
novus.Guild

async fetch_active_threads() → list[*Channel*]

Get the active threads from inside the guild.

Returns
A list of threads.

Return type
list[*novus.Channel*]

async fetch_all_emojis() → list[*Emoji*]

List all of the emojis for the guild.

See also:

novus.Emoji.fetch_all_for_guild()

Returns
A list of the guild's emojis.

Return type
list[*novus.Emoji*]

async fetch_all_stickers() → list[*Sticker*]

List all stickers associated with the guild.

See also:

novus.Sticker.fetch_all_for_guild()

Returns
The stickers associated with the guild.

Return type
list[*novus.Sticker*]

async fetch_audit_logs(*, *user_id: int | None = None*, *action_type: int | None = None*, *before: int | None = None*, *after: int | None = None*, *limit: int = 50*) → *AuditLog*

Get the audit logs for the guild.

Parameters

- **user_id** (*int | None*) – The ID of the moderator you want to to filter by.
- **action_type** (*int | None*) – The ID of an action to filter by.

  See also:

  novus.AuditLogEventType
- **before** (*int | None*) – The snowflake before which to get entries.
- **after** (*int | None*) – The snowflake after which to get entries.
- **limit** (*int*) – The number of entries to get. Max 100, defaults to 50.

Returns
The audit log for the guild.

Return type
novus.AuditLog

async fetch_auto_moderation_rules() → list[*AutoModerationRule*]

Get the auto moderation rules for this guild.

Returns
A list of the auto moderation rules for the guild.

Return type
list[*novus.AutoModerationRule*]

async fetch_ban(*user: AnySnowflake*) → *GuildBan*

Get an individual user's ban.

Parameters
user (*int | novus.abc.Snowflake*) – The user whose ban you want to get.

Returns
The ban for the user.

Return type
novus.GuildBan

async fetch_bans(*, *limit: int = 1000*, *before: int | None = None*, *after: int | None = None*) → list[*GuildBan*]

Get a list of bans from the guild.

Parameters

- **limit** (*str*) – The number of bans to get.
- **before** (*int | None*) – The snowflake to search around.
- **after** (*int | None*) – The snowflake to search around.

Returns
A list of bans from the guild.

Return type
list[novus.model.GuildBan]

async fetch_channels() → list[*Channel*]

Fetch all of the channels from a guild.

Returns
A list of channels from the guild.

Return type
list[*novus.Channel*]

async fetch_emoji(*id: int*) → *Emoji*

List all of the emojis for the guild.

See also:

novus.Emoji.fetch()

Returns
A list of the guild's emojis.

Return type
list[*novus.Emoji*]

async fetch_invites() → list[*Invite*]

Get the invites for the guild.

Requires the `MANAGE_GUILD` permission.

Returns
A list of invites.

Return type
list[*novus.Invite*]

async fetch_me() → *GuildMember*

Get the member object associated with the current guild and the current connection.

Note: Only usable via Oauth with the `guilds.members.read` scope. This is not usable as a bot.

See also:

novus.GuildMember.fetch_me()

Returns
The member object for the current user.

Return type
novus.GuildMember

async fetch_member(*member_id: int*) → *GuildMember*

Get a member from the guild.

See also:

novus.GuildMember.fetch()

Parameters
member_id (*int*) – The ID of the member you want to get.

Returns
The member object for the given user.

Return type
novus.GuildMember

async fetch_members(*, *limit: int = 1000*, *after: int = 0*) → list[*GuildMember*]

Get a list of members for the guild.

Note: This endpoint is restricted according to whether the `GUILD_MEMBERS` privileged intent is enabled for your application.

Note: This endpoint can return a maximum of 1000 members per request.

Parameters

- **limit** (*int*) – The number of guild members you want in the response payload.
- **after** (*int*) – The snowflake to get guild members after.

Returns
A list of members from the guild.

Return type
list[*novus.GuildMember*]

async fetch_roles() → list[*Role*]

Get a list of roles for the guild.

Returns
A list of roles in the guild.

Return type
list[novus.model.Role]

async fetch_scheduled_events(*, *with_user_count: bool = False*) → list[*ScheduledEvent*]

Get a list of all of the scheduled events for a guild.

See also:

novus.ScheduledEvent.fetch_all_for_guild()

Parameters
with_user_count (*bool*) – Whether or not to include the event's user count.

Returns
The scheduled events for the guild.

Return type
list[*novus.ScheduledEvent*]

async fetch_sticker(*id: AnySnowflake*) → *Sticker*

Get an individual sticker associated with the guild via its ID.

See also:

novus.Sticker.fetch()

Parameters
id (*str*) – The ID of the sticker.

Returns
The associated sticker instance.

Return type
novus.Sticker

async kick(*user: AnySnowflake*, *, *reason: str | None = None*) → None

Remove a user from the guild.

Requires the KICK_MEMBERS permission.

Parameters

- **user** (*int | novus.abc.Snowflake*) – The user you want to remove.
- **reason** (*str | None*) – The reason to be shown in the audit log.

async leave() → None

Leave the current guild.

async remove_member_role(*user: AnySnowflake*, *role: AnySnowflake*, **, reason: str | None = None*) → None

Remove a role from a member.

Requires the `MANAGE_ROLES` permission.

See also:

novus.GuildMember.remove_role()

Parameters

- **user** (*int | novus.abc.Snowflake*) – The user you want to add the role to.
- **role** (*int | novus.abc.Snowflake*) – The ID of the role you want to add.
- **reason** (*str | None*) – The reason shown in the audit log.

async search_members(**, query: str, limit: int = 1*) → list[*GuildMember*]

Get a list of members for the guild whose username of nickname starts with the provided string.

Note: This endpoint can return a maximum of 1000 members per request.

Parameters

- **query** (*str*) – the query string to match usernames and nicknames agains.
- **limit** (*int*) – The number of guild members you want in the response payload.

Returns
A list of members from the guild.

Return type
list[*novus.GuildMember*]

async unban(*user: AnySnowflake*, **, reason: str | None = None*) → None

Remove a user's ban

Parameters

- **user** (*int | novus.abc.Snowflake*) – The user who you want to ban.
- **reason** (*str | None*) – The reason to be shown in the audit log.

class novus.Invite(**, state: HTTPConnection, data: payloads.InviteWithMetadata | payloads.Invite*)

A model representing a guild invite.

code

The code associated with the invite.

Type
str

channel

The channel that the invite leads to.

Type
novus.Channel | None

uses

How many times the invite has been used.

Type
int | None

max_uses

The maximum number of times the invite can be used.

Type
int | None

max_age

Duration (in seconds) after which the invite expires.

Type
int | None

temporary

Whether the invite only grants temporary membership.

Type
bool | None

created_at

The time that the invite was created.

Type
datetime.datetime | None

guild

The guild that the invite leads to. Could be `None` if the invite leads to a group DM.

Type
novus.PartialGuild | None

async classmethod fetch(*state:* HTTPConnection, *code:* *str*) → *Invite*

Get an invite object via its code.

Parameters

- **state** (`HTTPConnection`) – The API connection.
- **code** (`str`) – The invite code, or URL. If an invalid code/URL is given, this is still passed over to the API for it to reject. This *will* count towards your API limit.

Returns
The invite associated with the code.

Return type
novus.Invite

async delete(*, *reason:* *str* | *None* = *None*) → *Invite*

Delete an instance of the invite.

Parameters
reason (`str | None`) – The reason shown in the audit log.

Returns
The deleted invite object.

Return type
novus.Invite

class novus.**Message**(*, *state:* HTTPConnection, *data: payloads.Message*)

A model representing a message from Discord.

id

The ID of the message.

Type
int

channel

A snowflake channel object.

Type
novus.Channel

guild

The guild associated with the message.

Note: If the message is fetched via the API, the guild will set to `None`.

Type
novus.Guild | None

author

The author of the message.

Type
novus.User | *novus.GuildMember*

content

The content of the message.

Type
str

timestamp

When the message was sent.

Type
datetime.datetime

edited_timestamp

When the message was last edited.

Type
datetime.datetime | None

tts

Whether or not the message was sent with TTS.

Type
bool

mention_everyone

Whehter the message mentions everyone.

Type
bool

`mentions`

A list of users who were mentioned in the message.

Type
list[*novus.User*]

`mention_roles`

A list of role IDs that were mentioned in the message.

Type
list[int]

`mention_channels`

A list of channels that mentioned in the message.

Type
list[*novus.Channel*]

`attachments`

A list of attachments on the message.

Type
list[*novus.Attachment*]

`embeds`

A list of embeds on the message.

Type
list[*novus.Embed*]

`reactions`

A list of reactions on the message.

Type
list[*novus.Reaction*]

`pinned`

If the message is pinned.

Type
bool

`webhook_id`

If the message was sent by a webhook, this would be the webhook's ID.

Type
int | None

`type`

The type of the message.

See also:

novus.MessageType

Type
int

`activity`

The message activity attached to the object.

Type
novus.MessageActivity | None

`application`

The application sent with RPC chat embeds.

Type
novus.Application | None

`application_id`

If the message is an interaction, or application-owned webhook, this is the ID of the application.

Type
int | None

`message_reference`

Data showing the source of a crosspost, channel follow, pin, or reply.

Type
novus.MessageReference | None

`interaction`

Interaction data associated with the message.

Type
novus.MessageInteraction | None

`flags`

The message flags.

Type
novus.MessageFlags

`referenced_message`

The message associated with the reference.

Type
novus.Message | None

`interaction`

Data referring to the interaction if the message is associated with one.

Type
novus.Interaction | None

`thread`

The thread that was started from this message.

Type
novus.Channel | None

`components`

The components associated with the message.

Type
list[*novus.ActionRow*]

sticker_items

The stickers sent with the message.

Type
list[*novus.Sticker*]

position

An integer representin the approximate position in the thread.

Type
int | None

role_subscription_data

Data of the role subscription purchase.

Type
novus.RoleSubscription | None

jump_url

A URL to jump to the message.

Type
str

async classmethod create(*state:* HTTPConnection, *channel:* *int* | *abc.Snowflake*, *content:* *str* = *MISSING*, **, tts:* *bool* = *MISSING*, *embeds:* *list[*Embed*]* = *MISSING*, *allowed_mentions:* AllowedMentions = *MISSING*, *components:* *list[*ActionRow*]* = *MISSING*, *message_reference:* Message = *MISSING*, *stickers:* *list[*Sticker*]* = *MISSING*, *files:* *list[*File*]* = *MISSING*, *flags:* *flags.MessageFlags* = *MISSING*) → *Message*

Send a message to the given channel.

See also:

novus.Channel.send()

Parameters

- **state** (novus.api.HTTPConnection) – The API connection.
- **channel** (*int | novus.abc.Snowflake*) – The channel to send the message to.
- **content** (*str*) – The content to be added to the message.
- **tts** (*bool*) – Whether the message should be sent with TTS.
- **embeds** (*list[*novus.Embed*]*) – A list of embeds to be added to the message.
- **allowed_mentions** (novus.AllowedMentions) – An object of users that you want to be pinged.
- **components** (*list[*novus.ActionRow*]*) – A list of components to add to the message.
- **message_reference** (novus.Message) – A message to reply to.
- **stickers** (*list[*novus.Sticker*]*) – A list of stickers to add to the message.
- **files** (*list[*novus.File*]*) – A list of files to be added to the message.
- **flags** (novus.MessageFlags) – Message send flags.

Returns
The created message.

Return type
novus.Message

async classmethod fetch(*state:* HTTPConnection, *channel:* *int* | *abc.Snowflake*, *message:* *int* | *abc.Snowflake*) → *Message*

Get an existing message.

See also:

novus.Channel.fetch_message()

Parameters

- **state** (novus.api.HTTPConnection) – The API connection.
- **channel** (*int* | *novus.abc.Snowflake*) – The channel to send the message to.
- **message** (*int* | *novus.abc.Snowflake*) – The message you want to get.

Returns
The created message.

Return type
novus.Message

async delete(*, *reason:* *str* | *None* = *None*) → None

Delete a message.

Parameters
reason (*str* | *None*) – The reason to be added to the audit log.

async crosspost() → *Message*

Crosspost a message.

async add_reaction(*emoji:* *str* | PartialEmoji) → None

Add a reaction to a message.

Parameters
emoji (*str* | novus.PartialEmoji) – The emoji to add to the message.

async remove_reaction(*emoji:* *str* | PartialEmoji, *user:* *int* | *abc.Snowflake*) → None

Remove a reaction from a message.

Parameters

- **emoji** (*str* | novus.PartialEmoji) – The emoji to remove from the message.
- **user** (*int* | *novus.abc.Snowflake*) – The user whose reaction you want to remove.

async fetch_reactions(*emoji:* *str* | PartialEmoji) → list[*User*]

Get a list of users who reacted to a message.

Parameters
emoji (*str* | novus.PartialEmoji) – The emoji to check the reactors for.

async clear_reactions(*emoji:* *str* | PartialEmoji | *None* = *None*) → None

Remove all of the reactions for a given message.

Parameters
emoji (*str* | novus.PartialEmoji | *None*) – The specific emoji that you want to clear. If not given, all reactions will be cleared.

async edit(*content: str | None = MISSING, *, tts: bool = MISSING, embeds: list[Embed] | None = MISSING, allowed_mentions: AllowedMentions = MISSING, components: list[ActionRow] | None = MISSING, message_reference: Message | None = MISSING, stickers: list[Sticker] | None = MISSING, files: list[File] | None = MISSING, attachments: list[Attachment] | None = MISSING, flags: flags.MessageFlags = MISSING*) → *Message*

Edit an existing message.

Parameters

- **content** (*str | None*) – The content to be added to the message.
- **tts** (*bool*) – Whether the message should be sent with TTS.
- **embeds** (*list[novus.Embed] | None*) – A list of embeds to be added to the message.
- **allowed_mentions** (novus.AllowedMentions) – An object of users that you want to be pinged.
- **components** (*list[novus.ActionRow] | None*) – A list of components to add to the message.
- **message_reference** (novus.Message | *None*) – A message to reply to.
- **stickers** (*list[novus.Sticker] | None*) – A list of stickers to add to the message.
- **files** (*list[novus.File] | None*) – A list of files to be appended to the message.
- **attachments** (*list[novus.Attachment] | None*) – A list of attachments currently on the message to keep.
- **flags** (novus.MessageFlags) – Message send flags.

Returns
The edited message.

Return type
novus.Message

async pin(**, reason: str | None = None*) → None

Pin the message to the channel.

Parameters
reason (*str | None*) – The reason shown in the audit log.

async unpin(**, reason: str | None = None*) → None

Unpin a message from the channel.

Parameters
reason (*str | None*) – The reason shown in the audit log.

async create_thread(*name: str, *, reason: str | None = None, auto_archive_duration: Literal[60, 1440, 4320, 10080] = MISSING, rate_limit_per_user: int = MISSING*) → *Channel*

Create a thread from the message.

Parameters

- **name** (*str*) – The name of the thread.
- **auto_archive_duration** (*int*) – The auto archive duration for the thread.
- **rate_limit_per_user** (*int*) – The number of seconds a user has to wait before sending another message.
- **reason** (*str | None*) – The reason shown in the audit log.

class novus.**PartialEmoji**(*, *data: payloads.PartialEmoji | payloads.Emoji | payloads.ForumDefaultReaction*)

Any generic emoji.

id

The ID of the emoji.

Type
int | None

name

The name of the emoji. Could be `None` in the case that the emoji came from a reaction payload and isn't unicode.

Type
str | None

animated

If the emoji is animated.

Type
bool

asset

The asset associated with the emoji, if it's a custom emoji.

Type
novus.Asset | None

classmethod from_str(*value: None*) → None

classmethod from_str(*value: str* | PartialEmoji) → *PartialEmoji*

Transform a string into an emoji object.

Parameters
value (*str | novus.PartialEmoji | None*) – The emoji you want converted. Can either be a Discord-style emoji string, a unicode emoji, or a ":smile:" style emoji via its name. If an emoji object is provided, then it is returned unchanged. If `None` is provided, it is returned as is.

Returns
The converted emoji, if a value was provided.

Return type
novus.PartialEmoji | None

Raises
ValueError – If the given value wasn't convertable to an emoji.

class novus.**PartialGuild**(*, *state:* HTTPConnection, *data: payloads.Guild*)

A model for a partial guild object, such as one retrieved from an invite link.

This model still implements the normal guild API methods, though does not contain all of the data a guild would (see attributes).

id

The ID of the guild.

Type
int

`name`

The name of the guild.

Type
str

`splash_hash`

The splash hash of the guild.

Type
str | None

`splash`

The splash asset for the guild.

Type
novus.Asset | None

`banner_hash`

The banner hash of the guild.

Type
str | None

`banner`

The banner asset for the guild.

Type
novus.Asset | None

`description`

A description of the guild.

Type
str | None

`icon_hash`

The icon hash for the guild.

Type
str | None

`icon`

An icon asset for the guild.

Type
novus.Asset | None

`features`

A list of features the guild implements.

Type
list[str]

`verification_level`

The guild's verification level.

Type
int

vainity_url_code

The guild's vainity URL code.

Type
str | None

nsfw_level

The guild's NSFW level.

See also:

novus.NSFWLevel

Type
int

premium_subscription_count

The number of nitro boosts the guild has.

Type
int

async add_member(*user_id: int*, *access_token: str*, **, nick: str = MISSING*, *mute: bool = MISSING*, *deaf: bool = MISSING*) → *GuildMember* | None

Add a member to the guild.

Note: This requires an Oauth access token, and the provided user ID must be the same one that matches the account.

Parameters

- **user_id** (*int*) – The ID of the user that you want to add. The user ID must match the ID of the oauth token.
- **access_token** (*str*) – The access token with the `guilds.join` scope to the bot's application for the user you want to add to the guild.
- **nick** (*str*) – The nickname youy want to set the user to.
- **mute** (*bool*) – Whether the user is muted in voice channels.
- **deaf** (*bool*) – Whether the user is deafened in voice channels.

Returns
The member for the user that was added to the guild, or `None` if the user was already present.

Return type
novus.GuildMember | None

async add_member_role(*user: AnySnowflake*, *role: AnySnowflake*, **, reason: str | None = None*) → None

Add a role to a user.

Requires the `MANAGE_ROLES` permission.

See also:

novus.GuildMember.add_role()

Parameters

- **user** (*int | novus.abc.Snowflake*) – The user you want to add the role to.
- **role** (*int | novus.abc.Snowflake*) – The role you want to add.
- **reason** (*str | None*) – The reason shown in the audit log.

async ban(*user: AnySnowflake, *, reason: str | None = None, delete_message_seconds: int = MISSING*) → None

Ban a user from the guild.

See also:

novus.GuildMember.ban()

Parameters

- **user** (*int | novus.abc.Snowflake*) – The user who you want to ban.
- **delete_message_seconds** (*int*) – The number of seconds of messages you want to delete.
- **reason** (*str | None*) – The reason to be shown in the audit log.

async chunk_members(*query: str = '', limit: int = 0, user_ids: list[int] | None = None, wait: bool = True*) → list[*GuildMember*] | None

Request member chunks from the gateway.

This will *only* work if you are connected to the gateway - this will not work with HTTP-only bots.

Parameters

- **query** (*str*) – A search string for usernames.
- **limit** (*int*) – A limit for the retrieved member count.
- **user_ids** (*list[int]*) – A list of user IDs to request.
- **wait** (*bool*) – Whether or not to wait for a response.

Returns
A list of requested users or `None` if you chose not to wait.

Return type
list[*novus.GuildMember*] | None

async classmethod create(*state: HTTPConnection, *, name: str*) → *Guild*

Create a guild.

Parameters

- **state** (*novus.HTTPConnection*) – The API connection to create the entity with.
- **name** (*str*) – The name for the guild that you want to create.

Returns
The created guild.

Return type
novus.Guild

async create_auto_moderation_rule(*, *reason: str | None = None*, *name: str*, *event_type: int*, *trigger_type: int*, *actions: list[AutoModerationAction]*, *trigger_metadata: AutoModerationTriggerMetadata | None = None*, *enabled: bool = False*, *exempt_roles: list[AnySnowflake] | None = None*, *exempt_channels: list[AnySnowflake] | None = None*) → *AutoModerationRule*

Create a new auto moderation rule.

Parameters

- **name** (*str*) – The new name for the role.
- **event_type** (*int*) – The event type.

  See also:

  novus.AutoModerationEventType
- **trigger_type** (*int*) – The trigger type.

  See also:

  novus.AutoModerationTriggerType
- **actions** (*list[novus.AutoModerationAction]*) – The actions to be taken on trigger.
- **trigger_metadata** (*novus.AutoModerationTriggerMetadata | None*) – The trigger metadata.
- **enabled** (*bool*) – Whether the rule is enabled or not.
- **exempt_roles** (*list[int | novus.abc.Snowflake] | None*) – A list of roles that are exempt from the rule.
- **exempt_channels** (*list[int | novus.abc.Snowflake] | None*) – A list of channels that are exempt from the rule.
- **reason** (*str | None*) – The reason shown in the audit log.

Returns
The created rule.

Return type
novus.AutoModerationRule

async create_channel(*, *name: str*, *type: int = MISSING*, *topic: str = MISSING*, *bitrate: int = MISSING*, *user_limit: int = MISSING*, *rate_limit_per_user: int = MISSING*, *position: int = MISSING*, *permission_overwrites: list[PermissionOverwrite] = MISSING*, *parent: AnySnowflake = MISSING*, *nsfw: bool = MISSING*, *default_auto_archive_duration: int = MISSING*, *default_reaction_emoji: Reaction = MISSING*, *available_tags: list[ForumTag] = MISSING*, *reason: str = MISSING*) → *Channel*

Create a channel within the guild.

Parameters

- **name** (*str*) – The name of the channel.
- **type** (*int*) – The type of the channel.

  See also:

  novus.ChannelType
- **bitrate** (*int*) – The bitrate for the channel. Only for use with voice channels.

- **user_limit** (*int*) – The user limit for the channel. Only for use with voice channels.
- **rate_limit_per_user** (*int*) – The slowmode seconds on the channel.
- **position** (*int*) – The channel position.
- **permission_overwrites** (*list[novus.PermissionOverwrite]*) – A list of permission overwrites for the channel.
- **parent** (*int | str | novus.abc.Snowflake*) – A parent object for the channel.
- **nsfw** (*bool*) – Whether or not the channel will be set to NSFW.
- **default_auto_archive_duration** (*int*) – The default duration that clients use (in minutes) to automatically archive the thread after recent activity. Only for use with forum channels.
- **default_reaction_emoji** (*Reaction*) – The default add reaction button to be shown on threads. Only for use with forum channels.
- **available_tags** (*list[ForumTag]*) – The tags available for threads. Only for use with forum channels.
- **reason** (*str*) – The reason to be shown in the audit log.

Returns
The created channel.

Return type
novus.model.Channel

async create_emoji(*, *name: str*, *image: FileT*, *roles: list[AnySnowflake] | None = None*, *reason: str | None = None*) → *Emoji*

Create an emoji within a guild.

Parameters

- **name** (*str*) – The name of the emoji you want to add.
- **image** (*str | bytes | io.IOBase*) – The image that you want to add.
- **roles** (*list[int | novus.abc.Snowflake] | None*) – A list of roles that are allowed to use the emoji.
- **reason** (*str | None*) – A reason you're adding the emoji.

Returns
The newly created emoji.

Return type
novus.Emoji

async create_role(*, *reason: str | None = None*, *name: str = MISSING*, *permissions: Permissions = MISSING*, *color: int = MISSING*, *hoist: bool = MISSING*, *icon: FileT = MISSING*, *unicode_emoji: str = MISSING*, *mentionable: bool = MISSING*) → *Role*

Create a role within the guild.

Parameters

- **name** (*str*) – The name of the role.
- **permissions** (*novus.Permissions*) – The permissions attached to the role.
- **color** (*int*) – The color of the role.
- **hoist** (*bool*) – Whether the role is displayed seperately in the sidebar.

- **icon** (*str | bytes | io.IOBase | None*) – The role icon image. Only usable if the guild has the ROLE_ICONS feature.
- **unicode_emoji** (*str*) – The role's unicode emoji. Only usable if the guild has the ROLE_ICONS feature.
- **mentionable** (*bool*) – Whether the role should be mentionable.
- **reason** (*str | None*) – The reason to be shown in the audit log.

async create_scheduled_event(*, *name: str*, *start_time: DiscordDatetime*, *entity_type: int*, *privacy_level: int*, *reason: str | None = None*, *channel: AnySnowflake | None = MISSING*, *location: str = MISSING*, *end_time: DiscordDatetime = MISSING*, *description: str | None = MISSING*, *status: int = MISSING*, *image: FileT | None = MISSING*) → *ScheduledEvent*

Create a new scheduled event.

See also:

novus.ScheduledEvent.create()

Parameters

- **name** (*str*) – The name of the event.
- **start_time** (*datetime.datetime*) – The time to schedule the event start.
- **entity_type** (*int*) – The type of the event.
- **privacy_level** (*int*) – The privacy level of the event.

  See also:

  novus.EventPrivacyLevel
- **channel** (*int | Snowflake | None*) – The channel of the scheduled event. Set to None if the event type is being set to external.
- **location** (*str*) – The location of the event.
- **end_time** (*datetime.datetime*) – The time to schedule the event end.
- **description** (*str | None*) – The description of the event.
- **status** (*int*) – The status of the event.

  See also:

  novus.EventStatus
- **image** (*str | bytes | io.IOBase | None*) – The cover image of the scheduled event.
- **reason** (*str | None*) – The reason shown in the audit log.

Returns
The new scheduled event.

Return type
novus.ScheduledEvent

async create_sticker(*, *reason: str | None = None*, *name: str*, *description: str | None = None*, *tags: str*, *image:* File) → *Sticker*

Create a new sticker.

See also:

novus.Sticker.create()

Parameters

- **name** (*str*) – The name of the sticker.
- **tags** (*str*) – Autocomplete/suggestion tags for the sitcker.
- **description** (*str | None*) – Description of the sticker.
- **image** (novus.File) – The image to be uploaded. All aside from the data itself is discarded - the name and description are taken from the other parameters.
- **reason** (*str | None*) – The reason shown in the audit log.
- **Reutrns**
- **-------**
- **novus.Sticker** – The created sticker instance.

async delete() → None

Delete the current guild permanently. You must be the owner of the guild to run this successfully.

async delete_role(*role: AnySnowflake*, *, *reason: str | None = None*) → None

A role to delete.

Parameters

- **role** (*int | novus.abc.Snowflake*) – The ID of the role to delete.
- **reason** (*str | None*) – The reason to be shown in the audit log.

async edit(*, *name: str = MISSING*, *verification_level: int | None = MISSING*, *default_message_notifications: int | None = MISSING*, *explicit_content_filter: int | None = MISSING*, *afk_channel: AnySnowflake | None = MISSING*, *icon: FileT | None = MISSING*, *owner: AnySnowflake = MISSING*, *splash: FileT | None = MISSING*, *discovery_splash: FileT | None = MISSING*, *banner: FileT | None = MISSING*, *system_channel: AnySnowflake | None = MISSING*, *system_channel_flags:* SystemChannelFlags *| None = MISSING*, *rules_channel: AnySnowflake | None = MISSING*, *preferred_locale: str | None = MISSING*, *public_updates_channel: AnySnowflake = MISSING*, *features: list[str] = MISSING*, *description: str | None = MISSING*, *premium_progress_bar_enabled: bool = MISSING*, *reason: str | None = None*) → *Guild*

Edit the guild parameters.

Note: The updated guild is not immediately put into cache - the bot waits for the guild update notification to be sent over the gateway before updating (which will not happen if you don't have the correct gateway intents).

Parameters

- **name** (*str*) – The name you want to set the guild to.
- **verification_level** (*int | None*) – The verification level you want to set the guild to.

  See also:

  novus.VerificationLevel

- **default_message_notifications** (*int | None*) – The default message notification level you want to set the guild to.

  See also:

  novus.NotificationLevel

- **explicit_content_filter** (*int | None*) – The content filter level you want to set the guild to.

  See also:

  novus.guild.ContentFilterLevel

- **afk_channel** (*int | novus.abc.Snowflake | None*) – The channel you want to set as the guild's AFK channel.
- **icon** (*str | bytes | io.IOBase | None*) – The icon that you want to set for the guild. Can be its bytes, a file path, or a file object.
- **owner** (*int | novus.abc.Snowflake*) – The person you want to set as owner of the guild. Can only be run if the current user is the existing owner.
- **splash** (*str | bytes | io.IOBase | None*) – The splash that you want to set for the guild. Can be its bytes, a file path, or a file object.
- **discovery_splash** (*str | bytes | io.IOBase | None*) – The discovery splash for the guild. Can be its bytes, a file path, or a file object.
- **banner** (*str | bytes | io.IOBase | None*) – The banner for the guild. Can be its bytes, a file path, or a file object.
- **system_channel** (*int | novus.abc.Snowflake | None*) – The system channel you want to set for the guild.
- **system_channel_flags** (*novus.guild.SystemChannelFlags | None*) – The system channel flags you want to set.
- **rules_channel** (*int | novus.abc.Snowflake | None*) – The channel you want to set as the rules channel.
- **preferred_locale** (*str | None*) – The locale you want to set as the guild's preferred.
- **public_updates_channel** (*int | novus.abc.Snowflake*) – The channel you want to set as the updates channel for the guild.
- **features** (*list[str]*) – A list of features for the guild.
- **description** (*str | None*) – A description for the guild.
- **premium_progress_bar_enabled** (*bool*) – Whether or not to enable the premium progress bar for the guild.
- **reason** (*str | None*) – A reason for modifying the guild (shown in the audit log).

Returns
The updated guild.

Return type
novus.Guild

async edit_member(*user: AnySnowflake, *, reason: str | None = None, nick: str | None = MISSING, roles: list[AnySnowflake] = MISSING, mute: bool = MISSING, deaf: bool = MISSING, voice_channel: AnySnowflake | None = MISSING, timeout_until: DiscordDatetime | None = MISSING*) → *GuildMember*

Edit a guild member.

See also:

novus.GuildMember.edit()

Parameters

- **user** (*int | novus.abc.Snowflake*) – The ID of the user you want to edit.
- **nick** (*str | None*) – The nickname you want to set for the user.
- **roles** (*list[int | novus.abc.Snowflake]*) – A list of roles that you want the user to have.
- **mute** (*bool*) – Whether or not the user is muted in voice channels. Will error if the user is not currently in a voice channel.
- **deaf** (*bool*) – Whether or not the user is deafened in voice channels. Will error if the user is not currently in a voice channel.
- **voice_channel** (*int | novus.abc.Snowflake | None*) – The voice channel that the user is in.
- **timeout_until** (*datetime.datetime | None*) – When the user's timeout should expire (up to 28 days in the future).

async edit_role(*role_id: int*, **, reason: str | None = None, name: str = MISSING, permissions: Permissions = MISSING, color: int = MISSING, hoist: bool = MISSING, icon: FileT = MISSING, unicode_emoji: str = MISSING, mentionable: bool = MISSING*) → *Role*

Edit a role.

Parameters

- **role_id** (*int*) – The ID of the role to be edited.
- **name** (*str*) – The new name of the role.
- **permissions** (*novus.Permissions*) – The permissions to be applied to the role.
- **color** (*int*) – The color to apply to the role.
- **hoist** (*bool*) – If the role should be displayed seperately in the sidebar.
- **icon** (*str | bytes | io.IOBase | None*) – The role's icon image. Only usable if the guild has the `ROLE_ICONS` feature.
- **unicode_emoji** (*str*) – The role's unicode emoji. Only usable if the guild has the `ROLE_ICONS` feature.
- **mentionable** (*bool*) – If the role is mentionable.
- **reason** (*str | None*) – The reason to be shown in the audit log.

async classmethod fetch(*state: HTTPConnection, guild: AnySnowflake*) → *Guild*

Get an instance of a guild from the API. Unlike the gateway's `GUILD_CREATE` payload, this method does not return members, channels, or voice states.

Parameters

- **state** (*HTTPConnection*) – The API connection.
- **guild** (*int | novus.abc.Snowflake*) – A reference to the guild that you want to fetch.

Returns
The guild associated with the given ID.

Return type
novus.Guild

async fetch_active_threads() → list[*Channel*]

Get the active threads from inside the guild.

Returns
A list of threads.

Return type
list[*novus.Channel*]

async fetch_all_emojis() → list[*Emoji*]

List all of the emojis for the guild.

See also:

novus.Emoji.fetch_all_for_guild()

Returns
A list of the guild's emojis.

Return type
list[*novus.Emoji*]

async fetch_all_stickers() → list[*Sticker*]

List all stickers associated with the guild.

See also:

novus.Sticker.fetch_all_for_guild()

Returns
The stickers associated with the guild.

Return type
list[*novus.Sticker*]

async fetch_audit_logs(*, *user_id: int | None = None*, *action_type: int | None = None*, *before: int | None = None*, *after: int | None = None*, *limit: int = 50*) → *AuditLog*

Get the audit logs for the guild.

Parameters

- **user_id** (*int | None*) – The ID of the moderator you want to to filter by.
- **action_type** (*int | None*) – The ID of an action to filter by.

  See also:

  novus.AuditLogEventType
- **before** (*int | None*) – The snowflake before which to get entries.
- **after** (*int | None*) – The snowflake after which to get entries.
- **limit** (*int*) – The number of entries to get. Max 100, defaults to 50.

Returns
The audit log for the guild.

Return type
novus.AuditLog

async fetch_auto_moderation_rules() → list[*AutoModerationRule*]

Get the auto moderation rules for this guild.

Returns
A list of the auto moderation rules for the guild.

Return type
list[*novus.AutoModerationRule*]

async fetch_ban(*user: AnySnowflake*) → *GuildBan*

Get an individual user's ban.

Parameters
user (*int | novus.abc.Snowflake*) – The user whose ban you want to get.

Returns
The ban for the user.

Return type
novus.GuildBan

async fetch_bans(*, *limit: int = 1000*, *before: int | None = None*, *after: int | None = None*) → list[*GuildBan*]

Get a list of bans from the guild.

Parameters

- **limit** (*str*) – The number of bans to get.
- **before** (*int | None*) – The snowflake to search around.
- **after** (*int | None*) – The snowflake to search around.

Returns
A list of bans from the guild.

Return type
list[novus.model.GuildBan]

async fetch_channels() → list[*Channel*]

Fetch all of the channels from a guild.

Returns
A list of channels from the guild.

Return type
list[*novus.Channel*]

async fetch_emoji(*id: int*) → *Emoji*

List all of the emojis for the guild.

See also:

novus.Emoji.fetch()

Returns
A list of the guild's emojis.

Return type
list[*novus.Emoji*]

async fetch_invites() → list[*Invite*]

Get the invites for the guild.

Requires the `MANAGE_GUILD` permission.

Returns
A list of invites.

Return type
list[*novus.Invite*]

async fetch_me() → *GuildMember*

Get the member object associated with the current guild and the current connection.

Note: Only usable via Oauth with the `guilds.members.read` scope. This is not usable as a bot.

See also:

novus.GuildMember.fetch_me()

Returns
The member object for the current user.

Return type
novus.GuildMember

async fetch_member(*member_id: int*) → *GuildMember*

Get a member from the guild.

See also:

novus.GuildMember.fetch()

Parameters
member_id (*int*) – The ID of the member you want to get.

Returns
The member object for the given user.

Return type
novus.GuildMember

async fetch_members(*, *limit: int = 1000*, *after: int = 0*) → list[*GuildMember*]

Get a list of members for the guild.

Note: This endpoint is restricted according to whether the `GUILD_MEMBERS` privileged intent is enabled for your application.

Note: This endpoint can return a maximum of 1000 members per request.

Parameters

- **limit** (*int*) – The number of guild members you want in the response payload.
- **after** (*int*) – The snowflake to get guild members after.

Returns
A list of members from the guild.

Return type
list[*novus.GuildMember*]

async fetch_roles() → list[*Role*]

Get a list of roles for the guild.

Returns
A list of roles in the guild.

Return type
list[novus.model.Role]

async fetch_scheduled_events(*, *with_user_count: bool = False*) → list[*ScheduledEvent*]

Get a list of all of the scheduled events for a guild.

See also:

novus.ScheduledEvent.fetch_all_for_guild()

Parameters
with_user_count (*bool*) – Whether or not to include the event's user count.

Returns
The scheduled events for the guild.

Return type
list[*novus.ScheduledEvent*]

async fetch_sticker(*id: AnySnowflake*) → *Sticker*

Get an individual sticker associated with the guild via its ID.

See also:

novus.Sticker.fetch()

Parameters
id (*str*) – The ID of the sticker.

Returns
The associated sticker instance.

Return type
novus.Sticker

async kick(*user: AnySnowflake*, *, *reason: str | None = None*) → None

Remove a user from the guild.

Requires the KICK_MEMBERS permission.

Parameters

- **user** (*int | novus.abc.Snowflake*) – The user you want to remove.
- **reason** (*str | None*) – The reason to be shown in the audit log.

async leave() → None

Leave the current guild.

async remove_member_role(*user: AnySnowflake*, *role: AnySnowflake*, ***, *reason: str | None = None*) → None

Remove a role from a member.

Requires the `MANAGE_ROLES` permission.

See also:

novus.GuildMember.remove_role()

Parameters

- **user** (*int | novus.abc.Snowflake*) – The user you want to add the role to.
- **role** (*int | novus.abc.Snowflake*) – The ID of the role you want to add.
- **reason** (*str | None*) – The reason shown in the audit log.

async search_members(***, *query: str*, *limit: int = 1*) → list[*GuildMember*]

Get a list of members for the guild whose username of nickname starts with the provided string.

Note: This endpoint can return a maximum of 1000 members per request.

Parameters

- **query** (*str*) – the query string to match usernames and nicknames agains.
- **limit** (*int*) – The number of guild members you want in the response payload.

Returns
A list of members from the guild.

Return type
list[*novus.GuildMember*]

async unban(*user: AnySnowflake*, ***, *reason: str | None = None*) → None

Remove a user's ban

Parameters

- **user** (*int | novus.abc.Snowflake*) – The user who you want to ban.
- **reason** (*str | None*) – The reason to be shown in the audit log.

class novus.**Reaction**(***, *state: HTTPConnection*, *data: payloads.Reaction | payloads.gateway.ReactionAddRemove*, *message_id: int | str | None = None*, *channel_id: int | str | None = None*)

A reaction container class.

message

A representation of the message that was reacted on.

Type
novus.abc.StateSnowflakeWithGuildChannel

emoji

The emoji that was added to the message. This will only ever be a partial emoji (ie it will only have ID, name, and animated attributes set).

Type
novus.PartialEmoji

burst

Whether the reaction was a burst reaction or not.

Type
bool

class novus.**Role**(*, *state:* HTTPConnection, *data: payloads.Role*, *guild_id: int* | *None* = *None*, *guild:* BaseGuild | *None* = *None*)

A model for a guild role.

id

The ID associated with the role.

Type
int

name

The name associated with the role.

Type
str

color

The color associated with the role, as a hex code.

Type
int

hoist

Whether the role is pinned in the user listing.

Type
bool

icon_hash

The hash associated with the role icon.

Type
str | None

icon

The asset associated with the role icon.

Type
novus.Asset | None

unicode_emoji

The role unicode emoji.

Type
str | None

position

The position of the role.

Note: The position of the role is calculated as a pair of the role's position attribute and it's ID attribute. Positions in a guild can be shared by multiple roles, or skipped entirely.

Type
int

permissions

The permissions for the role.

Type
novus.Permissions

managed

Whether the role is managed by an integration.

Type
bool

mentionable

Whether the role is mentionable.

Type
bool

tags

The tags associated with the role.

Type
list[dict]

guild

The guild (or a data container for the ID) that the emoji came from.

Type
novus.Guild

async delete(*, *reason: str | None = None*) → None

Delete the role from the guild.

Parameters
reason (*str | None*) – The reason shown in the audit log.

async edit(*, *reason: str | None = None*, *name: str = MISSING*, *permissions:* Permissions *= MISSING*, *color: int = MISSING*, *hoist: bool = MISSING*, *icon:* File *| None = MISSING*, *unicode_emoji: str | None = MISSING*, *mentionable: bool = MISSING*) → *Role*

Edit a role.

Parameters

- **name** (*str*) – The new name of the role.
- **permissions** (novus.Permissions) – The permissions to be applied to the role.
- **color** (*int*) – The color to apply to the role.
- **hoist** (*bool*) – If the role should be displayed seperately in the sidebar.
- **icon** (*discord.File | None*) – The role's icon image. Only usable if the guild has the ROLE_ICONS feature. All aside from the data itself is discarded.
- **unicode_emoji** (*str | None*) – The role's unicode emoji. Only usable if the guild has the ROLE_ICONS feature.
- **mentionable** (*bool*) – If the role is mentionable.
- **reason** (*str | None*) – The reason to be shown in the audit log.

class novus.**ScheduledEvent**(*, *state:* HTTPConnection, *data: payloads.GuildScheduledEvent*)

A model representing a scheduled event for a guild.

id

The ID of the event.

Type
int

guild

The guild that the event is taking place as part of.

Type
novus.Guild | *novus.Object*

channel

The channel that the event will take place in. Can be `None` if the event is taking place externally.

Type
novus.StageChannel | None

creator

The ID of the user that created the event. Will be `None` if the event was created before October 25th 2021.

Type
novus.User | None

name

The name of the event.

Type
str

description

The description given to the event.

Type
str | None

start_time

The scheduled start time of the event.

Type
datetime.datetime

end_time

The time the event will end. Required if the event type is external.

Type
datetime.datetime | None

privacy_level

The privacy level of the event.

See also:

novus.EventPrivacyLevel

Type
int

status

The status of the event.

See also:

novus.EventStatus

Type
int

entity_type

The tye of the scheduled event.

Type
int

entity_id

The ID of an entity associated with the event. Will not be set if the type is external.

Type
int | None

location

The given location of the event, if the event is external.

Type
str | None

user_count

The number of users subscribed to the scheduled event.

Type
int

image_hash

The cover image hash of the scheduled event.

Type
str | None

image

An asset associated with the cover image hash for the event.

Type
novus.Asset | None

async classmethod create(*state: HTTPConnection, guild: int | abc.Snowflake, *, name: str, start_time: dt, entity_type: int, privacy_level: int, reason: str | None = None, channel: int | abc.Snowflake | None = MISSING, location: str = MISSING, end_time: dt = MISSING, description: str | None = MISSING, status: int = MISSING, image: FileT | None = MISSING*) → *ScheduledEvent*

Create a new scheduled event.

See also:

`novus.Guild.create_scheduled_event()`

Parameters

- **state** (*novus.HTTPConnection*) – The API connection.

- **guild** (*int | novus.abc.Snowflake*) – A representation of the guild the event is to be created in.
- **name** (*str*) – The name of the event.
- **start_time** (*datetime.datetime*) – The time to schedule the event start.
- **entity_type** (*int*) – The type of the event.
- **privacy_level** (*int*) – The privacy level of the event.

  See also:

  novus.EventPrivacyLevel
- **channel** (*int | Snowflake | None*) – The channel of the scheduled event. Set to `None` if the event type is being set to external.
- **location** (*str*) – The location of the event.
- **end_time** (*datetime.datetime*) – The time to schedule the event end.
- **description** (*str | None*) – The description of the event.
- **status** (*int*) – The status of the event.

  See also:

  novus.EventStatus
- **image** (*str | bytes | io.IOBase | None*) – The cover image of the scheduled event.
- **reason** (*str | None*) – The reason shown in the audit log.

Returns
The new scheduled event.

Return type
novus.ScheduledEvent

async classmethod fetch(*state:* HTTPConnection, *guild:* *int* | *abc.Snowflake*, *id:* *int* | *abc.Snowflake*, **, with_user_count:* *bool* = *False*) → *ScheduledEvent*

Get a scheduled event via its ID.

Parameters

- **state** (HTTPConnection) – The API connection.
- **guild** (*int | novus.abc.Snowflake*) – A representation of the guild the event is from.
- **id** (*int | novus.abc.Snowflake*) – A representation of the ID of the event.
- **with_user_count** (*bool*) – Whether or not to include the event's user count.

Returns
The scheduled event associated with the ID.

Return type
novus.ScheduledEvent

async classmethod fetch_all_for_guild(*state:* HTTPConnection, *guild:* *int* | *abc.Snowflake*, **, with_user_count:* *bool* = *False*) → list[*ScheduledEvent*]

Get a list of all of the scheduled events for a guild.

See also:

novus.Guild.fetch_scheduled_events()

Parameters

- **state** (HTTPConnection) – The API connection.
- **guild** (*int | novus.abc.Snowflake*) – A representation of the guild the event is from.
- **with_user_count** (*bool*) – Whether or not to include the event's user count.

Returns
The scheduled events for the guild.

Return type
list[*novus.ScheduledEvent*]

async delete() → None

Delete the scheduled event.

async edit(*, *reason: str | None = None, channel: int | abc.Snowflake | None = MISSING, location: str = MISSING, name: str = MISSING, privacy_level: int = MISSING, start_time: dt = MISSING, end_time: dt = MISSING, description: str | None = MISSING, entity_type: int | None = MISSING, status: int = MISSING, image: FileT | None = MISSING*) → *ScheduledEvent*

Edit the scheduled event.

Parameters

- **channel** (*int | Snowflake | None*) – The channel of the scheduled event. Set to `None` if the event type is being set to external.
- **location** (*str*) – The location of the event.
- **name** (*str*) – The name of the event.
- **privacy_level** (*int*) – The privacy level of the event.

  See also:

  novus.EventPrivacyLevel
- **start_time** (*datetime.datetime*) – The time to schedule the event start.
- **end_time** (*datetime.datetime*) – The time to schedule the event end.
- **description** (*str | None*) – The description of the event.
- **entity_type** (*int | None*) – The type of the event.
- **status** (*int*) – The status of the event.

  See also:

  novus.EventStatus
- **image** (*str | bytes | io.IOBase | None*) – The cover image of the scheduled event.
- **reason** (*str | None*) – The reason shown in the audit log.

Returns
The updated scheduled event.

Return type
novus.ScheduledEvent

async fetch_users(*, *limit: int = 100, with_member: bool = False, before: int | None = None, after: int | None = None*) → list[*User* | *GuildMember*]

Get a scheduled event via its ID.

Parameters

- **limit** (*int*) – The number of users to return. Max 100.
- **with_member** (*bool*) – Whether to include guild member data if it exists.
- **before** (*int*) – Consider only users before the given ID.
- **after** (*int*) – Consider only users after the given ID.

Returns
The users/members subscribed to the event.

Return type
list[*novus.User* | *novus.GuildMember*]

class novus.**StageInstance**(*, *state:* HTTPConnection, *data: StageInstancePayload*)

A model for a stage instance, holding information about a live stage.

id

The ID of this stage instance.

Type
int

guild_id

The guild ID of the associated stage channel.

Type
int

channel_id

The ID of the associated stage channel.

Type
int

topic

The topic of the stage instance.

Type
str

privacy_level

The privacy level of the stage instance.

Type
int

event_id

The ID of the scheduled event for this stage instance.

Type
int | None

async classmethod create(*state:* HTTPConnection, *, *reason: str | None = None, channel: int | abc.Snowflake, topic: str, privacy_level: int = MISSING, send_start_notification: bool = MISSING*) → *StageInstance*

Create a stage instance.

Parameters

- **state** (HTTPConnection) – The API connection.

- **channel** (*int | Snowflake*) – The stage channel to be added to.
- **topic** (*str*) – The topic assigned to the stage.
- **privacy_level** (*int*) – The privacy level of the instance.

  See also:

  novus.EventPrivacyLevel
- **send_start_notification** (*bool*) – Notify @everyone that a stage instance has started.
- **reason** (*str | None*) – The reason shown in the audit log.

Returns
The stage instance.

Return type
novus.StageInstance

async classmethod fetch(*state:* HTTPConnection, *id: int*) → *StageInstance*

Get a created stage instance.

Parameters

- **state** (HTTPConnection) – The API connection.
- **id** (*int*) – The ID of the stage.

Returns
The stage instance.

Return type
novus.StageInstance

async edit(*, *topic: str = MISSING*, *privacy_level: int = MISSING*) → *StageInstance*

Update an existing stage instance.

Parameters

- **topic** (*str*) – The topic of the stage instance.
- **privacy_level** (*int*) – The privacy level of the stage instance.

  See also:

  novus.EventPrivacyLevel

Returns
The stage instance.

Return type
novus.StageInstance

class novus.**Sticker**(*, *state:* HTTPConnection, *data: payloads.Sticker | payloads.PartialSticker*)

A model for a sticker.

id

The ID of the sticker.

Type
int

pack_id

The ID of the pack that the sticker came in, for standard stickers.

Type
int | None

name

The name of the sticker.

Type
str

description

The description of the sticker.

Type
str

type

The type of the sticker.

See also:

novus.StickerType

Type
int

format_type

The format for the sticker.

See also:

novus.StickerFormat

Type
int

available

Whether or not the sticker can be used. May be `False` due to loss of nitro boosts.

Type
bool

asset

The asset associated with the sticker.

Type
novus.Asset

guild

The guild (or a data container for the ID) that the emoji came from. May be `None` if the sticker does not come from a guild.

Type
novus.Guild | None

async classmethod create(*state:* HTTPConnection, *guild:* *int* | *abc.Snowflake*, **, reason:* *str* | *None* = *None*, *name:* *str*, *description:* *str* | *None* = *None*, *tags:* *str*, *image:* File) → *Sticker*

Create a sticker within a guild.

Parameters

- **state** (*novus.HTTPConnection*) – The API connection.
- **guild** (*int | novus.abc.Snowflake*) – The guild you want to create the sticker within.
- **name** (*str*) – The name of the sticker.
- **tags** (*str*) – Autocomplete/suggestion tags for the sitcker.
- **description** (*str | None*) – Description of the sticker.
- **image** (novus.File) – The image to be uploaded. All aside from the data itself is discarded.
- **reason** (*str | None*) – The reason shown in the audit log.
- **Reutrns**
- **-------**
- **novus.Sticker** – The created sticker instance.

async classmethod fetch(*state:* HTTPConnection, *guild:* *int* | *abc.Snowflake*, *id:* *int*) → *Sticker*

Get an instance of a guild sticker from the API.

See also:

novus.Guild.fetch_sticker()

Parameters

- **state** (*novus.HTTPConnection*) – The API connection.
- **guild** (*int | novus.abc.Snowflake*) – The guild where the sticker resides.
- **id** (*int*) – The ID of the sticker you want to get.

Returns
The retrieved sticker instance.

Return type
novus.Sticker

async classmethod fetch_all_for_guild(*state:* HTTPConnection, *guild:* *int* | *abc.Snowflake*) → list[*Sticker*]

Get an instance of a guild sticker from the API.

See also:

novus.Guild.fetch_all_stickers()

Parameters

- **state** (*novus.HTTPConnection*) – The API connection.
- **guild** (*int | novus.abc.Snowflake*) – The guild where the stickers reside.

Returns
The retrieved sticker instances.

Return type
list[*novus.Sticker*]

async edit(*, *reason: str | None = None*, *name: str = MISSING*, *description: str | None = MISSING*, *tags: str = MISSING*) → *Sticker*

Edit a sticker from a guild.

Parameters

- **name** (*str*) – The new name for the sticker.
- **description** (*str | None*) – The new description for the sticker.
- **tags** (*str*) – The new tags for the sticker.
- **reason** (*str | None*) – The reason shown in the audit log.

Returns
The updated sticker instance.

Return type
novus.Sticker

async delete(*, *reason: str | None = None*) → *Sticker*

Delete a guild sticker.

Parameters
reason (*str | None*) – The reason shown in the audit log.

Returns
The updated sticker instance.

Return type
novus.Sticker

class novus.**Team**(*, *state:* HTTPConnection, *data: payloads.ApplicationTeam*)

A team associated with an application.

id

The ID of the team.

Type
int

icon_hash

The icon hash for the team icon.

Type
str | None

icon

An asset for the team icon.

Type
novus.Asset | None

members

A list of team members.

Type
list[*novus.TeamMember*]

name

The name of the team.

Type
str

owner_user_id

The ID of the owner of the team.

Type
int

class novus.**TeamMember**(*, *state:* HTTPConnection, *data: payloads.ApplicationTeamMember*)

A user within a team.

accepted

If the user has accepted the team invite.

Type
bool

permissions

The permissions that the team member has within the team.

Type
list[str]

team_id

The ID of the team that the user is part of.

Type
int

user

The team member. A partial object.

Type
novus.User

class novus.**ThreadMember**(*, *state:* HTTPConnection, *data: payloads.ThreadMember*, *guild_id: AnySnowflake | None = None*)

A model representing a member inside of a thread.

id

The ID of the user.

Type
int

thread_id

The ID of the thread.

Type
int

join_timestamp

The time that the user joined the thread.

Type
datetime.datetime

member

The guild member object.

Type
novus.GuildMember | None

class novus.**User**(*, *state:* HTTPConnection, *data: payloads.User | payloads.PartialUser*)

A model for a user object.

id

The ID of the user.

Type
int

username

The username of the user.

Type
str

global_name

The global name of the user.

Type
str | None

discriminator

The discriminator of the user.

Type
str

avatar_hash

The avatar hash of the user.

Type
str | None

avatar

The avatar of the user.

Type
novus.Asset | None

bot

Whether or not the user is associated with an Oauth2 application.

Type
bool

system

Whether or not the user is associated with a Discord system message.

Type
bool

mfa_enabled

Whether or not there's MFA available on the account. Only set properly for when you're receiving your own user via an Oauth2 application.

Type
bool

`banner_hash`

The hash for the user banner.

Type
str | None

`banner`

The asset for the user banner.

Type
novus.Asset | None

`accent_color`

The color associated with the user's accent color.

Type
int

`locale`

The locale for the user. Only set properly for when you're receiving your own user via an Oauth2 application.

Type
str | None

`verified`

Whether or not the user has a verified username attached. Only set properly for when you're receiving your own user via an Oauth2 application.

Type
bool

`email`

The email associated with the account. Only set properly for when you're receiving your own user via an Oauth2 application.

Type
str | None

`flags`

The flags associated with the user account. A combination of public and private.

Type
novus.UserFlags

`premium_type`

The premium type associated with the account.

See also:

novus.UserPremiumType

Type
int

status

The status of the user.

See also:

novus.Status

Type
str

activities

The activites of the user.

Type
list[*novus.Activity*]

property mention: str

A ping for the user.

async classmethod fetch(*state:* HTTPConnection, *id: int*) → *User*

Get an instance of a user from the API.

Parameters

- **state** (HTTPConnection) – The API connection.
- **id** (*int*) – The ID associated with the user you want to get.

Returns
The user associated with the given ID.

Return type
novus.User

async classmethod fetch_me(*state:* HTTPConnection) → *User*

Get the user associated with the current connection.

Parameters
state (HTTPConnection) – The API connection.

Returns
The user associated with the given ID.

Return type
novus.User

async classmethod fetch_my_guilds(*state:* HTTPConnection, *, *before: int* | *None* = *None*, *after: int* | *None* = *None*, *limit: int* = *200*) → list[OauthGuild]

Return a list of partial guild objects that the current user is a member of.

The endpoint returns 200 guilds by default, which is the maximum number of guilds that a non-bot can join.

Parameters

- **state** (HTTPConnection) – The API connection.
- **before** (*int | None*) – The snowflake before which to get guilds.
- **after** (*int | None*) – The snowflake after which to get guilds.
- **limit** (*int*) – The number of guilds you want to return.

Returns
A list of guilds associated with the current user.

Return type
list[novus.OauthGuild]

async create_dm_channel() → *Channel*

Open a DM channel with the given user.

Returns
The DM channel for the user.

Return type
novus.Channel

async send(*content: str = MISSING, *, tts: bool = MISSING, embeds: list[Embed] = MISSING, allowed_mentions: AllowedMentions = MISSING, components: list[ActionRow] = MISSING, message_reference: Message = MISSING, stickers: list[Sticker] = MISSING, files: list[File] = MISSING, flags: flags.MessageFlags = MISSING*) → *Message*

Send a message to the channel associated with the model.

Parameters

- **content** (*str*) – The content that you want to have in the message
- **tts** (*bool*) – If you want the message to be sent with the TTS flag.
- **embeds** (*list[novus.Embed]*) – The embeds you want added to the message.
- **allowed_mentions** (novus.AllowedMentions) – The mentions you want parsed in the message.
- **components** (*list[novus.ActionRow]*) – A list of action rows to be added to the message.
- **message_reference** (novus.Message) – A reference to a message you want replied to.
- **stickers** (*list[novus.Sticker]*) – A list of stickers to add to the message.
- **files** (*list[novus.File]*) – A list of files to be sent with the message.
- **flags** (novus.MessageFlags) – The flags to be sent with the message.

class novus.**VoiceState**(*, *state:* HTTPConnection, *data: payloads.VoiceState*, *guild_id: int | None = None*)

The voice state associated with a user.

guild

The guild that the voice state is attached to.

Type
novus.Guild | None

channel

The channel associated with the voice state.

Type
novus.Channel | None

user

The user associated with the voice state.

Type
novus.GuildMember | *User*

suppress

Type
bool

session_id

Type
str

self_video

Whether the user has video enabled.

Type
bool

self_mute

Whether the user has muted themselves.

Type
bool

self_deaf

Whether the user has deafened themselves.

Type
bool

request_to_speak_timestamp

When the user requested to speak.

Type
novus.utils.DiscordDatetime | None

mute

Whether the user is muted.

Type
bool

deaf

Whether the user is deafened.

Type
bool

class novus.**Webhook**(*, *state:* HTTPConnection, *data: WebhookPayload*)

A model for a webhook instance.

id

The ID of the webhook.

Type
int

guild_id

The guild ID this webhook is for, if any.

Type
int | None

channel_id

The channel ID this webhook is for, if any.

Type
int | None

name

The default of the webhook.

Type
str | None

avatar_hash

The hash associated with the user avatar.

Type
str | None

avatar

The avatar asset associated with the hash.

Type
novus.Asset | None

token

The token of the webhook.

Type
str | None

classmethod partial(*id: str | int, token: str | None = None, *, state: HTTPConnection | None = None*) → *Webhook*

Create a partial webhook state, allowing you to run webhook API methods.

Parameters

- **id** (*str | int*) – The ID of the webhook.
- **token** (*str | None*) – The auth token for the webhook.
- **state** (*HTTPConnection | None*) – The API connection, if one is made. Passing this enables API methods to be run on returned objects (eg a *Message.guild* from a message returned by executing a webhook). If no state is provided, one will be created for you to enable the sending of messages.

Returns
The created webhook instance.

Return type
novus.Webhook

classmethod from_url(*url: str, state: HTTPConnection | None = None*) → *Webhook*

Get a webhook object from a valid Discord URL. This won't get attributes like the name or channel ID, but will allow you to run API methods with the object.

Parameters

- **url** (*str*) – The URL of the webhook.
- **state** (*HTTPConnection | None*) – The API connection, if one is made. Passing this enables API methods to be run on returned objects (eg a *Message.guild* from a message returned by executing a webhook). If no state is provided, one will be created for you to enable the sending of messages.

Returns
The created webhook instance.

Return type
novus.Webhook

Raises
ValueError – The provided URL is not valid.

async classmethod fetch(*state:* HTTPConnection, *id:* *int*, *token:* *str* | *None* = *None*) → *Webhook*

Get a webhook instance.

Parameters

- **state** (*novus.HTTPConnection*) – The API connection.
- **id** (*int*) – The ID of the webhook.
- **token** (*str*) – The webhook token.

Returns
The webhook instance.

Return type
novus.Webhook

async edit(*, *reason:* *str* | *None* = *None*, *name:* *str* = *MISSING*, *avatar:* File | *None* = *MISSING*, *channel:* *int* | *abc.Snowflake* = *MISSING*) → *Webhook*

Edit the webhook.

Parameters

- **name** (*str*) – The new name of the webhook.
- **avatar** (*novus.File | None*) – The avatar of the webhook.
- **channel** (*int | novus.abc.Snowflake*) – The channel to move the webhook to.
- **reason** (*str | None*) – The reason shown in the audit log.

Returns
The updated webhook instance.

Return type
novus.Webhook

async edit_with_token(*, *reason:* *str* | *None* = *None*, *name:* *str* = *MISSING*, *avatar:* File | *None* = *MISSING*) → *Webhook*

Edit the webhook.

Parameters

- **name** (*str*) – The new name of the webhook.
- **avatar** (*novus.File | None*) – The avatar of the webhook.
- **reason** (*str | None*) – The reason shown in the audit log.

Returns
The updated webhook instance.

Return type
novus.Webhook

async send(*content: str*, *, *wait: Literal[False] = False*, *thread: int | abc.Snowflake | None*, *tts: bool*, *embeds: list[Embed]*, *components: list[ActionRow] = MISSING*, *allowed_mentions: AllowedMentions*, *message_reference: Message*, *stickers: list[Sticker]*, *files: list[File]*, *flags: MessageFlags*) → None

async send(*content: str*, *, *wait: Literal[True] = True*, *thread: int | abc.Snowflake | None*, *tts: bool*, *embeds: list[Embed]*, *components: list[ActionRow] = MISSING*, *allowed_mentions: AllowedMentions*, *message_reference: Message*, *stickers: list[Sticker]*, *files: list[File]*, *flags: MessageFlags*) → *Message*

Send a message to the channel associated with the webhook. Requires a token inside of the webhook.

Parameters

- **wait** (*bool*) – Whether or not to wait for a message response.
- **thread** (*int | Snowflake | None*) – A reference to a thread to send a message in.
- **content** (*str*) – The content that you want to have in the message
- **tts** (*bool*) – If you want the message to be sent with the TTS flag.
- **embeds** (*list[novus.Embed]*) – The embeds you want added to the message.
- **components** (*list[novus.ActionRow]*) – The components that you want added to the message.
- **allowed_mentions** (novus.AllowedMentions) – The mentions you want parsed in the message.
- **message_reference** (*novus.MessageReference*) – A reference to a message you want replied to.
- **stickers** (*list[novus.Sticker]*) – A list of stickers to add to the message.
- **files** (*list[novus.File]*) – A list of files to be sent with the message.
- **flags** (novus.MessageFlags) – The flags to be sent with the message.

class novus.**WebhookMessage**(*, *webhook: Webhook*, ***kwargs: Any*)

async classmethod fetch(*state: HTTPConnection*, *webhook: int | abc.Snowflake*, *webhook_token: str*, *message: int | abc.Snowflake*) → *WebhookMessage*

Get an existing message using the webhook.

Parameters

- **state** (novus.api.HTTPConnection) – The API connection.
- **webhook** (*int | novus.abc.Snowflake*) – The webhook that sent the message.
- **webhook_token** (*str*) – The token associated with the webhook.
- **message** (*int | novus.abc.Snowflake*) – The message you want to get.

Returns
The created message.

Return type
novus.Message

async delete() → None

Delete a webhook message.

Parameters

- **webhook** (*int | novus.abc.Snowflake*) – The webhook that sent the message.

- **webhook_token** (*str*) – The token associated with the webhook.
- **message** (*int | novus.abc.Snowflake*) – The message you want to delete.

async edit(*content: str | None = MISSING, *, tts: bool = MISSING, embeds: list[Embed] | None = MISSING, allowed_mentions: AllowedMentions = MISSING, components: list[ActionRow] | None = MISSING, message_reference: Message | None = MISSING, stickers: list[Sticker] | None = MISSING, files: list[File] | None = MISSING, attachments: list[Attachment] | None = MISSING, flags: flags.MessageFlags = MISSING*) → *Message*

Edit an existing message sent by the webhook.

Parameters

- **content** (*str | None*) – The content to be added to the message.
- **tts** (*bool*) – Whether the message should be sent with TTS.
- **embeds** (*list[novus.Embed] | None*) – A list of embeds to be added to the message.
- **allowed_mentions** (*novus.AllowedMentions*) – An object of users that you want to be pinged.
- **components** (*list[novus.ActionRow] | None*) – A list of components to add to the message.
- **message_reference** (*novus.Message | None*) – A message to reply to.
- **stickers** (*list[novus.Sticker] | None*) – A list of stickers to add to the message.
- **files** (*list[novus.File] | None*) – A list of files to be appended to the message.
- **attachments** (*list[novus.Attachment] | None*) – A list of attachments currently on the message to keep.
- **flags** (*novus.MessageFlags*) – Message send flags.

Returns
The edited message.

Return type
novus.Message

async add_reaction(*emoji: str | PartialEmoji*) → None

Add a reaction to a message.

Parameters
emoji (*str | novus.PartialEmoji*) – The emoji to add to the message.

async clear_reactions(*emoji: str | PartialEmoji | None = None*) → None

Remove all of the reactions for a given message.

Parameters
emoji (*str | novus.PartialEmoji | None*) – The specific emoji that you want to clear. If not given, all reactions will be cleared.

async classmethod create(*state: HTTPConnection, channel: int | abc.Snowflake, content: str = MISSING, *, tts: bool = MISSING, embeds: list[Embed] = MISSING, allowed_mentions: AllowedMentions = MISSING, components: list[ActionRow] = MISSING, message_reference: Message = MISSING, stickers: list[Sticker] = MISSING, files: list[File] = MISSING, flags: flags.MessageFlags = MISSING*) → *Message*

Send a message to the given channel.

See also:

novus.Channel.send()

Parameters

- **state** (novus.api.HTTPConnection) – The API connection.
- **channel** (*int | novus.abc.Snowflake*) – The channel to send the message to.
- **content** (*str*) – The content to be added to the message.
- **tts** (*bool*) – Whether the message should be sent with TTS.
- **embeds** (*list[novus.Embed]*) – A list of embeds to be added to the message.
- **allowed_mentions** (novus.AllowedMentions) – An object of users that you want to be pinged.
- **components** (*list[novus.ActionRow]*) – A list of components to add to the message.
- **message_reference** (novus.Message) – A message to reply to.
- **stickers** (*list[novus.Sticker]*) – A list of stickers to add to the message.
- **files** (*list[novus.File]*) – A list of files to be added to the message.
- **flags** (novus.MessageFlags) – Message send flags.

Returns
The created message.

Return type
novus.Message

async create_thread(*name: str, *, reason: str | None = None, auto_archive_duration: Literal[60, 1440, 4320, 10080] = MISSING, rate_limit_per_user: int = MISSING*) → *Channel*

Create a thread from the message.

Parameters

- **name** (*str*) – The name of the thread.
- **auto_archive_duration** (*int*) – The auto archive duration for the thread.
- **rate_limit_per_user** (*int*) – The number of seconds a user has to wait before sending another message.
- **reason** (*str | None*) – The reason shown in the audit log.

async crosspost() → *Message*

Crosspost a message.

async fetch_reactions(*emoji: str* | PartialEmoji) → list[*User*]

Get a list of users who reacted to a message.

Parameters
emoji (*str | novus.PartialEmoji*) – The emoji to check the reactors for.

async pin(*, *reason: str | None = None*) → None

Pin the message to the channel.

Parameters
reason (*str | None*) – The reason shown in the audit log.

async remove_reaction(*emoji: str* | PartialEmoji, *user: int* | *abc.Snowflake*) → None

Remove a reaction from a message.

Parameters

- **emoji** (*str* | *novus.PartialEmoji*) – The emoji to remove from the message.
- **user** (*int* | *novus.abc.Snowflake*) – The user whose reaction you want to remove.

async unpin(*, *reason: str* | *None* = *None*) → None

Unpin a message from the channel.

Parameters

reason (*str* | *None*) – The reason shown in the audit log.

class novus.**WelcomeScreen**(*, *data: WelcomeScreenPayload*)

A welcome screen for a guild.

class novus.**WelcomeScreenChannel**(*, *data: WelcomeScreenChannelPayload*)

A channel shown in a guild's welcome screen.

1.1.2 User-Creatable Models

Models that itneract with Discord. These may be user-created, but some can still be returned by the API on certain methods.

class novus.**AllowedMentions**(*, *users: AMI* = *False*, *roles: AMI* = *False*, *everyone: bool* = *False*)

Allowed mentions for a particular message. Have more fine-grained control over what and who you mention.

Parameters

- **users** (*bool* | *list[int]* | *list[novus.abc.Snowflake]*) – A list of users (or IDs) that you want to mention.
- **roles** (*bool* | *list[int]* | *list[novus.abc.Snowflake]*) – A list of roles (or IDs) that you want to mention.
- **everyone** (*bool*) – Whether or not you want @everyone and @here pings to be parsed.

classmethod none() → Self

An allowed mentions object with no pings allowed.

classmethod all() → Self

An allowed mentions object with all pings allowed.

classmethod only(*target:* Role | User | GuildMember) → Self

Users or roles that you want to be the only parsed mention in a given message.

Parameters

target (novus.Role | novus.User | novus.GuildMember) – The

class novus.**Embed**(*, *title: str* | *None* = *None*, *type: str* = *'rich'*, *description: str* | *None* = *None*, *url: str* | *None* = *None*, *timestamp: datetime* | *None* = *None*, *color: int* | *None* = *None*)

A model for an embed object.

Parameters

- **title** (*str*) – The title on the embed.
- **description** (*str*) – The description of the embed.

- **url** (*str*) – The url of the embed, attached to the title.
- **timestamp** (*datetime.datetime*) – The timestamp in the footer of the bot.
- **color** (*int*) – The color of the embed.

title

The title of the embed.

Type
str | None

type

The type of the embed.

Type
str | None

description

The description of the embed.

Type
str | None

url

The URL of the embed.

Type
str | None

timestamp

The timestamp in the embed footer.

Type
datetime.datetime | None

color

The colour integer of the embed.

Type
int | None

footer

The footer of the embed. An object containing the following attributes:

- `text`: *str*
- `icon_url`: *str* | *None*
- `proxy_icon_url`: *str* | *None*

Type
object | None

image

The image added to the embed. An object containing the following attributes:

- `url`: *str*
- `proxy_url`: *str* | *None*
- `height`: *int* | *None*

- `width`: *int* | *None*

Type
object | None

`thumbnail`

The image added to the embed. An object containing the following attributes:

- `url`: *str*
- `proxy_url`: *str* | *None*
- `height`: *int* | *None*
- `width`: *int* | *None*

Type
object | None

`video`

The video added to the embed. An object containing the following attributes:

- `url`: *str* | *None*
- `proxy_url`: *str* | *None*
- `height`: *int* | *None*
- `width`: *int* | *None*

Type
object | None

`provider`

The provider information. An object containing the following attributes:

- `name`: *str* | *None*
- `url`: *str* | *None*

Type
object | None

`author`

The author of the embed. An object containing the following attributes:

- `name`: *str*
- `url`: *str* | *None*
- `icon_url`: *str* | *None*
- `proxy_icon_url`: *str* | *None*

Type
object | None

fields

A list of fields added to the embed. An a field is an object containing the following attributes:

- `name`: *str*
- `value`: *str*
- `inline`: *bool*

Type
list[object]

update(*, *title: str | None = MISSING*, *description: str | None = MISSING*, *url: str | None = MISSING*, *timestamp: datetime | None = MISSING*, *color: int | None = MISSING*) → Self

Set an attribute of the embed via a single `.update` method.

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.
- **timestamp** (*datetime.datetime | None*) – The timestamp in the embed footer.
- **color** (*int | None*) – The colour integer of the embed.

Returns
The embed instance.

Return type
novus.Embed

set_footer(*text: str*, *, *icon_url: str | None = None*) → Self

Set the footer of the embed.

Parameters

- **text** (*str*) – The text to be added to the footer. Does not support markdown.
- **icon_url** (*str | None*) – The url of the icon to be used in the footer. Only supports HTTP(S) and attachments.

remove_footer() → Self

Remove the footer of the embed.

set_image(*url: str*) → Self

Set an image for the embed.

Parameters
url (*str*) – The source url of the image. Only supports HTTP(S) and attachments.

remove_image() → Self

Remove the image of the embed.

set_thumbnail(*url: str*) → Self

Set an thumbnail for the embed.

Parameters
url (*str*) – The source url of the thumbnail. Only supports HTTP(S) and attachments.

remove_thumbnail() → Self

Remove the thumbnail of the embed.

set_author(*name: str*, *, *url: str | None = None*, *icon_url: str | None = None*) → Self

Set the author of the embed.

Parameters

- **name** (*str*) – The name of the author in the embed.
- **url** (*str | None*) – The URL attached to the author's name in the embed.
- **icon_url** (*str | None*) – The url of the author's icon.

set_author_from_user(*user: User | GuildMember*) → Self

Set the author of the embed with the attributes present on a user.

Parameters

user (*novus.User | novus.GuildMember*) – The user that you want to set into the embed.

remove_author() → Self

Remove the author of the embed.

add_field(*name: str*, *value: str*, *, *inline: bool = True*) → Self

Add a field to the embed.

Parameters

- **name** (*str*) – The name of the field.
- **value** (*str*) – The value of the embed.
- **inline** (*bool*) – Whether or not the field should be inline.

remove_field(*index: int*) → Self

Remove a field at a given index.

Parameters

index (*int*) – The index of the field.

insert_field_at(*index: int*, *name: str*, *value: str*, *, *inline: bool = True*) → Self

Add a field to the embed at a specified location.

Parameters

- **index** (*int*) – The index that you want to add the field at.
- **name** (*str*) – The name of the field.
- **value** (*str*) – The value of the embed.
- **inline** (*bool*) – Whether or not the field should be inline.

clear_fields() → Self

Remove all of the fields from the embed.

class novus.**File**(*data: bytes | str | Path | IOBase*, *filename: str*, *, *description: str | None = None*, *spoiler: bool = False*)

A representation for an uploaded file to Discord.

Parameters

- **data** (*bytes | str | pathlib.Path | io.IOBase*) – The data that should be inserted into the file. Raw bytes can be provided (and is recommended), though a filename or *pathlib.Path* object can be used if you want the library to read the content directly, or a file handle can be provided directly.
- **filename** (*str*) – The filename given to the attachment. Any leading "SPOILER_" names will be automatically removed.
- **spoiler** (*bool*) – Whether or not the file should be marked as a spoiler on send.

data

The data associated with the attachment.

Type
bytes

filename

The filename for the attachment.

Type
str

description

The description of the file, if one is given.

Type
str | None

spoiler

Whether or not the attachment is a spoiler.

Type
bool

content_type

The type of the data.

Type
str

class novus.**Object**(*id: int | str, *, state: HTTPConnection, guild: abc.Snowflake | None = None, guild_id: int | None = None*)

An abstract class that you can pass around to other classes requiring IDs and a state.

class novus.**PermissionOverwrite**(*id: AnySnowflake, type: int | type[Role] | type[User] | None = None, *, allow: Permissions | None = None, deny: Permissions | None = None*)

A class representing a permission overwrite for a guild channel.

Parameters

- **id** (*int*) – The ID of the target.
- **type** (*int*) – The type of the target.

  See also:

  novus.PermissionOverwriteType

- **allow** (novus.Permissions) – The permissions that the target is explicitly allowed.
- **deny** (novus.Permissions) – The permissions that the target is explicitly denied.

id

The ID of the target.

Type
int

type

The type of the target.

See also:

novus.PermissionOverwriteType

Type
int

allow

The permissions that the target is explicitly allowed.

Type
novus.Permissions

deny

The permissions that the target is explicitly denied.

Type
novus.Permissions

1.1.3 Components

Models that relate to message components.

class novus.**ActionRow**(*components: Iterable[*InteractableComponent*] = MISSING*)

A generic layout component that holds other components.

This class implements a `__getitem__` and an `__iter__` method to allow for eady indexing and iterating.

Parameters
components (*Iterable[*novus.Component*]*) – A list of components to be initially added to the action row.

components

The components inside of the action row.

Type
list[*novus.Component* | None]

add(*component:* InteractableComponent) → Self

Add a component to the end of the action row.

Parameters
component (novus.Component) – The component that you want to add to the action row.

Returns
The action row instance, allowing for easy chaining.

Return type
novus.ActionRow

set(*index: int*, *component:* InteractableComponent | *None*) → Self

Set a component at a specified index. If the index given is larger than the current number of components, the components list will be filled with `None` values, which will be removed upon sending.

Parameters

- **index** (*int*) – The index that you want to set the component at.
- **component** (novus.Component | *None*) – The component that you want to set.

Returns
The action row instance, allowing for easy chaining.

Return type
novus.ActionRow

pop() → Self

Pop a component from the end of the action row.

Returns
The action row instance, allowing for easy chaining.

Return type
novus.ActionRow

clear() → Self

Clear all of the components from the action row.

Returns
The action row instance, allowing for easy chaining.

Return type
novus.ActionRow

__getitem__(*index: int*) → *InteractableComponent* | None

Get an item at the specified index.

Parameters
index (*int*) – The index that you want to get the component at.

Returns
The item at the index.

Return type
novus.Component | None

Raises
IndexError – If the given index does not exist.

__setitem__(*index: int*, *value:* InteractableComponent | *None*) → None

Set an item at the specified index.

Parameters

- **index** (*int*) – The index that you want to set the component at.
- **component** (novus.Component | *None*) – The component that you want to set.

__iter__() → Iterator[*InteractableComponent*]

An iterator over the components of the action row.

class novus.**Button**(*label: str | None = None*, *, *style: int = 2*, *custom_id: str*, *emoji: str | PartialEmoji | None = None*, *url: str | None = None*, *disabled: bool = False*)

A button component.

Parameters

- **label** (*str*) – The label of the button. Either this or `emoji` needs to be set.
- **style** (*int*) – The style of the button.

  See also:

  novus.ButtonStyle
- **custom_id** (*str*) – The custom ID of the component.
- **emoji** (`novus.PartialEmoji` | `novus.Emoji` | *str*) – The emoji attached to the button. Either this or `label` needs to be set.
- **url** (*str*) – The URL that the button leads to, if the style is *novus.ButtonStyle.url*.
- **disabled** (*bool*) – Whether or not the button is disabled.

label

The label of the button. Either this or `emoji` needs to be set.

Type
str | None

style

The style of the button.

See also:

novus.ButtonStyle

Type
int

custom_id

The custom ID of the component.

Type
str

emoji

The emoji attached to the button. Either this or `label` needs to be set.

Type
novus.PartialEmoji | *novus.Emoji* | None

url

The URL that the button leads to, if the style is *novus.ButtonStyle.url*.

Type
str | None

disabled

Whether or not the button is disabled.

Type
bool

class novus.**StringSelectMenu**(*, *options: Iterable[SelectOption] = MISSING*, *custom_id: str*, *placeholder: str | None = None*, *min_values: int = 1*, *max_values: int = 1*, *disabled: bool = False*)

A string select menu component.

Parameters

- **options** (*Iterable[SelectOption]*) – A list of options to be added to the select menu by default.
- **custom_id** (*str*) – The custom ID for the menu.
- **placeholder** (*str | None*) – Placeholder text to be shown in the menu when nothing is selected.
- **min_values** (*int | None*) – The minimum number of values to be selected before the menu will submit.
- **max_values** (*int | None*) – The maximum number of values to be selected before the menu will submit.
- **disabled** (*bool*) – If the component is disabled.

options

A list of options to be added to the select menu by default.

Type
list[*SelectOption*]

custom_id

The custom ID for the menu.

Type
str

placeholder

Placeholder text to be shown in the menu when nothing is selected.

Type
str | None

min_values

The minimum number of values to be selected before the menu will submit.

Type
int

max_values

The maximum number of values to be selected before the menu will submit.

Type
int

disabled

If the component is disabled.

Type
bool

add(*option: SelectOption*) → Self

Add an option to the select menu.

Parameters
option (novus.SelectOption) – The option that you want to add.

Returns
The menu instance, allowing for easy chaining.

Return type
novus.StringSelectMenu

set(*index: int*, *component:* SelectOption | *None*) → Self

Set an option at a specified index. If the index given is larger than the current number of components, the components list will be filled with `None` values, which will be removed upon sending.

Parameters

- **index** (*int*) – The index that you want to set the component at.
- **component** (novus.SelectOption | *None*) – The option that you want to set.

Returns
The menu instance, allowing for easy chaining.

Return type
novus.StringSelectMenu

pop() → Self

Pop an from the end of the menu.

Returns
The menu instance, allowing for easy chaining.

Return type
novus.StringSelectMenu

clear() → Self

Clear all of the options from the menu.

Returns
The menu instance, allowing for easy chaining.

Return type
novus.StringSelectMenu

__getitem__(*index: int*) → *SelectOption* | None

Get an option at the specified index.

Parameters
index (*int*) – The index that you want to get the option at.

Returns
The item at the index.

Return type
novus.SelectOption | None

Raises
IndexError – If the given index does not exist.

__setitem__(*index: int*, *value:* SelectOption | *None*) → None

Set an item at the specified index.

Parameters

- **index** (*int*) – The index that you want to set the option at.

- **component** (novus.SelectOption | None) – The option that you want to set.

__iter__() → Iterator[*SelectOption*]

An iterator over the options of the select menu.

class novus.**ChannelSelectMenu**(*, *channel_types: Iterable[int] = MISSING*, *custom_id: str*, *placeholder: str | None = None*, *min_values: int = 1*, *max_values: int = 1*, *disabled: bool = False*)

A select menu for channels within a guild.

Parameters

- **custom_id** (*str*) – The custom ID for the menu.
- **placeholder** (*str* | *None*) – Placeholder text to be shown in the menu when nothing is selected.
- **min_values** (*int* | *None*) – The minimum number of values to be selected before the menu will submit.
- **max_values** (*int* | *None*) – The maximum number of values to be selected before the menu will submit.
- **disabled** (*bool*) – If the component is disabled.

custom_id

The custom ID for the menu.

Type
str

placeholder

Placeholder text to be shown in the menu when nothing is selected.

Type
str | None

min_values

The minimum number of values to be selected before the menu will submit.

Type
int

max_values

The maximum number of values to be selected before the menu will submit.

Type
int

disabled

If the component is disabled.

Type
bool

class novus.**RoleSelectMenu**(*, *custom_id: str*, *placeholder: str | None = None*, *min_values: int = 1*, *max_values: int = 1*, *disabled: bool = False*)

A select menu for roles within a guild.

Parameters

- **custom_id** (*str*) – The custom ID for the menu.

- **placeholder** (*str | None*) – Placeholder text to be shown in the menu when nothing is selected.
- **min_values** (*int | None*) – The minimum number of values to be selected before the menu will submit.
- **max_values** (*int | None*) – The maximum number of values to be selected before the menu will submit.
- **disabled** (*bool*) – If the component is disabled.

custom_id

The custom ID for the menu.

Type
str

placeholder

Placeholder text to be shown in the menu when nothing is selected.

Type
str | None

min_values

The minimum number of values to be selected before the menu will submit.

Type
int

max_values

The maximum number of values to be selected before the menu will submit.

Type
int

disabled

If the component is disabled.

Type
bool

class novus.**UserSelectMenu**(*, *custom_id: str*, *placeholder: str | None = None*, *min_values: int = 1*, *max_values: int = 1*, *disabled: bool = False*)

A select menu for users within a guild.

Parameters

- **custom_id** (*str*) – The custom ID for the menu.
- **placeholder** (*str | None*) – Placeholder text to be shown in the menu when nothing is selected.
- **min_values** (*int | None*) – The minimum number of values to be selected before the menu will submit.
- **max_values** (*int | None*) – The maximum number of values to be selected before the menu will submit.
- **disabled** (*bool*) – If the component is disabled.

custom_id

The custom ID for the menu.

Type
str

placeholder

Placeholder text to be shown in the menu when nothing is selected.

Type
str | None

min_values

The minimum number of values to be selected before the menu will submit.

Type
int

max_values

The maximum number of values to be selected before the menu will submit.

Type
int

disabled

If the component is disabled.

Type
bool

class novus.**MentionableSelectMenu**(*, *custom_id: str*, *placeholder: str | None = None*, *min_values: int = 1*, *max_values: int = 1*, *disabled: bool = False*)

A select menu for both roles and users within a guild.

Parameters

- **custom_id** (*str*) – The custom ID for the menu.
- **placeholder** (*str | None*) – Placeholder text to be shown in the menu when nothing is selected.
- **min_values** (*int | None*) – The minimum number of values to be selected before the menu will submit.
- **max_values** (*int | None*) – The maximum number of values to be selected before the menu will submit.
- **disabled** (*bool*) – If the component is disabled.

custom_id

The custom ID for the menu.

Type
str

placeholder

Placeholder text to be shown in the menu when nothing is selected.

Type
str | None

min_values

The minimum number of values to be selected before the menu will submit.

Type
int

max_values

The maximum number of values to be selected before the menu will submit.

Type
int

disabled

If the component is disabled.

Type
bool

class novus.**SelectOption**(*label: str*, *value: str*, *, *emoji: str* | PartialEmoji | *None* = *None*, *description: str* | *None* = *None*, *default: bool* = *False*)

An option inside of a select menu.

Parameters

- **label** (*str*) – The label shown on the option.
- **value** (*str*) – The value given back to your application when selected.
- **description** (*str*) – The description shown on the option.
- **emoji** (novus.PartialEmoji / novus.Emoji / *str*) – The emoji shown on the option.
- **default** (*bool*) – Whether the option is selected by default or not.

label

The label shown on the option.

Type
str

value

The value given back to your application when selected.

Type
str

description

The description shown on the option.

Type
str | None

emoji

The emoji shown on the option.

Type
novus.PartialEmoji | *novus.Emoji* | None

default

Whether the option is selected or not.

Type
bool

class novus.**TextInput**(*label: str, *, style: int = 1, custom_id: str, min_length: int = 0, max_length: int = 4000, required: bool = True, value: str | None = None, placeholder: str | None = None*)

A text input component for inside of modals.

Parameters

- **label** (*str*) – The label on the text component.
- **style** (*int*) – The style of the component.

  See also:

  novus.TextInputStyle
- **custom_id** (*str*) – The custom ID for the input.
- **min_length** (*int*) – The minimum length of the user's input.
- **max_length** (*int*) – The maximum length of the user's input.
- **required** (*bool*) – Whether or not the input is required to be filled before you can submit the interaction.
- **value** (*str | None*) – The value filling the component.
- **placeholder** (*str | None*) – A placeholder string shown in the component when there's no value.

label

The label on the text component.

Type
str

style

The style of the component.

See also:

novus.TextInputStyle

Type
int

custom_id

The custom ID for the input.

Type
str

min_length

The minimum length of the user's input.

Type
int

max_length

The maximum length of the user's input.

Type
int

required

Whether or not the input is required to be filled before you can submit the interaction.

Type
bool

value

The value filling the component.

Type
str | None

placeholder

A placeholder string shown in the component when there's no value.

Type
str | None

1.1.4 Application Commands

Models that relate to application commands.

class novus.**PartialApplicationCommand**(*name: str, description: str | None, type: int, *, name_localizations: LocType = MISSING, description_localizations: LocType = MISSING, options: list[ApplicationCommandOption] = MISSING, default_member_permissions: Permissions | None = MISSING, dm_permission: bool = True, nsfw: bool = False*)

A partial application command object. This is an object that can be created and used by general users.

Parameters

- **name** (*str*) – The name of the command.
- **description** (*str*) – The description of the command.
- **type** (*int*) – The command type.

  See also:

  novus.ApplicationCommandType
- **name_localizations** (*dict[str, str] | novus.Localization*) – Localizations for the command name.
- **description_localizations** (*dict[str, str] | novus.Localization*) – Localizations for the description.
- **options** (*list[novus.ApplicationCommandOption]*) – Thje options for the command.
- **default_member_permissions** (novus.Permissions) – The permissions required (by default) to run the command.
- **dm_permission** (*bool*) – Whether or not the command can be run in DMs.
- **nsfw** (*bool*) – Whether or not the command is marked as NSFW.

name

The name of the command.

Type
str

description

The description of the command.

Type
str

type

The command type.

See also:

novus.ApplicationCommandType

Type
int

name_localizations

Localizations for the command.

Type
dict[str, str] | novus.Localization

description_localizations

Localizations for the description.

Type
dict[str, str] | novus.Localization

options

Thje options for the command.

Type
list[*novus.ApplicationCommandOption*]

default_member_permissions

The permissions required (by default) to run the command.

Type
novus.Permissions

dm_permission

Whether or not the command can be run in DMs.

Type
bool

nsfw

Whether or not the command is marked as NSFW.

Type
bool

class novus.**ApplicationCommand**(*, *state:* HTTPConnection, *data: payloads.ApplicationCommand*)

An application command object.

Note: This should not be user instantiated.

id

The ID of the command.

Type
int

type

The type of the command.

See also:

novus.ApplicationCommandType

Type
int

application_id

The ID of the application that the command is registered to.

Type
int

guild_id

The ID of the guild that the command is registered to.

Type
int | None

name

The name of the command.

Type
str

name_localizations

The command's name localizations.

Type
novus.Localization

description

The description of the command.

Type
str

description_localizations

The command's description localizations.

Type
novus.Localization

options

The options of the command.

Type
list[*novus.ApplicationCommandOption*]

default_member_permissions

The permissions required to run the command by default.

Type
novus.Permissions

dm_permission

Whether or not the command can be run in DMs.

Type
bool

nsfw

Whether or not the command is marked as NSFW.

Type
bool

version

The version of the command. An auto-updating snowflake.

Type
int

class novus.**ApplicationCommandChoice**(*name: str*, *value: str* | *int* | *float* | *None* = *None*, *, *name_localizations: LocType* = *None*)

A choice object for application commands.

Parameters

- **name** (*str*) – The name of the choice.
- **value** (*str | int | float*) – The value associated with the choice. If not provided, the given name will be used instead.

  Note: Large numbers (those in BIGINT range, including IDs) will be truncated by Discord - it's recommended that you use a string in their place.

- **name_localizations** (*dict[str, str] | novus.Localization*) – Localizations for the choice name.

name

The name of the choice.

Type
str

value

The value associated with the choice.

Type
str | int | float | None

name_localizations

Localizations for the command name.

Type
novus.Localization

class novus.**ApplicationCommandOption**(*name: str, description: str, type: int, *, name_localizations: LocType = MISSING, description_localizations: LocType = MISSING, required: bool = True, choices: list[ApplicationCommandChoice] = MISSING, options: list[ApplicationCommandOption] = MISSING, channel_types: list[int] = MISSING, min_value: int | float | None = MISSING, max_value: int | float | None = MISSING, min_length: int | None = MISSING, max_length: int | None = MISSING, autocomplete: bool = False*)

An option for use inside of an application command.

Parameters

- **name** (*str*) – The name of the option.
- **description** (*str*) – The description of the option.
- **type** (*int*) – The type of the option.

  See also:

  novus.ApplicationOptionType
- **name_localizations** (*dict[str, str] | novus.Localization*) – Localizations for the option's name.
- **description_localizations** (*dict[str, str] | novus.Localization*) – Localizations for the option's description.
- **required** (*bool*) – Whether or not the option is required.
- **choices** (*list[novus.ApplicationCommandChoice]*) – Choices for the option. Only valid if the option is neither a subcommand or a subcommand group.
- **options** (*list[novus.ApplicationCommandOption]*) – Options for the option. Only valid if the option is a subcommand or subcommand group.
- **channel_types** (*list[int]*) – The channel types that are supported by the option, if the option type is a channel.

  See also:

  novus.ChannelType
- **min_value** (*int*) – The minimum allowed value for the option if the option type is an integer or number.
- **max_value** (*int*) – The maximum allowed value for the option if the option type is an integer or number.
- **min_length** (*int*) – The minimum length of the input string if the option type is string.
- **max_length** (*int*) – The maximum length of the input string if the option type is string.
- **autocomplete** (*bool*) – Whether or not autocomplete interactions are enabled for this option.

name

The name of the option.

Type

str

description

The description of the option.

Type
str

type

The type of the option.

See also:

novus.ApplicationOptionType

Type
int

name_localizations

Localizations for the option's name.

Type
novus.Localization

description_localizations

Localizations for the option's description.

Type
novus.Localization

required

Whether or not the option is required.

Type
bool

choices

Choices for the option. Only valid if the option is neither a subcommand or a subcommand group.

Type
list[*novus.ApplicationCommandChoice*]

options

Options for the option. Only valid if the option is a subcommand or subcommand group.

Type
list[*novus.ApplicationCommandOption*]

channel_types

The channel types that are supported by the option, if the option type is a channel.

See also:

novus.ChannelType

Type
list[int]

min_value

The minimum allowed value for the option if the option type is an integer or number.

Type
int

`max_value`

The maximum allowed value for the option if the option type is an integer or number.

Type
int

`min_length`

The minimum length of the input string if the option type is string.

Type
int

`max_length`

The maximum length of the input string if the option type is string.

Type
int

`autocomplete`

Whether or not autocomplete interactions are enabled for this option.

Type
bool

1.1.5 Interactions

Models that relate to interactions.

`class novus.Interaction(*, state: HTTPConnection, data: payloads.Interaction)`

An interaction received from Discord.

Note: Novus silently ignores the possibility of `PING` interactions existing. It is simply easier that way.

`id`

The ID of the interaction.

Type
int

`application_id`

The ID of the application receiving the interaction. Usually your bot's ID.

Type
int

`type`

The type of the interaction.

See also:

novus.InteractionType

Type
int

data

The data associated with the interaction.

Type
ApplicationComandData | *ContextComandData* | *MessageComponentData* | *ModalSubmitData* | None

guild

The guild that the interaction was run in.

Type
novus.Guild | None

channel

The channel that the interaction was run in.

Type
novus.Channel

user

The user who ran the interaction.

Type
novus.GuildMember | *novus.User*

token

The token associated with the interaction response.

Type
str

message

The message associated with the interaction. Will only be set if the interaction was spawned from a message (ie from a component).

Type
novus.Message | None

app_permissions

The application's permissions within the channel where the interaction was called.

Type
novus.Permissions

locale

The user's locale.

Type
str

guild_locale

The locale of the guild where the interaction was run.

Type
str | None

async pong() → None

Send a pong interaction response.

async send(*content: str = MISSING, *, tts: bool = MISSING, embeds: list[Embed] = MISSING, allowed_mentions: AllowedMentions = MISSING, components: list[ActionRow] = MISSING, message_reference: Message = MISSING, stickers: list[Sticker] = MISSING, files: list[File] = MISSING, flags: MessageFlags = MISSING, ephemeral: bool = False*) → None

Send a message associated with the interaction response.

Parameters

- **content** (*str*) – The content that you want to have in the message
- **tts** (*bool*) – If you want the message to be sent with the TTS flag.
- **embeds** (*list[novus.Embed]*) – The embeds you want added to the message.
- **allowed_mentions** (novus.AllowedMentions) – The mentions you want parsed in the message.
- **components** (*list[novus.ActionRow]*) – A list of action rows to be added to the message.
- **message_reference** (novus.Message) – A reference to a message you want replied to.
- **stickers** (*list[novus.Sticker]*) – A list of stickers to add to the message.
- **files** (*list[novus.File]*) – A list of files to be sent with the message.
- **flags** (novus.MessageFlags) – The flags to be sent with the message.
- **ephemeral** (*bool*) – Whether the message should be sent so only the calling user can see it. This is ignored if this is the first message you're sending relating to this interaction and you've previously deferred.

async followup(*content: str = MISSING, *, tts: bool = MISSING, embeds: list[Embed] = MISSING, allowed_mentions: AllowedMentions = MISSING, components: list[ActionRow] = MISSING, message_reference: Message = MISSING, stickers: list[Sticker] = MISSING, files: list[File] = MISSING, flags: MessageFlags = MISSING, ephemeral: bool = False*) → *Message*

Send a message associated with the interaction response.

Parameters

- **content** (*str*) – The content that you want to have in the message
- **tts** (*bool*) – If you want the message to be sent with the TTS flag.
- **embeds** (*list[novus.Embed]*) – The embeds you want added to the message.
- **allowed_mentions** (novus.AllowedMentions) – The mentions you want parsed in the message.
- **components** (*list[novus.ActionRow]*) – A list of action rows to be added to the message.
- **message_reference** (novus.Message) – A reference to a message you want replied to.
- **stickers** (*list[novus.Sticker]*) – A list of stickers to add to the message.
- **files** (*list[novus.File]*) – A list of files to be sent with the message.
- **flags** (novus.MessageFlags) – The flags to be sent with the message.
- **ephemeral** (*bool*) – Whether the message should be sent so only the calling user can see it. This is ignored if this is the first message you're sending relating to this interaction and you've previously deferred.

async defer(*, *ephemeral: bool = False*) → None

Send a defer response.

async defer_update() → None

Send a defer update response.

async update(*, *content: str | None = MISSING*, *tts: bool = MISSING*, *embeds: list[Embed] | None = MISSING*, *allowed_mentions: AllowedMentions | None = MISSING*, *components: list[ActionRow] | None = MISSING*, *message_reference: Message | None = MISSING*, *stickers: list[Sticker] | None = MISSING*, *files: list[File] | None = MISSING*, *flags: flags.MessageFlags | None = MISSING*) → None

Send an update response.

Parameters

- **content** (*str*) – The content that you want to have in the message
- **tts** (*bool*) – If you want the message to be sent with the TTS flag.
- **embeds** (*list[novus.Embed]*) – The embeds you want added to the message.
- **allowed_mentions** (novus.AllowedMentions) – The mentions you want parsed in the message.
- **components** (*list[novus.ActionRow]*) – A list of action rows to be added to the message.
- **message_reference** (novus.Message) – A reference to a message you want replied to.
- **stickers** (*list[novus.Sticker]*) – A list of stickers to add to the message.
- **files** (*list[novus.File]*) – A list of files to be sent with the message.
- **flags** (novus.MessageFlags) – The flags to be sent with the message.

async send_autocomplete(*options: list[ApplicationCommandChoice]*) → None

Send an autocomplete response.

Parameters

options (*list[novus.ApplicationCommandChoice]*) – A list of choices to to populate the autocomplete with.

async send_modal(*, *title: str*, *custom_id: str*, *components: list[ActionRow]*) → None

Send a modal response. Not valid on modal interactions.

Parameters

- **title** (*str*) – The title to be shown in the modal.
- **custom_id** (*str*) – The custom ID of the modal.
- **components** (*list[novus.ActionRow]*) – The components shown in the modal.

async delete_original() → None

Delete the original message that is associated with the interaction.

async fetch_original() → *WebhookMessage*

Get the original message associated with the interaction.

async edit_original(**kwargs: Any*) → *WebhookMessage*

Edit the original message associated with the interaction.

class novus.**GuildInteraction**(*, *state:* HTTPConnection, *data: payloads.Interaction*)

A type-hinting version of interactions that have all guild attributes set properly for you.

async defer(*, *ephemeral: bool = False*) → None

Send a defer response.

async defer_update() → None

Send a defer update response.

async delete_original() → None

Delete the original message that is associated with the interaction.

async edit_original(**kwargs: Any*) → *WebhookMessage*

Edit the original message associated with the interaction.

async fetch_original() → *WebhookMessage*

Get the original message associated with the interaction.

async followup(*content: str = MISSING*, *, *tts: bool = MISSING*, *embeds: list[*Embed*] = MISSING*, *allowed_mentions:* AllowedMentions *= MISSING*, *components: list[*ActionRow*] = MISSING*, *message_reference:* Message *= MISSING*, *stickers: list[*Sticker*] = MISSING*, *files: list[*File*] = MISSING*, *flags:* MessageFlags *= MISSING*, *ephemeral: bool = False*) → *Message*

Send a message associated with the interaction response.

Parameters

- **content** (*str*) – The content that you want to have in the message
- **tts** (*bool*) – If you want the message to be sent with the TTS flag.
- **embeds** (*list[novus.Embed]*) – The embeds you want added to the message.
- **allowed_mentions** (novus.AllowedMentions) – The mentions you want parsed in the message.
- **components** (*list[novus.ActionRow]*) – A list of action rows to be added to the message.
- **message_reference** (novus.Message) – A reference to a message you want replied to.
- **stickers** (*list[novus.Sticker]*) – A list of stickers to add to the message.
- **files** (*list[novus.File]*) – A list of files to be sent with the message.
- **flags** (novus.MessageFlags) – The flags to be sent with the message.
- **ephemeral** (*bool*) – Whether the message should be sent so only the calling user can see it. This is ignored if this is the first message you're sending relating to this interaction and you've previously deferred.

async pong() → None

Send a pong interaction response.

async send(*content: str = MISSING*, *, *tts: bool = MISSING*, *embeds: list[*Embed*] = MISSING*, *allowed_mentions:* AllowedMentions *= MISSING*, *components: list[*ActionRow*] = MISSING*, *message_reference:* Message *= MISSING*, *stickers: list[*Sticker*] = MISSING*, *files: list[*File*] = MISSING*, *flags:* MessageFlags *= MISSING*, *ephemeral: bool = False*) → None

Send a message associated with the interaction response.

Parameters

- **content** (*str*) – The content that you want to have in the message
- **tts** (*bool*) – If you want the message to be sent with the TTS flag.
- **embeds** (*list[novus.Embed]*) – The embeds you want added to the message.
- **allowed_mentions** (novus.AllowedMentions) – The mentions you want parsed in the message.
- **components** (*list[novus.ActionRow]*) – A list of action rows to be added to the message.
- **message_reference** (novus.Message) – A reference to a message you want replied to.
- **stickers** (*list[novus.Sticker]*) – A list of stickers to add to the message.
- **files** (*list[novus.File]*) – A list of files to be sent with the message.
- **flags** (novus.MessageFlags) – The flags to be sent with the message.
- **ephemeral** (*bool*) – Whether the message should be sent so only the calling user can see it. This is ignored if this is the first message you're sending relating to this interaction and you've previously deferred.

async send_autocomplete(*options: list[*ApplicationCommandChoice*]*) → None

Send an autocomplete response.

Parameters

options (*list[novus.ApplicationCommandChoice]*) – A list of choices to to populate the autocomplete with.

async send_modal(*, *title: str*, *custom_id: str*, *components: list[*ActionRow*]*) → None

Send a modal response. Not valid on modal interactions.

Parameters

- **title** (*str*) – The title to be shown in the modal.
- **custom_id** (*str*) – The custom ID of the modal.
- **components** (*list[novus.ActionRow]*) – The components shown in the modal.

async update(*, *content: str | None = MISSING*, *tts: bool = MISSING*, *embeds: list[*Embed*] | None = MISSING*, *allowed_mentions:* AllowedMentions *| None = MISSING*, *components: list[*ActionRow*] | None = MISSING*, *message_reference:* Message *| None = MISSING*, *stickers: list[*Sticker*] | None = MISSING*, *files: list[*File*] | None = MISSING*, *flags: flags.MessageFlags | None = MISSING*) → None

Send an update response.

Parameters

- **content** (*str*) – The content that you want to have in the message
- **tts** (*bool*) – If you want the message to be sent with the TTS flag.
- **embeds** (*list[novus.Embed]*) – The embeds you want added to the message.
- **allowed_mentions** (novus.AllowedMentions) – The mentions you want parsed in the message.
- **components** (*list[novus.ActionRow]*) – A list of action rows to be added to the message.
- **message_reference** (novus.Message) – A reference to a message you want replied to.
- **stickers** (*list[novus.Sticker]*) – A list of stickers to add to the message.

- **files** (*list[novus.File]*) – A list of files to be sent with the message.
- **flags** (novus.MessageFlags) – The flags to be sent with the message.

class novus.**MessageInteraction**(*, *state:* HTTPConnection, *data: payloads.MessageInteraction*, *guild:* Guild | *None*)

An interaction attached to a message.

id

The ID of the interaction.

Type
int

type

The type of the interaction.

See also:

novus.InteractionType

Type
int

name

The name of the invoked application command.

Type
str

user

The user who invoked the interaction.

Type
novus.GuildMember | *novus.User*

class novus.**ContextComandData**(*, *parent:* Interaction, *data: payloads.ApplicationComandData*)

Data associated with a context command interaction.

id

The ID of the command that was run.

Type
int

name

The name of the command that was run.

Type
str

type

The type of the command that was run.

See also:

novus.ApplicationCommandType

Type
int

resolved

A resolved set of models for the interaction.

Type
InteractionResolved

options

A list of options that was run.

Type
list[*novus.ApplicationCommandOption*]

guild

The guild that the command was run in.

Type
novus.Guild | None

target

The entity that was targeted by the command.

Type
novus.Message | *novus.User* | *novus.GuildMember*

class novus.**ApplicationCommandData**(*, *parent:* Interaction, *data: payloads.ApplicationComandData*)

Data associated with an application command interaction.

id

The ID of the command that was run.

Type
int

name

The name of the command that was run.

Type
str

type

The type of the command that was run.

See also:

novus.ApplicationCommandType

Type
int

resolved

A resolved set of models for the interaction.

Type
InteractionResolved

options

A list of options that was run.

Type
list[*novus.ApplicationCommandOption*]

guild

The guild that the command was run in.

Type
novus.Guild | None

class novus.**MessageComponentData**(*, *parent:* Interaction, *data: payloads.MessageComponentData*)

Data associated with a message component interaction.

custom_id

Type
str

component

Type
novus.Component

values

Type
list[*novus.SelectOption*]

class novus.**ModalSubmitData**(*, *parent:* Interaction, *data: payloads.ModalSubmitData*)

Data associated with a modal interaction.

custom_id

Type
str

components

Type
list[*novus.ActionRow*]

class novus.**InteractionData**

Data associated with an interaction.

class novus.**InteractionOption**(*, *data: payloads.InteractionDataOption*)

Data from an option in an interaction.

class novus.**InteractionResolved**(*, *state:* HTTPConnection, *data: payloads.InteractionResolved* | *None*, *guild:* BaseGuild | *None* = *None*)

An object containing resolved models from interactions.

users

The users from the interaction.

Type
list[*novus.User*]

members

The members from the interaction.

Type
list[*novus.GuildMember*]

roles

The roles from the interaction.

Type
list[*novus.Role*]

channels

The channels from the interaction.

Type
list[*novus.Channel*]

messages

The messages from the interaction.

Type
list[*novus.Message*]

attachments

The attachments from the interaction.

Type
list[*novus.Attachment*]

1.2 ABCs

class novus.**Component**

Abstract base class for all Discord UI components.

class novus.**LayoutComponent**

Abstract base class for Discord UI components dealing with layout.

class novus.**InteractableComponent**

Abstract base class for Discord UI components that users can interact with.

1.3 Enums

class novus.**ApplicationCommandType**

CHAT_INPUT

MESSAGE

USER

class novus.**ApplicationOptionType**

ATTACHMENT

BOOLEAN

CHANNEL

INTEGER

MENIONABLE

NUMBER

ROLE

STRING

SUB_COMMAND

SUB_COMMAND_GROUP

USER

class novus.**AuditLogEventType**

APPLICATION_COMMAND_PERMISSION_UPDATE

AUTO_MODERATION_BLOCK_MESSAGE

AUTO_MODERATION_FLAG_TO_CHANNEL

AUTO_MODERATION_RULE_CREATE

AUTO_MODERATION_RULE_DELETE

AUTO_MODERATION_RULE_UPDATE

AUTO_MODERATION_USER_COMMUNICATION_DISABLED

BOT_ADD

CHANNEL_CREATE

CHANNEL_DELETE

CHANNEL_OVERWRITE_CREATE

CHANNEL_OVERWRITE_DELETE

CHANNEL_OVERWRITE_UPDATE

CHANNEL_UPDATE

EMOJI_CREATE

EMOJI_DELETE

EMOJI_UPDATE

GUILD_SCHEDULED_EVENT_CREATE

GUILD_SCHEDULED_EVENT_DELETE

GUILD_SCHEDULED_EVENT_UPDATE

GUILD_UPDATE

INTEGRATION_CREATE

INTEGRATION_DELETE

INTEGRATION_UPDATE

INVITE_CREATE

INVITE_DELETE

INVITE_UPDATE

MEMBER_BAN_ADD

MEMBER_BAN_REMOVE

MEMBER_DISCONNECT

MEMBER_KICK

MEMBER_MOVE

MEMBER_PRUNE

MEMBER_ROLE_UPDATE

MEMBER_UPDATE

MESSAGE_BULK_DELETE

MESSAGE_DELETE

MESSAGE_PIN

MESSAGE_UNPIN

ROLE_CREATE

ROLE_DELETE

ROLE_UPDATE

STAGE_INSTANCE_CREATE

STAGE_INSTANCE_DELETE

STAGE_INSTANCE_UPDATE

STICKER_CREATE

STICKER_DELETE

STICKER_UPDATE

THREAD_CREATE

THREAD_DELETE

THREAD_UPDATE

WEBHOOK_CREATE

WEBHOOK_DELETE

WEBHOOK_UPDATE

class novus.**AutoModerationActionType**

BLOCK_MESSAGE

SEND_ALERT_MESSAGE

TIMEOUT

class novus.**AutoModerationEventType**

MESSAGE_SEND

class novus.**AutoModerationKeywordPresetType**

PROFANITY

SEXUAL_CONTENT

SLURS

class novus.**AutoModerationTriggerType**

KEYWORD

KEYWORD_PRESET

MENTION_SPAM

SPAM

class novus.**ButtonStyle**

BLURPLE

CTA

DANGER

GRAY

GREEN

GREY

LINK

PRIMARY

RED

SECONDARY

SUCCESS

URL

class novus.**ChannelType**

ANNOUNCEMENT_THREAD

DM

GROUP_DM

GUILD_ANNOUNCEMENT

GUILD_CATEGORY

GUILD_DIRECTORY

GUILD_FORUM

GUILD_STAGE_VOICE

GUILD_TEXT

GUILD_VOICE

PRIVATE_THREAD

PUBLIC_THREAD

class novus.ComponentType

ACTION_ROW

BUTTON

CHANNEL_SELECT

MENTIONABLE_SELECT

ROLE_SELECT

STRING_SELECT

TEXT_INPUT

USER_SELECT

class novus.ContentFilterLevel

ALL_MEMBERS

DISABLED

MEMBERS_WITHOUT_ROLES

class novus.EventEntityType

EXTERNAL

STAGE_INSTANCE

VOICE

class novus.EventPrivacyLevel

GUILD_ONLY

class novus.EventStatus

ACTIVE

CANCELLED

COMPLETED

SCHEDULED

class novus.ForumLayout

GALLERY_VIEW

LIST_VIEW

NOT_SET

class novus.VideoQualityMode

AUTO

FULL

class novus.ForumSortOrder

CREATION_DATE

LATEST_ACTIVITY

class novus.GatewayOpcode

DISPATCH

GUILD_SYNC

HEARTBEAT

HEARTBEAT_ACK

HELLO

IDENTIFY

INVALIDATE_SESSION

PRESENCE

RECONNECT

REQUEST_MEMBERS

RESUME

VOICE_PING

VOICE_STATE

class novus.InteractionResponseType

APPLICATION_COMMAND_AUTOCOMPLETE_RESULT

CHANNEL_MESSAGE_WITH_SOURCE

DEFERRED_CHANNEL_MESSAGE_WITH_SOURCE

DEFERRED_UPDATE_MESSAGE

MODAL

PONG

UPDATE_MESSAGE

class novus.InteractionType

APPLICATION_COMMAND

AUTOCOMPLETE

MESSAGE_COMPONENT

MODAL_SUBMIT

PING

class novus.MFALevel

ELEVATED

NONE

class novus.MessageType

AUTO_MODERATION_ACTION

CALL

CHANNEL_FOLLOW_ADD

CHANNEL_ICON_CHANGE

CHANNEL_NAME_CHANGE

CHANNEL_PINNED_MESSAGE

CHAT_INPUT_COMMAND

CONTEXT_MENU_COMMAND

DEFAULT

GUILD_APPLICATION_PREMIUM_SUBSCRIPTION

GUILD_BOOST

GUILD_BOOST_TIER_1

GUILD_BOOST_TIER_2

GUILD_BOOST_TIER_3

GUILD_DISCOVERY_DISQUALIFIED

GUILD_DISCOVERY_GRACE_PERIOD_FINAL_WARNING

GUILD_DISCOVERY_GRACE_PERIOD_INITIAL_WARNING

GUILD_DISCOVERY_REQUALIFIED

GUILD_INVITE_REMINDER

INTERACTION_PREMIUM_UPSELL

RECIPIENT_ADD

RECIPIENT_REMOVE

REPLY

ROLE_SUBSCRIPTION_PURCHASE

THREAD_CREATED

THREAD_STARTER_MESSAGE

USER_JOIN

class novus.**NSFWLevel**

AGE_RESTRICTED

DEFAULT

EXPLICIT

SAFE

class novus.**NotificationLevel**

ALL_MESSAGES

ONLY_MENTIONS

class novus.**PermissionOverwriteType**

MEMBER

ROLE

class novus.**PremiumTier**

NONE

TIER_1

TIER_2

TIER_3

class novus.**StickerFormat**

APNG

GIF

LOTTIE

PNG

class novus.StickerType

GUILD

STANDARD

class novus.TextInputStyle

LONG

PARAGRAPH

SHORT

class novus.TimestampFormat

LONG_DATE

LONG_DATETIME

LONG_TIME

RELATIVE

SHORT_DATE

SHORT_DATETIME

SHORT_TIME

class novus.UserPremiumType

NITRO

NITRO_BASIC

NITRO_CLASSIC

NONE

class novus.VerificationLevel

HIGH

LOW

MEDIUM

NONE

VERY_HIGH

class novus.Status

AFK

DND

DO_NOT_DISTURB

IDLE

INVISIBLE

OFFLINE

ONLINE

class novus.ActivityType

COMPETING

CUSTOM

GAME

LISTENING

STREAMING

WATCHING

1.4 Flags

class novus.Intents

guilds

guild_members

guild_moderation

guild_emojis_and_stickers

guild_integrations

guild_webhooks

guild_invites

guild_voice_states

guild_presences

guild_messages

guild_message_reactions

guild_message_typing

direct_messages

direct_message_reactions

direct_message_typing

message_content

guild_scheduled_events

auto_moderation_configuration

auto_moderation_execution

class novus.**ApplicationFlags**

The public flags for an application.

application_command_badge

Indicates if an app has registered global application commands.

embedded

Indicates if an app is embedded within the Discord client (currently unavailable publicly).

gateway_guild_members

Intent required for bots in 100 or more servers to receive member-related events like `GUILD_MEMBER_ADD`.

gateway_guild_members_limited

Intent required for bots in under 100 servers to receive member-related events like `GUILD_MEMBER_ADD`.

gateway_message_content

Intent required for bots in 100 or more servers to receive message content.

gateway_message_content_limited

Intent required for bots in under 100 servers to receive message content.

gateway_presence

Intent required for bots in 100 or more servers to receive `PRESENCE_UPDATE` events.

gateway_presence_limited

Intent required for bots in under 100 servers to receive `PRESENCE_UPDATE` events.

verification_pending_guild_limit

Indicates unusual growth of an app that prevents verification.

class novus.**MessageFlags**

suppress_join_notifications

suppress_premium_subscriptions

suppress_guild_reminder_notifications

suppress_join_notification_replies

class novus.**SystemChannelFlags**

Flags for a system channel within a guild.

suppress_join_notifications

suppress_premium_subscriptions

suppress_guild_reminder_notifications

suppress_join_notification_replies

class novus.**Permissions**

A permission set from Discord's API.

create_instant_invite

kick_members

ban_members

administrator

manage_channels

manage_guild

add_reactions

view_audit_log

priority_speaker

stream

view_channel

send_messages

send_tts_messages

manage_messages

embed_links

attach_files

read_message_history

mention_everyone

use_external_emojis

view_guild_insights

connect

speak

mute_members

deafen_members

move_members

use_vad

change_nickname

manage_nicknames

manage_roles

manage_webhooks

manage_emojis_and_stickers

use_application_commands

request_to_speak

manage_events

manage_threads

create_public_threads

create_private_threads

use_external_stickers

send_messages_in_threads

use_embedded_activites

moderate_members

class novus.UserFlags

staff

partner

hypesquad

bug_hunter_level_1

hypesquad_house_bravery

hypesquad_house_brilliance

hypesquad_house_balance

premium_early_supporter

team_pseudo_user

bug_hunter_level_2

verified_bot

verified_developer

certified_moderator

bot_http_interactions

active_developer

class novus.AuditLogEntryType

guild_update

channel_create

channel_update

channel_delete

channel_overwrite_create

channel_overwrite_update

channel_overwrite_delete

member_kick

member_prune

member_ban_add

member_ban_remove

member_update

member_role_update

member_move

member_disconnect

bot_add

role_create

role_update

role_delete

invite_create

invite_update

invite_delete

webhook_create

webhook_update

webhook_delete

emoji_create

emoji_update

emoji_delete

message_delete

message_bulk_delete

message_pin

message_unpin

integration_create

integration_update

integration_delete

stage_instance_create

stage_instance_update

stage_instance_delete

sticker_create

sticker_update

sticker_delete

guild_scheduled_event_create

guild_scheduled_event_update

guild_scheduled_event_delete

thread_create

thread_update

thread_delete

application_command_permission_update

auto_moderation_rule_create

auto_moderation_rule_update

auto_moderation_rule_delete

auto_moderation_block_message

auto_moderation_flag_to_channel

auto_moderation_user_communication_disabled

1.5 Utils

novus.utils.**try_snowflake**(*given: list[str] | list[int]*) → list[int]

novus.utils.**try_snowflake**(*given: str | int*) → int

novus.utils.**try_snowflake**(*given: None*) → None

Try and turn a given string into a snowflake, returning `None` if the given value is `None`.

Parameters

given (*str | int | list[str] | list[int] | None*) – The given "snowflake".

Returns

If the given value was castable to an int, that value. Otherwise, `None`.

Return type

int | list[int] | None

novus.utils.**try_id**(*given: Literal[None]*) → None

novus.utils.**try_id**(*given: int | Snowflake | str*) → int

`novus.utils.try_id`(*given: list[int] | list[Snowflake] | list[str]*) → list[int]

Get the ID from the given object if it is a snowflake; return the object unchanged otherwise.

Parameters
given (*int | novus.abc.Snowflake | str | None*) – The object you want an ID from.

Returns
The ID from the model if the object is one already (or contains one); `None` otherwise.

Return type
int | None

`novus.utils.try_object`(*given: int | str | Snowflake*) → Snowflake

`novus.utils.try_object`(*given: None*) → None

Wrap the given ID in a `novus.Object`, or return `None` if the item is not an int (or if the object is a snowflake already, return it unchanged).

Parameters
given (*int | novus.abc.Snowflake | None*) – The object you want to wrap.

Returns
The wrapped object.

Return type
Snowflake | None

`novus.utils.walk_components`(*rows: list[*ActionRow*] | None*) → Generator[*InteractableComponent*, None, None]

1.6 API

class `novus.api._route.Route`(*method: str, resource: str, **kwargs: str | int*)

A route to access the API with.

Parameters

- **method** (*str*) – The HTTP method to use.
- **resource** (*str*) – The resource that you want to access.
- ****kwargs** – Any additional kwargs to be formatted into the resource.

method

The HTTP method to use.

Type
str

resource

The resource that you want to access.

Type
str

path

The resource with the kwargs mapped onto it.

Type
str

url

The fully built route path.

Type
str

kwargs

Any additional kwargs to be formatted into the resource.

Type
dict

class novus.api.**HTTPConnection**(*token: str | None = None, client_id: str | None = None, client_secret: str | None = None*)

A wrapper around the API for HTTP handling.

As well as this class (and it's subclasses), each applicable model should have an API mixin.

The mixins are intended to be user facing, so have all of the parameters added.

The HTTP connection classes have positional parameters for the resource params; a keyword only parameter for the reason (if applicable) and any GET parameters; and the kwargs are added to the JSON body. For the most part, kwargs get fixed up into a valid payload. Known kwargs are assumed to be the correct type (things like a `channel` kwarg will be fixed into a `channel_id` via the `.id` attribute; a flags class will get its `.value` attribute and cast to a string for the payload, etc). Any unknown kwargs will be added as is to the payload, which allows you to add any additional arguments should Discord update before the library.

Parameters
token (*str | None*) – The token to use.

application_role_connection_metadata

Type
ApplicationRoleHTTPConnection

audit_log

Type
AuditLogHTTPConnection

auto_moderation

Type
AutoModerationHTTPConnection

channel

Type
ChannelHTTPConnection

emoji

Type
EmojiHTTPConnection

guild

Type
GuildHTTPConnection

guild_scheduled_event

Type
GuildEventHTTPConnection

guild_template

Type
GuildTemplateHTTPConnection

interaction

Type
InteractionHTTPConnection

invite

Type
InviteHTTPConnection

oauth2

Type
Oauth2HTTPConnection

stage_instance

Type
StageHTTPConnection

sticker

Type
StickerHTTPConnection

user

Type
UserHTTPConnection

voice

Type
VoiceHTTPConnection

webhook

Type
WebhookHTTPConnection

class novus.api.**APIIterator**(*method: Any*, *before: AnySnowflake*, *after: AnySnowflake*, *limit: int | None*, *method_limit: int*)

An async iterator class for Discord API methods that return multiple - but limited - items in their return.

async flatten() → list[T]

Flatten the entire item from its original API generator into a list.

class novus.api.application_role_connection_metadata.**ApplicationRoleHTTPConnection**(*parent: HTTPConnection*)

async get_application_role_records(*application_id: int*) → list[dict]

Get the application role connection metadata objects for the given application.

async update_application_role_records(*application_id: int*, *records: list[Any]*) → list[dict]

Get one guild emoji.

class novus.api.audit_log.**AuditLogHTTPConnection**(*parent:* HTTPConnection)

async get_guild_audit_log(*guild_id: int*, *, *user_id: int | None = None*, *action_type: int | None = None*, *before: int | None = None*, *after: int | None = None*, *limit: int = 50*) → *AuditLog*

Get guild audit logs.

class novus.api.auto_moderation.**AutoModerationHTTPConnection**(*parent:* HTTPConnection)

async list_auto_moderation_rules_for_guild(*guild_id: int*) → list[*AutoModerationRule*]

Get a list of automoderator rules for the guild.

async get_auto_moderation_rule(*guild_id: int*, *rule_id: int*) → *AutoModerationRule*

Get a specific automoderator rule for the guild.

async create_auto_moderation_rule(*guild_id: int*, *, *reason: str | None = None*, ***kwargs: dict[str, Any]*) → *AutoModerationRule*

Create an automoderation rule.

async modify_auto_moderation_rule(*guild_id: int*, *rule_id: int*, *, *reason: str | None = None*, ***kwargs: dict[str, Any]*) → *AutoModerationRule*

Edit an automoderation rule.

async delete_auto_moderation_rule(*guild_id: int*, *rule_id: int*, *, *reason: str | None = None*) → None

Delete an automoderation rule.

class novus.api.channel.**ChannelHTTPConnection**(*parent:* HTTPConnection)

async get_channel(*channel_id: int*) → *Channel*

Get a channel object by its ID.

async modify_channel(*channel_id: int*, *, *reason: str | None = None*, ***kwargs: dict[str, Any]*) → *Channel*

Modify a channel's settings.

async delete_channel(*channel_id: int*, *, *reason: str | None = None*) → None

Delete a channel (or close a DM channel).

async get_channel_messages(*channel_id: int*, *, *around: int = MISSING*, *before: int = MISSING*, *after: int = MISSING*, *limit: int = MISSING*) → list[*Message*]

Get a channel object by its ID.

async get_channel_message(*channel_id: int*, *message_id: int*) → *Message*

Get a specific message inside of a channel.

async create_message(*channel_id: int*, ***kwargs: dict[str, Any]*) → *Message*

Create a new message inside of a channel.

async crosspost_message(*channel_id: int*, *message_id: int*) → *Message*

Crosspost a message.

async create_reaction(*channel_id: int*, *message_id: int*, *emoji: str* | PartialEmoji) → None

Create a reaction on a message.

async delete_own_reaction(*channel_id: int*, *message_id: int*, *emoji: str* | PartialEmoji) → None

Remove your own reaction to a message.

async delete_user_reaction(*channel_id: int*, *message_id: int*, *emoji: str* | PartialEmoji, *user_id: int*) → None

Remove another user's reaction from a message.

async get_reactions(*channel_id: int*, *message_id: int*, *emoji: str* | PartialEmoji) → list[*User*]

Get a list of users who reacted to a message with a particular emoji.

async delete_all_reactions(*channel_id: int*, *message_id: int*) → None

Remove all reactions from a message.

async delete_all_reactions_for_emoji(*channel_id: int*, *message_id: int*, *emoji: str* | PartialEmoji) → None

Remove all reactions for the given emoji on a message.

async edit_message(*channel_id: int*, *message_id: int*, ***kwargs: dict[str, Any]*) → *Message*

Edit an existing message.

async delete_message(*channel_id: int*, *message_id: int*, **, reason: str* | *None* = *None*) → None

Delete an existing message.

async bulk_delete_messages(*channel_id: int*, **, reason: str* | *None* = *None*, *message_ids: list[int]*) → None

Delete multiple messages.

async edit_channel_permissions(*channel_id: int*, *overwrite_id: int*, **, reason: str* | *None* = *None*, *allow: Permissions* = *Permissions(0b0)*, *deny: Permissions* = *Permissions(0b0)*, *type: Type[*User*]* | *Type[*GuildMember*]* | *Type[*Role*]*) → None

Update the permissions for a given item in the channel.

async get_channel_invites(*channel_id: int*) → list[*Invite*]

Get the invites for a channel.

async create_channel_invite(*channel_id: int*, **, reason: str* | *None* = *None*, ***kwargs: dict[str, Any]*) → *Invite*

Create an invite for the channel.

async delete_channel_permission(*channel_id: int*, *overwrite_id: int*, **, reason: str* | *None* = *None*) → None

Delete channel permission.

async follow_announcement_channel(*channel_id: int*, *webhook_channel_id: int*) → int

Follow an announcement channel. Returns the created webhook ID.

async trigger_typing_indicator(*channel_id: int*) → None

Trigger the typing indicator for the specified channel.

async get_pinned_messages(*channel_id: int*) → list[*Message*]

Return all of the pinned messages in the channel.

async pin_message(*channel_id: int*, *message_id: int*, **, reason: str | None = None*) → None

Pin a message.

async unpin_message(*channel_id: int*, *message_id: int*, **, reason: str | None = None*) → None

Unpin a message.

async start_thread_from_message(*channel_id: int*, *message_id: int*, **, reason: str | None = None*, ***kwargs: dict[str, Any]*) → *Channel*

Create a thread from a message.

async start_thread_without_message(*channel_id: int*, **, reason: str | None = None*, ***kwargs: dict[str, Any]*) → *Channel*

Create a thread not connected to an existing message.

async start_thread_in_forum_channel(*channel_id: int*, **, reason: str | None = None*, ***kwargs: dict[str, Any]*) → *Channel*

Create a thread inside of a forum channel.

async add_thread_member(*channel_id: int*, *user_id: int | Literal['@me']*) → None

Add a member to a thread.

async remove_thread_member(*channel_id: int*, *user_id: int | Literal['@me']*) → None

Remove another user from a thread.

async get_thread_member(*channel_id: int*, *user_id: int*, **, with_member: bool = False*) → *ThreadMember*

Get a member in a thread.

async list_thread_members(*channel_id: int*, **, with_member: bool = False*, *after: int = MISSING*, *limit: int = 100*) → list[*ThreadMember*]

Get an array of thread members.

async list_public_archived_threads(*channel_id: int*, **, before: datetime = MISSING*, *limit: int = MISSING*) → list[*Channel*]

Get the archived threads inthe channel that are public.

async list_private_archived_threads(*channel_id: int*, **, before: datetime = MISSING*, *limit: int = MISSING*) → list[*Channel*]

Get the archived threads inthe channel that are private.

async list_joined_private_archived_threads(*channel_id: int*, **, before: datetime = MISSING*, *limit: int = MISSING*) → list[*Channel*]

Get the archived threads inthe channel that are private.

class novus.api.emoji.**EmojiHTTPConnection**(*parent:* HTTPConnection)

async list_guild_emojis(*guild_id: int*) → list[*Emoji*]

Get all guild emojis.

async get_emoji(*guild_id: int*, *emoji_id: int*) → *Emoji*

Get one guild emoji.

async create_guild_emoji(*guild_id: int*, */*, **, reason: str | None = None*, ***kwargs: Any*) → *Emoji*

Create a guild emoji.

async modify_guild_emoji(*guild_id: int, emoji_id: int, /, *, reason: str | None = None, **kwargs: Any*) → *Emoji*

Modify guild emoji.

async delete_guild_emoji(*guild_id: int, emoji_id: int, /, *, reason: str | None = None*) → None

Delete guild emoji.

class novus.api.guild.**GuildHTTPConnection**(*parent:* HTTPConnection)

async create_guild(***kwargs: Any*) → *Guild*

Create a guild.

async get_guild(*guild_id: int, /, with_counts: bool = False*) → *Guild*

Get a guild.

async get_guild_preview(*guild_id: int, /*) → *GuildPreview*

Get a guild preview.

async modify_guild(*guild_id: int, /, *, reason: str | None = None, **kwargs: Any*) → *Guild*

Edit a guild.

async delete_guild(*guild_id: int, /*) → None

Delete a guild.

async get_guild_channels(*guild_id: int, /*) → list[*Channel*]

Get guild channels.

async create_guild_channel(*guild_id: int, /, *, reason: str | None = None, **kwargs: Any*) → *Channel*

Create a guild channel.

async get_active_guild_threads(*guild_id: int*) → list[*Channel*]

Get the threads from the guild.

async get_guild_member(*guild_id: int, member_id: int*) → *GuildMember*

Get a guild member.

async get_guild_members(*guild_id: int, *, limit: int = 1, after: int = 0*) → list[*GuildMember*]

Get a guild member.

async search_guild_members(*guild_id: int, *, query: str, limit: int = 1*) → list[*GuildMember*]

Get a guild member.

async add_guild_member(*guild_id: int, user_id: int, *, access_token: str, **kwargs: Any*) → *GuildMember* | None

Add a member to the guild. Only works if you have a valid Oauth2 access token with the guild.join scope.

async modify_guild_member(*guild_id: int, user_id: int, /, *, reason: str | None = None, **kwargs: Any*) → *GuildMember*

Update a guild member.

async add_guild_member_role(*guild_id: int, user_id: int, role_id: int, /, *, reason: str | None = None*) → None

Add a role to a guild member.

async remove_guild_member_role(*guild_id: int, user_id: int, role_id: int, /, *, reason: str | None = None*) → None

Remove a role from a guild member.

async remove_guild_member(*guild_id: int*, *user_id: int*, /, *, *reason: str | None = None*) → None

Remove a member from the guild.

async get_guild_bans(*guild_id: int*, /, *, *limit: int | None = None*, *before: int | None = None*, *after: int | None = None*) → list[*GuildBan*]

Get the bans from a guild.

async get_guild_ban(*guild_id: int*, *user_id: int*, /) → *GuildBan*

Get a ban for a particular member.

async create_guild_ban(*guild_id: int*, *user_id: int*, /, *, *reason: str | None = None*, ***kwargs: Any*) → None

Ban a user.

async remove_guild_ban(*guild_id: int*, *user_id: int*, /, *, *reason: str | None = None*) → None

Unban a user.

async get_guild_roles(*guild_id: int*) → list[*Role*]

List the roles for the guild.

async create_guild_role(*guild_id: int*, *, *reason: str | None = None*, ***kwargs: Any*) → *Role*

Create a role in the guild.

async modify_guild_role(*guild_id: int*, *role_id: int*, /, *, *reason: str | None = None*, ***kwargs: Any*) → *Role*

Edit a guild role.

async delete_guild_role(*guild_id: int*, *role_id: int*, /, *, *reason: str | None = None*) → None

Delete a guild role.

async get_guild_invites(*guild_id: int*, /) → list[*Invite*]

Get the invites for a guild.

class novus.api.guild_scheduled_event.**GuildEventHTTPConnection**(*parent: HTTPConnection*)

async list_scheduled_events_for_guild(*guild_id: int*, *, *with_user_count: bool = MISSING*) → list[*ScheduledEvent*]

Get the scheduled events for the guild.

async create_guild_scheduled_event(*guild_id: int*, *, *reason: str | None*, ***kwargs: Any*) → *ScheduledEvent*

Get the scheduled events for the guild.

async get_guild_scheduled_event(*guild_id: int*, *event_id: int*, *, *with_user_count: bool = MISSING*) → *ScheduledEvent*

Get a particular guild scheduled event.

async modify_guild_scheduled_event(*guild_id: int*, *event_id: int*, *, *reason: str | None*, ***kwargs: Any*) → *ScheduledEvent*

Modify a particular scheduled event.

async delete_guild_scheduled_event(*guild_id: int*, *event_id: int*) → None

Delete a given guild scheduled event.

async get_guild_scheduled_event_users(*guild_id: int*, *event_id: int*, *, *limit: int = 100*, *with_member: bool = False*, *before: int | None = None*, *after: int | None = None*) → list[*User* | *GuildMember*]

Get the event users for a guild scheduled event.

class novus.api.guild_template.**GuildTemplateHTTPConnection**(*parent:* HTTPConnection)

class novus.api.invite.**InviteHTTPConnection**(*parent:* HTTPConnection)

async get_invite(*invite_code: str, *, with_counts: bool | None = None, with_expiration: bool | None = None, guild_scheduled_event_id: int | None = None*) → *Invite*

Get an invite from the API.

async delete_invite(*invite_code: str, *, reason: str | None = None*) → *Invite*

Get an invite from the API.

class novus.api.stage_instance.**StageHTTPConnection**(*parent:* HTTPConnection)

async create_stage_instance(*, *reason: str | None = None, **kwargs: dict[str, Any]*) → *StageInstance*

Create a stage instance aassociated with a stage channel.

async get_stage_instance(*channel_id: int*) → *StageInstance*

Get a stage instance.

async modify_stage_instance(*channel_id: int, *, reason: str | None = None, **kwargs: dict[str, Any]*) → *StageInstance*

Update a stage instance.

async delete_stage_instance(*channel_id: int, *, reason: str | None = None*) → None

Delete a stage instance.

class novus.api.sticker.**StickerHTTPConnection**(*parent:* HTTPConnection)

async get_sticker(*sticker_id: int*) → *Sticker*

Get a sticker from the API via its ID.

This route shouldn't really be used, but is included for completeness sake.

async list_guild_stickers(*guild_id: int*) → list[*Sticker*]

List the stickers within a guild.

async get_guild_sticker(*guild_id: int, sticker_id: int*) → *Sticker*

Get a sticker form a guild.

async create_guild_sticker(*guild_id: int, *, reason: str | None = None, **kwargs: dict[str, Any]*) → *Sticker*

Create a sticker within a guild.

async modify_guild_sticker(*guild_id: int, sticker_id: int, *, reason: str | None = None, **kwargs: dict[str, Any]*) → *Sticker*

Edit a sticker within a guild.

async delete_guild_sticker(*guild_id: int, sticker_id: int, *, reason: str*) → None

Delete a guild sticker.

class novus.api.user.**UserHTTPConnection**(*parent:* HTTPConnection)

async get_current_user() → *User*

Get the current user from the API.

async get_user(*user_id: int*) → *User*

Get a user from the API.

async modify_current_user(***kwargs: Any*) → *User*

Edit the current user.

async get_current_user_guilds(**, before: int | None = None, after: int | None = None, limit: int = 200*) → list[OauthGuild]

Get a list of guilds from the oauth user.

async get_current_user_guild_member(*guild_id: int*) → *GuildMember*

Get the current user's guild member object.

async leave_guild(*guild_id: int*) → None

Leave a given guild.

async create_dm(*recipient_id: int*) → *Channel*

Create a DM with a user.

async get_user_connections() → Any

Get a list of connection objects from the API.

async get_user_application_role_connection(*application_id: int*) → Any

Get the current role connection for an application.

async update_user_application_role_connection(*application_id: int*) → Any

Update a role connection for an application.

class novus.api.voice.**VoiceHTTPConnection**(*parent:* HTTPConnection)

class novus.api.webhook.**WebhookHTTPConnection**(*parent:* HTTPConnection)

async create_webhook(*channel_id: int, *, reason: str | None = None, **kwargs: dict[str, Any]*) → *Webhook*

Create a webhook instance.

async get_channel_webhooks(*channel_id: int*) → list[*Webhook*]

Get all webhooks associated with a channel.

async get_guild_webhooks(*guild_id: int*) → list[*Webhook*]

Get all webhooks associated with a guild.

async get_webhook(*webhook_id: int, *, token: str | None = None*) → *Webhook*

Get a webhook via its ID.

async modify_webhook(*webhook_id: int, *, reason: str | None = None, token: str | None = None, **kwargs: dict[str, Any]*) → *Webhook*

Modify a webhook.

async delete_webhook(*webhook_id: int, *, reason: str | None = None, token: str | None = None*) → None

Delete a webhook.

async execute_webhook(*webhook_id: int, token: str, *, wait: Literal[True] = False, thread_id: int | None = None, **kwargs: dict[str, Any]*) → *WebhookMessage*

async execute_webhook(*webhook_id: int, token: str, *, wait: Literal[False] = False, thread_id: int | None = None, **kwargs: dict[str, Any]*) → None

Create a webhook message.

async get_webhook_message(*webhook_id: int*, *token: str*, *message_id: int*, **, thread_id: int | None = None*) → *WebhookMessage*

Get a webhook message.

async edit_webhook_message(*webhook_id: int*, *token: str*, *message_id: int*, **, thread_id: int | None = None*, ***kwargs: dict[str, Any]*) → *WebhookMessage*

Edit a webhook message.

async delete_webhook_message(*webhook_id: int*, *token: str*, *message_id: int*, **, thread_id: int | None = None*) → None

Delete a webhook message.

CHAPTER

TWO

CLIENT

The Novus client ext is the easiest way to streamline your bot creation. The client ext automates aspects of running your bot and handling command execution, while still allowing you flexibility in how your bot runs and what it processes.

```
class Pings(client.Plugin):

    @client.command()
    async def pingme(self, ctx: novus.types.CommandI):
        """Ping yourself."""
        await ctx.send(ctx.user.mention)

    @client.command(
        options=[
            novus.ApplicationCommandOption(
                name="user",
                type=novus.ApplicationOptionType.USER,
                description="The user you want to ping.",
            )
        ]
    )
    async def ping(self, ctx: novus.types.CommandI, user: novus.User):
        """Ping someone."""
        await ctx.send(user.mention)
```

2.1 Client Quickstart

todo

2.2 Command-Line Interface

Novus implements a few menus via CLI.

2.2.1 novus run

Run a bot instance with config either passed in as CLI arguments, or from a config file (either found in the current directory, or passed as a CLI arg).

2.2.2 novus run-webserver

Run a bot webserver instance (interactions only) with config either passed in as CLI arguments, or from a config file (either found in the current directory, or passed as a CLI arg).

2.2.3 novus config-dump

Output a config (with arguments optionally supplied via CLI). This can be piped into a file (`novus config-dump --token test > config.yaml`).

2.2.4 novus new-plugin

Output a new plugin (with arguments optionally supplied via CLI). This can be piped into a file (`novus new-plugin Test > test.py`).

2.3 API Reference

2.3.1 Bots

class novus.ext.client.**Client**(*config:* Config, *, *load_plugins:* *bool* = *False*)

A gateway and API connection into Discord.

config

The config file for the bot.

Type
novus.ext.client.Config

state

The connection for the bot.

Type
novus.api.HTTPConnection

plugins

A list of plugins that have been added to the bot. This does *not* mean that the plugin has been loaded necessarily, just that it has been added.

Type
set[*novus.ext.client.Plugin*]

me

The user associated with the bot. Will be `None` if the bot has not connected to the gateway.

Type
novus.User | None

commands

A list of commands that have been loaded into the bot.

Type
set[*novus.ext.client.Command*]

get_guild(*id: AnySnowflake*) → n.Guild | None

Get a guild from the bot's internal cache.

Parameters
id (*int | str*) – The ID of the guild that you want to get.

Returns
The object from the cache.

Return type
novus.Guild | None

get_user(*id: AnySnowflake*) → n.User | None

Get a user from the bot's internal cache.

Parameters
id (*int | str*) – The ID of the user that you want to get.

Returns
The object from the cache.

Return type
novus.User | None

get_channel(*id: AnySnowflake*, *or_partial: Literal[True]*) → *Channel*

get_channel(*id: AnySnowflake*, *or_partial: Literal[False]*) → *Channel* | None

Get a channel from the bot's internal cache.

Parameters

- **id** (*int | str*) – The ID of the channel that you want to get.
- **or_partial** (*bool*) – If set to `True`, the bot will return a partial object if a channel was not available.

Returns
The object from the cache.

Return type
novus.Channel | None

async change_presence(*activities: list[*Activity*]* | *None* = *None*, *status: str* = *'online'*) → None

Change the presence of the client.

Parameters

- **activities** (*list[novus.Activity]*) – The activities that you want to set the client as displaying.
- **status** (*str*) – The status of the client.

  See also:

  novus.Status

async wait_until_ready() → None

Wait until all of the shards for the bot have received the "ready" message from the gateway.

add_command(*command:* Command) → None

Add a command to the bot's internal cache.

Parameters
command (novus.ext.client.Command) – The command you want to add.

Raises
NameError – You have tried to add a command with a duplicate name (and guild status) as an already cached command.

get_command(*name: str*, *guild_id: int | None = None*) → *Command* | *CommandGroup* | None

Get a command that's been loaded into the bot from the cache.

Parameters

- **name** (*str*) – The name of the command thaty ou want to get.
- **guild_id** (*int | None*) – The ID of the guild where the command is registered.

Returns
The command from the command cache.

Return type
novus.ext.client.Command | *novus.ext.client.CommandGroup*

get_plugin(*name: str*) → *Plugin* | None

Get a loaded plugin from the bot's internal cache.

Parameters
name (*str*) – The name of the plugin, case sensitive.

Returns
The loaded plugin, if one could be found.

Return type
novus.ext.client.Plugin | None

add_plugin(*plugin: Type[*Plugin*]*, **, load: bool = False*) → None

Load a plugin into the bot.

Parameters
plugin (*Type[*novus.ext.client.Plugin*]*) – The plugin that you want to add to the bot.

async load_plugins() → None

Load all of the plugins that have been added to the bot.

remove_command(*command:* Command) → None

Remove a command from the bot's internal cache.

Parameters
command (novus.ext.client.Command) – The command you want to remove.

Raises
NameError – You have tried to remove a command that has not been loaded or is not in the cache.

remove_plugin(*plugin: Type[*Plugin*]*) → None

Remove a plugin from the bot.

Parameters
plugin (*Type[*novus.ext.client.Plugin*]*) – The plugin type that you want to remove.

Raises
TypeError – There are no plugins with that plugin type loaded.

add_plugin_file(**plugin: str, load: bool = False, reload_import: bool = False*) → None

Add a plugin via its filename:ClassName pair.

Parameters
plugin (*str*) – The plugin reference.

Raises
TypeError – No plugin could be loaded from the given reference.

remove_plugin_file(**plugin: str*) → None

Remove a plugin via its filename:ClassName pair.

Parameters
plugin (*str*) – The plugin reference.

Raises
TypeError – No plugin could be loaded from the given reference.

dispatch(*event_name: str, *args: Any, **kwargs: Any*) → None

Dispatch an event to all loaded plugins.

async sync_commands(**, create: bool = True, edit: bool = True, delete: bool = True*) → None

Get all commands from Discord. Determine if they all already exist. If not, PUT them there. If so, save command IDs. This command is required to run to be able to dispatch command events properly.

async connect(*check_concurrency: bool = False, sleep: bool = False*) → None

Connect the bot to the gateway, running the connection in the background.

async connect_webserver(**, port: int = 8000*) → BaseSite

Open a webserver to receive interactions from Discord.

Parameters
port (*int*) – The port to open the server on.

async close() → None

Close the gateway and session connection.

async run(**, sync: bool = True*) → None

Connect the bot to the gateway, keeping the bot's connection to the websocket alive.

Parameters
sync (*bool*) – Whether or not to sync the bot's commands to Discord.

async run_webserver(**, port: int = 8000, sync: bool = True*) → None

Run a webserver for the bot so as to receive interactions from Discord.

Parameters

- **port** (*int*) – The port that you want to open the webserver on.
- **sync** (*bool*) – Whether or not to sync the bot's commands to Discord.

class novus.ext.client.**Config**(**, token: str = '', pubkey: str = '', shard_ids: list[int] | None = None, shard_count: int = 1, intents: Intents | None = None, plugins: list[str] | None = None, oauth: dict[str, str | None] | None = None, **kwargs: Any*)

2.3.2 Plugins

class novus.ext.client.**Plugin**(**args: Any, **kwargs: Any*)

The base command class for any novus client commands.

async on_load() → None

A function to be run when the plugin is loaded. The plugin is not automatically loaded on its addition to the bot, but on its connection to the gateway. If the bot does not connect to the gateway, this function will never be called.

async on_unload() → None

A function to be run automatically where the plugin is removed from the bot. This function is not run if there is not an event loop running.

dispatch(*event_name: str, *args: Any, **kwargs: Any*) → None

Send data to an event within the plugin.

2.3.3 Commands

novus.ext.client.**command**(*name: str | None = None, description: str | None = None, type: int = 1, *, options: list[n.ApplicationCommandOption] | None = None, default_member_permissions: n.Permissions | None = None, dm_permission: bool = True, nsfw: bool = False, guild_ids: list[int] | None = None, cls: Type[Command] = <class 'novus.ext.client.command.Command'>, **kwargs: Any*) → Callable[[CommandCallback], *Command*]

Wrap a function in a command.

Parameters

- **name** (*str | None*) – The name of the command. If not provided, the name of the function is used. If a name with spaces is given, then it is automatically implemented with subcommands (unless it is not a `chat_input` type).
- **description** (*str | None*) – The description associated with the command. If not provided, the docstring for the function is used. If the command is built as a subcommand, you can give descriptions and localizations using the *novus.ext.client.CommandDescription* class.
- **type** (*int*) – The type of the command that you want to create.

  See also:

  novus.ApplicationCommandType
- **options** (*list[novus.ApplicationCommandOption]*) – A list of options to be added to the slash command. If the option names and the function parameter names don't match up, an error will be raised.
- **default_member_permissions** (novus.Permissions) – The permissions that are required (by default) to run this command. These can be changed by server admins.
- **dm_permission** (*bool*) – Whether the command can be run in DMs.
- **nsfw** (*bool*) – Whether the comamnd is set to only work in NSFW channels or not.
- **guild_ids** (*list[int]*) – The guilds that the command will be added to. If not set, then the command will be added globally.

Raises

- **ValueError** – The command is missing a description.
- **Exception** – The command's parameters and the options don't match up.

Examples

```
@client.command()
async def ping(self, ctx):
    '''Command description goes here.'''
    await ctx.send("Pong")
```

```
@client.command(
    options=[
        novus.ApplicationCommandOption(
            name="user",
            type=novus.ApplicationOptionType.USER,
            description="The user you want to mention.",
        )
    ]
)
async def mention(self, ctx, user):
    '''Command description goes here.'''
    await ctx.send(user.mention)
```

class novus.ext.client.**Command**(*name: str*, *type: int*, *application_command: n.PartialApplicationCommand*, *callback: CommandCallback*, *guild_ids: list[int]*)

A command object for Novus command handling.

Parameters

- **name** (*str*) – The name of the command
- **type** (*int*) – The type of the command.

  See also:

  novus.ApplicationCommandType
- **callback** (*CommandCallback*) – The function that acts as the command.
- **application_command** (novus.PartialApplicationCommand) – The application command used to generate the command.
- **guild_ids** (*list[int]*) – A list of guild IDs that the command is present in.

name

The name of the command

Type
str

type

The type of the command.

See also:

novus.ApplicationCommandType

Type
int

`callback`

The function that acts as the command.

Type
CommandCallback

`application_command`

The application command used to generate the command.

Type
novus.PartialApplicationCommand

`guild_ids`

A list of guild IDs that the command is present in.

Type
list[int]

`command_ids`

A list of IDs that refer to the command.

Type
set[int]

`is_subcommand`

Whether or not the command is implemented as a subcommand.

Type
bool

`to_application_command_option()` → *ApplicationCommandOption*

Convert this instance of the command into a command option. This should only be used on subcommands.

Returns
Although the command was generated as an application command, this will convert that application command into an option.

Return type
novus.ApplicationCommandOption

`add_id`(*guild_id: int | None, id: int*) → None

Add an ID to the command. This means that any interaction invokations with the given command ID will be routed to this command instance.

Parameters
`id` (*int*) – The ID that you want to add.

`get_mention`(*guild_id: int | None = None*) → str

Get a mention for the given command.

Parameters
`guild_id` (*int | None*) – The ID of the guild that the command exists in.

Returns
A mention for the command.

Return type
str

async run(*interaction:* Interaction[ApplicationCommandData] | Interaction[ContextComandData], *options:* *list*[InteractionOption] | *None* = *None*) → None

Run the command with the given interaction.

Parameters

- **interaction** (novus.Interaction) – The interaction that invoked the command.
- **options** (*list*[novus.InteractionOption] | *None*) – The list of options that the command is to be called with. If not provided, then the options are taken from the interaction itself. This is primarily used as a helper for subcommands.

autocomplete(*func: AutocompleteCallback*) → AutocompleteCallback

Add an autocomplete to this command.

Examples

```
@client.command(...)
async def echo(self, ctx: novus.Interaction, text: str):
    ...

@echo.autocomplete
async def echo_autocomplete(self, ctx: novus.Interaction):
    return [
        novus.ApplicationCommandChoice("name", "value"),
        novus.ApplicationCommandChoice("name2", "value2"),
        novus.ApplicationCommandChoice("name3", "value3"),
    ]
```

async run_autocomplete(*interaction:* Interaction[ApplicationCommandData], *options:* *list*[InteractionOption] | *None* = *None*) → None

This interaction has been triggered to autocomplete! Work out what parameter needs autocompleting, and then run that :)

Parameters

interaction (novus.Interaction) – The interaction that needs completing.

class novus.ext.client.**CommandDescription**(*description: str* = *MISSING*, *, *name_localizations: LocType* = *MISSING*, *description_localizations: LocType* = *MISSING*, *default_member_permissions: n.Permissions* = *MISSING*, *dm_permission: bool* = *MISSING*, *nsfw: bool* = *MISSING*, *guild_ids: list[int]* | *None* = *MISSING*, *children: dict[str,* CommandDescription] | *None* = *None*)

A description class to wrap around command groups.

class novus.ext.client.**CommandGroup**(*application_command:* PartialApplicationCommand, *commands:* *Iterable*[Command], *guild_ids: list[int]*)

A group of commands and subcommands.

name

The name of the command.

Type

str

application_command

The application command that builds the command.

Type
novus.PartialApplicationCommand

guild_ids

The IDs of the guilds that the command is set to.

Type
list[int]

command_ids

A list of command IDs that are associated with this command.

Type
list[int]

commands

A dict of names and command objects that make up the child commands of this command group.

Type
dict[str, *novus.ext.client.Command*]

add_description(*d:* CommandDescription) → None

Add a description to the command group.

Parameters
d (novus.ext.client.CommandDescription) – The description that you want to add to the command group.

classmethod from_commands(*commands: Iterable[*Command*], run_checks: bool = True*) → Self

Generate a group from a list of commands.

Parameters

- **commands** (*Iterable[*novus.ext.client.Command*]*) – The commands that will make up the group.
- **run_checks** (*bool*) – Whether to check that all implemented commands have the same basic attributes (dm_permission, NSFW, etc).

Raises
novus.ext.client.CommandError – An error was encountered trying to group the commands together.

async run(*interaction:* Interaction*[*ApplicationCommandData*]*) → None

Run the command with the given interaction.

Parameters
interaction (novus.Interaction) – The interaction that invoked the command.

async run_autocomplete(*interaction:* Interaction*[*ApplicationCommandData*]*) → None

Run the autocomplete for the given command with the given options.

Parameters
interaction (novus.Interaction) – The interaction that invoked the autocomplete.

add_id(*guild_id: int | None, id: int*) → None

Add an ID to the command. This means that any interaction invokations with the given command ID will be routed to this command instance.

Parameters
id (*int*) – The ID that you want to add.

autocomplete(*func: AutocompleteCallback*) → AutocompleteCallback

Add an autocomplete to this command.

Examples

```
@client.command(...)
async def echo(self, ctx: novus.Interaction, text: str):
    ...

@echo.autocomplete
async def echo_autocomplete(self, ctx: novus.Interaction):
    return [
        novus.ApplicationCommandChoice("name", "value"),
        novus.ApplicationCommandChoice("name2", "value2"),
        novus.ApplicationCommandChoice("name3", "value3"),
    ]
```

get_mention(*guild_id: int | None = None*) → str

Get a mention for the given command.

Parameters
guild_id (*int | None*) – The ID of the guild that the command exists in.

Returns
A mention for the command.

Return type
str

to_application_command_option() → *ApplicationCommandOption*

Convert this instance of the command into a command option. This should only be used on subcommands.

Returns
Although the command was generated as an application command, this will convert that application command into an option.

Return type
novus.ApplicationCommandOption

2.3.4 Events

novus.ext.client.**event**(*predicate: Callable[..., bool] | None = None*) → WEL

class novus.ext.client.**EventListener**(*event_name: str, func: Callable[[...], Awaitable[Any]], predicate: Callable[[...], bool] | None = None*)

An object that listens for an event.

2.3.5 Loops

novus.ext.client.**loop**(*loop_time: float*, *start_behavior:* LoopBehavior = *LoopBehavior.immediate*, *end_behavior:* LoopBehavior = *LoopBehavior.end*, *autostart: bool* = *True*, *wait_until_ready: bool* = *True*) → Callable[[...], *Loop*]

A wrapper to create a loop object.

Parameters

- **loop_time** (*float*) – The number of seconds between each loop. This is not guarenteed to be accurate, but is used internally for the wait function.
- **start_behavior** (LoopBehavior) – How the loop should behave on its start.
- **end_behavior** (LoopBehavior) – How the loop should behave on its end.
- **autostart** (*bool*) – Whether the loop should start immediately when the plugin is loaded.
- **wait_until_ready** (*bool*) – If the plugin should wait until the bot has received the ready payload before beginning its task.

Examples

```
@client.loop(60)
async def every_minute(self):
    self.log.info("Ping")
```

class novus.ext.client.**Loop**(*func: Callable[[], Coroutine[None, None, Any]]*, *loop_time: float*, *start_behavior:* LoopBehavior = *LoopBehavior.immediate*, *end_behavior:* LoopBehavior = *LoopBehavior.end*, *autostart: bool* = *True*, *wait_until_ready: bool* = *True*)

A piece of code that should be looped within a plugin.

func

The function that is part of the loop.

Type
Callable[…, Awaitable[Any]]

loop_time

The number of seconds between each loop iteration.

Type
float

start_behavior

The behaviour for how each loop will end.

Type
novus.ext.client.LoopBehaviour

end_behavior

The behaviour for how each loop will end.

Type
novus.ext.client.LoopBehaviour

autostart

If the plugin should start automatically.

Type
bool

wait_until_ready

If the loop should wait until the bot receives the ready payload.

Type
bool

owner

The plugin where the loop resides.

Type
novus.ext.client.Plugin

task

The last instance of the given function that the bot ran in this loop.

Type
asyncio.Task | None

before(*func: Callable[[], Coroutine[None, None, Any]]*) → None

Set a function that is to run before the loop starts.

start(**args: Any, **kwargs: Any*) → None

Start the loop.

stop() → None

Stop the loop. This will not stop the currently running task instance, if one is running.

See also:

novus.ext.client.Loop.cancel()

cancel() → None

Stop the loop and cancel any running tasks.

See also:

novus.ext.client.Loop.stop()

class novus.ext.client.**LoopBehavior**(*value*)

How a loop should behave on its ending.

end

The previous loop should have finished executing before a new loop is triggered to start.

immediate

A new loop should be triggered immediately after the previous one is triggered.

2.3.6 Errors

class novus.ext.client.**CommandError**

Base error for all other command errors.

CHAPTER

THREE

CONFIGURATION

Novus uses a few environment variables to configure itself. These support `.env` files via use of `python-dotenv`. These variables change how the library works at a base level, such as changing routes.

- `NOVUS_API_URL`: The base URL for the API. Defaults to `https://discord.com/api/v10`
- `NOVUS_CDN_URL`: The base URL for the CDN. Defaults to `https://cdn.discordapp.com`
- `NOVUS_GATEWAY_URL`: The base URL for the gateway. Defaults to `wss://gateway.discord.gg/?v=10&encoding=json`

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W